



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

НАУКОВІ ЗАПИСКИ НАУКМА

КОМП'ЮТЕРНІ НАУКИ

Том 7 ♦ 2024

Науковий журнал ♦ Щорічник ♦ Заснований у 1996 р.

Київ
2024

**Національний університет «Києво-Могилянська академія»
заснував видання «Наукові записки НаУКМА» 1996 року**

«Наукові записки НаУКМА. Комп'ютерні науки» (англ. NaUKMA Research Papers. Computer Science) — науковий рецензований журнал відкритого доступу, що висвітлює результати наукових досліджень у галузі комп'ютерних наук та інформаційних технологій.

До 2018 р. виходив друком як частина багатосерійного видання «Наукові записки НаУКМА» (серія «Комп'ютерні науки»). З 2018 р. — «Наукові записки НаУКМА. Комп'ютерні науки». Виходить раз на рік.

Пропонований том висвітлює творчі здобутки вчених, аспірантів, студентів факультету інформатики Національного університету «Києво-Могилянська академія», а також фахівців з інших наукових центрів України, що співпрацюють з НаУКМА в галузях інформатики, кібернетики, програмування, за останній рік. Дослідження авторів статей спрямовані на розв'язання теоретичних проблем і практичних завдань у галузі сучасної інформатики.

Журнал адресовано науковцям, фахівцям, викладачам, докторантам, аспірантам, студентам, а також усім зацікавленим у наукових дослідженнях у галузі комп'ютерних наук та інформаційних технологій.

Мови видання: українська, англійська

Редакційна колегія

Глибовець Микола Миколайович, доктор фізико-математичних наук, професор кафедри інформатики (НаУКМА) — головний редактор

Кириєнко Оксана Валентинівна, старший викладач кафедри інформатики (НаУКМА) — відповідальний секретар
Анісімов Анатолій Васильович, доктор фізико-математичних наук, професор, декан факультету кібернетики (Київський національний університет імені Тараса Шевченка)

Буй Дмитро Борисович, доктор фізико-математичних наук, професор, старший науковий співробітник (Київський національний університет імені Тараса Шевченка)

Глибовець Андрій Миколайович, доктор технічних наук, доцент, декан факультету інформатики (НаУКМА)

Горлач Сергій, Dr. rer. nat., професор (Університет Мюнстера, Німеччина)

Дорошенко Анатолій Юхимович, доктор фізико-математичних наук, професор (НаУКМА)

Іваненко Віктор Іванович, доктор технічних наук, професор (НТУУ «КПІ» імені Ігоря Сікорського)

Кондратенко Юрій Пантелійович, доктор технічних наук, професор (Чорноморський державний університет імені Петра Могили)

Малашонюк Геннадій Іванович, доктор фізико-математичних наук, професор, завідувач кафедри мережних технологій (НаУКМА)

Тереженко Василь Миколайович, доктор фізико-математичних наук, професор, завідувач кафедри математичної інформатики, професор (Київський національний університет імені Тараса Шевченка)

Фісун Микола Тихонович, доктор технічних наук, професор (Чорноморський державний університет імені Петра Могили)

Шкільняк Степан Степанович, доктор фізико-математичних наук, професор кафедри теорії та технології програмування (Київський національний університет імені Тараса Шевченка)

Здійснюється подвійне анонімне рецензування матеріалів

Засновник і видавець:

Національний університет
«Києво-Могилянська академія»

Ідентифікатор у Реєстрі суб'єктів у сфері
медіа: R40-04349

Внесено до Переліку наукових фахових видань України, в яких можуть публікуватися результати дисертаційних робіт на здобуття наукових ступенів доктора наук, кандидата наук і ступеня доктора філософії, категорія «Б» (наказ МОН України від 02.07.2020 № 886)

ПЕРЕДМОВА

Маємо честь представити новий випуск наукового журналу «Наукові записки НаУКМА. Комп'ютерні науки», який висвітлює наукові здобутки викладачів, аспірантів і магістрів факультету інформатики Національного університету «Києво-Могилянська академія» у сферах інформатики, кібернетики та програмування.

У цьому збірнику розглянуто теоретичні аспекти програмної інженерії, а також практичні застосування програмних технологій у сучасних системах штучного інтелекту та інтелектуальних інформаційних системах. Цей випуск є результатом наполегливих досліджень наших науковців і відображає наше прагнення до розширення меж знань у цих стратегічно важливих галузях.

Ми щиро вдячні керівництву НаУКМА за їхню щедрі фінансову та організаційну підтримку, завдяки якій це видання стало можливим. Також висловлюємо подяку всім авторам за їхній внесок та співпрацю.

Голова редколегії збірника
М. М. Глибовець

Олецький О. В., Моголівський В. О.

КООРДИНАЦІЯ МІКРОСЕРВІСІВ ІЗ ВИКОРИСТАННЯМ МАШИН СТАНІВ

У статті розглянуто підхід до координації мікросервісів на основі використання машин станів як абстракції для управління розподіленим потоком виконання програми з мікросервісною архітектурою. Реалізовано прототип бібліотеки, загальну архітектуру і принципи роботи якого описано в статті.

У разі застосування описаного підходу полегшується та упорядковується робота з паралельними потоками виконання і довготривалими фоновими задачами. Крім того, покращується прозорість процесів, що полегшує їх моніторинг, візуалізацію, тестування і відлагодження. Для використання машин станів у мікросервісному середовищі реалізовано інтеграцію з брокером повідомлень і належне довготривале зберігання їхнього стану.

Ключові слова: мікросервісна архітектура, машини станів, координація мікросервісів, подійно-орієнтований підхід, управління потоком виконання, моніторинг мікросервісів, візуалізація мікросервісів, тестування мікросервісів.

1. Вступ і постановлення задачі

Від моменту своєї появи у 2011 р. [6] мікросервісна архітектура набула значного поширення завдяки масштабованості, гнучкості та надійності. Згідно з опитуванням, проведеним О'Reilly серед своїх підписників у 2020 р., 77 % респондентів зазначили, що їхні компанії використовують мікросервісну архітектуру [7]. Більшість респондентів, а саме 55 %, вважають, що впровадження мікросервісів є доволі успішним, однак зауважили про труднощі, з якими доводиться стикатися [7]. На складність управління багатьма сервісами скаржаться 29 % опитаних, на непрозорість і труднощі у моніторингу — 27 %, на клопоти з тестуванням — 20 %, на складність відлагодження — 16 % [7]. Інші дослідження також виявляють схожі проблеми [5; 9–11].

Отже, актуальною є проблема пом'якшення та/або усунення цих та інших проблем, характерних для мікросервісної архітектури. Якщо зробити певне узагальнення, то слід звернути увагу на те, що ці проблеми значною мірою пов'язані з недостатнім розвитком засобів координації мікросервісів. У цій царині переважають евристичні підходи.

У цій статті розвинуто підхід до координації мікросервісів на основі використання машин станів.

2. Забезпечення належного функціонування машин станів у мікросервісній архітектурі

Машину станів слід розглядати як абстракцію для управління розподіленим потоком виконання програм [13], і для застосунків із мікросервісною архітектурою це набуває особливого значення. При цьому важливими є питання задовільного вибору моделі та реалізації машини станів, забезпечення механізмів комунікації між мікросервісами і машинами станів, а також забезпечення належного рівня персистентності, тобто довготривале і надійне зберігання як станів розподіленої системи, так і власне машини станів.

Як реалізацію машини станів було обрано бібліотеку XState [13]. Ця бібліотека спирається на класичний теоретичний апарат на основі скінчених автоматів Мілі та Мура, але доповнює їх підходами, запропонованими Девідом Харелем, Граді Бучем, а також використовує W3C стандарт SCXML [13]. Ідеться про механізми на основі вкладених і паралельних (складених) станів, внутрішніх і зовнішніх подій, умовних переходів, контекстної залежності (по суті — на основі пам'яті) та навіть побічних ефектів [3; 4; 12]. Маючи таку багату функціональність, машини станів на цій основі можуть бути максимально корисними під час розроблення сучасного програмного забезпечення, зокрема на основі мікросервісів.

Для забезпечення комунікації між машинами станів і мікросервісами було вирішено використати подійно-орієнтований підхід, який можна застосовувати як у машинах станів, так і у мікросервісах [8; 12]. Було реалізовано можливість надсилати події з машини станів до мікросервісів і навпаки. Для цього було використано брокер повідомлень Apache Kafka. Такий вибір обґрунтовано двома аспектами. По-перше, брокер повідомлень — це необхідний інструмент у мікросервісній архітектурі, який забезпечує надійність, відновлюваність і масштабованість [6]. По-друге, було використано таку властивість Apache Kafka, як гарантований порядок повідомлень з однаковим ключем партиції [1]. Для усіх подій, пов'язаних із певною машиною станів, як ключ партиції встановлюється унікальний ідентифікатор машини станів. У такий спосіб певною мірою забезпечується порядок повідомлень, що суттєво спрощує опис усіх потенційних подій для кожного зі станів у машині станів.

Читання повідомлень (подій) може відбуватися в одному з двох режимів, а саме зі збереженням відступу від початку черги для кожної події або для групи подій. Перший режим спрямований на максимальну надійність, адже за критичного збою доведеться повторно обробляти тільки одну подію. Другий режим дає змогу пришвидшити роботу системи, зберігаючи відступ після оброблення певної кількості подій, проте призводить до надмірної складності у разі критичного збою. Саме задля подолання цієї складності режимом за замовчуванням було обрано перший режим — збереження відступу для кожної події. Цей режим дає змогу значно легше зробити машину станів, наскільки це можливо, ідемпотентною, оскільки значно легше забезпечити ідемпотентність для кожної окремо взятої події, ніж для серії подій. Якби брокер повідомлень міг гарантувати семантику доставлення повідомлень «рівно один раз», то не було б потреби забезпечувати ідемпотентність оброблення подій, проте жодний із на сьогодні відомих комерційних брокерів повідомлень не надає такої гарантії у контексті мікросервісної архітектури. А вона необхідна для забезпечення відмовостійкості.

Для довготривалого зберігання стану машини станів було використано багатомодельну СУБД ArangoDB [2]. У ній зберігається метайнформація про машину станів і її серіалізований стан. У принципі можна використовувати будь-яку СУБД, але вибір саме ArangoDB пов'язаний із її перспективністю для подальших досліджень. Ця перспективність полягає у багатомодельності цієї бази даних і наявній підтримці у ній графів [2]. Машини станів легко моделюються графами, що може дати змогу додатково їх аналізувати для надання корисних порад розробникам.

Отже, для використання машин станів у мікросервісній архітектурі було створено бібліотеку з архітектурою, зображеною на рис. 1.

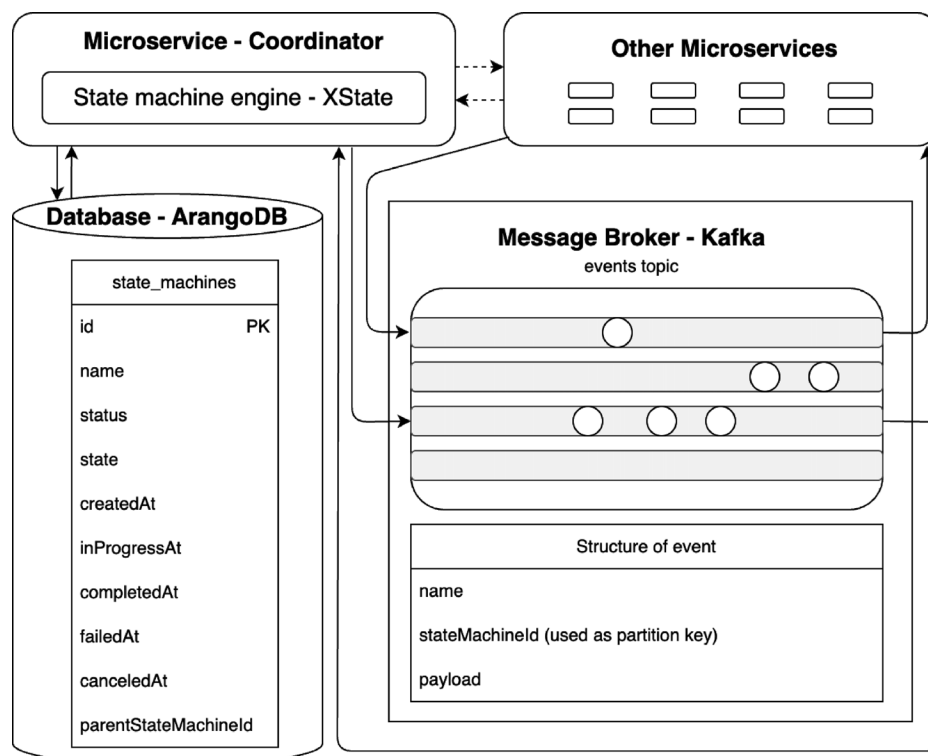


Рис. 1. Реалізація мікросервісної архітектури з використанням машини станів

У базі даних ArangoDB зберігається представлення машини станів, яке складається з унікального ідентифікатора, назви, статусу, стану та ідентифікатора батьківської машини станів (якщо така є). Також зберігається дата і час створення, запуску, помилки, завершення чи скасування виконання машини станів. Поле «назва» по суті зберігає тип машини станів, як-от `GDPRCleanUpMachine` чи `MonthlyReportMachine`. З назвою пов'язана визначена розробником машина станів (стани, переходи, дії тощо). Статус зберігає високорівневий стан виконання, він може мати такі значення, як `NEW`, `IN_PROGRESS`, `COMPLETED`, `FAILED`, `CANCELLED`. У полі «стан» зберігається серіалізований стан, створений бібліотекою `XState`. У ньому міститься стан, у якому перебуває машина, поточний контекст (об'єкт, який є пам'яттю машини станів) та історія попередніх станів.

Під час старту застосунку буде ініціалізовано екземпляр машини станів, після чого його буде збережено у базу даних із початковим станом. Далі брокеру повідомлень буде надіслано подію `STATE_MACHINE_START`, яку пізніше прочитає один із мікросервісів, який буде у ролі координатора; він десеріалізує машину станів із бази даних в оперативну пам'ять і надішле їхню початкову подію. Машина станів почне своє виконання, під час якого вона змінюватиме стани й надсилатиме події для інших мікросервісів до брокера. Вона продовжуватиме своє виконання, допоки не досягне кінцевого стану або не вичерпає усі можливі переходи. Коли це станеться, вона буде серіалізована та збережена у базу даних.

Якщо усі можливі переходи вичерпано, а кінцевого стану не досягнуто — це означає, що машина очікує на події від інших мікросервісів, якщо, звісно, розробник не припустився помилок при оголошенні машини станів. Коли інші мікросервіси оброблять раніше надіслані події з машини станів мікросервісом-координатором, вони надішлють свою відповідь у вигляді подій аналогічних до першої розглянутої події `STATE_MACHINE_START`, так коло замкнеться: машину станів буде знов десеріалізовано й запущено. Цей цикл відбуватиметься до того моменту, поки машина станів не досягне свого кінцевого стану.

3. Переваги застосування машин станів у мікросервісній архітектурі

Центральною ідеєю мікросервісної архітектури є розподіл логіки застосунку між мікросервісами. Такий підхід підвищує гнучкість, масштабованість і надійність системи, але водночас і ускладнює управління даними та координацію процесів.

Прикладом типової задачі, для якої проявляються зазначені проблеми, є видалення даних для дотримання регламенту GDPR. При проєктуванні мікросервісів основну увагу приділяють забезпеченню високої продуктивності системи, що є звісно цілком виправданим, але це своєю чергою призводить до розпорошення даних між мікросервісами. Через це виникають складнощі при імплементації дотримання GDPR, оскільки дані можуть бути розкидані по багатьох мікросервісах. Інформація про те, які дані мають бути видалені, може міститися на одному наборі мікросервісів, а самі дані на іншому. Тому може постати потреба у забезпеченні розподіленого потоку виконання між мікросервісами, за якого спершу буде зібрано інформацію про дані, які потребують очищення, а уже потім буде видалено дані з усіх мікросервісів, які їх містять. Приклад такого потоку виконання зображено на рис. 2.

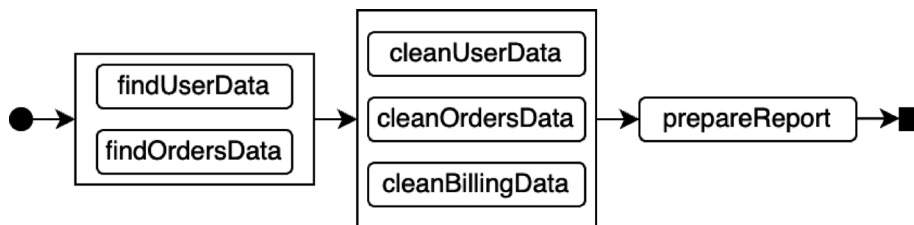


Рис. 2. Приклад потоку виконання мікросервісного застосунку при видаленні даних згідно з GDPR

Спершу необхідно знайти інформацію, яка потребує видалення. Для цього, як правило, потрібно залучити декілька мікросервісів. Далі необхідно дочекатися, поки кожен із залучених мікросервісів виконає пошук. Коли пошук буде завершено, необхідно здійснити видалення даних, у якому теж, як правило, мають бути залучені декілька мікросервісів. Більше того, процес очищування може бути тривалим і відбуватися протягом декількох годин, а то й днів. І тільки коли всі залучені мікросервіси видалять дані, можна перейти до завершальних дій, наприклад формування звіту. За це завдання, як правило, відповідає тільки один із мікросервісів.

Для вирішення таких задач типовим є застосування подійно-орієнтованого підходу, за якого мікросервіси надсилають події, щоб обмінюватися інформацією для координації своєї роботи. За такого підходу перед розробниками постають дві основні проблеми:

1. Координація очікування мікросервісами один одного:
 - а) один чекає на багатьох;
 - б) багато чекають на одного;
 - в) багато чекають на багатьох.
2. Забезпечення контролю та прозорості поточного статусу системи, особливо для довготривалих завдань.

Доповнення подійно-орієнтованого підходу машинами станів може сприяти вирішенню цих проблем, оскільки сучасні реалізації машин станів легко оперують вкладеними та паралельними (складеними станами), а також чітко зберігають поточний стан, що вирішує першу та другу проблеми відповідно.

Іншою значною перевагою запропонованого підходу є наявність візуалізаторів машин станів, які дають змогу уявляти логіку застосунку у формі схем та чітко ідентифікувати поточний стан системи. Така наочність значно збільшує зрозумілість логічних потоків і станів системи. Графічні діаграми станів можуть слугувати відмінною технічною документацією, яка ніколи не втрачає актуальності, оскільки автоматично генерується з програмного коду. Така документація може бути особливо корисною для нових розробників у команді, яким зазвичай доводиться витратити багато часу для ознайомлення з усіма наявними подіями у подійно-орієнтованому підході. Для розуміння подій і взаємозв'язків між ними доводиться повністю перечитувати програмний код їх опрацювання. Натомість із діаграмами станів вивчати взаємозв'язки між подіями значно легше. Також візуальне представлення може значно посприяти покращенню комунікації між різними учасниками проєкту. Воно буде корисним не тільки для технічних, а й для нетехнічних спеціалістів: проєктних менеджерів, бізнес-аналітиків, спеціалістів із підтримки та ін. Особливо корисною візуалізація буде для тестувальників, які зможуть чітко бачити, у якому стані перебувала система під час виявлення вади, та зможуть легко ділитися цією інформацією з розробниками. Приклад такої візуалізації зображено на рис. 3. Її було автоматично згенеровано з програмного коду.

Візуалізація показує діаграму станів машини станів, яка містить спрощений приклад процесу очищення даних. Спершу відбувається збір інформації про дані, які мають бути видалені, це робить один із мікросервісів. Далі паралельно відбуваються два процеси: видалення даних користувачів і видалення платіжної інформації, пов'язаної з цими користувачами. Цими задачами займаються два різні мікросервіси відповідно до своєї предметної області. Коли обидва завершать виконання, буде сформовано звіт і машина станів перейде у стан «completed». Також наявне оброблення помилок на кожному з кроків виконання (станів). У разі помилки інформацію про помилку буде записано в контекст машини станів (внутрішню пам'ять), і машина завершить своє виконання у стані «errored». Отже, усі процеси, які відбуваються під час очищення даних, є наочними та зрозумілими. Чим більші масштаби процесів, які відбуваються між мікросервісами, тим кориснішими будуть такі візуалізації.

Також запропонований підхід покращує можливості модульного і ручного тестування. Перевіряти взаємодію між мікросервісами досить складно. Найчастіше для досягнення цієї мети використовують інтеграційне або системне тестування, яке є значно складнішим, ніж модульне тестування. Покрити модульними тестами інтеграційну взаємодію мікросервісів досить складно, оскільки модульні тести будуть розкидані по всіх мікросервісах і будуть ніяк не пов'язані між собою, що робить їх написання та підтримку затратним завданням. Та якщо взаємодія мікросервісів буде представлена машиною станів, тоді таку взаємодію досить легко протестувати модульними тестами, оскільки машина станів є незалежною сутністю, яку може бути протестовано окремо (звісно, за умови використання заглушок для логіки, яка виходить за рамки машини станів).

Також виконання машини станів може бути симульовано, тобто можливе ручне тестування різних сценаріїв подій без побічних ефектів та необхідності запуску мікросервісів. Запуск машини станів у такому режимі може зекономити багато часу розробників, оскільки вони зможуть переконатися, що їхній логічний ланцюг потоку виконання програми правильний і не містить помилок. За використання традиційного подійно-орієнтованого підходу такого домогтися дуже складно; у переважній більшості випадків, щоб протестувати взаємодію між всіма мікросервісами, вони мають бути запуснені. Водночас у разі використання машин станів переважну частину взаємодії можна перевірити, тільки симулюючи виконання машини станів, без необхідності запуску мікросервісів.

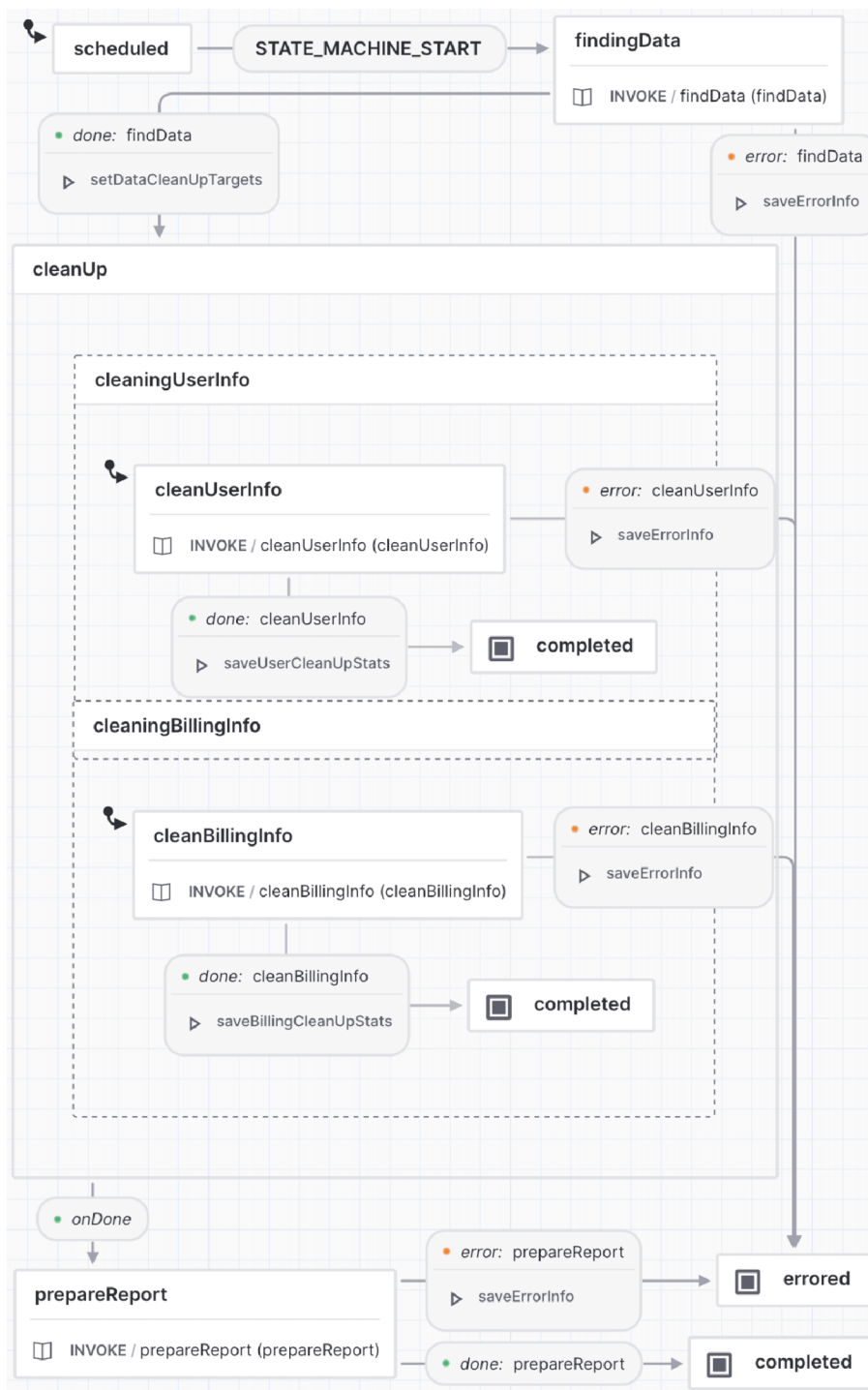


Рис. 3. Візуалізація машини станів, автоматично згенерована за допомогою візуалізатора бібліотеки XState

4. Висновки й обговорення

У статті описано підхід до інтеграції машин станів і мікросервісів із метою управління потоком виконання застосунку з мікросервісною архітектурою. Було створено бібліотеку, яка інтегрує між собою три ключові компоненти: реалізацію машин станів XState, брокер повідомлень Kafka і багатомодельну базу даних ArangoDB.

Створено також прототип, який ілюструє використання створеної бібліотеки для низки типових задач. Роботу прототипу проілюстровано на прикладі задачі видалення даних відповідно до регламенту GDPR. Показано шляхи до вирішення деяких із цих проблем в рамках розглянутої архітектури.

Запропонований підхід спрощує розроблення логіки виконання мікросервісного застосунку, а саме полегшує та упорядковує роботу з паралельними потоками виконання та довготривалими фоновими задачами. Крім того, він покращує прозорість процесів у мікросервісній архітектурі, а саме їх моніторинг, візуалізацію, тестування та відлагодження.

Водночас необхідно звернути увагу на певні проблемні аспекти, пов'язані з удосконаленням і практичним використанням розробленої бібліотеки. Очевидною є потреба в розширенні спектра типових задач, для яких її може бути застосовано. Вище йшлося про те, що здійснюється оброблення кожної окремої події, але це може створювати потенційні ускладнення при обробленні логічно пов'язаних послідовностей подій, тобто транзакцій, особливо за високого рівня завантаженості. Корисним видається також підвищення рівня формалізації моделей, що лежать в основі підходу. Все це має стати предметом подальших досліджень.

Список літератури

1. Apache Kafka documentation [Electronic resource]. — Mode of access: <https://kafka.apache.org/documentation/> (date of access: 01.06.2024). — Title from screen.
2. ArangoDB documentation [Electronic resource]. — Mode of access: <https://docs.arangodb.com/stable/> (date of access: 01.06.2024). — Title from screen.
3. Booch G. Unified modeling language user guide, the (2nd edition) (the addison-wesley object technology series) / Grady Booch, James Rumbaugh, Ivar Jacobson. — 2nd ed. — [S. l.] : Addison-Wesley Professional, 2005. — 496 p.
4. Harel D. Statecharts: a visual formalism for complex systems [Electronic resource] / David Harel // Science of computer programming. — 1987. — Vol. 8, no. 3. — Pp. 231–274. — [https://doi.org/10.1016/0167-6423\(87\)90035-9](https://doi.org/10.1016/0167-6423(87)90035-9).
5. Kurian George Cheripurathu. Integrating microservices and microfrontends: a comprehensive literature review on architecture, design patterns, and implementation challenges [Electronic resource] / Kurian George Cheripurathu, Sanjeev Kulkarni // Journal of scientific research and technology. — 2024. — Pp. 1–12. — <https://doi.org/10.61808/jsrt115>.
6. Lewis J. Microservices [Electronic resource] / James Lewis, Martin Fowler. — Mode of access: <https://martinfowler.com/articles/microservices.html> (date of access: 01.06.2024). — Title from screen.
7. Loukides M. Microservices adoption in 2020 [Electronic resource] / Mike Loukides, Steve Swoyer // O'Reilly Media. — Mode of access: <https://www.oreilly.com/radar/microservices-adoption-in-2020/> (date of access: 02.07.2024). — Title from screen.
8. Michelson B. Event-Driven architecture overview [Electronic resource] / Brenda Michelson. — Boston, MA: Patricia Seybold Group, 2006. — <https://doi.org/10.1571/bda2-2-06cc>.
9. Microservices in practice: a survey study / Markos Viggiano [et al.] // VI workshop on software visualization, evolution and maintenance, São Carlos, 19 September 2018. — [S. l.].
10. Montelius A. An exploratory study of micro frontends [Electronic resource]: thesis / Montelius Anna. — [S. l.], 2021. — Mode of access: <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-176963> (date of access: 11.09.2024). — Title from screen.
11. Söylemez M. Challenges and solution directions of microservice architectures: a systematic literature review [Electronic resource] / Mehmet Söylemez, Bedir Tekinerdogan, Ayça Kolukısa Tarhan // Applied sciences. — 2022. — Vol. 12, no. 11. — Pp. 5507. — <https://doi.org/10.3390/app12115507>.
12. State chart XML (SCXML): state machine notation for controlabstraction [Electronic resource] // W3C. — Mode of access: <https://www.w3.org/TR/scxml/> (date of access: 01.06.2024). — Title from screen.
13. XState documentation [Electronic resource] // XState — JavaScript State Machines and Statecharts. — Mode of access: <https://xstate.js.org/docs/> (date of access: 01.06.2024). — Title from screen.

References

- Apache Kafka documentation. (n.d.). Apache Kafka. <https://kafka.apache.org/documentation/>.
- ArangoDB documentation. (n.d.). ArangoDB. <https://docs.arangodb.com/stable/>.
- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified modeling language user guide, the (2nd edition) (the addison-wesley object technology series)* (2nd ed.). Addison-Wesley Professional.
- Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8 (3), 231–274. [https://doi.org/10.1016/0167-6423\(87\)90035-9](https://doi.org/10.1016/0167-6423(87)90035-9).
- Kurian, George Cheripurathu, & Kulkarni, Sanjeev. (2024). Integrating microservices and microfrontends: A comprehensive literature review on architecture, design patterns, and implementation challenges. *Journal of Scientific Research and Technology*, 1–12. <https://doi.org/10.61808/jsrt115>.
- Lewis, J., & Fowler, M. (2014, March 25). *Microservices*. [martinfowler.com](https://martinfowler.com/articles/microservices.html). <https://martinfowler.com/articles/microservices.html>.
- Loukides, M., & Swoyer, S. (2020, July 15). *Microservices adoption in 2020*. O'Reilly Media. <https://www.oreilly.com/radar/microservices-adoption-in-2020/>.
- Michelson, B. (2006). *Event-Driven architecture overview*. Patricia Seybold Group. <https://doi.org/10.1571/bda2-2-06cc>.
- Montelius, A. (2021). *An exploratory study of micro frontends* [Thesis, Linköpings universitet, Programvara och system]. <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-176963>.
- Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Challenges and solution directions of microservice architectures: A systematic literature review. *Applied Sciences*, 12 (11), 5507. <https://doi.org/10.3390/app12115507>.
- State chart XML (SCXML): State machine notation for controlabstraction. (n.d.). W3C. <https://www.w3.org/TR/scxml/>.
- Viggiano, M., Terra, R., Rocha, H., Tulio Valente, M., & Figueiredo, E. (n. d.). Microservices in practice: A survey study. In *VI workshop on software visualization, evolution and maintenance*.
- XState documentation. (n.d.). XState — JavaScript State Machines and Statecharts. <https://xstate.js.org/docs/>.

O. Oletsky, V. Moholivskyi

COORDINATION OF MICROSERVICES USING STATE MACHINES

The article describes the use of state machines to coordinate microservices. The lack of well-developed microservice coordination tools presents a significant number of challenges for developers building applications with a microservice architecture. Among these challenges are the difficulty of managing multiple services and their distributed execution flow, the lack of transparency in monitoring and debugging, complications in testing, and other issues. These challenges can be addressed by utilizing state machines.

State machines serve as an abstraction to control the distributed execution flow of a program in a microservice architecture. This approach simplifies working with parallel execution flows and long-running background tasks. It also improves process transparency, as well as monitoring, visualization, testing, and debugging.

Integration with a message broker and adequate long-term state storage has been implemented to use state machines in a microservice environment. The paper describes the developed library, which connects three key components required for the effective functioning of state machines within microservices. These components include the XState implementation of the state machine, the Kafka message broker, and the ArangoDB multi-model database. Additionally, a prototype has been created to illustrate the usage of the developed library for a number of typical tasks. The prototype showcases the process of deleting data in accordance with GDPR regulations.

The proposed approach simplifies the development of execution logic in a microservice application, specifically by facilitating and organizing the management of parallel execution flows and long-running background tasks. In addition, it improves process transparency in the microservice architecture, namely in monitoring, visualization, testing, and debugging.

Keywords: microservice architecture, state machines, coordination of microservices, event-oriented approach, execution flow management, monitoring of microservices, visualization of microservices, testing of microservices.

Матеріал надійшов 12.08.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

СИСТЕМА ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ НОРМАЛІЗАЦІЇ БАЗ ДАНИХ

У статті розкрито поняття процесу нормалізації баз даних, проведено аналіз наявних інструментів для нормалізації, виділено основні їхні переваги та недоліки. Описано функціонал і реалізацію нової системи для автоматичної нормалізації структури бази даних з урахуванням недоліків наявних систем.

Ключові слова: реляційні бази даних, процес нормалізації, цілісність баз даних, функціональні залежності, алгоритм декомпозиції, властивість з'єднання без втрат, властивість збереження залежностей.

Вступ

У сучасному світі обсяг даних зростає з неймовірною швидкістю, і їхнє ефективне управління стає дедалі важливішим завданням для організацій різних галузей. Реляційні бази даних є одним із ключових інструментів для збереження даних для проєктів, де є важливою структурованість даних, забезпечення цілісності даних, а також підтримка складних запитів і транзакцій. Прикладами таких систем можуть слугувати фінансові системи, системи охорони здоров'я, системи електронної комерції, CRM-системи, ERP-системи та багато інших.

Правильне проєктування структури бази даних відіграє ключову роль у забезпеченні її ефективності, продуктивності та цілісності. Одним із основних методів досягнення цих цілей є нормалізація бази даних.

Процес нормалізації баз даних допомагає організувати дані так, щоб зменшити надлишковість даних, яка потенційно може призвести до виникнення аномалій оновлення, що своєю чергою спричиняє порушення цілісності даних. Цей процес сприяє зменшенню обсягу бази даних і витрат на зберігання даних, підвищенню швидкості виконання запитів і ефективності роботи бази даних.

Однак цей процес є складним завданням, що потребує глибоких знань і досвіду фахівців, тому є актуальним використання систем для автоматизації цього процесу. Сьогодні на ринку є недостатня кількість таких інструментів, тож у цій роботі запропоновано нову систему для реалізації процесу нормалізації бази даних.

Теоретичні аспекти нормалізації баз даних

Нормалізація — це формальний метод аналізу відношень на основі їхнього первинного ключа (або потенційних ключів) і наявних функціональних залежностей [1].

Процес нормалізації полягає у виконанні декомпозиції схем відношень відповідно до алгоритму декомпозиції на декілька відношень так, щоб кожне з результуючих відношень перебувало в певній нормальній формі. Своєю чергою, нормальна форма — це певний набір обмежень, і відношення перебуває в деякій нормальній формі, якщо задовольняє властивий їй набір обмежень. У теорії реляційних баз даних, як правило, виділяють таку послідовність нормальних форм: 1НФ, 2НФ, 3НФ, Нормальна форма Бойса-Кодда (НФБК), 4НФ, 5НФ [2].

Варто зазначити, що в процесі проєктування баз даних повністю усувати надлишковість даних не потрібно, оскільки при цьому неможливо буде підтримувати базу даних як єдине ціле, а нормальні форми є лише вказівками, а не абсолютними вимогами. Інколи постає необхідність відхилитися від них, щоб задовольнити практичні вимоги бізнесу. Є деякі випадки, коли навмисне порушення правил нормалізації є хорошою практикою.

Базу даних вважають нормалізованою, якщо її відношення перебувають як мінімум у 3НФ чи НФБК.

Одним із алгоритмів, які використовують для процесу нормалізації, є алгоритм декомпозиції, що базується на теоремі Хіса і дає змогу проводити розбиття до 3НФ та НФБК [5]. Цей алгоритм було вибрано для реалізації процесу нормалізації у заданій системі.

Для того щоб упевнитись у правильності декомпозиції, потрібно перевірити, чи отримана декомпозиція задовольняє властивість з'єднання без втрат [5] і властивість збереження залежностей [6]. Властивість з'єднання без втрат гарантує, що будь-який екземпляр початкового відношення може бути відновлено за допомогою відповідних екземплярів декомпованих відношень із використанням лише операцій природного з'єднання. Згаданий вище алгоритм декомпозиції гарантує, що результуюче розбиття задовольнятиме цю властивість. Властивість збереження залежностей гарантує, що обмеження, які накладали на початкове відношення, можна підтримувати, застосовуючи ті самі обмеження до кожного з дрібніших відношень.

Ця система на основі поданих на вхід атрибутів реляції та множини функціональних залежностей реалізує процес пошуку мінімальної множини функціональних залежностей, пошук квазі-ключів, зазначений алгоритм декомпозиції до 3НФ, а також явно проводить перевірку виконання властивості з'єднання без втрат і збереження залежностей.

Огляд наявних систем

На ринку нині є такі системи для нормалізації баз даних, кожна з яких має свої переваги й недоліки:

1. Functional Dependencies Checker [4]. Система реалізована у вигляді вебзастосунку з відкритим програмним кодом з використанням програмної платформи Node.js. Інтерфейс інструменту подано на рис. 1.

Functional Dependencies Checker

Enter Functional Dependencies in the form of $\{a,b,c\} \rightarrow \{d\}$, $\{d\} \rightarrow \{a\}$

$\{a\} \rightarrow \{c,d\}, \{b\} \rightarrow \{c\}, \{b,d\} \rightarrow \{c\}$

Attribute Closure Functional Dependency Closure Minimal Cover

Normal Forms

$\{\{a\} \rightarrow \{c\}, \{a\} \rightarrow \{d\}, \{b\} \rightarrow \{c\}\}$

Created by arjo129. Source code available on [github](#). Found an error? [Report an issue](#)

Рис. 1. Інтерфейс системи Functional Dependencies Checker

Цей застосунок дає можливість обчислити замикання усіх можливих комбінацій атрибутів відносно заданої множини функціональних залежностей, знайти потенційні та суперключі, побудувати замикання заданої множини функціональних залежностей, визначити мінімальну множину функціональних залежностей, а також перевірити, чи реляція перебуває у 2, 3 нормальних формах та НФБК. Проте у цьому застосунку не можна провести декомпозицію реляції до певної нормальної форми.

2. Relational Database Tools [7]. Систему реалізовано у вигляді вебзастосунку з відкритим програмним кодом з використанням мови програмування Java. Інтерфейс інструменту подано на рис. 2.

У цій системі можна обчислити замикання усіх можливих комбінацій атрибутів відносно заданої множини функціональних залежностей, знайти потенційні та суперключі, побудувати замикання заданої множини функціональних залежностей, визначити мінімальну множину функціональних залежностей, перевірити, чи ця реляція перебуває у 2, 3 нормальних формах, НФБК, а також 4 нормальної формі, надає можливість зведення реляції до 3 нормальної форми з виконанням властивостей з'єднання без втрат і збереження залежностей та НФБК без гарантії виконання властивості збереження залежностей. 3-поміж недоліків системи можна виділити незручний інтерфейс користувача й відсутність зведення реляції до 4 нормальної форми.

Enter the relation schema in form R(A,B,C,AB,ABC)

R(A,B,C,D)

Use commas to separate attributes. Spaces are optional. Inputs are case-insensitive.

Enter all given functional dependencies in form A -> B; AB -> C; B,C -> A

A->C,D; B->C; B,D->C

Use commas to separate attributes that belong to the same side of the same functional dependency.
Use semi-colons to separate different functional dependencies.

Enter all given multivalued dependencies in form A ->> B; AB ->> C; B,C ->> A (same as functional dependencies)

Leave blank if there are none.

Calculate

Calculating information for Relation R having attributes: A, B, C, D.
Given input functional dependencies: $A \rightarrow C,D$; $B \rightarrow C$; $B,D \rightarrow C$.
No input multivalued dependencies.

Calculating attribute closures:

$\{A\}^+ = \{A, C, D\}$
 $\{B\}^+ = \{B, C\}$
 $\{C\}^+ = \{C\}$
 $\{D\}^+ = \{D\}$
 $\{A, B\}^+ = \{A, B, C, D\}$ <--- Composite minimum candidate key
 $\{A, C\}^+ = \{A, C, D\}$
 $\{B, C\}^+ = \{B, C\}$
 $\{A, D\}^+ = \{A, C, D\}$
 $\{B, D\}^+ = \{B, C, D\}$
 $\{C, D\}^+ = \{C, D\}$
 $\{A, B, C\}^+ = \{A, B, C, D\}$ <--- Superkey
 $\{A, B, D\}^+ = \{A, B, C, D\}$ <--- Superkey
 $\{A, C, D\}^+ = \{A, C, D\}$
 $\{B, C, D\}^+ = \{B, C, D\}$
 $\{A, B, C, D\}^+ = \{A, B, C, D\}$ <--- Superkey

Found 1 composite minimum candidate key. There are no non-composite minimum candidate keys.
Found 3 superkeys (excluding minimum candidate keys).

List of prime attributes (attributes that are part of a minimum candidate key): A, B.
List of non-prime attributes (attributes that are not part of any minimum candidate key): C, D.

Calculating a minimal cover set (F_min) of functional dependencies from given input: (note that functional dependencies with common left-hand sides have their right-hand sides combined)

Рис. 2. Інтерфейс системи Relational Database Tools

3. Boyce-Codd relation solver [3]. Систему реалізовано у вигляді вебзастосунку з відкритим програмним кодом з використанням мови програмування JavaScript. Інтерфейс інструменту подано на рис. 3.

Relation A,B,C,D
Use "," as separator

Dependencies A->C,D; B->C; B,D->C
Use "," as separator, ">" marks dependency

Calculate

Solution:
R(A,C,D)
R(A,B)

Evaluating R(A,B,C,D)
A->C,D is not BCNF
Decomposing into R1(A,C,D) and R2(A,B)
Projecting FD's onto A,C,D
Evaluating fd's on basis of A
A->C,D
Evaluating fd's on basis of C
Evaluating fd's on basis of A,C
Evaluating fd's on basis of D
Evaluating fd's on basis of A,D
Evaluating fd's on basis of C,D
Projecting FD's onto A,B
Evaluating fd's on basis of A
Evaluating fd's on basis of B

Decomposed result:
R1(A,C,D):
A->C,D
R2(A,B):
None
-- end of iteration --
Evaluating R(A,C,D)
A->C,D is BCNF
Evaluating R(A,B)
no FD's. Relation is BCNF

Рис. 3. Інтерфейс застосунку Boyce-Codd relation solver [3]

Функціонал системи дає змогу здійснити лише декомпозицію реляції до НФБК. Усі інші функції, що є доступними у системах, описаних вище, не передбачено.

Реалізація логіки системи

Для реалізації серверної частини системи було використано програмну платформу Node.js.

Основна частина коду, яка створює необхідні для процесу нормалізації об'єкти, складається з декількох класів: Relation, Dependency, Data та NormalizationData. Кожен клас призначений для оброблення певної частини представлення схеми бази даних і операцій, які мають бути виконані в процесі нормалізації.

Серверна частина системи використовує маршрутизатори фреймворку Express.js для ефективного управління HTTP-запитами.

Маршрутизатор CalculationRouter, створений як окремий клас, використовується для групування всіх маршрутів запитів, пов'язаних з обчисленнями нормалізації. Цей маршрутизатор прив'язується до маршруту «/calc».

Кожен маршрут у CalculationRouter налаштовано на оброблення POST-запитів для різних типів операцій нормалізації:

- /min — обчислення мінімальної множини функціональних залежностей;
- /nf — проведення декомпозиції до 3НФ;
- /lossless — перевірка властивості з'єднання без втрат;
- /preserve — перевірка властивості збереження залежностей.

Ці маршрути обробляються відповідними методами в CalculationController, які інкапсулюють логіку для кожного типу обчислень. Сам клас CalculationController містить методи, які безпосередньо взаємодіють із логікою нормалізації, інкапсульованою у класі NormalizedData.

Кожен метод працює за схожим шаблоном:

- отримуються необхідні дані з тіла запиту;
- створюються екземпляри класів Relation і Dependency для кожної реляції і залежності, заданих у запиті;
- створюється екземпляр NormalizedData зі створеними в попередньому пункті реляціями і залежностями;
- виконуються обчислення процесу нормалізації за допомогою методів класу NormalizedData;
- результати переводяться у формат, придатний для відповіді, і вони надсилаються на клієнтську частину для відображення.

```
findMinimalCover() {
  this.#singleAttributeRightPart()
  this.data.dependencies = this.#removeDuplicates(this.data.dependencies)
  this.#removingRedundantDeps()
  this.#removingRedundantLeftDeps()
  while (this.#changedLeft) {
    this.#removingRedundantDeps()
    this.#changedLeft = this.#checkMultipleLefts(this.data.dependencies)
    if (this.#changedLeft) this.#removingRedundantLeftDeps()
  }
}
```

Рис. 4. Метод знаходження мінімальної множини функціональних залежностей

```
get3NF() {
  const minimalCover = this.data.dependencies;
  const attributes = this.#getAllAttrs();
  const quasikey = this.getQuasikey();

  if (!quasikey) {
    console.log("Error: Quasikey not found.");
    return;
  }

  const relations3NF = this.#decomposeTo3NF(minimalCover, attributes, quasikey);
  this.data.relations = relations3NF
}
```

Рис. 5. Метод декомпозиції реляції до 3НФ

Також було проведено автоматизоване тестування системи, що передбачає unit-тестування з використанням фреймворку для тестування JavaScript Jest та ручне тестування коректності роботи застосунку.

Реалізація інтерфейсу користувача

Для реалізації клієнтської частини системи було використано відкриту JavaScript бібліотеку для створення інтерфейсів користувача React.js.

На головній сторінці подано опис можливостей системи, а саме: можливість знаходження мінімальної множини функціональних залежностей, зведення реляції до 3НФ, явна перевірка виконання властивості з'єднання без втрат та збереження залежностей (рис. 6). При натисканні кнопки «Алгоритм» можна побачити опис використаних алгоритмів для реалізації процесу нормалізації (рис. 7).

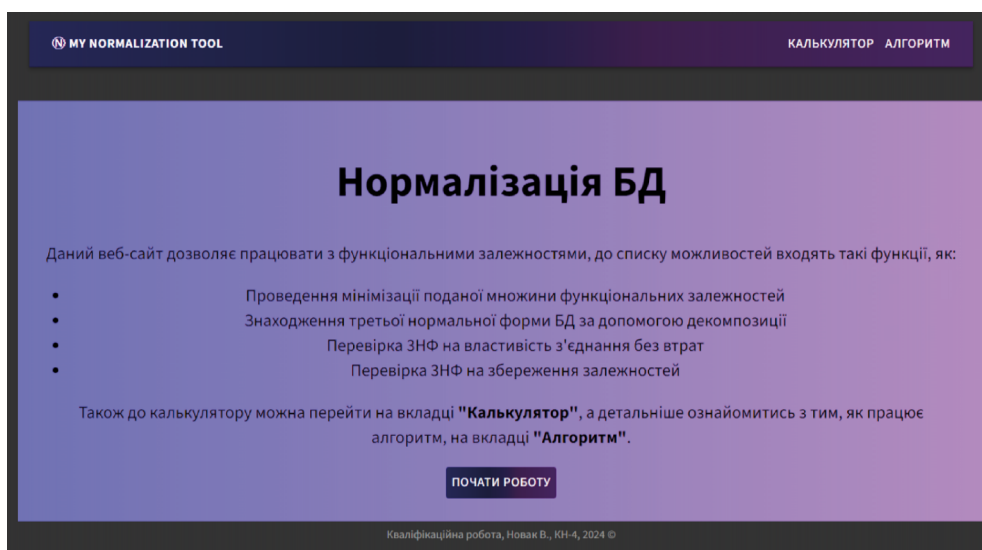


Рис. 6. Головна сторінка клієнтської частини застосунку

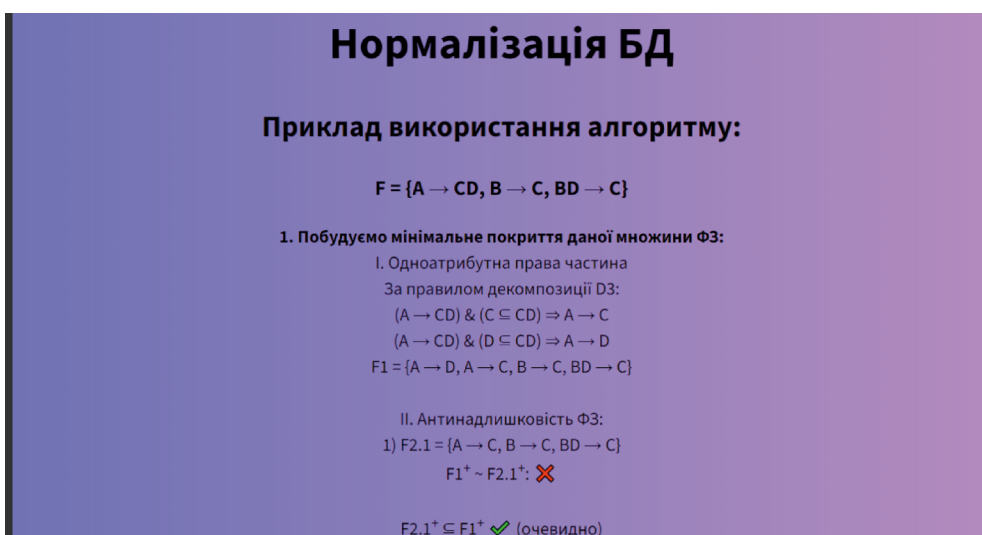


Рис. 7. Опис алгоритмів

При переході на сторінку обчислень «Калькулятор» з'являється набір текстових полів, де перше поле призначене для введення усіх атрибутів реляції, а усі інші пари текстових полів — для введення функціональних залежностей (рис. 8). Клієнт має змогу виконати певну операцію, використовуючи відповідні кнопки на інтерфейсі користувача (рис. 9).

Рис. 8. Сторінка калькулятора

Рис. 9. Зведення прикладу рис. 8. до 3НФ

Висновок

У статті описано основні теоретичні аспекти процесу нормалізації баз даних, розглянуто наявні інструменти, виділено їхні основні переваги й недоліки. Також на прикладах надано опис функціонування нової системи для нормалізації баз даних, яка враховує деякі недоліки наявних систем.

Ця система не лише є актуальною для розробників баз даних, аналітиків та інших фахівців, що працюють із великими обсягами даних, забезпечуючи швидке та ефективне досягнення оптимальної структури бази даних, спрощуючи підтримку та розширення бази даних, а й може слугувати навчальним інструментом для студентів і новачків у галузі інформаційних технологій, допомагаючи їм зрозуміти процеси нормалізації через можливість розв'язання практичних прикладів та ознайомлення з використаними алгоритмами.

Подальшими кроками покращення системи є реалізація декомпозиції реляції до НФБК та 4НФ.

Список літератури

1. Бардус І. О. Бази даних у схемах (на основі фундаменталізованого підходу) : навч. посіб. / І. О. Бардус, М. І. Лазарев, А. О. Ніценко. — Харків : Вид-во «Діса плюс», 2017. — 133 с.
2. Гайджари В. І. Основи проектування та використання баз даних : навч. посіб. / В. І. Гайджари, О. А. Дацюк. — 2-ге вид., виправл. і доповн. — Київ : ІВЦ «Видавництво «Політехніка», ТОВ «Фірма «Періодика», 2004. — 256 с.
3. Boyce-Codd relation solver [Electronic resource]. — Mode of access: <https://gamgi.github.io/js-bcnf/>.
4. Functional Dependencies Checker [Electronic resource]. — Mode of access: <https://arjo129.github.io/functionalDependencyCalculator/>.
5. Garcia-Molina H. Database systems. The complete book / H. Garcia-Molina, J. D. Ullman, J. Widom. — Upper Saddle River, New Jersey : Person Prentice Hall, Person Education, Inc., 2009, 2002.
6. Ramakrishnan, R. Database management systems / R. Ramakrishnan, J. Gehrke. — Singapore : McGraw-Hill Higher Education, 2003.
7. Relational Database Tools [Electronic resource]. — Mode of access: <http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools.html>.

References

- Bardus, I. O., Lazarev, M. I., & Nitsenko, A. O. (2017). *Bazy danykh u skhemakh (na osnovi fundamentalizovanoho pidkhodu)*. Disa plus. BCNF-js | Boyce-Codd relation solver. (n. d.). [gamgi.github.io](https://gamgi.github.io/js-bcnf/). <https://gamgi.github.io/js-bcnf/>.
- BCNF-js | Boyce-Codd relation solver. (n. d.). [gamgi.github.io](https://gamgi.github.io/js-bcnf/). <https://gamgi.github.io/js-bcnf/>.
- Functional Dependencies Checker. (n. d.). Arjo's Blog. <https://arjo129.github.io/functionalDependencyCalculator/>.
- Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database systems, The complete book*. Person Prentice Hall, Person Education, Inc.
- Haidzhary, V. I., & Datsiuk, O. A. (2004). *Osnovy proektuvannia ta vykorystannia baz danykh*. Vydavnytstvo "Politekhnik", TOV "Firma "Periodyka".
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems*. McGraw-Hill Higher Education.
- Relational Database Tools. (n. d.). Raymondcho.net. <http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools.html>.

S. Yaremko, V. Novak

SYSTEM FOR AUTOMATING THE DATABASE NORMALIZATION PROCESS

As data volumes grow rapidly, efficient database management has become critical for organizations. Relational databases play an essential role in ensuring data integrity, enabling complex queries, and supporting various applications, including financial, healthcare, e-commerce, and CRM systems.

Database normalization, a key technique for structuring data and reducing redundancy, improves database efficiency and performance. However, the normalization process can be complex and demands expert knowledge. The article outlines the theoretical foundations of normalization, explaining various normal forms, including 1NF, 2NF, 3NF, and Boyce-Codd Normal Form (BCNF). It emphasizes that while normalization is essential, eliminating redundancy entirely is impractical when maintaining database cohesion.

The proposed system automates normalization using an algorithm based on Heath's theorem, which guarantees a lossless decomposition and dependency preservation. The system can identify minimal sets of functional dependencies, search for quasi-keys, and perform decompositions up to 3NF, ensuring that the database meets lossless join and dependency preservation requirements.

The authors compare the new system with existing tools, highlighting key advantages such as its user-friendly interface and comprehensive functionality, including decomposition capabilities and result integrity verification. The system is designed with Node.js for the backend and React.js for the user interface, providing a web-based platform for database normalization.

The article also explores potential use cases, noting that the system is beneficial for database developers, analysts, and students learning about database management. It simplifies the normalization process, making it faster and more user-friendly. The authors conclude by discussing future improvements, including support for BCNF and 4NF decompositions.

This system offers a practical solution for addressing database normalization challenges, reducing process complexity while enhancing data integrity and performance.

Keywords: relational databases, normalization process, database integrity, functional dependencies, decomposition algorithm, lossless join property, dependency preservation property.

Матеріал надійшов 30.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Касьяненко М. В., Нагірна А. М.

АЛГОРИТМІЧНИЙ ПІДХІД ДО АНАЛІЗУ МЕТРИК У ГАЛУЗІ РОЗРОБЛЕННЯ ВІДЕОІГОР З ВІДКРИТИМ СВІТОМ

Статтю присвячено розгляду наявних метрик у галузі розроблення відеоігор із відкритим світом та побудові алгоритмічного підходу до аналізу цього типу метрик. Для відеоігор із відкритим світом запропоновано низку числових і графічних метрик, що дають змогу аналізувати продукт ще на етапі його розроблення, однак мають низку недоліків. Також представлено програмну реалізацію алгоритмічного підходу, що дає можливість обчислювати й аналізувати метрики для ігор із відкритим світом. Роботу системи було перевірено шляхом аналізу реальної відеогри.

Ключові слова: алгоритм, відеогра, відкритий світ, метрика, теплова мапа, маршрут, діаграма, мапа світу, квест, стороння активність.

Вступ

Одними з найпоширеніших продуктів інженерії програмного забезпечення є відеоігри. Згідно з Еммою МакДоналд [7] із видання Newzoo, що займається аналітикою відеоігрового ринку, обсяги ринку за 2022 рік становили значну суму — понад 182 мільярди доларів.

Але з великими інвестиціями з'являються великі ризики. Тому дуже важливо розробляти якісні продукти. Проте оцінити якість продукту вкрай важко, особливо до появи робочого прототипу. Отже, необхідно дослідити підходи до розроблення вимог для відеоігор і розробити метрики, що дають змогу аналізувати продукт на основі вимог.

Одними з найширших відеоігор є ігри з відкритим світом. Вони дають змогу гравцеві вільно досліджувати світ, зазвичай із незначними обмеженнями або зовсім без них [3, с. 5]. При цьому «відкритий світ» не є жанром. Це підхід до побудови ігрової структури, який можна застосовувати до будь-якого жанру [3, с. 4]. Тож обмеження предметної області роботи до відеоігор із відкритим світом дасть можливість зробити більш детальні метрики і при цьому зберегти широку сферу застосування результатів роботи.

Особливості і проблеми наявних метрик

Загальноприйнятим підходом до розрахунків метрик для відеоігор є використання телеметрії. Прикладами ігрової телеметрії можуть слугувати будь-які дані про дії гравця [1, с. 6].

Загалом виділяють три категорії відеоігрових метрик, які розраховують за допомогою телеметрії: метрики внутрішньоігрового росту, метрики залученості та монетизації, метрики процесів розроблення [8, с. 4]. Метрики внутрішньоігрового росту аналізують ігровий процес, метрики залученості вказують на ступінь зацікавленості, метрики монетизації забезпечують отримання доходу (або заробітку) від продуктів відеоігор, метрики процесів розроблення характеризують командну роботу.

Проте всі згадані метрики для розрахунку потребують даних телеметрії від реальних гравців. Це унеможливає їх використання на етапі розроблення і планування. Для вирішення цієї проблеми у роботі, пропонують алгоритмічний підхід аналізу метрик, що передбачає використання опису вимог для їхньої безпосередньої побудови.

Основні поняття і ключові елементи, що описують відкритий світ для побудови метрик

На основі аналізу наявних ігор із відкритим світом було виділено такий набір складових ігрового світу: загальна мапа світу, квест, ворог, стороння активність (place of interest) і перепона.

Загальна мапа світу являє собою простір, яким переміщується гравець. У ній розташовуються усі інші складові [2]. Квест — це структурна одиниця прогресу гравця у проходженні відеогри. Гру вважають пройденою після того, як гравець виконує усі сюжетні квести [4]. Вороги розміщуються на мапі з метою ускладнити маршрут гравця між квестами. Для того щоб зробити шлях більш цікавим, на мапу світу додають сторонні активності (places of interest), як-от скриня з корисними речами, невелика головоломка, персонаж, із яким можна вступити в діалог [5, с. 5].

Для обмеження пересування гравця передбачено два види перепон: заборонена територія, що має форму замкнутого багатокутника, і бар'єр, що має форму ламаної лінії, яку не можна перетинати у жодній точці. Приклад розміщення квестів, ворогів, сторонніх активностей і перепон на мапі зображено на рис. 1.

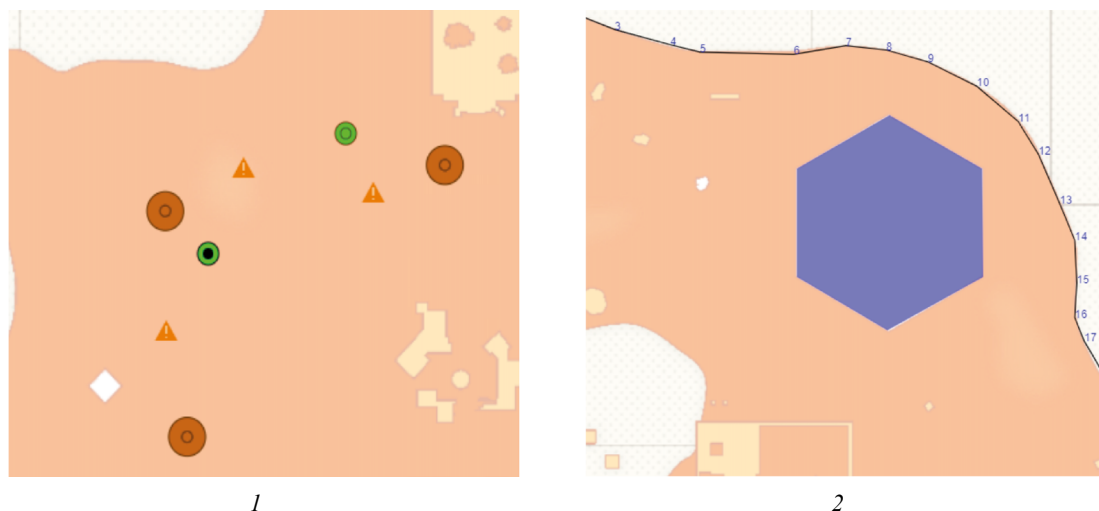


Рис. 1. Фрагменти мапи світу: 1 — мапа з квестами (круги більшого розміру), ворогами (трикутники) та сторонніми активностями (круги меншого розміру); 2 — мапа з забороненою територією і бар'єром

На основі цих даних можливо змодельовати маршрут, яким рухатиметься гравець у відкритому світі. Точки, через які проходить маршрут, використовують як вхідні дані для побудови метрик.

Опис запропонованих метрик

Категорії метрик

Після побудови можливих маршрутів гравця можна робити висновки про його потенційний ігровий процес і починати розраховувати метрики. В рамках цієї роботи було розроблено дві категорії метрик: числові та графічні.

Числові метрики

Першою числовою метрикою є загальний хронометраж. Ця метрика вказує на загальний час, який знадобиться гравцеві для проходження гри від самого початку, до останнього квесту. У результаті часу враховують усі види дій, які може виконувати гравець: ходьба, проходження квестів і сторонніх активностей, битви з ворогами.

Друга числова метрика має назву «Час, витрачений на кожну активність» (рис. 2). Вона має форму кругової діаграми й показує, скільки часу витрачає гравець на кожен із можливих типів діяльності у грі. За допомогою цієї метрики розробники можуть зробити висновок, чи достатньо подій кожного типу трапляється на шляху гравця. Або навпаки, чи не забагато часу гравця витрачається на однотипні завдання.

Третьою метрикою в цій категорії є «Час, витрачений на кожен квест». Ця метрика у формі стовпчикової діаграми візуалізує, скільки часу займає проходження кожного квесту в маршруті. За її допомогою можна проаналізувати, наскільки сильно відрізняються за тривалістю квести.

Четверта метрика — «Відстань між кожним квестом» — у вигляді стовпчикової діаграми показує, скільки метрів шляху потрібно подолати гравцеві, щоб дістатися кожного наступного квесту (рис. 3). За її допомогою розробники можуть побачити, чи немає у грі занадто довгих переходів між квестами.

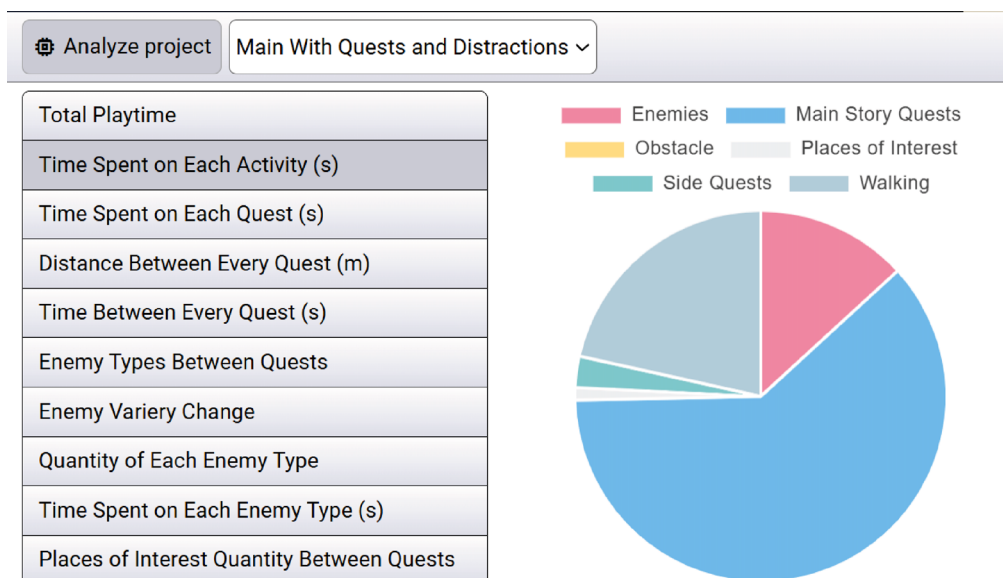


Рис. 2. Кругова діаграма метрики «Час, витрачений на кожну активність»

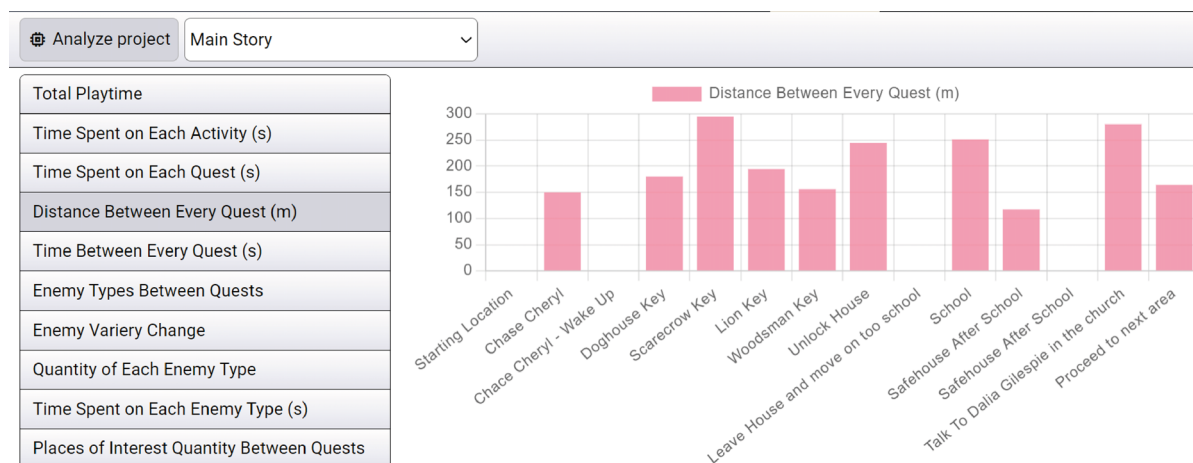


Рис. 3. Стовпчикова діаграма метрики «Відстань між кожним квестом»

«Час між кожним квестом» є п'ятою числовою метрикою. Її стовпчикова діаграма відображає, скільки часу проводить гравець у відкритому світі перед проходженням кожного з квестів. Окрім часу, витраченого на подолання маршруту, ця метрика також ураховує час на проходження сторонніх активностей і бої з ворогами. Розробники можуть використовувати цю метрику для балансування темпу гри (pacing). Просування гравця у сюжеті гри відбувається лише у квестах, тож важливо розташувати квести так, щоб гравець не почав нудьгувати, займаючися другорядними завданнями.

«Типи ворогів між квестами» — це шоста числова метрика. Вона представлена у формі стовпчикової діаграми, кожне значення якої дорівнює кількості видів ворогів, що трапляються на шляху між двома квестами. Ця метрика може допомогти в балансуванні складності. На ній можна побачити місця, що мають незвично високе різноманіття ворогів, або навпаки, місця, де видів ворогів замало, що може призвести до втрати цікавості гравця.

Сьома метрика є круговою діаграмою — «Кількість ворогів кожного типу». В ній підраховано загальну кількість ворогів кожного типу, яких зустрів гравець під час проходження гри.

«Час, витрачений на кожен тип ворогів» — восьма метрика у формі кругової діаграми, що аналізує час, витрачений на боротьбу з кожним типом ворогів. Вона аналізує бойову складову гри з іншого боку, порівняно з попередньою метрикою.

Останньою числовою метрикою є «Кількість сторонніх активностей між кожним квестом». Ця стовпчикова діаграма показує кількість сторонніх активностей, що їх виконав гравець на маршруті до певного квесту.

Графічні метрики

Першою метрикою є «Покриття мапи сторонніми активностями і ворогами». Цю теплову мапу повністю зафарбовано червоним у місцях, що не мають поблизу ані сторонніх активностей, ані місць розташування ворогів. Що ближчими до координати є ворог або стороння активність, то менш зафарбованою є ця точка. За допомогою цієї метрики можна аналізувати, наскільки вдало розташовані на мапі завдання для гравця. Бажано, щоб у відеоіграх із відкритим світом не було місць, які зовсім не мають сторонніх активностей і ворогів, оскільки в такому разі гравець не має причин їх досліджувати.

Друга метрика має назву «Досліджені території». Цю теплову мапу зафарбовано червоним у місцях, біля яких гравець не проходив. Що ближчою координата мапи до маршруту гравця, то прозорішою вона є на тепловій мапі. Завдяки цій мапі можна бачити місця, якими пересувався гравець, і помітити території відкритого світу, які залишаються невивченими. Проаналізувавши цю метрику, розробники гри можуть визначити потребу додавання додаткових квестів у певні регіони мапи.

Третя графічна метрика має назву «Занадто досліджені території». За її допомогою можна побачити, якими територіями гравець ходить забагато разів. Ця мапа використовує спектр кольорів для відображення, наскільки сильно дослідженою є певна місцевість. Територія, якою гравець ходив лише один раз або тільки бачив здалека, має синій колір. Місця, де гравець бував один або кілька разів, — зелені. А території, які відвідували небажано велику кількість разів, мають червоний колір. Теплові мапи «Покриття мапи сторонніми активностями і ворогами» і «Занадто досліджені території» зображено на рис. 4.

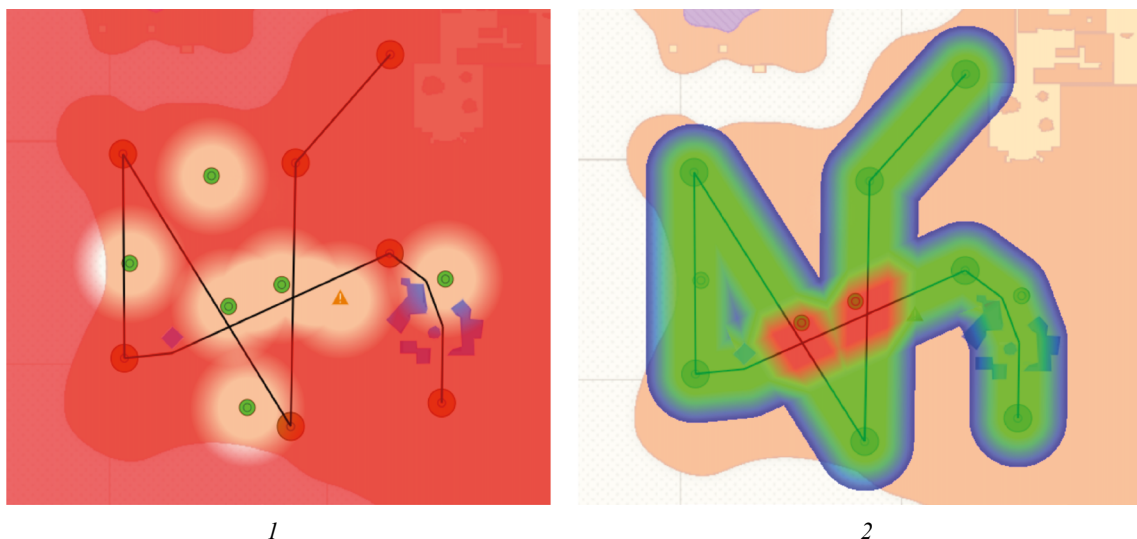


Рис. 4. Теплові мапи: 1 — мапа «Покриття мапи сторонніми активностями і ворогами»; 2 — мапа «Занадто досліджені території»

Програмна реалізація алгоритмічного підходу аналізу метрик

Алгоритми розрахунку метрик реалізовано у вигляді вебдодатка. Він містить графічний редактор мапи світу, що дає змогу розміщувати елементи відкритого світу у просторі (рис. 5). За допомогою панелі інструментів унизу можна вибрати, який елемент відкритого світу буде розміщено на мапі наступним. Вибравши будь-який з елементів за допомогою інструмента Select, можна вносити детальну інформацію про елемент у відповідній формі. Після того як усі елементи та їхні характеристики внесено на мапу, система буде маршрути гравця й теплові мапи. Панель інструментів зверху дає змогу вибрати маршрут і мапу, що буде відображена у редакторі.

Також реалізовано алгоритми побудови числових метрик. Результати їхнього обчислення для кожного з маршрутів можна переглянути в окремому інтерфейсі.

Перевірка ефективності розроблених метрик

Щоби перевірити ефективність розроблених метрик, необхідно за їх допомогою виконати аналіз відеоігор з відкритим світом.

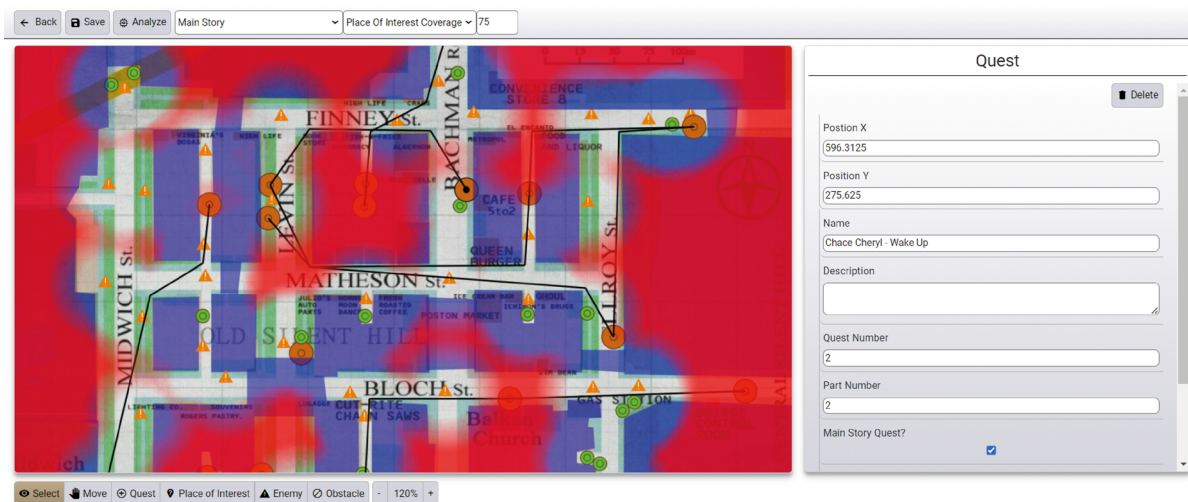


Рис. 5. Інтерфейс редактора мапи: 1 — мапа світу та панелі інструментів; 2 — форма редагування вибраного елементу

Аналіз проведено на прикладі гри Silent Hill, випущеної 1999 р. для PlayStation [9]. Гра складається з кількох лінійних секцій, що об'єднані між собою за допомогою відкритих зон. Це надає їй статусу гри з елементами відкритого світу [6]. З використанням офіційної книги стратегії усі елементи відкритого світу внесено у графічний редактор мапи світу і на основі цих даних проведено аналіз метрик [10].

За результатами аналізу графічних метрик, вибрана мапа є добре побудованою. Із метрик «Покриття мапи сторонніми активностями» та «Досліджені території» видно, що розташування квестів спонукає гравця до дослідження більшої частини мапи. Сторонні активності й вороги розташовані так, щоб наповнювати маршрут подіями, навіть існують окремі частини мапи, які нагороджують гравців, що готові докласти додаткових зусиль до дослідження світу. За допомогою метрики «Занадто досліджені території» помічено один з головних недоліків гри — backtracking. Цей підхід до дизайну світу полягає в тому, щоб змушувати гравця тим чи тим способом повертатися до вже досліджених територій [11]. Він є характерною рисою для ігор цього жанру, які розробляли у 1990-ті. На мапі існує декілька місць, якими гравець проходить три або більше разів. Теплову мапу «Занадто досліджені території» зображено на рис. 6.



Рис. 6. Теплова мапа «Занадто досліджені території»

Результат метрики «Загальний хронометраж» перевірено шляхом порівняння його з записом проходження гри longplay гри Silent Hill від спільноти World Of Longplays [12]. Отриманий результат мав похибку 7,6 %.

У результаті аналізу числових метрик гри виявлено декілька проблем. Відстань між квестами є нерівномірною, і гравець іноді вимушений долати великі відстані між квестами. Також сторонні активності розташовані занадто далеко від місць, якими проходить гравець. Це було виявлено за метрикою «Кількість сторонніх активностей між кожним квестом», що відобразила проходи між квестами, поряд з якими не було сторонніх активностей.

Проте числові метрики також показали низку позитивних характеристик. Усі метрики, що стосуються балансування ворогів, указують на те, що розробники правильно наповнювали світ ворогами. Також розподіл ігрового часу між різними активностями підкреслює особливості жанру горор.

Успішно проведений аналіз засвідчив гнучкість розробленої системи. Незважаючи на те, що розглянута гра не мала чіткої структури відкритого світу, використання системи для аналізу дало змогу зробити велику кількість висновків про особливості гри.

Висновки

Представлено програмну реалізацію алгоритмічного підходу аналізу метрик у галузі розроблення відеоігор з відкритим світом.

Розроблений додаток надає користувацький інтерфейс для введення вимог до гри з відкритим світом, забезпечує відображення та розрахунок усіх розроблених метрик. Опис вимог імплементовано у вигляді графічного редактора й мапи світу.

Для підтвердження ефективності роботи додатка для перевірки запропонованих метрик проведено експеримент за вимогами відеоігри Silent Hill. Система показала високу гнучкість використання, забезпечивши аналіз гри лише за наявності елементів відкритого світу без чіткої структури відкритого світу.

Список літератури

1. Ameli S. R. A descriptive-analytical study of the open-world game design “grand theft auto V and online version” with an emphasis on role-playing capabilities / Saied Reza Ameli, Milad Mousavi Haghsheenas // *Journal of cyberspace studies*. — 2023. — Т. 7, № 1. — Pp. 1–22. — <https://doi.org/10.22059/jcss.2023.353081.1083>.
2. Chądzyńska D. Maps in video games — range of applications / Dominika Chądzyńska, Dariusz Gotlib // *Polish cartographical review*. — 2015. — Т. 47, № 3. — Pp. 137–145. — <https://doi.org/10.1515/pcr-2015-0011>.
3. Farra T. J. Between freedom and success — A study of gamer experience of level design in open-world games: Bachelor Thesis in Game Design / Farra Trinity Jacob. — [Б. м.], 2023.
4. Howard J. *Quests: design, theory, and history in games and narratives* / Jeff Howard. — [S. l.] : CRC Press LLC, 2008.
5. Jaber M. The 40 seconds rule and points of interest in the witcher 3: wild hunt : thesis [Electronic resource] / Jaber Maher, Cojanu Andrei. — [Б. м.], 2021. — Mode of access: <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-446386> (date of access: 17.10.2024). — Title from screen.
6. Kirkland E. Restless dreams in Silent Hill: approaches to video game analysis / Ewan Kirkland // *Journal of media practice*. — 2005. — Т. 6, № 3. — Pp. 167–178. — <https://doi.org/10.1386/jmpr.6.3.167/1>.
7. McDonald E. Newzoo’s key numbers | games, esports, mobile [Electronic resource] / Emma McDonald // Newzoo. — Mode of access: <https://web.archive.org/web/20190509014637/https://newzoo.com/key-numbers/> (date of access: 17.10.2024). — Title from screen.
8. Perfecting A video game with game metrics / Junaidi Junaidi [та ін.] // *TELKOMNIKA (telecommunication computing electronics and control)*. — 2018. — Т. 16, № 3. — P. 1324. — <https://doi.org/10.12928/telkomnika.v16i0.7209>.
9. Silent hill // *Playstation magazine*. — 1999. — Т. 3, № 18. — Pp. 88–92.
10. Silent hill strategy guide. — [Б. м.] : Cyber Press Publishing, 1999.
11. Woodbury D. Backtracking and Non-Essential Areas [Electronic resource] / Dylan Woodbury // *Game Developer | Game Industry News, Deep Dives, and Developer Blogs*. — Mode of access: <https://www.gamedeveloper.com/design/backtracking-and-non-essential-areas> (date of access: 17.10.2024). — Title from screen.
12. World of Longplays. PSX longplay [181] silent hill (EU) [Electronic resource] / World of Longplays // YouTube. — Mode of access: <https://www.youtube.com/watch?v=WY34ChMlvow> (date of access: 17.10.2024). — Title from screen.

References

- Ameli, S. R., & Haghsheenas, M. M. (2023). A descriptive-analytical study of the open-world game design “grand theft auto V and online version” with an emphasis on role-playing capabilities. *Journal of Cyberspace Studies*, 7 (1), 1–22. <https://doi.org/10.22059/jcss.2023.353081.1083>.
- Chądzyńska, D., & Gotlib, D. (2015). Maps in video games — range of applications. *Polish Cartographical Review*, 47 (3), 137–145. <https://doi.org/10.1515/pcr-2015-0011>.
- Farra, T. J. (2023). *Between freedom and success — A study of gamer experience of level design in open-world games*. [Unpublished Bachelor Thesis in Game Design]. Uppsala.

- Howard, J. (2008). *Quests: Design, theory, and history in games and narratives*. CRC Press LLC.
- Jaber, M., & Cojanu, A. (2021). *The 40 seconds rule and points of interest in the witcher 3: Wild hunt* [Thesis, Uppsala universitet, Institutionen för speldesign]. <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-446386>.
- Junaidi, J., Julianto, A., Anwar, N., Safrizal, S., Spits Warnars, H. L. H., & Hashimoto, K. (2018). Perfecting A video game with game metrics. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 16 (3), 1324. <https://doi.org/10.12928/telkomnika.v16i0.7209>.
- Kirkland, E. (2005). Restless dreams in Silent Hill: Approaches to video game analysis. *Journal of Media Practice*, 6 (3), 167–178. <https://doi.org/10.1386/jmpr.6.3.167/1>.
- McDonald, E. (n. d.). *Newzoo's key numbers | games, esports, mobile*. Newzoo. <https://web.archive.org/web/20190509014637/https://newzoo.com/key-numbers/>.
- Silent hill. (1999). *Playstation Magazine*, 3 (18), 88–92.
- Silent hill strategy guide*. (1999). Cyber Press Publishing.
- Woodbury, D. (2010, 22 грудня). *Backtracking and Non-Essential Areas*. Game Developer | Game Industry News, Deep Dives, and Developer Blogs. <https://www.gamedeveloper.com/design/backtracking-and-non-essential-areas>.
- World of Longplays. (2013, 31 жовтня). *PSX longplay [181] silent hill (EU)*. YouTube. <https://www.youtube.com/watch?v=Wy34ChMlvow>.

M. Kasianenko, A. Nahirna

METRIC QUALITY ANALYSIS IN OPEN-WORLD VIDEO GAMES

Video games are a very popular aspect of software engineering. However, they require a large time commitment and investment. Creating a modern video game requires a large budget. However, there is a significant risk of not recouping the investment if the game becomes a commercial failure. To ensure that the final product is high-quality, developers use different analytical tools, such as metrics.

Despite being used for a long time, metrics still have a limited scope of application. All common metric analysis approaches focus on retaining the audience of an already released product. Moreover, these metrics require end-user data for their calculation, which means they cannot be used for products that are still in development.

The solution proposed in the article offers an approach that allows metric analysis for open-world games to begin in early stages of development, based on game requirements. It provides a simple classification of open-world elements that allows developers to describe an open-world map and model a route for the future player. This enables the calculation of a variety of metrics, both graphic and numeric. The metrics help analyze multiple aspects of gameplay, such as difficulty, pacing, and map coverage.

To facilitate the application of the metrics, a solution was developed. It contains a graphic editor to populate the world map with open-world elements, as well as algorithms for metric calculation. Both visual and numeric metrics can be easily observed and analyzed through the app's interface. The article contains a detailed description of all the metrics, an overview of the developed application, and an analysis of an existing video game using this tool. The analysis successfully identified numerous key characteristics of a game.

Keywords: videogame, open world, metric, heatmap, algorithm, route, diagram, world map, quest, place of interest.

Матеріал надійшов 14.10.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

A NUCLEOLUS-BASED APPROACH FOR CLOUD RESOURCE ALLOCATION

Cloud computing has transformed organizational operations by enabling flexible resource allocation and reducing upfront hardware investments. However, the growing complexity of resource management, particularly for computing instances, has led to challenges in cost control and resource allocation. Fair allocation policies, such as max-min fairness and Dominant Resource Fairness, aim to distribute resources fairly among users. In recent years, the FinOps framework has emerged to address cloud cost management, empowering teams to manage their own resource usage and budgets. The allocation of resources among competing product teams within an organization can be modelled as a cooperative game, where teams with competing priorities must negotiate resource allocation based on their claims and the available budget.

The article explores cloud resource allocation as a cooperative game, particularly in situations where the total budget is insufficient to meet all teams' demands. Several resource allocation methods are discussed, including the proportional rule and the nucleolus-based approach, which seeks to minimize the coalitions' incentives to deviate. The nucleolus method offers a stable and fair solution by distributing resources in a way that maximizes stability and reduces the likelihood of coalitions deviating from the overall allocation. This approach ensures that no team is allocated more than its claim and maintains fairness by adhering to principles such as claim boundaries, monotonicity, and resource constraints. Ultimately, the nucleolus-based method is proposed as an effective solution for allocating cloud resources in a cooperative and stable manner, ensuring that resource allocation is both fair and efficient.

Keywords: cloud computing, resource allocation, bankruptcy issue, cooperative game, nucleolus.

Introduction

Cloud computing is a game-changing technology that revolutionized the way many organizations operate. With cloud technologies, organizations can operate more flexibly, easily scaling and reducing up-front investments in hardware. Thus, the cost of cloud computing usage could quickly escalate [3].

Resource allocation is an important part of any shared system, not just clouds. One of the most popular allocation policies proposed so far has been max-min fairness, which maximizes the minimum allocation received by a user in the system. Assuming each user has enough demand, this policy gives them an equal share of resources. Round-robin, proportional resource sharing, and weighted fair queueing are known algorithms of this approach. A shared-system specialized allocation method, like Dominant Resource Fairness, was proposed for multiple resource types [4] and could be used for clouds as well [2].

These methods approach the idea of fairness, where each user is allocated more or less in proportion to their demand and weight of resources. In other words, allocation is viewed as a non-cooperative game in which alliances cannot be formed, or all agreements need to be self-enforcing [1].

In recent years a new profession, 'FinOps practitioners,' emerged, along with a framework to address cloud cost management. One of the main FinOps principles is that 'individual feature and product teams are empowered to manage their own cloud usage within their budget.' [6]

In other words, each team should take part in computing resource allocations for their needs and managing costs. Whereas some resources (object storage, selective databases) are dynamically allocated, computing instances require provisioning time and are often allocated in advance.

Usually, resources need to be allocated among product teams. These teams have competing priorities, and each product team is focused on its respective product, not the system as a whole. The teams can form coalitions, and there is a possibility of external enforcement of cooperative behavior (formal or informal). Even though 'common goods' could be facilitated within the organization, factions could easily form and could be considered.

Cloud Resource Allocation as a Cooperative Game

Suppose there are N product teams, each demanding computing resources such as CPU, memory size, and memory bandwidth. Within serverless computing, most resources can scale on demand, while some need to be allocated in advance. Let's consider allocating computing instances in a specific cloud.

Most major cloud vendors provide a selection of instances with various hardware and price options. In a cloud environment, all resources can be summarized by a single number: price (per usage). Table 1 provides an example of pricing for various general-purpose computing instances (EC2) of AWS in the US East (Ohio) retrieved on 2024-04-01. To eliminate cents, let us consider hourly prices multiplied by 168 (weekly), with \$/w representing USD per week.

Table 1. AWS EC2 prices and key performance indicators

Instance	v CPU	Clock Speed	Memory	EBS Mbps	Price \$/w
m5.large	2	3.1 GHz	8 GiB	<2,120	17
m4.large	2	2.4 GHz	8 GiB	450	18
m5.xlarge	4	3.1 GHz	16 GiB	<2,120	34
m4.xlarge	4	2.4 GHz	16 GiB	750	37
m5.2xlarge	8	3.1 GHz	32 GiB	<2,120	68
m4.2xlarge	8	2.4 GHz	32 GiB	1,000	74
m5.4xlarge	16	3.1 GHz	64 GiB	2,120	137
m4.4xlarge	16	2.4 GHz	64 GiB	2,000	148
m5.8xlarge	32	3.1 GHz	128 GiB	5,000	274
m4.10xlarge	40	2.4 GHz	160 GiB	4,000	336
m5.12xlarge	48	3.1 GHz	192 GiB	7,000	411
m4.16xlarge	64	2.4 GHz	256 GiB	10,000	538
m5.16xlarge	64	3.1 GHz	256 GiB	10,000	548
m5.24xlarge	96	3.1 GHz	384 GiB	12,000	774

Each product team can select the main characteristics of instances and choose desired instances based on the available list of computing instances and actual needs. Let us assume all product teams use the same utilization factor and other metrics to make their decision. For example, there are three teams ($N=3$) with specific requests (see Table 2), and allocation is on a weekly basis. Then, based on each team's request, FinOps (assuming it is responsible for resource allocation) estimates the total budget. For simplicity, let us assume it is also calculated on a weekly basis.

The task is: how should we allocate resources when the total requested budget (total demand) is higher than the available budget (estate)? Obviously, if the estate is greater than the total demand, it is not a problem.

Table 2. Product teams' desired instances

Team	Main characteristic	Asked instance	Ask, \$/w
1	Memory	m5.8xlarge	274
2	EBS	m4.16xlarge	538
3	CPU	m5.24xlarge	774
Total			1,586

For example, let the estate be \$600/w, whereas the total demand (see Table 2) is \$1,586/w. Please notice that Team 3's request (\$774/w) alone is higher than the available budget (\$600/w).

This task can be rewritten as a claims problem (also called a bankruptcy problem). For that, the budget should be considered as a divisible source. Each product team is a player, and each team's desired instance price can be considered its claim or request. If all product teams use the same metrics and have the same value, we can consider a claim to be a team's weight as well. The available budget is the estate. Deficits are the differences between a team's claims and awards.

Is cloud resource allocation between product teams an example of a cooperative game? A cooperative game is one where players are able to make enforceable contracts outside of those specifically modelled within the game. FinOps should find a solution that minimizes the chances of any team 'going upstairs' or teams merging/splitting just to cheat the system. In other words, no single team or coalition will leave the grand coalition (all teams combined). Thus, the problem can be considered a cooperative game. Excess is

the difference between the payment given to the coalition for its allocated resources and the value the coalition would receive by deviating.

What are the properties of a good solution?

1. Claim boundaries: No team gets more than its claim or less than \$0.
2. Equal treatment of equals: If two teams have the same claim, they should be given (roughly) equal awards.
3. Monotonicity: Teams with higher original claims should not receive fewer awards.
4. Resource monotonicity: If the estate increases, each team should not get less.
5. Cloud constraints: A finite number of available resource types and possible awards.

Maximum Award

Let us start with the goal of finding a solution that maximizes the total award alone. The solution can be found by brute force (see Table 3. Hereafter, a coalition is described by a tuple of teams, for example, (1,2,3) represents a grand coalition for all three teams. When all coalitions have only positive excesses, then deviation is not profitable for any coalition.

How is excess calculated? Let us take coalition (2,3) as an example. If it deviates, then the total amount allocatable to it will be \$600/w minus \$538/w (claim of the remaining coalition (1)) giving \$62/w to allocate. What could this coalition buy with the allocated funds? The best option is an m4.large (\$18/w) and an m4.xlarge (\$37/w) with a total award of \$55/w. Thus, the excess is its award of \$55/w minus \$55/w, resulting in \$527/w.

Thus, monotonicity is not maintained, as the award for coalition (2) is less than the award of coalition (1), so additional conditions must be added.

Table 3. Maximum award solution

Coalition	Award, \$/w	Instance(s)	Excess, \$/w
(1)	34	[m5.xlarge]	34
(2)	18	[m4.large]	18
(3)	548	[m5.16xlarge]	548
(1, 2)	52	[m5.xlarge, m4.large]	52
(1, 3)	582	[m5.xlarge, m5.16xlarge]	527
(2, 3)	566	[m4.large, m5.16xlarge]	255
(1, 2, 3)	600	[m5.xlarge, m4.large, m5.16xlarge]	

Constraint Equal Award Rule

One common approach is to distribute available resources proportionally to the estate, divided equally among all teams (without exceeding their claims). An example of the rule's application is given in Table 4.

Table 4. Constraint equal award rule allocation

Coalition	Award, \$/w	Instance(s)	Excess, \$/w
(1)	148	[m4.4xlarge]	148
(2)	148	[m4.4xlarge]	148
(3)	148	[m4.4xlarge]	148
(1, 2)	296	[m4.4xlarge, m4.4xlarge]	296
(1, 3)	296	[m4.4xlarge, m4.4xlarge]	241
(2, 3)	296	[m4.4xlarge, m4.4xlarge]	-15
(1, 2, 3)	444	[m4.4xlarge, m4.4xlarge, m4.4xlarge]	

As seen in Table 4, the excess of coalition (2,3) is negative. How is it possible? Suppose that coalition deviates from the grand coalition. The claim of the remaining coalition (1) is \$274/w. It can be fully granted, as the m5.8xlarge instance price is exactly \$274/w. The remaining budget is \$600 - \$274 = \$326/w. We can allocate the total budget by allocating an m5.8xlarge (\$274/w) to Team 2 and an m4.xlarge (\$37/w) to Team 3. This means that coalition (2,3) will receive \$311/w, which is higher than the allocated \$296/w, making deviation from the grand coalition beneficial. Thus, the equal award is not a stable solution, as for some of the product team coalitions, deviation is more beneficial.

Proportional Rule

Many authors [2] consider the proportional rule a reasonably fair allocation method. In this approach, each player receives a share of the estate proportional to its weight. An example of the rule is given in Table 5. Obviously, for product team (3), this approach is much better than the previous one. Please note, that for the coalition of teams (2,3) it is more profitable to deviate as excess is negative. Even so, competing coalition (1) receives less, making this solution appear to be more stable.

Table 5. Proportional rule

Coalition	Award, \$/w	Instance(s)	Excess, \$/w
(1)	74	[m4.2xlarge]	74
(2)	148	[m4.4xlarge]	148
(3)	148	[m4.4xlarge]	148
(1, 2)	222	[m4.2xlarge, m4.4xlarge]	222
(1, 3)	222	[m4.2xlarge, m4.4xlarge]	167
(2, 3)	296	[m4.4xlarge, m4.4xlarge]	-15
(1, 2, 3)	370	[m4.2xlarge, m4.4xlarge, m4.4xlarge]	

Nucleolus for Computing Cloud Resource Allocation

The nucleolus is a solution concept in cooperative game theory that maximizes stability by minimizing coalitions' incentives to deviate [5].

The objective is to find an imputation whose excesses vector is maximized in leximin order within the available range of choices per player.

Table 6 shows that the nucleolus solution provides both the best excess and deficit values for the worst coalitions, making it the most stable choice.

Table 6. Nucleolus

Coalition	Award, \$/w	Instance(s)	Excess, \$/w
(1)	148	[m4.4xlarge]	148
(2)	148	[m4.4xlarge]	148
(3)	274	[m5.8xlarge]	274
(1, 2)	296	[m4.4xlarge, m4.4xlarge]	296
(1, 3)	422	[m4.4xlarge, m5.8xlarge]	367
(2, 3)	422	[m4.4xlarge, m5.8xlarge]	111
(1, 2, 3)	570	[m4.4xlarge, m4.4xlarge, m5.8xlarge]	570

Thus, the nucleolus should be determined within a limited choice space, after which FinOps can calibrate resource allocation based on each product team's main computing characteristics.

The suggested approach is as follows:

1. Define claims (instance price) and budget in game theory terms.
2. Find a solution that maximizes the smallest excess of a coalition (nucleolus) in relation to available cloud resources.
3. For each product team select an instance with a price lower than the allocated profit based on the product team's desired criteria.

The suggested approach is presented as a BPMN model in Figure 1.

Found nucleolus per each estate and possible instance types taken from Table 1 are shown in Table 7. One may notice that claim boundaries, monotonicity, resource monotonicity, and cloud constraints are maintained in the given examples.

Table 7. Nucleolus-based solutions for different estates

Estate, \$/w	Awards, \$/w		
	Product team 1	Product team 2	Product team 3
222	74	74	74
370	74	148	148
570	148	148	274
758	148	274	336

Estate, \$/w	Awards, \$/w		
	Product team 1	Product team 2	Product team 3
1107	148	411	548
1586	274	538	774

Based on the identified nucleolus-based solution, appropriate instances can be assigned (see Table 8).

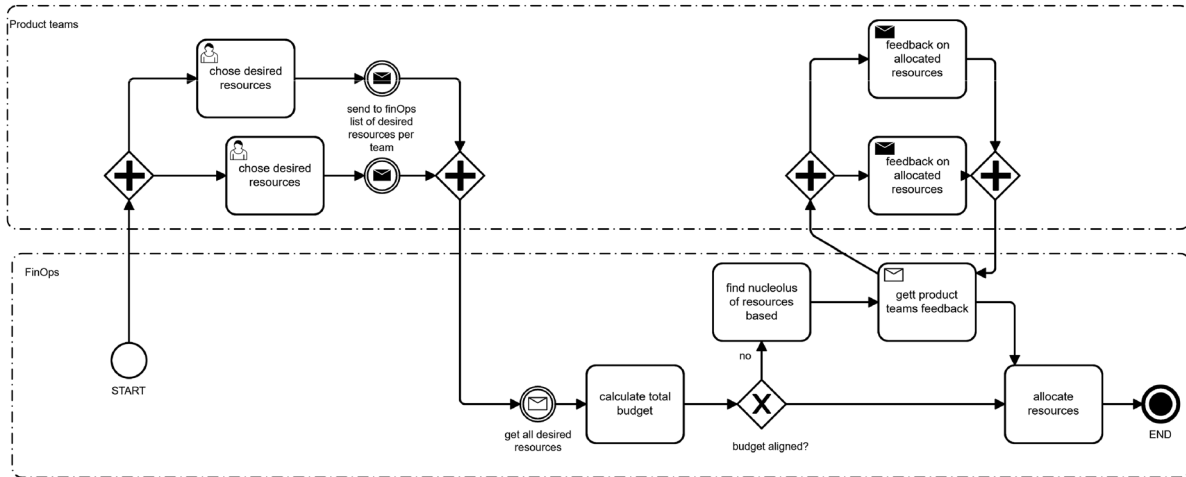


Figure 1. Suggested approach

Table 8. Allocated instances

Estate, \$/w	Product team 1	Product team 2	Product team 3
222	m4.2xlarge	m4.2xlarge	m4.2xlarge
370	m4.2xlarge	m4.4xlarge	m4.4xlarge
570	m4.4xlarge	m4.4xlarge	m5.8xlarge
758	m4.4xlarge	m5.8xlarge	m4.10xlarge
1107	m4.4xlarge	m5.12xlarge	m5.16xlarge
1586	m5.8xlarge	m4.16xlarge	m5.24xlarge

Thus, the m4.4xlarge has lower EBS than the m5.4xlarge, and product team 2's priority is EBS. The same applies to product team 3: CPU is their highest priority. Based on these requirements, some product teams may be offered an alternative solution (see Table 9), as their main characteristic requirements sometimes prove to be more flexible at later stages.

Table 9. Allocated instances based on total budgets and usage

Estate, \$/w	Product team 1	Product team 2	Product team 3
222	m4.2xlarge	m4.2xlarge	m5.2xlarge
370	m4.2xlarge	m4.4xlarge	m4.4xlarge
570	m4.4xlarge	m4.4xlarge	m5.8xlarge
758	m4.4xlarge	m5.2xlarge	m5.8xlarge
1107	m4.4xlarge	m5.12xlarge	m5.16xlarge
1586	m5.8xlarge	m4.16xlarge	m5.24xlarge

Conclusions

A game-theoretic approach to allocating cloud-based computing resources has been presented. In the computational service (or resource) market, each service incurs a cost based on the amount of resources used. Specifically, the allocation of cloud computing instances among product teams is examined. As each resource is allocated to a specific team and teams can cooperate with one another, the task can be considered a cooperative game.

A nucleolus-based approach is suggested to solve the cloud resource allocation problem. First, a nucleolus is determined based on the available cloud resource options, and then the solution is adjusted based on the product teams' objectives. Using the nucleolus maximizes the stability of resource allocation by minimizing the incentives of product team coalitions to deviate.

Список літератури

1. A game-theoretic method of fair resource allocation for cloud computing services / G. Wei et al. // *The journal of supercomputing*. — 2010. — Vol. 54. — Pp. 252–269. — <https://doi.org/10.1007/s11227-009-0318-1>.
2. Bei X. Fair and efficient multi-resource allocation for cloud computing / X. Bei, Z. Li, J. Luo // *Web and internet economics 18th international conference*. — Troy, NY, 2022. — Pp. 169–186.
3. Deochake S. Cloud cost optimization: a comprehensive review of strategies and case studies / S. Deochake. — 2023. — 36 p. (Preprint). — <https://doi.org/10.48550/arXiv.2307.12479>.
4. Dominant resource fairness: fair allocation of multiple resource types [Electronic resource] / A. Ghodsi et al. // 8th USENIX symposium on networked systems design and implementation. — Boston, MA, 2011. — Mode of access: <https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types>. — Title from screen.
5. Schmeidler D. The nucleolus of a characteristic function game / D. Schmeidler // *SIAM journal on applied mathematics*. — 1969. — Vol. 17, no. 6. — Pp. 1163–1170. — <https://doi.org/10.1137/0117107>.
6. What is FinOps? [Electronic resource] // FinOps Foundation. — Mode of access: <https://www.finops.org/framework/> (date of access: 28.12.2024). — Title from screen.

References

- Bei, X., Li, Z., & Luo, J. (2022). Fair and efficient multi-resource allocation for cloud computing. *Web and internet economics 18th international conference* (pp. 169–186). Springer.
- Deochake, S. (2023). *Cloud cost optimization: a comprehensive review of strategies and case studies*. <https://doi.org/10.48550/arXiv.2307.12479>.
- Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011). Dominant resource fairness: fair allocation of multiple resource types. *8th USENIX symposium on networked systems design and implementation*. USENIX Association. <https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types>.
- Schmeidler, D. (1969). The nucleolus of a characteristic function game. *SIAM journal on applied mathematics*, 17 (6), 1163–1170. <https://doi.org/10.1137/0117107>.
- Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The journal of supercomputing*, 54, 252–269. <https://doi.org/10.1007/s11227-009-0318-1>.
- What is FinOps? FinOps Foundation. <https://www.finops.org/framework/>.

Артюшенко Б. А.

РОЗПОДІЛ ХМАРНИХ РЕСУРСІВ НА БАЗІ НУКЛЕОЛУСА

Хмарні обчислення змінили процес менеджменту інфраструктури та ввели нові виклики, зокрема керування витратами на хмару. В роботі розглянуто розподіл хмарних ресурсів із метою оптимізації витрат. Показано, що розподіл хмарних ресурсів можливо розглядати як приклад кооперативної гри. Запропоновано підхід на базі нуклеолуса для розв'язання задачі максимізації найгіршого ексцесу коаліції при розподілі хмарних ресурсів. Розглянуто й досліджено приклади наближені до реальних із порівнянням з поширеними підходами.

Ключові слова: хмарні технології, розподіл ресурсів, кооперативна гра, нуклеолус.

Матеріал надійшов 14.03.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Колінько П. В.

АВТОМАТИЗОВАНА ГЕНЕРАЦІЯ І НАЛАШТУВАННЯ МІКРОСЕРВІСІВ ДЛЯ СПРОЩЕННЯ ПРОЦЕСУ РОЗРОБЛЕННЯ

У статті розглянуто підходи і методи автоматизованої генерації коду та структури застосунків, зокрема таких, що базуються на мікросервісній архітектурі. Описано розроблений програмний застосунок для генерації мікросервісної архітектури засобами платформи Node.js. В основу розробки покладено використання автоматизованої генерації програмного коду та архітектури на базі поняття scaffolding.

Ключові слова: scaffolding, мікросервісна архітектура, генерація коду.

Вступ

Розроблення вебзастосунків завжди починається з аналізу бізнес-вимог і підбору архітектури. Одним із важливих питань, що постає на початку розроблення бекенд-застосунків, є вибір поміж монолітною і мікросервісною архітектурами, кожна з яких має свої переваги та недоліки. Проте варто зазначити, що у кожній із зазначених архітектур є спільний етап конфігурації, що передбачає налаштування середовища, встановлення бібліотек, підключення бази даних і багато супутніх кроків. У разі розроблення мікросервісного застосунку, процес конфігурації повторюється велику кількість разів, адже кожен сервіс є окремим застосунком в ізольованому середовищі, що значно сповільнює процес імплементації.

Для вирішення проблеми повільного розроблення і запобігання помилкам у момент конфігурації використовують scaffolding-утиліти. Scaffolding передбачає певну автоматизацію генерації коду, шаблону, архітектури проєкту. На базі платформи Node.js існують такі генеративні утиліти, як Nest CLI, Yeoman, Prisma CLI тощо. Однак жодна з перелічених бібліотек не вирішує проблеми автоматизованої генерації мікросервісів.

Монолітна архітектура — це класичний підхід у розробленні програмного забезпечення, особливо для бекенд-застосунків, який передбачає, що проєкт є цілісною та незалежною сутністю. На противагу, мікросервісна архітектура передбачає, що найбільшою незалежною одиницею розподіленої системи є один мікросервіс, що представлений у формі атомарного застосунку. Зазвичай кожен такий сервіс побудований довкола конкретної бізнес-логіки, наприклад, окремий сервіс для аутентифікації, листування тощо. Кожен з них є розгорнутим в окремому середовищі, зазвичай має власну базу даних.

Структура проєкту, оснований на мікросервісній архітектурі

Через особливості своєї структури розподілені системи мають додатковий функціонал для правильної взаємодії між собою, оскільки під час розроблення мікросервісів легко втратити контроль над їхньою кількістю. Завершений мікросервісний додаток має структуру, представлену на рис. 1.

Найвищим рівнем завершеного доробку є клієнт. Зазвичай фронтенд-застосунок, що має графічний інтерфейс, зрозумілий людині, з якою взаємодіє користувач. Комунікація між клієнтом і сервером у монолітній і мікросервісній архітектурах не відрізняється, тобто запити надсилають за допомогою API.

Наступним рівнем є функціональна бізнес-логіка, що відповідає за оброблення інформації, збір статистики, впровадження відкладених задач тощо. Його поділяють на два підтипи:

- experience;
- domain.



Рис. 1. Структура мікросервісного застосунку

Рівень *experience* відповідає за бізнес-логіку і взаємодію з даними, клієнтом та іншими навантаженими частинами застосунку, натомість *domain* відповідає за вузькоспеціалізований функціонал, як-от шифрування даних.

Використання окремого середовища для сервісів передбачає необхідність комунікації між ними. Обмін даними та повідомленнями між сервісами відбувається за допомогою *Event Brokers*, *Message Queues* або інших патернів взаємодії сервісів. Такі додаткові утиліти дають змогу асинхронно передавати інформацію від сервісу до сервісу. Черги повідомлень, такі як *RabbitMQ*, *Apache Kafka*, — це зовнішні утиліти, що дають змогу асинхронно обмінюватись даними між різними компонентами застосунку безпосередньо. Окрім мікросервісної архітектури, можуть бути використані в рамках моноліту чи хмарної архітектури. Вони працюють за принципом обміну повідомленнями від одного ресурсу (який називають *publisher* або *producer*) до іншого (*consumer*). Оскільки операція є асинхронною, тобто такою, що час для її виконання є невідомим, повідомлення публікуються у так звану чергу, де зберігаються за принципом «отримати і забути». Брокери подій (*event brokers*) є схожим механізмом до черги повідомлень, тобто це такі посередники, що обробляють дані в момент їх передавання від *publisher* до *subscriber*. Вони мають доступ до контексту та метаданих самого повідомлення й можуть робити з ними такий набір операцій, як читання та запис. Проте, на відміну від черг повідомлень, брокери подій реалізовані так, що *publisher* і *subscriber* не обов'язково мають бути напряму пов'язані між собою, адже за керування «напрямком» інформації відповідає саме такий посередник, що надсилає дані від ресурсу-публікатора повідомлення до одного або декількох підписників.

Комунікація клієнта і сервера в рамках мікросервісної архітектури відбувається або з кожним сервісом окремо, або за допомогою шлюзу *API Gateway*. Шлюз — це єдина точка входу в додаток. Фактично, це окремий застосунок, який розсилає запити на дочірні сервіси всього застосунку і повертає відповідь клієнту.

Кожен сервіс у рамках мікросервісної архітектури є незалежним додатком, тому він має бути розгорнутим в окремому середовищі. Для спрощення цього процесу використовують контейнеризацію, яка передбачає розгортання програмного застосунку в ізольованому віртуальному середовищі, умовному «контейнері», що не взаємодіє із зовнішніми компонентами системи, проте має змогу звертатися до них. Контейнеризація реалізується найчастіше за допомоги спеціальних програмних продуктів типу *Docker* або *Kubernetes*. Останнє програмне забезпечення є оркестратором, тобто «центром керування» всіх контейнеризованих сервісів, що спрощує і пришвидшує процес інтеграції, розроблення та паралельного розгортання декількох версій продукту.

Scaffolding у програмуванні

Підхід *scaffolding* передбачає генерацію за якимось загальним шаблоном або «скелетом». Ідеться не обов'язково про генерацію кінцевої версії продукту, що є готовим до використання, навпаки, частіше згенерована частина є лише основою, що спрощує та значно пришвидшує подальше розроблення застосунку.

Scaffolding можна поділити на дві основні категорії: генерацію коду і генерацію архітектури.

У першому випадку утиліта допомагає генерувати шматки типового коду, такі як інтерфейси, функції *getter* і *setter* тощо. Цей підхід підтримується деякими *model-view-controller* (*MVC*) фреймворками.

Типовий приклад генерації коду — scaffolding у RoR (Ruby on Rails), що працює за допомогою CLI (command line interface) і передбачає набір консольних команд, які виконують будь-який функціонал у рамках проєкту, зазвичай різноманітні програмні скрипти.

На противагу генерації в рамках окремого фреймворку, така утиліта, як Yeoman, має набагато ширший спектр використання, наприклад генерацію фронтенд- і бекенд-застосунків на різних мовах програмування. Проте наразі не створено такого yeoman-генератора, що вирішував би задачу генерації мікросервісної архітектури та окремих сервісів. Розробити такий генератор цілком можливо, проте такий підхід має декілька недоліків. По-перше, необхідно повністю підв'язуватись під можливості Yeoman, тобто в разі припинення підтримки коду генератор буде так само важко підтримувати. По-друге, можливості SDK є обмеженими і нерідко застарілими, тоді як Node.js має ширший і новіший функціонал.

Отже, розроблення застосунку для генерації мікросервісів, що є глобально незалежним від сторонніх бібліотек, є більш оптимальним і стабільним рішенням, адже у цьому випадку єдина технологія, що потрібна для «життя» проєкту, — це платформа Node.js. Ця платформа з часом усе більше і більше набирає популярності, що означає її постійну підтримку з боку розробників. Тому кінцевим варіантом застосунку для генерації мікросервісів стало поєднання найкращих практик із кожної дослідженої утиліти — простота у використанні за допомогою CLI команд і узагальнений широкий підхід, оснований на філософії yeoman-generators.

Архітектура програмного застосунку

Перед початком розроблення власного scaffolding-застосунку для генерації мікросервісів необхідно визначитися з архітектурою. Потрібно створити максимально зрозумілий користувацький інтерфейс за допомогою CLI, в якому не буде потреби довго розбиратись. Тому з метою уникнення потреби проходити «опитування», як у випадку роботи застосунку Yeoman, було вирішено описувати всі необхідні конфігурації в одному джерелі — конфігураційному файлі. Такий опис конфігурацій запозичено з підходу Iac (Infrastructure as a Code, інфраструктура як код) для опису та керування інфраструктурою хмарних сервісів за допомогою конфігураційних файлів. Первинний вигляд архітектури програмного застосунку та структуру згенерованого проєкту схематично зображено на рис. 2 і 3.

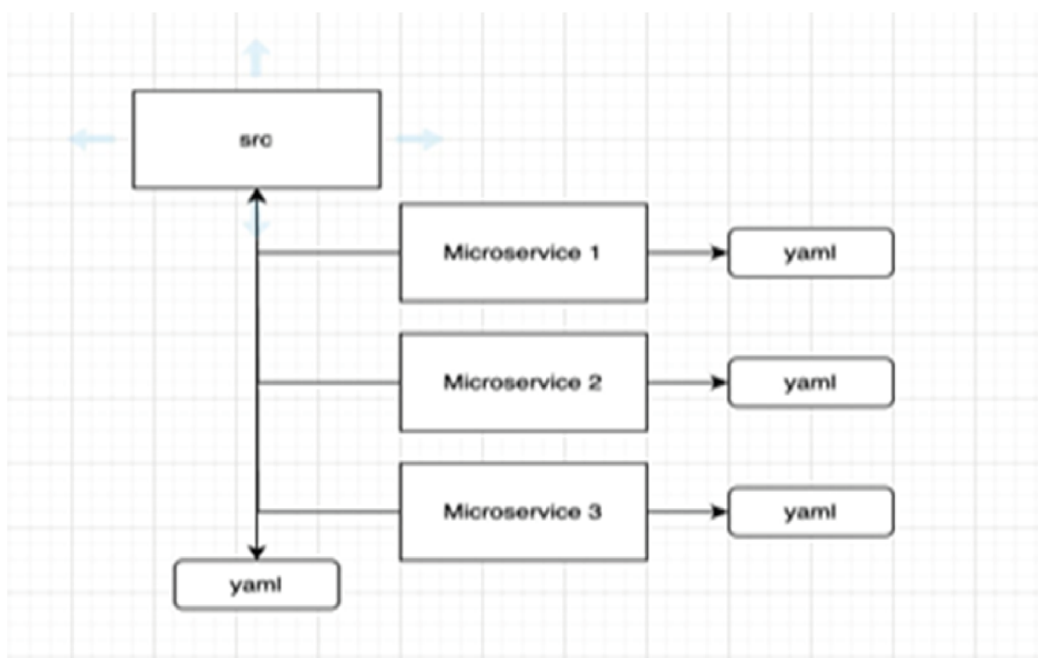


Рис. 2. Первинна архітектура проєкту

Проте під час роботи над застосунком багато початкових рішень було замінено на більш оптимальні. Наприклад, було прийнято рішення про спільний файл, що відповідатиме за налаштування проєкту й кожного сервісу. У такому разі весь додаток можна буде контролювати з «одного джерела» і користувачеві не доведеться робити зайву роботу, як-от створювати різні папки та файли.

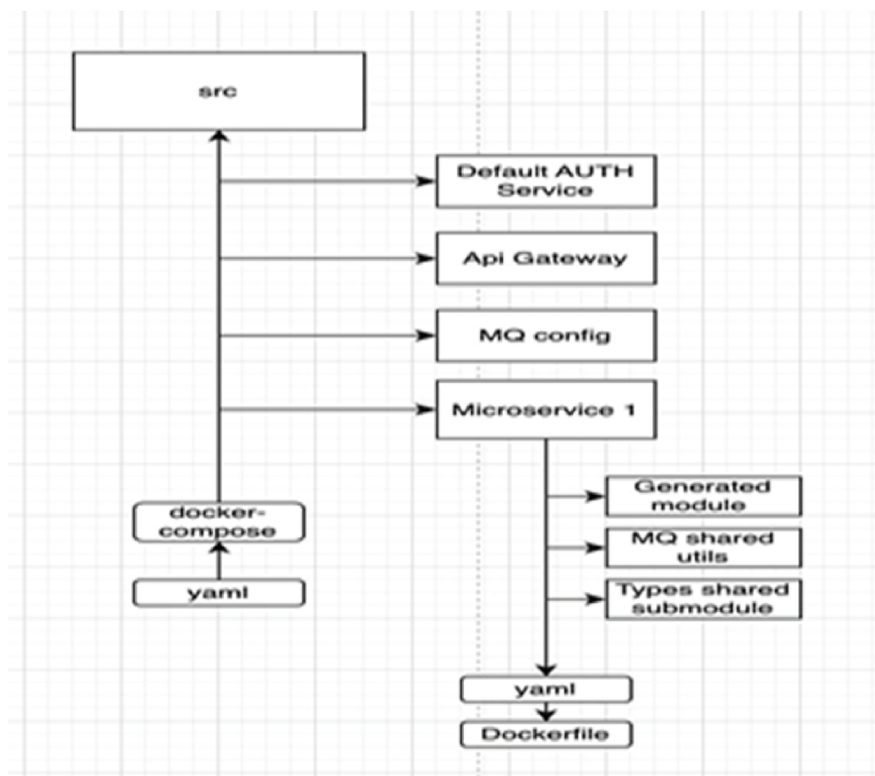


Рис. 3. Архітектура проєкту після виконаної генерації

Друга важлива зміна, яку було впроваджено, — це зміна формату конфігураційного файлу `.yaml` на формат `.json`, адже він є ідеально сумісним із мовою програмування JavaScript.

```

1  microgen.json x
2  microgen.json > [ ] microservices
3  {
4    "name": "My first app",
5    "externalModules": [],
6    "gateway": true,
7    "eventBroker": {
8      "driver": "kafka"
9    },
10   "microservices": [
11     {
12       "name": "user",
13       "framework": "express",
14       "language": "ts",
15       "deps": ["rxjs", "body-parser", "yup"],
16       "devDeps": ["nodemon"],
17       "database": {
18         "driver": "mysql",
19         "orm": "prisma"
20       },
21       "envFilePath": "",
22       "dockerFile": true
23     },
24     {
25       "name": "profile",
26       "framework": "nest",
27       "language": "ts",
28       "deps": ["yup", "yarn"],
29       "devDeps": [],
30       "dockerFile": true,
31       "database": {
32         "driver": "psql"
33       }
34     },
35     {
36       "name": "profileNest",
37       "framework": "nest",
38       "language": "ts",
39       "deps": ["yup", "yarn", "body-parser"],
40       "devDeps": ["nodemon"],
41       "dockerFile": true,
42       "database": {
43         "driver": "psql",
44         "orm": "prisma"
45       }
46     }
47   ]
48 }

```

Рис. 4. Приклад конфігураційного файлу `microgen.json`

Для старту генерації необхідно запустити команду `npm run generate - -path=/path-to-microgen.json`. Консольна змінна `-path` вказує на абсолютний шлях до кореневого конфігураційного файлу.

Опис використаних технологій

Як мову програмування було вибрано Typescript. Розроблення будь-якого програмного застосунку за допомогою Typescript має низку переваг і недоліків. З-поміж недоліків — потреба витрачати більше часу на розроблення безпосередньо. Проте для такого застосунку, як *microgen*, критично важливим є правильний, односторонній потік даних і «виразний» опис кожної окремої сутності проєкту. Такий вибір мови програмування дає змогу підтримувати наявний код, розширювати наявний функціонал, мінімізує кількість помилок, пов'язаних із неоднорідністю кодової бази.

Для старту генерації необхідно звернути увагу на консольні аргументи, що передаються під час ініціалізації, а саме змінну *path*. Оскільки скрипт має бути універсальним і дозволяти генерацію для будь-якої файлової системи, без прив'язки до конкретної директорії, необхідно дати змогу користувачеві динамічно зазначати абсолютне розташування *microge.json* файлу. І для того, щоб отримати такий шлях до файлу, потрібно передати консольні аргументи і розпарсити їх. Для оброблення такого випадку вирішили відмовитися від сторонніх бібліотек і запровадити власне рішення, адже всі наявні варіанти не працюють належно.

Для генерації файлів, копіювання, зчитування, встановлення залежностей та інших операцій необхідно було обрати модулі їх виконання. В платформі Node.js існують дві вбудовані бібліотеки, які добре підійдуть для виконання поставленої задачі, — *fs* і *node:child_process*.

Для процесу логування даних вибрали зовнішню бібліотеку *Winston*, що імплементує головні методи виводу даних у консоль, відповідно до патерну *Logger*, такі як *log*, *info*, *warn* та *error*.

Модулі програмної реалізації

У цьому розділі описано, які рішення було впроваджено під час розроблення програмного забезпечення та аргументовано причину їх використання. Роботу такого застосунку організовано за принципом переходу від найменшої задачі до найбільшої. Такий підхід є необхідним, адже кожний наступний крок у генерації є фактично надбудовою над попереднім. Для спрощення розуміння та читабельності коду процес генерації було розбито на атомарні модулі, кожен з яких відповідає виключно за конкретну одиницю функціоналу.

Перший етап передбачає парсинг і валідацію головного конфігураційного файлу *microgen.json*. Читання файлу відбувається засобами бібліотеки *fs:promises* і записом даних у RAM. Оскільки формат файлу є *.json*, немає потреби в додаткових кроках, адже такий формат добре підходить для оброблення мовою Javascript.

```
export class ChildProcessBuilder {
  private query: string = "";
  private readonly executionHandler = null;
  constructor(customHandler?: any) {
    this.executionHandler = customHandler;
  }

  append(command: string): typeof this {
    if (!command) {
      return this;
    }
    if (!this.query) {
      this.query = command;
    } else {
      this.query = `${this.query} && ${command}`;
    }
    return this;
  }

  exec(): void {
    logger.info(`Executing query: ${this.query}`);
    nodeExec(this.query, (error, stdout) => {
      stdout && logger.info(stdout);
      error && logger.error(error);
    });
  }

  async execAsync(): Promise<any> {
    return util.promisify(require("node:child_process").exec)(this.query);
  }
}
```

Рис. 5. Імплементация класа ChildProcessBuilder

Валідація файлу впроваджується окремою функцією і є простою перевіркою на відповідність даних заданому шаблону: чи заповнені всі обов'язкові поля в загальній конфігурації проєкту або окремого мікрсервісу, чи існує імплементація вибраного варіанта. У разі невідповідності даних такому шаблону користувачу буде виведено повідомлення про помилку, де буде описано, що необхідно змінити для продовження роботи.

Важливо розглянути процес генерації коду засобами Node.js безпосередньо. Генерація коду відбувається в рамках атомарного модуля, що представлено як окрему функцію генерації. Для виконання консольних команд розроблено клас `ChildProcessBuilder`, що є імплементацією патерну `Builder`. Це такий патерн, що породжує шаблон об'єкта, або, як у випадку генерації коду, шаблон консольної команди, що потім буде виконана за допомогою метода `exec`.

За допомогою методу `append` клас об'єднує консольні команди в одну та зберігає їх у формі рядка. Цей клас має на меті виконання ланцюга команд послідовно. Оскільки процес, створений методом `exec`, є ізольованим, такий підхід — єдина можливість виконати набір команд послідовно.

Наступна проблема, яка виникла під час розроблення застосунку, — асинхронність виконання команд. Команда `exec` працює так, що час на виконання команди є невизначеним, проте виклик цього самого методу виконується синхронно. Для вирішення цієї проблеми було використано пакет `utils.promisify`, який працює подібно до `fs.promises` і дає змогу обробляти проміси за допомогою `async/await` синтаксису, що дає можливість очікувати нового завершення виконання консольної команди. Нижче наведено приклад створення команд для кодогенерації за допомогою `ChildProcessBuilder`.

```
export const initializeNodeProjects = (): Promise<any>[] => {
  const promises = __PROJECT_METADATA__.microservices
    .filter(({ framework }) => framework !== Frameworks.NEST)
    .map(({ absolutePath }) => {
      return new ChildProcessBuilder()
        .append(osExecutableCommands.changeDirectory(absolutePath))
        .append(npmExecutableCommands.npmInit())
        .execAsync();
    });

  logger.info("Initializing Node projects");

  return promises;
};
```

Рис. 6. Генерація директорій та ініціалізація порожнього Node.js проєкту

Список консольних команд було описано у формі пари ключ–значення, де ключ — назва команди, а значення — функція, яка повертає текстовий рядок із сигнатурою команди. Вони розділені на два типи — `npm`-команди та команди `os`. Список `npm`-команд відповідає за весь менеджмент, що пов'язаний із Node.js та `npm`, тоді як список `os` відповідає за команди операційної системи, наприклад створення директорії, копіювання файлів тощо. Кожна з цих команд описана у формі функції, а не у формі рядка, тому що деяким командам для роботи необхідно динамічно приймати аргументи. Завдяки такому підходу та можливостям мови програмування `Typescript` є змога уникнути помилок у разі, якщо у функцію буде передано аргумент, що не буде використаним. Тож можливо зіставити варіант із конфігураційного файлу і відповідну директиву, наприклад ініціалізацію фреймворку або мови програмування.

```
const frameworkConfigs: Record<string, CallableFunction> = {
  [Frameworks.EXPRESS]: npmExecutableCommands.npmInitExpress,
  [Frameworks.NEST]: npmExecutableCommands.npmInitNest,
  [Languages.TS]: npmExecutableCommands.npmInitTypeScript,
  [Languages.JS]: npmExecutableCommands.npmNoop,
  [Languages.DEFAULT]: npmExecutableCommands.npmNoop,
};
```

Рис. 7. Приклад словника `npm`-команд для ініціалізації фреймворку

Оскільки проєкт написано мовою програмування `Typescript`, який своєю чергою транспілюється, тобто перекладається в `Javascript`, усі файли, що мали розширення не `.js` або `.ts`, не потрапляли в кінцевий `Javascript bundle`, тобто побудований проєкт, готовий до використання. Такий підхід є стан-

дартною поведінкою tsc білдера, за допомогою якого Typescript перетворюється на Javascript, тому необхідно було розробити підхід, щоби всі ресурси проєкту зберігались у кінцевому продукті. Для цього розробили скрипт, що на етапі білд-проєкту дає змогу копіювати всю директорію із Typescript директорії в аналогічну директорію Javascript білда.

Вибудована архітектура проєкту має чималу кількість переваг, як-от стабільність і безвідмовність роботи, універсальність, а головне — розширюваність, адже у розроблену кодову базу дуже легко додати нові фреймворки та конфігурації.

Аналіз розробленого програмного забезпечення

У результаті розроблення програмного забезпечення було досягнуто основної мети зі створення стабільного застосунку, що є мінімально залежним від сторонніх бібліотек і має простий у використанні інтерфейс.

Єдина «точка входу», що представлена конфігураційним файлом, дає змогу не тільки полегшити процес розроблення на початкових етапах, а й легко керувати розширенням продукту, що використовує утиліту microgen. Наприклад, першочергові бізнес-умови проєкту вимагали створення лише двох сервісів, під назвою user і bookings.

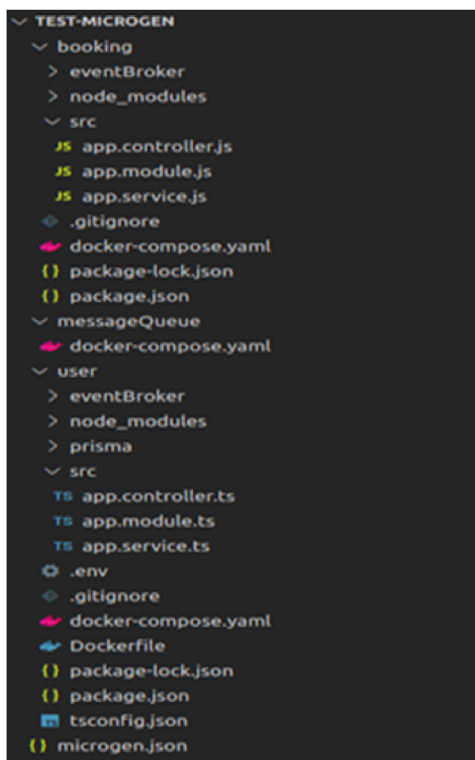


Рис. 8. Згенерована кодова база проєкту

Окрім мови програмування, в конфігураціях сервісів було вибрано фреймворк Express, необхідні node_modules пакети, які треба встановити на етапі генерації, бази даних postgresql та mysql із додатковими ORM prisma та typeorm відповідно. Для мікросервісу під назвою user також був вибрано параметр dockerFile, що відповідає за створення контейнера для згенерованого сервісу. Як показано на рис. 6, бажані сервіси було згенеровано з дотриманням конфігураційних вимог. Також було додано сервіс для взаємодії із чергою повідомлень Apache kafka, зазначений у загальній конфігурації проєкту.

Якщо бізнес-вимоги до проєкту буде розширено, може постати потреба створити новий мікросервіс, наприклад засобами фреймворку Nest.js. Якщо додати в конфігураційний microgen.json файл опис такого сервісу та повторно використати CLI команду старту генерації, сервіс буде додано, а всі інші, що вже були згенеровані, не матимуть жодних змін, адже всі сервіси, що містять package.json файл, будуть проігноровані для повторної генерації. Такий підхід є необхідним для запобігання програмного втручання в зміни, внесені розробниками.

Висновок

Розроблений додаток microgen має низку вагомих переваг, таких як незалежність від бібліотек, що є нестабільними у підтримці, але і використовує вже розроблені рішення, такі як Nest CLI або Prisma CLI. Такий підхід дає змогу позбутись зайвої розробки додаткового коду та ресурсів застосунку, згенерувавши скелет мікросервісу, що використовує фреймворк Nest.js або ж ініціалізацію конфігураційного Prisma файлу, за допомогою однієї команди, без необхідності власноруч прописувати код. Гнучка архітектура застосунку microgen дозволяє швидко додавати нові ресурси та розширювати функціонал.

Проте розроблений додаток також має певні недоліки. Наприклад, наразі функціонал додатку є обмеженим певним набором технологій, тобто він здатен працювати лише з двома мовами програмування, двома фреймворками — Express і Nest.

Ще одним великим недоліком є брак функціоналу збереження версіонування (попереднього стану) проєкту. Підхід Iac та утиліта terraform, з яких було запозичено ідею єдиного конфігураційного файлу як основи генерації проєкту, мають функціонал, що дає змогу порівнювати попередню версію застосунку із теперішньою конфігурацією на наступному етапі генерації. Наприклад, такий функціонал є корисним для видалення конфігурацій, таких як зміна бази даних, відмова від використання docker-compose.yaml тощо.

Підсумовуючи, варто зазначити, що розроблена утиліта не має чітких відповідників у рамках поставленої задачі. Вона спрощує та пришвидшує розроблення мікросервісів на базі платформи Node.js, що можуть бути згенеровані на базі найпопулярніших технологій цієї платформи. Проте можливості застосунку обмежуються використанням невеликого переліку найпопулярніших технологій і відсутністю версіонування.

Список використаних джерел

1. Altcademy Team. What is Scaffolding in Ruby on Rails? [Electronic resource] // Altcademy Blog. — Mode of access: <https://www.altcademy.com/blog/what-is-scaffolding-in-ruby-on-rails/#:~:text=Scaffolding%20in%20Ruby%20on%20Rails%20is%20a%20helpful%20feature%20that,practices,%20and%20speeds%20up%20development,2024>. — Title from screen.
2. Global Logic. Mikroservisna arkhitektura dlia pochatkivtsiv. Chastyna I [Electronic resource] // GlobalLogic Ukraine. — Mode of access: <https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/>. — Title from screen.
3. Hoang C. Monolith Architecture [Electronic resource] / C. Hoang // Medium. — Mode of access: <https://tech.tamara.co/monolith-architecture-5f00270f384e>. — Title from screen.
4. IBM. What Is a Message Queue? [Electronic resource] // IBM in Deutschland, Österreich und der Schweiz. — Mode of access: <https://www.ibm.com/topics/message-queues>. — Title from screen.
5. Microsoft. What is infrastructure as code (IaC)? Azure DevOps. Microsoft Learn: Build skills that open doors in your career [Electronic resource]. — Mode of access: <https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code>. — Title from screen.
6. Nagpal A. Monolithic vs Microservices Architecture: Advantages, Disadvantages, And Differences [Electronic resource] / A. Nagpal // Medium. — Mode of access: <https://medium.com/@jasminepuno/monolithic-vs-microservices-architecture-advantages-disadvantages-and-differences-2bee6d1da8ca#:~:text=Monolithic%20architecture%20is%20a%20conventional,closely%20connected%20and%20centralized%20system>. — Title from screen.
7. Naik A. How to scaffold ExpressJS server and test it [Electronic resource] / A. Naik // Medium. — Mode of access: <https://medium.com/craft-academy/how-to-scaffold-expressjs-server-and-test-it-d2a2ab1d30e0>. — Title from screen.
8. NestJS. Documentation | NestJS — A progressive Node.js framework [Electronic resource]. — Mode of access: <https://docs.nestjs.com/>. — Title from screen.
9. NodeJS. Child process | Node.js v22.2.0 Documentation. Node.js – Run JavaScript Everywhere [Electronic resource]. — Mode of access: https://nodejs.org/api/child_process.html. — Title from screen.
10. Solace. What is an Event Broker? [Electronic resource] // Solace. — Mode of access: <https://solace.com/what-is-an-event-broker/>. — Title from screen.
11. Yeoman. Getting started with Yeoman [Electronic resource] // Yeoman. — Mode of access: <https://yeoman.io/learning/>. — Title from screen.

References

- Documentation: Nestjs — a progressive node.js framework. NestJS. (n. d.). <https://docs.nestjs.com/>.
- Getting started with yeoman. Yeoman. (n. d.). <https://yeoman.io/learning/>.
- GlobalLogic Ukraine. Microservice arcitecture for the bigginers. (2023, March 16). <https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/>.
- Hoang, C. (2023, December 27). *Monolith architecture*. Medium. <https://tech.tamara.co/monolith-architecture-5f00270f384e>.
- Mijacobs, E. (n. d.). *What is infrastructure as code (IAC)? — azure DevOps*. Azure DevOps | Microsoft Learn. <https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code>.
- Nagpal, A. (2024, July 30). *Monolithic vs microservices architecture: Advantages, disadvantages, and differences*. Medium. <https://medium.com/@jasminepuno/monolithic-vs-microservices-architecture-advantages-disadvantages-and-differences-2bee6d1da8ca#:~:text=Monolithic%20architecture%20is%20a%20conventional,closely%20connected%20and%20centralized%20system>.

- Naik, A. (2020, June 17). *How to scaffold expressjs server and test it*. Medium. <https://medium.com/craft-academy/how-to-scaffold-expressjs-server-and-test-it-d2a2ab1d30e0>.
- Node.js V22.8.0 documentation. Child process | Node.js v22.8.0 Documentation. (n.d.). https://nodejs.org/api/child_process.html.
- Team, A. (2023, May 3). *What is scaffolding in Ruby on Rails?*. Altcademy Blog. <https://www.altcademy.com/blog/what-is-scaffolding-in-ruby-on-rails/#:~:text=Scaffolding%20in%20Ruby%20on%20Rails%20is%20a%20helpful%20feature%20that,practices,%20and%20speeds%20up%20development>.
- What is a message queue? IBM. (2021, October 8). <https://www.ibm.com/topics/message-queues>.
- What is an event broker? Solace. (2024, April 3). <https://solace.com/what-is-an-event-broker/>.

P. Kolinko

AUTOMATED GENERATION AND CONFIGURATION OF MICROSERVICES TO SIMPLIFY THE DEVELOPMENT PROCESS

The article “Automated Generation and Configuration of Microservices to Simplify the Development Process” explores how automated code generation and structured design streamline microservice-based application development. It introduces a custom software application built with Node.js, designed to automate the creation of a microservice architecture through scaffolding. This method simplifies the setup of core structures, speeding up development. The article starts with a comparison between monolithic and microservice architectures. In monolithic systems, all components—the user interface, business logic, and databases—are tightly integrated, making scaling and updates challenging as the system grows. Microservices break the application into independent services, allowing developers to scale, test, and deploy each part individually. This flexibility leads to improved fault tolerance and adaptability.

It also delves into microservice architecture layers, detailing how services communicate via lightweight protocols like HTTP and message queues. The importance of service discovery and load balancing, which ensure smooth communication in dynamic environments, is highlighted.

Node.js is chosen for its event-driven, non-blocking architecture, which is well-suited for handling multiple requests. Other technologies, such as Express.js, MongoDB, and Docker, further enhance scalability and efficiency. A key focus is on scaffolding, a technique that automates the generation of boilerplate code and project structure. This reduces repetitive tasks, ensures a consistent architecture, and improves development speed.

The article concludes by outlining the development of the custom application, showing how automation accelerates microservice creation, making the process more efficient and scalable.

Keywords: scaffolding, microservice architecture, code generation.

Матеріал надійшов 02.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Верета В. В., Ткаченко В. О.

АВТОМАТИЗОВАНА ЛОКАЛІЗАЦІЯ ЗАСТОСУНКІВ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

У цій статті проведено аналіз різних інструментів і сервісів для здійснення локалізації (перекладу) вебзастосунків, методів і підходів до їх інтеграції та масштабування. Розглянуто наявні рішення для управління локалізаціями вебсервісів. Описано розроблену архітектуру, яка дає змогу швидко адаптувати сервіс під потреби різних користувачів і різних проєктів та ефективно інтегруватися з різними платформами з урахуванням простого масштабування сервісу.

На базі цієї архітектури реалізовано вебсервіс EchoLocal. Він покращує взаємодію учасників процесу локалізації вебзастосунків, а також дає змогу оптимально інтегрувати цей процес у розподілену мікросервісну архітектуру сучасних застосунків і просто розгортати у своїй закритій екосистемі. EchoLocal надає можливість своїм користувачам інтегрувати та управляти локалізаціями своїх продуктів в одному місці в реальному часі.

EchoLocal допоможе знизити витрати на локалізацію, спростити процес адаптації контенту під різні мовні ринки та підвищити ефективність комунікації між усіма учасниками процесу локалізації.

Ключові слова: локалізація, інтернаціоналізація, глобалізація, мікросервісна архітектура, Kubernetes, Google Cloud Platform, GCP, CI/CD.

Вступ

Локалізація, або переклад, вебсервісів стає вирішальною задачею у процесі глобалізації програмних продуктів, онлайн-рішень та автоматизації процесів оброблення інформації [1]. Нині дедалі більшого значення набувають своєчасність і чіткість адаптації контенту до потреб різномовних користувачів. Вебсервіси, які мають більше локалізацій, зазвичай здатні залучити ширшу аудиторію і забезпечити кращий, клієнтоорієнтований досвід. Однак управління багатьма мовами може стати складним, ресурсоємним завданням без належних інструментів та підходів. Водночас, потреба у своєчасній і точній локалізації лише зростає, а ефективність її реалізації може значно впливати на успіх продукту на міжнародному рівні.

Саме тому необхідно підвищувати обізнаність бізнесу та продуктових рішень щодо локалізації та забезпечити розуміння не меншої важливості цього, ніж конкретні особливості систем.

Нині існує декілька взаємопов'язаних понять, які безпосередньо стосуються приведення та перенесення онлайн-продукту на нові ринки, але не є чимось одним і тим самим. Є поняття локалізації (localisation, l10n), яке буде досліджене у цій роботі, і поняття інтернаціоналізації (internationalisation, i18n). Їх часто використовують як взаємозамінні, однак іноді це не так.

Цифровий продукт можна придбати онлайн, про нього можна дізнатись онлайн і він не вимагає якогось локального представництва. Тобто тепер, маючи ідею продукту для споживача, нам не обов'язково створювати собі рамки і кордони. Ось тут нас і чекають ускладнення, адже що ми знаємо про весь світ? Ми знаємо і розуміємо мову нашого регіону, форматування дат, валют і чисел; однак знайомі нам образи і формати можуть бути незрозумілими нашим покупцям на іншому кінці світу. Отже, ми маємо подумати про інтернаціоналізацію, вбудувати її і додати локалізацію для підтримки інших регіонів.

Інтернаціоналізація — узагальнене поняття, що означає принципи проєктування та реалізацію програмного продукту або документації в такий спосіб, який максимально спростить подальшу локалізацію [11]. Можна сказати, що це технічне вдосконалення продукту для того, щоб забезпечити, по-перше, можливість його адаптації, а по-друге, легкість локалізації для різних ринків без необхідності виконання дорогих змін. Це фундамент, на якому будують подальші налаштування відкритості світу, іншим цінностям і форматам.

Локалізація — процес адаптації програмного продукту до мови та культури клієнта [11]. Тобто саме локалізація стосується процесу зміни вже інтернаціоналізованого (інколи глобалізованого) продукту для конкретного ринку. Це конкретні зміни виду, форматів та мови для споживачів конкретної країни або регіону.

Наведемо кілька прикладів, щоб уточнити поняття інтернаціоналізації:

- створення продукту з урахуванням можливості використання різних мов, які зберігаються в окремих джерелах і не вбудовані у саму систему;
- проєктування продукту з урахуванням нестандартного розміщення тексту справа наліво, зверху вниз; проєктування компонентів і використання вже перевірених компонентів, які вміють працювати з такими змінами;
- використання у продукті локалізованих знаків валют, дат, чисел тощо, що зазвичай досягається вбудованими утилітами мов програмування.

Тепер, розуміючи поняття інтернаціоналізації, визначимо наше розуміння локалізації [17]:

- процес адаптації продукту або контенту для конкретного ринку, регіону;
- передбачає переклад, адаптацію та зміну продукту, щоби врахувати конкретні лінгвістичні, культурні або регуляторні вимоги конкретної країни, регіону або ринку, наприклад використання локалізованого формату дат, валют та одиниць вимірювання;
- виконують за конкретним запитом, відповідає поточному етапу розширення та впливу.

Обидва процеси надзвичайно важливі для бізнесу або продукту, який хоче розширитися та бути досяжним глобально. Ці інтегровані процеси дають змогу компаніям надавати більш персоналізований і відповідний досвід для своїх користувачів.

Отже, локалізація — це процес адаптації продукту або контенту під конкретну культуру або мовну аудиторію. Локалізація забезпечує не тільки переклад тексту, а й адаптацію графічних елементів, валют, форматів дати та часу, використання правильних культурних референцій, а також технічних вимог, як-от кодування символів. Процес локалізації потрібен для того, щоб продукт легко сприймався користувачем, підвищуючи тим самим його зручність та ефективність використання в різних регіонах і ринках. Така адаптація дає змогу компаніям ефективніше взаємодіяти з міжнародною аудиторією та розширювати свій вплив.

Сучасні підходи до локалізації вебзастосунків

Вибір підходу до локалізації або перекладу вебзастосунків може бути непростю задачею. Звісно, кожен розробник, менеджер або власник проєкту обирає сам, як проходитиме процес локалізації, наскільки це критично і хто за це відповідатиме. Кількість інформації щодо цього, а також інструментів, теж немала, тож розглянемо загальні підходи і конкретно запропонуємо своє рішення на стику цих підходів.

Починаючи зі вбудованих, пов'язаних із кодом підходів до повноцінних сервісів, які повністю переписують ваш контент, немає єдиного правильного рішення, яке підходитиме в кожному випадку. Від загального підходу до бізнесу, налаштованих бізнес-процесів і навіть орієнтації самої компанії буде залежати, який підхід до локалізації служитиме краще. Наприклад, є повноцінно комп'ютерно налаштовані організації, де все йде від розробників, де всі говорять зрозумілою лише технічним спеціалістам мовою. Тут мабуть не буде абсолютною проблемою, якщо процес налаштування і зміни локалізованого контенту проходитиме через розробника із фактичними змінами у кодову базу. Але ж є і такі компанії, які все ще потребують послуг розробників, але не є тільки комп'ютерно налаштованими, де міркують більше самою ідеєю, заробітком та маркетингом. Напевно тут було б краще вибрати підхід, який потребує меншої участі розробника або ж взагалі все можна зробити без нього.

Перелічимо орієнтовні запитання, які варто поставити перед вибором підходу. Хто створює і коректує контент на вебсайті? Як часто планують зміни до вже готового тексту? Хто відповідає за переклад? Наскільки компанія орієнтована на ІТ і скільки розробників можуть бути задіяні? Чи важливою є продуктивність додатку? Чи є інші системи, з якими потрібно інтегруватися? Чи є вимоги до безпеки та системи авторизації?

Розглянувши запитання, можна перейти до переліку підходів, яких насправді лише три: локалізація, вбудована у код (традиційна, ручна); локалізація, винесена в окрему CMS (напівавтоматична); проксі-локалізація (автоматична).

Залежно від технологічного стеку самого вебзастосунку можуть бути різні рішення, що підходять саме для нього. Можна вважати їх базовими, з яких усе починається і які підтримані розробниками. Підійдуть ці варіанти чи ні у кожному окремому випадку використання, потрібно дивитись на конкретних прикладах. Наприклад, природним вибором для React-коду є використання бібліотек `i18n`, таких як `react-i18next`, `react-intl` або `lingui` [16]. Існують також спеціалізовані бібліотеки для конкретних фреймворків, наприклад `gatsby-plugin-i18n` для Gatsby [5] та `next-intl` для Next.js [10]. На високому рівні архітектура більшості цих бібліотек схожа. Перевагами традиційної локалізації є:

1. Налаштування — розробники мають повний контроль і відповідають за реалізацію локалізації у повному обсязі. Окрім того, це роблять лише при налаштуванні самого проєкту.
2. Оптимізація — переклад отримується зі статичного файлу, який розміщений безпосередньо в проєкті або завантажується зі стороннього ресурсу. Це означає, що після того, як дані файли доступні, переклад можливий миттєво.

Недоліками ж можна вважати такі пункти:

1. Підтримка розробників — потрібна постійна участь розробників у налаштуванні, перевірці та корекції контенту. Для будь-якої зміни потрібна взаємодія безпосередньо з кодом проєкту.
2. Необхідність тримати файли перекладу поряд — компроміс між розміром файлів коду та часом на завантаження певного мовного файлу.

Ми розглянули підхід і його імплементацію зі сторони розробників проєкту, як він влаштований і як його підтримують. Однак сама локалізація — переклад текстів — відбувається в іншому місці. Для невеликих проєктів із малою кількістю тексту це може виконуватись вручну перекладачами і потім знову збиратись у файли, які повернуться у код. Більші ж проєкти вже потребують TMS (Translation Management System) для управління локалізаціями. Це сервіси, які зазвичай відірвані від розробки і більше налаштовані на роботу і управління командою перекладачів, адже вони також мають свої процеси, які потрібно контролювати.

Однією з таких систем є Localazy [4]. Сервіс має багату функціональність і змістовну документацію, але тут я пропоную розглянути ту його частину, яка відповідає саме за закриття прогалу в локалізації, що у нас утворилась, — переклад контенту. Localazy спрощує та організовує процес перекладу, допомагає ефективно розподілити і проконтролювати роботу, щоб забезпечити якісний результат. Подібних сервісів багато, всі вони пропонують як додаткові просунуті функції на зразок рішень на основі штучного інтелекту та різного роду автоматизації.

Напіваавтоматична локалізація, винесена в окрему CMS

Розглянемо підхід до локалізації, який вважають напіваавтоматичним. У контексті цієї роботи він є таким, що відбувається за допомогою стороннього сервісу — CMS (Content Management System). Такий підхід підтримує написання контенту кількома мовами, які має підтримувати вебзастосунок, і спрощує редагування контенту для нетехнічних спеціалістів і перекладачів, які можуть керувати контентом через цей окремий сервіс.

CMS-системи мають бути глибоко інтегровані у кодову базу застосунку. Особливості першої конфігурації залежать від самого сервісу, але потім нетехнічні спеціалісти зможуть створювати новий контент або редагувати наявний безпосередньо через CMS. На платформі можна керувати локалізаціями або мовами, які доступні у вебзастосунку, можна додати новий переклад для наявної структури. Розробники отримують локалізований контент через вказаний ідентифікатор локалізації.

Традиційний підхід і CMS мають багато спільного, зокрема обидва зберігають перекладений контент як структуровані дані, які застосунок може отримати за ідентифікатором. Однак вони відрізняються підходом до програмування та необхідною архітектурою кодової бази, адже мають різні API (Application programming interface) для взаємодії:

- для реалізації традиційного підходу потрібна реалізація і робота з контентом на рівні рядків;
- системи CMS дають змогу визначати типи самого контенту, групувати текстові дані у так звані компоненти — публікація у блозі, картка товару тощо. Тобто коли застосунок отримує дані з CMS — це результат у вигляді об'єкта, що являє собою перекладений контент компоненту, який безпосередньо розміщений на сторінці. Такий підхід потребує завчасно продуманої архітектури застосунку, і його не можна просто замінити іншим підходом.

Перевагою напіваавтоматичної локалізації на базі CMS є те, що розробники мають повний контроль над реалізацією перекладу, відображенням локалізованого контенту, і його легко редагувати.

Головний недолік — складність початкового налаштування та відповідний дизайн архітектури, який створений для роботи з конкретною CMS та запрограмованими компонентами. Типовим представником CMS є сервіс Contentful [2], що являє собою повноцінну CMS із безліччю функцій для управління даними, ролями, перекладами тощо.

Автоматична локалізація

Цей підхід є найцікавішим і майже нічого не потребує зі сторони розробників застосунку. Але водночас він більш ризиковий і не дає повної гарантії щодо повноцінної локалізації, адже для ідеального виконання цього підходу мають бути виконані певні умови. Для досягнення автоматичної локалізації багато сервісів зробили крок уперед і автоматизували збирання тексту з активного сайту, який доступний для користувачів, і навіть динамічну заміну тексту.

Ці автоматизовані сервіси зазвичай мають свої бібліотеки, які мають бути включені напряму в кодову базу застосунку. Під час завантаження ресурсу скрипт знайде увесь видимий текст і завантажить їх у панель керування. Сервіс покаже список і запропонує машинний переклад або дозволить додати його вручну. Тільки-но контент буде локалізовано, при завантаженні вебзастосунку вбудований скрипт знаходитиме відповідний до локалізації користувача переклад і замінюватиме DOM (Document object model) елементи, щоб відобразити текст із відповідним перекладом.

На перший погляд, цей підхід є такою собі срібною кулею, яка вирішує всі питання перекладу для вебсайтів, майже все автоматизовано і з мінімумом налаштувань. Потрібно тільки перекладати автоматично знайдений текст, але на практиці все може бути не так райдужно через обмежену кастомізацію, неможливість зберігати й повторно використовувати частки контенту, передати контекст тощо.

Отже, можна назвати такі переваги підходу: легке налаштування (підключити бібліотеку або скрипт напряму в застосунок, зазвичай на рівні базового HTML), легке використання (текст автоматично знайдений, вивантажений на платформу та показаний користувачу після перекладу; оскільки все робиться через окремий сервіс, то в основному застосунку жодні зміни не потрібні і можна дотримуватися свого поточного процесу без змін).

Недоліки також очевидні й впливають із переваг: оптимізація застосунку (оскільки текст замінюється напряму у DOM після завантаження перекладу, це відбувається з певною затримкою, і користувач це може помітити), обмежена кастомізація (оскільки сервіс автоматично зчитує написані текстові рядки у застосунку, він має чітко «розуміти» структуру кодової бази, щоб їх знайти; така автоматизація значно зменшує можливості індивідуальних проектних налаштувань, адже має бути універсальною для всього застосунку).

Як ілюстрацію автоматичного варіанта розглянемо сервіс Gtranslate [3]. Цей сервіс застосовують у випадках, коли потрібна не обов'язково дуже точна і якісна локалізація, але її наявність може відігравати важливу роль. Для роботи із сервісом майже нічого не потрібно зі сторони розробника, зокрема жодних змін до кодової бази. Є певні налаштування DNS (Domain Name System), які потрібно зробити, щоб пропустити сайт через сервіс і виконати переклад. Основним є налаштування CNAME (Canonical Name), яке відповідатиме за перенаправлення вашого сайту на сервіс локалізації, а також перенаправлення локалізованих версій.

Сервіс пропонує безкоштовну версію із машинним перекладом, а також декілька варіантів із розміщенням локалізованих версій сайту, із використанням субдоменів або субдиректорій. Переклад, який було створено за допомогою параметра “language_edit=1”, доступний до редагування прямо на сторінці. Також сервіс надає мінімальні можливості для встановлення селектора мов і має плагіни для налаштувань для певних популярних систем, наприклад Wordpress.

Підсумовуючи інформацію щодо різних підходів із локалізації вебзастосунків, можна сказати, що немає єдиного рішення для всіх випадків. Часто рішення залежить від того, наскільки тісно з цим працювати може сам розробник, чи можлива його підтримка. Автоматизована локалізація може бути гарним варіантом, коли потрібна найменша підтримка зі сторони розробників. Зазначимо, що перехід між будь-якими підходами теж не є легким завданням. Для кожного з них кодова база має мати певну структуру та по-різному викликати локалізовані елементи, а для автоматизованої не всі технології підійдуть.

Важливо виділити окремо ситуацію, коли можлива інтеграція у вже наявний проєкт без локалізації взагалі. В такому разі важливо зорієнтуватись, якого типу застосунок, його архітектура та яка кількість тексту для локалізації.

Вебсервіс EchoLocal для оптимізації локалізації

Отже, врахувавши згадані в попередньому розділі підходи до локалізації та особистий досвід роботи, у нас виникла ідея розробити застосунок, який покращив би, оптимізував цей процес загалом.

У чому головна складність перекладу? На нашу думку, у передаванні інформації у ланцюжку замовник — дизайнер — розробник — перекладач. Те, що здається на перший погляд логічним і очевидним, може значно змінити своє значення у процесі, якщо немає спілкування між учасниками. Також складності додає те, що цей процес асинхронний і потребує певного часу для виконання роботи кожним учасником. Потрібна певна база знань, де буде збережено контекст, щоб максимально зручно й ефективно передати значення кожного тексту перекладачу.

На ринку існують, з одного боку, інтегровані рішення для технологічного аспекту, бібліотеки, які реалізують сам функціонал перекладу у вебзастосунку, а з іншого — гарні вебрішення для перекладачів, які спрощують роботу саме для них. Наша ж ідея полягала в тому, щоб почати реалізовувати щось середнє між цим, даючи можливість ефективно співпрацювати розробникам і людям, які відповідають за контент, без додаткових, непотрібних дій.

Один із підходів до перекладу контенту — групування ключів та їхніх значень у файл. Цей файл лежить на сервері разом із проектом і завантажувється разом із ним у браузер користувача. Це найшвидший і найлегший спосіб для проекту отримати нові значення, але процес перекладу ключів, знання контексту цих ключів залишається абсолютно незручним і неоптимізованим. Наша ідея проекту вирішує цю проблему. Тепер цей файл ключів має зберігатись не у проекті всередині, а на окремому ресурсі, який бере на себе всю роботу з організації перекладу та наповнення вебсторінки контентом. Самому проекту залишається лише підвантажити перелік необхідних ключів із перекладами, це можуть бути всі ключі або лише частина. Також за такого підходу може бути доступна певна метаінформація про переклад, як-от дата створення, дата останнього оновлення тощо.

При цьому ми не зменшуємо зручність для самих перекладачів, адже саме для них на ринку дуже багато платформ, так званих менеджерів перекладів. Нова платформа для менеджменту перекладів, до якої допускають самих розробників проєктів, які потребують перекладу, надає певні переваги та поліпшення. Наприклад, тепер ключі можуть створювати люди, які фактично їх додають у проєкт, є можливість завантажити картинку для контексту та іншу додаткову інформацію, яка допоможе перекладачеві краще зорієнтуватись.

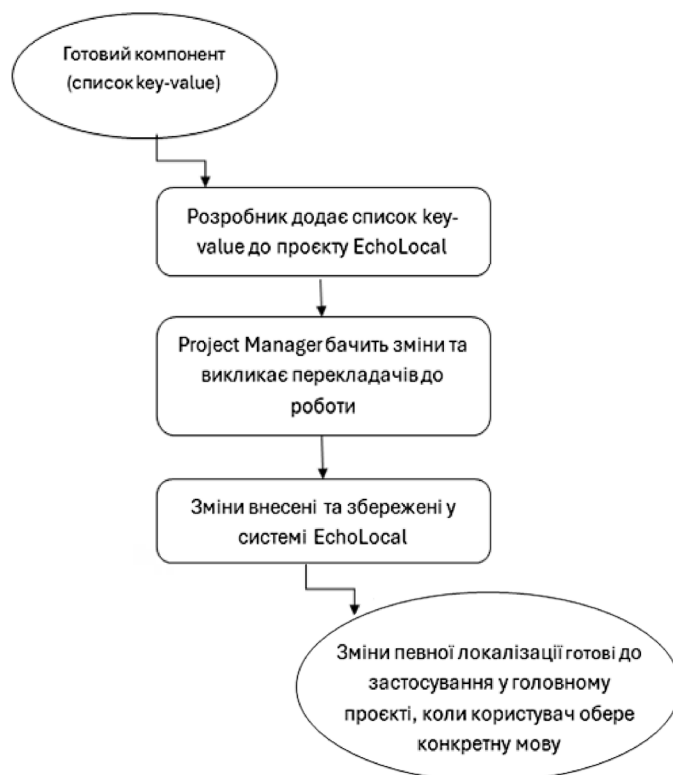


Рис. 1. Сценарій роботи сервісу

Сценарій роботи сервісу такий (рис. 1). Уявімо, що є дизайн і розробник уже спроектував і вніс необхідні зміни до коду. Маємо робочий функціонал на вебсторінці з текстом англійською мовою. Текст не є вбудованим, а підв'язаний до ключів, які своєю чергою перетворюються на реальний текст необхідною мовою. Саме ці ключі і текст необхідною мовою будуть завантажуватись у проєкт. У самому проєкті є налаштований функціонал для роботи із сервісом перекладів, де отримують ключі, а також один файл перекладів основною мовою (так званий fallback, переклади англійською мовою згідно з цим прикладом). Розробник самостійно вносить цей ключ і переклад у запрограмовану систему EchoLocal і прикріплює необхідні допоміжні елементи, щоб передати контекст. На цьому робота замовника закінчується. Далі штат перекладачів вносить зміни, додаючи переклади на інші мови, а застосунок уже сам підтягне ці зміни залежно від вибраної користувачем мови.

У чому полягає основний вигравш від використання платформи EchoLocal? На нашу думку, у вирішенні проблеми ефективності шляхом зняття зайвої, непрофільної роботи з розробника. Тепер кожен займається своєю роботою і не залежить від часу й роботи іншого. Спочатку зробив свою роботу (запрограмував), підготував роботу для іншого (завантажив ключі) і пішов, далі все автоматично запрацює, коли буде готово (після оновлення контенту перекладачами). Також є неочевидна перевага у контролі, адже тепер не потрібно шукати текст на вебсайті або, перевіряючи за якимось критерієм, мати доступ до всіх частин сайту. Можна просто переглянути список усіх ключів і їх перекладів на зручному вересурсі та зробити необхідні зміни без будь-якої зовнішньої допомоги.

Основними можливостями сервісу є: зберігання даних проєкту, для якого роблять переклад, і списку усіх доступних перекладів, зберігання листів перекладів на різні мови для зручної роботи перекладачів і менеджерів проєкту, можливість оновлення контенту за межами сайту і автоматичного оновлення на сайті, незалежність всіх учасників процесу у часі, ефективність роботи всіх учасників, кожен з яких займається своєю справою.

Далі пропонуємо розглянути загальну архітектуру проєкту, схему бази даних і підхід до розгортання проєкту.

Загальна архітектура EchoLocal складається із окремих сервісів для клієнта та сервера, які співпрацюють один з одним за допомогою обміну HTTP запитами (рис. 2). Клієнт і сервер являють собою два окремі застосунки, розміщені в одному репозиторії (монорепозиторії), незалежні частини, які співпрацюють. Оскільки однією із цілей застосування є підтримкування роботи з мікросервісами, то такий розподіл є необхідним, адже API має бути доступним для зовнішніх джерел: запити від зовнішніх проєктів мають йти напряму до серверної частини застосунку і завантажувати переклад або його частину. Тож користувачі не прив'язані до фронтенд-частини і мають можливість користуватись API напряму.

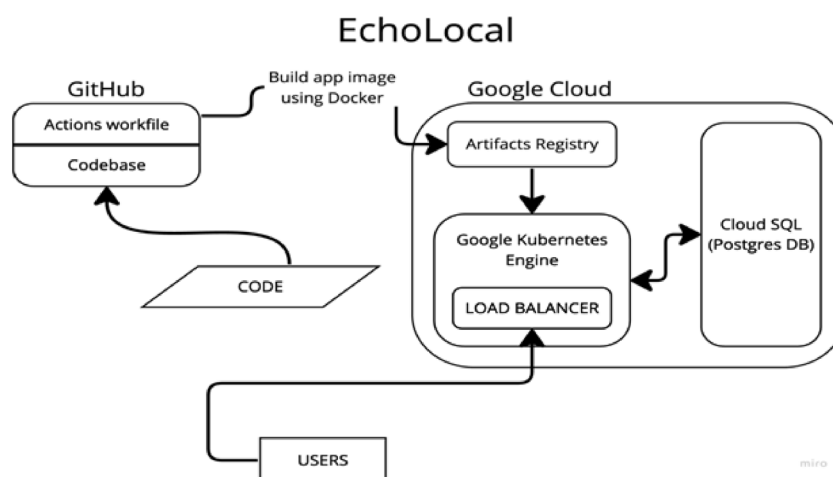


Рис. 2. Загальна архітектура проєкту EchoLocal

Проект EchoLocal запущений у хмарі на потужностях Google Cloud. Складовими частинами проєкту є: Google Kubernetes Engine [8], Artifacts Registry [7], Cloud SQL [9], Docker [12], GitHub [13], GitHub Actions [6].

Частина хмарного рішення Google Kubernetes Engine є сервісом для роботи із Kubernetes [15] від Google Cloud [14]. Головний тримач застосунку — автоматизований Kubernetes кластер, який має

декілька реплікацій і доступ ззовні, може гнучко налаштовуватись під потреби застосунку залежно від навантаження. Сервіс Artifacts Registry використовують для зберігання та управління образами застосунків для подальшого використання у Kubernetes. Cloud SQL є частиною хмарного рішення для баз даних від Google Cloud. Для нашого застосунку ми вибрали базу даних Postgres (детальну схему буде показано далі). Рішення GitHub використано для збереження кодової бази та подальшого розгортання проєкту. Репозиторій містить весь код проєкту і дає змогу зручно його переглядати та оновлювати. Для проєктів із відкритим кодом він дає можливість переглядати код іншим спеціалістам і пропонувати зміни та покращення.

Допоміжні інструменти Docker, GitHub Actions використано для зручнішого розгортання застосунку на платформі Google Cloud. Docker — інструмент контейнеризації, який дає змогу «упакувати» застосунок у контейнер із заданими налаштуваннями оточення так, що будь-який інший розробник зможе відтворити це оточення, якщо у нього встановлений Docker. Також було налаштовано CI/CD процес, який автоматично оновлює застосунок до останньої версії у Google Cloud за допомогою GitHub Actions. Загалом, рішення розмістити проєкт у хмарі було пов'язане з гнучкістю і легкістю налаштування необхідної потужності. Порівняно просто можна налаштувати оновлення застосунку за допомогою процесів Continuous Integration і Continuous Delivery. Також потрібно згадати про те, що застосунок є доступним, зручно показувати демо-версію та налаштовувати необхідні доступи. Google надає певний кредит для навчальних цілей, тож дуже вчасно і ефективно ми використали його для тестування цього рішення.

Kubernetes — платформа з відкритим кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Розроблена та активно підтримується компанією Google. Цей інструмент має забезпечити гнучкість у розгортанні та можливість масштабування за потреби. Система автоматично зможе підлаштовуватись під навантаження, а також забезпечує необхідні оновлення і резервні копії станів застосунку. Для повноцінної роботи із Kubernetes потрібен спеціальний інструмент, який має підготувати образ застосунку для роботи. Оновлення цього образу у сховищі дасть команду кластеру на перезбір.

Postgres як реляційну базу даних було вибрано з огляду на її надійну роботу, підтримку шардінгу та простоту реплікацій у разі необхідності. У майбутньому можливе використання особливостей цієї бази даних, зокрема підтримки enum, для розподілу ролей або підтримки певного переліку мов, доступних до перекладу. Реляційна база даних має нормалізовану структуру даних, які будуть зберігатись у ній. Усі записи й зв'язки між даними досить очевидні та прямолінійні. Варіант NoSQL бази даних також розглядали, адже він мав би певну перевагу при зчитуванні: можна було простіше створити запит на отримання даних для проєкту у заданій мові, але процес запису і налаштування зв'язків потребував би більше дій і контролю.

На нинішньому етапі імплементації застосунку база даних має такі таблиці як структурні елементи:

- users містить дані про зареєстрованих користувачів системи — first_name, last_name, email, password_hash, created_at;
- organizations містить дані про створені користувачами організації — name, owner_id, created_at;
- projects містить дані про створені користувачами проєкти, які належать конкретній організації, — name, description, org_id, created_at;
- translations містить дані про листи перекладів, створені та доступні конкретному проєкту, — project_id, language, updated_at, created_at;
- translationKeys містить дані про перелік ключів і їх перекладів для конкретного листа перекладу — sheet_id, key, value, updated_at, created_at.

У процесі розвитку проєкту передбачено певні зміни у структурі та зв'язках між таблицями. Схему бази даних подано на рис. 3.

Щоб окреслити функціональні можливості вебзастосунку EchoLocal, розглянемо основні сторінки інтерфейсу. На момент написання доступною була лише одна роль користувача, який реєструється і має доступ до всього функціоналу. У подальшому заплановано введення більш гранулярного доступу.

Портал EchoLocal умовно розділений на дві частини залежно від статусу користувача — авторизований він чи ні. Головна сторінка доступна для всіх і має містити загальну інформацію про сервіс. Для неавторизованого користувача доступні сторінки входу та реєстрації (рис. 4).

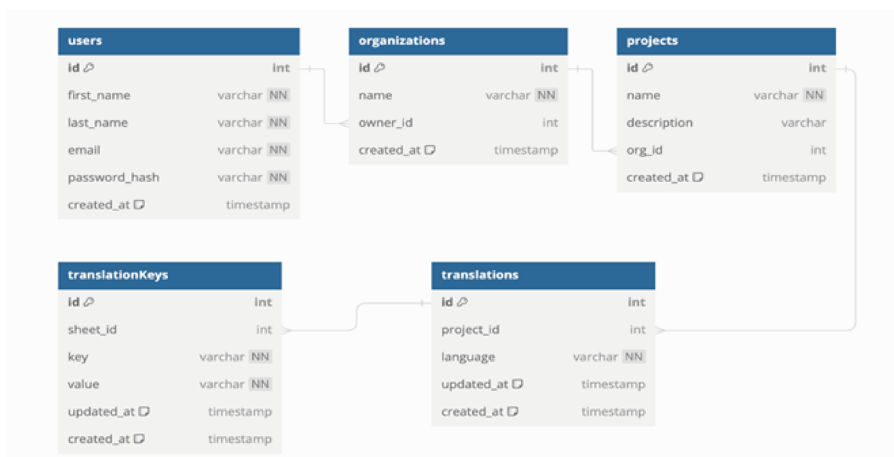


Рис. 3. Схема бази даних

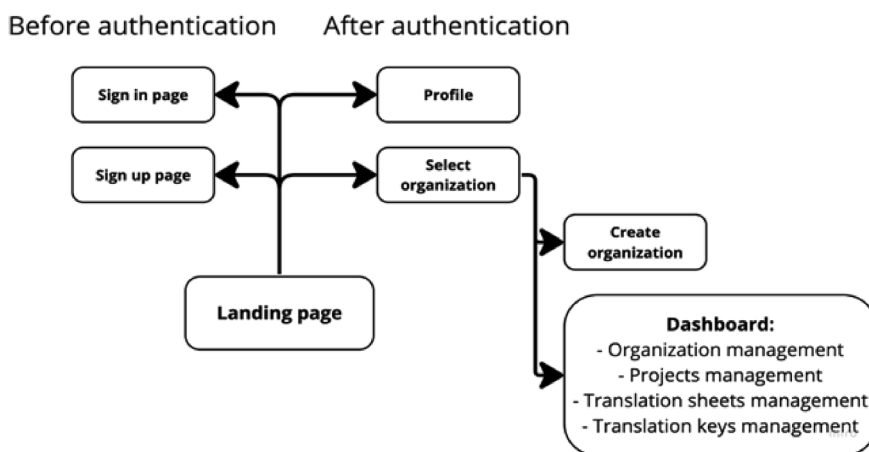


Рис. 4. Головна сторінка

Якщо користувач авторизований, то він має повний доступ до системи — до сторінки профілю, вибору та створення організації та панелі керування — сторінка, яка містить все необхідне для взаємодії з сервісом. Далі розглянемо кожну сторінку та її функціональність окремо для більшого розуміння загальної функціональності та можливостей системи.

На головній сторінці відображається простий інтерфейс із логотипом EchoLocal та меню навігації, яке дає змогу неавторизованому користувачеві зареєструватись або увійти до системи, а авторизованому — переходити між сторінками системи. Варто зазначити, що візуальна частина сторінки не готова і показує лише те, що сторінка у процесі розроблення, а також кнопку перевірки статусу сервера.

Після входу в систему користувач потрапляє на сторінку вибору організації. Тут можна вибрати наявну організацію або створити нову.

Сторінка налаштувань організації дає можливість користувачу керувати організацією, зокрема видалити її.

На панелі керування користувач може налаштовувати проекти і переклади. Це головна сторінка сервісу, де відбувається основна взаємодія і доступний такий функціонал: вибір проекту (можливість вибрати проект зі спадного списку), список перекладів (перегляд наявних перекладів для вибраного проекту), вибір листа перекладу (вибір конкретного листа перекладу для детальної роботи із ним), додавання нового перекладу (користувач може додати новий ключ перекладу та його значення), зміна та видалення наявного перекладу.

Користувач може створювати проекти з власним набором мов і деталями та редагувати їх у подальшому.

Огляд функціональних можливостей EchoLocal дає уявлення про те, як саме користувач взаємодіє із системою і які можливості вона надає для управління процесом локалізації.

Переваги використання EchoLocal

EchoLocal підтримує різні технології та платформи, що дає можливість легко інтегрувати сервіс у будь-який проєкт. Завдяки можливостям масштабування він зручний як для малих стартапів, так і для великих компаній із багатьма проєктами та мовами. Всі ключі локалізації та переклади зберігаються в одному місці, що забезпечує легкий доступ та управління. Централізований контроль дає змогу уникнути дублювання зусиль і помилок, забезпечуючи єдність контенту. Завдяки можливостям синхронізації в реальному часі всі зміни в локалізації миттєво відображаються в проєкті. API сервісу дозволяє автоматизувати процес завантаження та оновлення перекладених текстів, знижуючи потребу в ручній праці. Власник проєкту може легко додавати та керувати ролями членів команди, забезпечуючи ефективну співпрацю між розробниками, перекладачами та менеджерами. Інтуїтивно зрозумілий інтерфейс дає змогу кожному члену команди швидко зрозуміти свої задачі та розпочати роботу. Наявність коментарів і контексту для перекладачів значно підвищує якість перекладів, зменшуючи кількість помилок та непорозумінь. Можливість затвердження перекладів менеджерами забезпечує додатковий контроль якості.

Сервіс EchoLocal може стати хорошим інструментом для управління локалізацією, який забезпечує гнучкість, ефективність та високу якість перекладів. Він дає змогу знизити витрати й час на локалізацію, підвищуючи при цьому загальну ефективність роботи команди. Використовуючи його, компанії можуть легко адаптувати свої продукти під нові ринки, розширюючи свою аудиторію та збільшуючи доходи.

Майбутні напрями розвитку та удосконалення EchoLocal

Звичайно, у рамках цього дослідження не було реалізовано всі ідеї. Основний наголос було зроблено на дослідження проблеми та перевірки методів можливого покращення і підвищення ефективності крос-командної співпраці у роботі над локалізацією вебзастосунків.

Ми визначили такі напрями покращення сервісу:

1. Можливість спільної роботи декількох користувачів на перекладом одного проєкту.
2. Введення ролей користувачів і закріплення за ними певних можливостей, які доступні на порталі, для покращення контролю за доступом. Це стане більш актуальним зі збільшенням аудиторії та розмірів проєктів.
3. Суттєвим покращенням сервісу було б додавання перекладу у реальному часі. Це можливо зробити з ускладненням підключення сервісу та проєкту за допомогою WebSockets або long-polling підходів. Можна запропонувати створення спеціально налаштованих бібліотек для популярних технологій, які вже мали такий функціонал.
4. Можливість попереднього перегляду сторінки з певною локалізацією. Це допомогло б виявити проблеми зі структурою сторінки, довжиною тексту та інші можливі невідповідності.

Висновки

У цій статті розглянуто різні підходи до локалізації сучасних вебзастосунків, їхні переваги та недоліки, а також концепцію нового сервісу для локалізації — EchoLocal. Сервіс покликаний покращити процес і зайняти свою нішу серед подібних рішень на ринку. Детально представлено архітектуру прототипу та функціонал сервісу.

Сервіс EchoLocal було розроблено з урахуванням основних недоліків наявних підходів, він надає такі переваги:

1. Підтримує різні технології та платформи, що дає змогу легко інтегрувати сервіс у будь-який проєкт. Прототип не залежить від технологій і платформ, адже надає API для роботи, яка своєю чергою може бути побудована різним чином — налаштована вручну або із використанням сторонніх бібліотек. Завдяки можливостям масштабування, EchoLocal підходить як для малих стартапів, так і для великих компаній із багатьма проєктами та мовами.
2. Усі ключі локалізації та переклади зберігаються в одному місці, що забезпечує легкий доступ та управління. Це допомагає уникнути повторення помилок і неефективної співпраці, забезпечуючи цілісність контенту.

3. Можна легко додавати та керувати ролями членів команди, забезпечуючи ефективну співпрацю між розробниками, перекладачами та іншими менеджерами. Інтуїтивно зрозумілий інтерфейс дає змогу кожному члену команди швидко зрозуміти свої задачі та розпочати роботу.
4. Наявність нотаток і контексту для перекладачів значно підвищує якість перекладів, зменшуючи кількість помилок та непорозумінь. Можливість затвердження перекладів менеджерами забезпечує додатковий контроль якості.
5. Завдяки можливостям синхронізації в реальному часі, всі зміни в локалізації миттєво відображаються в проєкті. API сервісу дає можливість автоматизувати процес завантаження та оновлення перекладених текстів.

EchoLocal, як сучасний інструмент для управління локалізацією, забезпечує гнучкість, масштабованість і високу якість перекладів, що робить його ідеальним рішенням для компаній, які прагнуть ефективно локалізувати свій контент і вийти на нові ринки. Однак подальші дослідження та розвиток технологій залишаються важливими для подолання майбутніх викликів.

Список літератури

1. Глибовець А. М. Агентно-базовані програмні системи пошуку та аналізу інформації : монографія / А. М. Глибовець. — Київ : Видавничий дім «Києво-Могилянська академія», 2019. — 281 с.
2. Документація сервісу Contentful [Електронний ресурс]. — Режим доступу: <https://www.contentful.com/help/contentful-101/>.
3. Документація сервісу Gtranslate [Електронний ресурс]. — Режим доступу: <https://gtranslate.io/docs/58-gtranslate-tdn-documentation>.
4. Документація сервісу Localazy [Електронний ресурс]. — Режим доступу: <https://localazy.com/docs/general/getting-started-with-localazy>.
5. Документація Gatsby [Електронний ресурс]. — Режим доступу: <https://www.gatsbyjs.com/docs>.
6. Документація GitHub Actions [Електронний ресурс]. — Режим доступу: <https://docs.github.com/en/actions>.
7. Документація Google Cloud Artifact Registry [Електронний ресурс]. — Режим доступу: <https://cloud.google.com/artifact-registry/docs>.
8. Документація Google Cloud Kubernetes Engine [Електронний ресурс]. — Режим доступу: <https://cloud.google.com/kubernetes-engine/docs>.
9. Документація Google Cloud SQL [Електронний ресурс]. — Режим доступу: <https://cloud.google.com/sql/docs>.
10. Документація Next.js [Електронний ресурс]. — Режим доступу: <https://nextjs.org/docs>.
11. Інтернаціоналізація та локалізація [Електронний ресурс]. — Режим доступу: <https://qalight.ua/baza-znaniy/internatsionalizatsiya-ta-lokalizatsiya/>.
12. Офіційний сайт Docker [Електронний ресурс]. — Режим доступу: <https://www.docker.com/>.
13. Офіційний сайт GitHub [Електронний ресурс]. — Режим доступу: <https://github.com/>.
14. Офіційний сайт Google Cloud Platform [Електронний ресурс]. — Режим доступу: <https://cloud.google.com/>.
15. Офіційний сайт Kubernetes [Електронний ресурс]. — Режим доступу: <https://kubernetes.io/uk/>.
16. Cheng R. How to choose a localization approach for your React application [Electronic resource] / Raymond Cheng. — 2022. — Mode of access: <https://www.plasmic.app/blog/react-localization>.
17. Krukowski I. Internationalization vs. localization (i18n vs l10n): The differences [Electronic resource] / Ilya Krukowski. — 2022. — Mode of access: <https://lokalise.com/blog/internationalization-vs-localization/>.

References

- Cheng, R. (2022). *How to choose a localization approach for your React application*. <https://www.plasmic.app/blog/react-localization/>.
- Contentful service documentation. (n.d.). [tps://www.contentful.com/help/contentful-101/](https://www.contentful.com/help/contentful-101/).
- Gatsby documentation. (n.d.). <https://www.gatsbyjs.com/docs>.
- GitHub Actions documentation. (n.d.). <https://docs.github.com/en/actions>.
- Google Cloud Artifact Registry documentation. (n.d.). [tps://cloud.google.com/artifact-registry/docs](https://cloud.google.com/artifact-registry/docs).
- Google Cloud Kubernetes Engine documentation. (n.d.). <https://cloud.google.com/kubernetes-engine/docs>.
- Google Cloud SQL documentation. (n.d.). <https://cloud.google.com/sql/docs>.
- Gtranslate service documentation. (n.d.). <https://gtranslate.io/docs/58-gtranslate-tdn-documentation/>.
- Hlybovets, A. M. (2019). *Ahentno-bazovani prohramni systemy poshuku ta analizu informatsii* (p. 281). Vydavnychiy dim "Kyievo-Mohylianska akademiia" [in Ukrainian].
- Internatsionalizatsiia ta lokalizatsiia. (n.d.). <https://qalight.ua/baza-znaniy/internatsionalizatsiya-ta-lokalizatsiya/>.
- Krukowski, I. (2023). *Internationalization vs. localization (i18n vs l10n): What's the difference?* <https://lokalise.com/blog/internationalization-vs-localization/>.
- Localazy service documentation. (n.d.). <https://localazy.com/docs/general/getting-started-with-localazy/>.
- Next.js documentation. (n.d.). <https://nextjs.org/docs>.
- Official Docker website. (n.d.). <https://www.docker.com/>.
- Official GitHub website. (n.d.). <https://github.com/>.
- Official Google Cloud Platform website. (n.d.). <https://cloud.google.com/>.
- Official Kubernetes website. (n.d.). <https://kubernetes.io/>.

V. Vereta, V. Tkachenko

AUTOMATED LOCALIZATION OF APPLICATIONS IN MICROSERVICE ARCHITECTURE

The article provides an in-depth analysis of various tools and services available for the localization and translation of web applications, as well as methods and approaches for their integration and scaling. It examines existing solutions for managing web service localization, highlighting their strengths and weaknesses. The article also describes an architecture that allows for the quick adaptation of services to meet the diverse needs of different users and projects. This architecture facilitates efficient integration with various platforms, while ensuring ease of scaling.

Based on this architecture, the EchoLocal web service was developed. EchoLocal enhances interaction among participants in the web application localization process, allowing for optimal integration into distributed microservice architectures common in modern applications. It supports the easy deployment within a closed ecosystem, making it suitable for organizations with specific security or operational requirements. EchoLocal enables users to integrate and manage localizations of their products on one centralized platform in real-time.

EchoLocal helps reduce localization costs by simplifying the adaptation of content for different language markets. It increases the efficiency of communication between all participants in the localization process, fostering stronger collaboration. By centralizing localization management, EchoLocal streamlines workflows for developers, translators, and project managers alike.

In conclusion, the article underscores the importance of efficient localization in today's global market and presents EchoLocal as a robust solution to many challenges in web application localization. Implementing the described architecture allows organizations to significantly improve their localization processes, leading to faster market entry and a more enhanced user experience across different regions.

Keywords: localisation, internationalisation, globalisation, microservice architecture, Kubernetes, Google Cloud Platform, GCP, CI/CD.

Матеріал надійшов 18.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Єщенко М. С.

МОДЕЛЮВАННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙНУ

У статті розглянуто ключові характеристики технології блокчейну і підходи до комплексного моделювання блокчейну з відтворенням усіх його атрибутів. Визначено найбільш перспективний спосіб моделювання для подальшого дослідження.

Ключові слова: блокчейн, мультиагентні системи, моделювання.

Вступ

Блокчейн є дуже привабливою технологією, оскільки він надає публічний, відкритий тільки для додавання, незмінний і впорядкований журнал транзакцій. Системи блокчейну за своєю суттю є міждисциплінарними, оскільки вони поєднують різні сфери, такі як криптографія, мультиагентні системи, розподілені системи, соціальні системи, економіка та фінанси. Крім того, вони мають дуже активну та динамічну екосистему, де постійно розробляються нові блокчейн-платформи та алгоритми завдяки інтересу громадськості та галузей до цієї технології. З огляду на складність і багатогранність блокчейну, його представлення — моделювання — іншими, більш відомими засобами має посприяти поліпшенню розуміння можливостей та особливостей цієї технології.

Ця стаття покликана дослідити основні способи моделювання технології блокчейну і визначити найбільш перспективний для подальшого дослідження можливостей моделювання всіх аспектів технології блокчейну.

Попередні відомості про блокчейн-системи

Структура даних блокчейну

Структуру даних блокчейну можна моделювати як динамічне дерево, відкрите лише для додавання, в якому кожен блок містить криптографічне посилання на попередній блок. Кореневий блок також відомий як блок генезису. Найдавший блок від блоку генезису називають головою блокчейну.

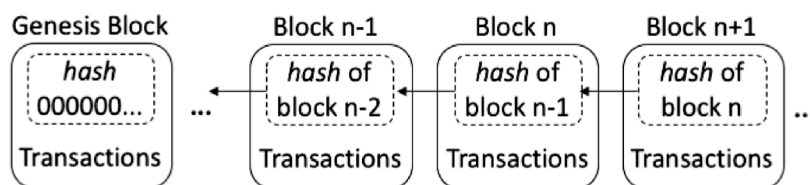


Рис. 1. Структура даних блокчейну [22]

Основи систем блокчейну

Із технічного погляду, усі учасники зберігають непідтвержені транзакції у власних пулах пам'яті, а підтвержені транзакції — у своїх локальних блокчейнах.

Є два основні типи учасників для всіх типів блокчейн-систем: користувачі та автори блоків. Користувачі створюють транзакції із певною винагородою і пропонують їх, поширюючи мережу блокчейну для підтвердження (тобто повного впорядкування та криптографічного пов'язання з блокчейном). Кожен учасник, отримуючи запропоновану транзакцію, підтверджує та поширює її для своїх сусідів. Після отримання певної кількості транзакцій автори блоків вибирають транзакції для підтвердження та впорядковують їх, створюючи спеціальний блок за допомогою механізму консен-

сусу блокчейну, наприклад, Proof-of-Work (PoW), Proof-of-Stake (PoS), Делегованого Proof-of-Stake (DPoS), Візантійської відмовостійкості (BFT). Залежно від механізму, який використовують, і технології блокчейну, ініціаторів (пропонентів) створення блоків називають «майнерами» [15], «валідаторами» [24], «пекарями» [7], «замовниками» [10], «членами комітетів» [2] тощо. Успішний пропонент блоку пропонує свій блок, поширюючи його в мережу для додавання до локальних блокчейнів. Кожен учасник, який отримує запропонований блок, перевіряє його на свій локальний блокчейн і поширює його для своїх сусідів. Після включення свого блоку всіма учасниками відповідний успішний ініціатор блоку отримує винагороду $R + F_i$, де R — статична винагорода за блок, а F_i — загальна сума винагород за транзакції, включені в блок i . Отже, користувач і учасники, що пропонують блок, разом підтримують спільну структуру даних, яку називають блокчейном.

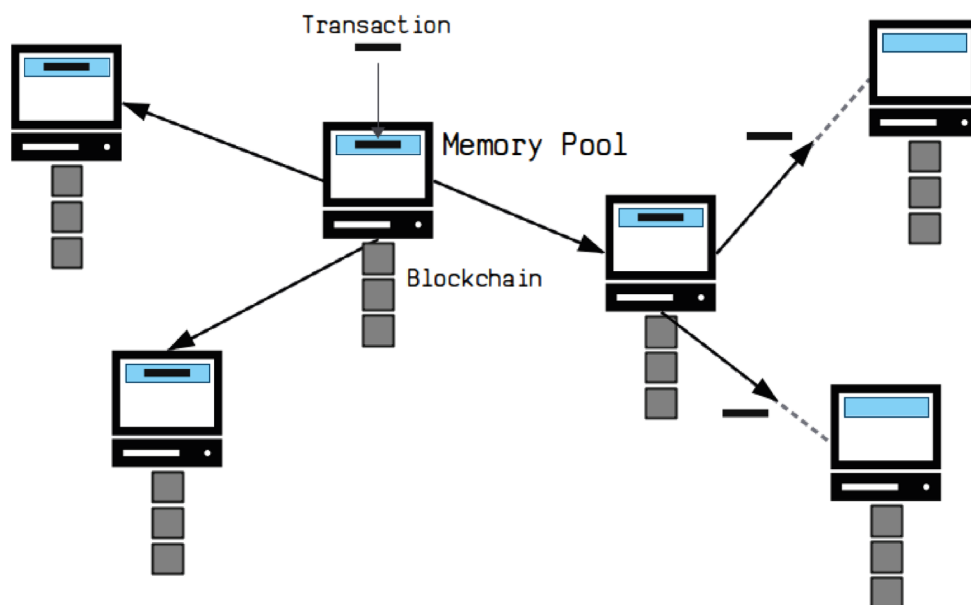


Рис. 2. Зберігання транзакцій

Щоб покращити свої можливості створення блоків, автори блоків можуть інвестувати в обладнання, капітал або інші агенти залежно від механізму консенсусу, який використовують. Наприклад, у блокчейні PoW вони можуть інвестувати в нове обладнання, оскільки що більша обчислювальна потужність у них є, то легше їм створювати блоки. У блокчейні PoS вони можуть інвестувати в ставки, оскільки що більше у них заблокованих ставок, то більше шансів увійти до комітету. На додаток до PoS, у блокчейнах DPoS вони також можуть інвестувати в інших пропонентів блоків, делегуючи свої частки.

Хоча блокчейни спочатку забезпечували лише операції, пов'язані з криптовалютою, у 2014 р. було представлено підтримку повних за Тьюрингом смарт-контрактів (SC), які кодують довільну логіку оброблення даних [24]. Завдяки цьому прогресу блокчейни еволюціонували від простих криптовалютних платформ до розподілених транзакційних і логічних систем. У блокчейнах із підтримкою смарт-контрактів транзакції, що складаються із викликів смарт-контрактів, виконують усі учасники, які бажають постійно підтримувати стан блокчейну. Сьогодні такі блокчейн-системи можна розглядати як децентралізовані комп'ютери світового масштабу, найвідомішим прикладом яких є Ethereum [3].

Підходи до моделювання технології блокчейн

Модель — це абстракція деяких аспектів наявної чи планованої системи. Моделі слугують певним цілям, наприклад для представлення зрозумілого людині опису деяких аспектів системи або інформації у формі, яку можна ефективно проаналізувати. У контексті блокчейну існує невелика кількість досліджень, які спрямовані безпосередньо на моделювання [8; 9]. На основі наявних досліджень у літературі про блокчейн було визначено такі парадигми моделювання: процесно-орієнтова-

ну, об'єктно-орієнтовану, теоретико-графову та агентно-орієнтовану парадигми. Кожну буде описано з метою висвітлення інформації про те, як вони моделюють учасників (тобто користувачів і ініціаторів блоків), взаємодії, поведінку та структури даних (наприклад, блокчейни, транзакції) за допомогою надаваних ними абстракцій.

Процесно-орієнтована парадигма

У процесно-орієнтованій парадигмі (вона ж парадигма розподіленого програмування) система охоплює кілька розподілених процесів (абстракція процесу може представляти фізичний або віртуальний комп'ютер, процесор у комп'ютері або певний потік виконання в паралельній системі), пов'язаних комунікаційними каналами (наприклад, спільною пам'яттю або консенсусом), які співпрацюють над певним спільним завданням [4].

Процеси виконують призначений їм розподілений алгоритм через набір компонентів, що реалізують алгоритм у цих процесах. Канали дають змогу процесам транслювати повідомлення, запускаючи події. Спільна пам'ять дозволяє локально здійснювати прямий доступ до ресурсу з, можливо, багатьох процесів. Механізми консенсусу спрямовані на надання процесу для узгодження спільного рішення/результату в рамках децентралізованої системи. Ця парадигма спрямована на створення та/або аналіз систем, які є надійними (пропонуючи надійність і безпеку) і мають передбачувану поведінку навіть за негативного впливу навколишнього середовища (пропонуючи толерантність до несправностей).

Багато досліджень використовують цю парадигму для аналізу [5; 18] та/або побудови блокчейнів [15; 10; 24], залучаючи такі пов'язані абстракції: учасники (наприклад, користувачі та пропоненти блоків) моделюються як процеси, взаємодії — як канали, блокчейн — як спільна пам'ять, а прийняття рішення про загальний блок представлено за допомогою консенсусних абстракцій.

Наприклад, [5] показує, що деякі ініціатори блоків у блокчейні біткойн (також відомі як майнери) можуть відхилятися від своєї номінальної поведінки, таким чином отримуючи несправедливу перевагу, яка, як наслідок, знижує надійність системи. Пізніше [19] будує та вдосконалює обидва [5; 18], надаючи гібридну стратегію, що відхиляється як на рівні створення блоку, так і на мережевому рівні, ефективно досягаючи несправедливої переваги для майнерів, водночас змушуючи інших агентів працювати на майнерів, які відхиляються.

Теоретико-графова парадигма

Теоретико-графова парадигма зосереджена на топології, а отже, на сполучних властивостях алгебраїчних/математичних об'єктів. У контексті розподілених обчислень ці об'єкти є узагальненнями графів, а їхні властивості зв'язування пов'язані з обчислюваністю розподілених алгоритмів. Використання певних топологічних властивостей геометричних об'єктів більшої розмірності для підтвердження результатів розподілених алгоритмів називають топологічним підходом до розподілених обчислень. Методи з комбінаторної та алгебраїчної топології мають розширену характеристику синхронних та асинхронних розподілених алгоритмів, а також їх розв'язуваність [16]. У теорії графів вершина — це точка в графі. Вершини з'єднані між собою ребрами, які представляють відношення між двома вершинами.

У цій парадигмі учасники моделюються за допомогою абстракцій вершин, транзакції — за допомогою абстракцій країв, взаємодії — за допомогою абстракцій симплексу та/або граней, а шахрайство моделюється як абстракції просторово-часового шаблону.

Здатність структур теорії пучків розшифровувати глобальну інформацію з локальної інформації призвела до різноманітності застосувань, як-от тих, що були запропоновані для моделювання одночасних процесів у розподілених системах [12], семантики для об'єктно-орієнтованих мов програмування [23] та представлень інформаційних систем [17]. У [14] досліджуються топологічні моделі розподілених обчислень для технологій блокчейну, орієнтованих на масштабованість. Для цього автор моделює блок як пучок і розробляє теорію розподілених консенсусних протоколів.

Об'єктно-орієнтована парадигма

В об'єктно-орієнтованій парадигмі система складається з кількох об'єктів (також відомих як екземпляри класів), які взаємодіють із локальними або віддаленими викликами методів (також їх нази-

вають передаванням повідомлень) [11]. Класи мають певні обов'язки, і вони інкапсулюють дані (у формі абстракцій атрибутів) і код (у формі абстракцій методів), який маніпулює цими даними, і пов'язані один з одним за допомогою абстракцій асоціацій. Ця парадигма спрямована на створення та/або аналіз систем, які можна розвивати (тобто легко розширювати) і підтримувати (тобто легко виправляти).

Кілька досліджень явно використовують цю парадигму в літературі про блокчейн [1; 13]. У цих дослідженнях учасники моделюються як клас, взаємодії — як асоціації, блокчейн — як клас, а рішення щодо спільного блоку — як абстракції методу.

Наприклад, [1] пропонує загальну модель блокчейну PoW із метою створення розширюваного симулятора блокчейну. Наразі вони можуть моделювати та імітувати як Bitcoin, так і Ethereum 1.0 і перевіряти свої результати на історичні дані. Вони сумісні з блокчейном PoS, враховуючи кілька модифікацій. Як інший приклад, [13] пропонує нотацію на основі UML для підтримки дизайну смарт-контракту. Зокрема, вони додають стереотипи для діаграм класів і послідовностей UML, щоб краще виразити сутності та взаємодію між ними.

Агентно-орієнтована парадигма

У агентно-орієнтованій парадигмі система складається з кількох автономних агентів, які здатні сприймати своє оточення, міркувати незалежно (реактивно чи проактивно) і впливати на своє оточення [6; 25], утворюючи таким чином так звану мультиагентну систему (MAS). Ця парадигма спрямована саме на створення та/або аналіз систем, які мають деякий ступінь відкритості, автономності, розумності та складності. Моделювання через MAS розглядалося з двох основних поглядів: орієнтованого на агента та орієнтованого на організацію. У той час як перший зосереджується на внутрішній архітектурі агента, другий спирається на структуру системи та першочергово розглядає агентів як порожні оболонки, щоб зосередитися на організаційних аспектах MAS. Агент — це суб'єкт системи, який покладається на певний ступінь автономії для досягнення своєї мети або виконання своїх функцій пасивно чи активно. Координація — це засіб, за допомогою якого агенти обмінюються інформацією або ресурсами для досягнення мети.

Багато досліджень використовують цю парадигму як агентоцентризований засіб для аналізу блокчейнів [20; 21]. У цих дослідженнях учасники моделюються як агенти, взаємодії — як координаційні абстракції, блокчейн — як спільні знання, а прийняття рішення щодо спільного блоку моделюється за допомогою цілей, стратегій або ігрових абстракцій.

Наприклад, [21] використовує загальний підхід Reinforcement Learning (RL) для виявлення атак на різні системи блокчейнів за допомогою симуляції на основі RL і стратегічного пошуку, водночас досить обмежений процесами створення блоків у блокчейнах PoW. Мета подібна до [5], тобто виявлення векторів атак і слабких місць у блокчейні. Хоча цей пошук є експериментальним, на відміну від аналітичного, він здатний зрозуміти деякі багатоагентні взаємодії та обмеження, що виникають через багато різних агентів, кожен з яких має певну мету. Інший приклад: [20] фокусується на метаагентах, тобто агентах, які діють на кілька гіперпараметрів блокчейну, щоб збалансувати компроміс між безпекою та пропускну здатністю. Це спосіб оптимізації продуктивності блокчейну, що зберігає безпеку й надійність системи через RL.

Висновки

Моделі забезпечують абстракції, які використовують для представлення та передавання важливої інформації, без зайвих деталей, і допомагають впоратися зі складністю досліджуваної проблеми або розробленого рішення. Отже, вкрай важливо використовувати оптимальний підхід до моделювання. Беручи до уваги попередні дослідження, можна побачити, що всі розглянуті підходи до моделювання блокчейну розроблено з метою детального моделювання конкретних аспектів та/або задач, розв'язуваних за допомогою застосування блокчейну (наприклад, цільові дослідження топології мережі, як правило, використовують парадигму теорії графів), що, своєю чергою, робить їх досить специфічними й унеможливає загальне застосування.

З усіх розглянутих методів моделювання можна виділити агентно-орієнтований як найбільш перспективний з погляду його універсальності та близькості парадигми мультиагентних систем до структури блокчейну. Враховуючи всі попередньо наголошені особливості, наступним логічним

кроком у моделюванні блокчейну можна назвати визначення загального способу моделювання блокчейну шляхом зміщення акценту в агентно-орієнтованому підході рівнем вище — з внутрішніх деталей на структурні та організаційні особливості. Пропонується розглянути так званий організаційно-орієнтований підхід для моделювання технології блокчейну з усіма її особливостями, який мав би логічно доповнити та, ймовірно, поліпшити раніше розглянуті досягнення з моделювання технології блокчейну.

Список літератури

1. Alharby M. BlockSim: An Extensible Simulation Tool for Blockchain Systems / Maher Alharby, Aad van Moorsel // *Frontiers in Blockchain*. — 2020. — Vol. 3. — <https://doi.org/10.3389/fbloc.2020.00028>.
2. Buchman E. The latest gossip on BFT consensus / E. Buchman, J. Kwon, Z. Milosevic. — 2018. — <https://doi.org/10.48550/arXiv.1807.04938>.
3. Buterin V. A Next-Generation Smart Contract and Decentralized Application Platform [Electronic resource] / V. Buterin. — 2014. — Mode of access: https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (date of access: 22.07.2024). — Title from screen.
4. Cachin C. Introduction to Reliable and Secure Distributed Programming / C. Cachin. — 2nd ed. — New York, NY, USA : Springer Publishing Company, Incorporated, 2011. — 386 p.
5. Eyal I. Majority is not enough: Bitcoin mining is vulnerable / I. Eyal, E. G. Sirer // *International Conference on Financial Cryptography and Data Security*, Christ Church, 3–7 March 2014. — [S. l.], 2014. — <https://doi.org/10.48550/arXiv.1311.0243>.
6. Ferber J. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence / J. Ferber. — Boston, MA, USA : Addison-Wesley Reading, 1999. — 509 p.
7. Goodman L. M. Tezos—A Self-Amending Crypto-Ledger White Paper [Electronic resource] / L. M. Goodman. — 2014. — Mode of access: <https://www.tezos.com/static/papers/whitepaper.pdf> (date of access: 22.07.2024). — Title from screen.
8. Gürçan Ö. Multi-Agent Modelling of Fairness for Users and Miners in Blockchains / Önder Gürçan // *Communications in Computer and Information Science*. — Cham, 2019. — Pp. 92–99. — https://doi.org/10.1007/978-3-030-24299-2_8.
9. Gürçan Ö. On Using Agent-based Modeling and Simulation for Studying Blockchain Systems / O. Gürçan // *JFMS 2020—Les Journées Francophones de la Modélisation et de la Simulation—Convergences entre la Théorie de la Modélisation et la Simulation et les Systèmes Multi-Agents*. — [S. l.], 2020. — <https://doi.org/10.48550/arXiv.2405.01574>.
10. Hyperledger fabric / Elli Androulaki [et al.] // *EuroSys'18: Thirteenth EuroSys Conference 2018*, Porto Portugal. — New York, NY, USA, 2018. — <https://doi.org/10.1145/3190508.3190538>.
11. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development / C. Larman. — 3rd ed. — Upper Saddle River, NJ, USA : Prentice Hall PTR, 2004. — 736 p.
12. Malcolm G. Sheaves, Objects, and Distributed Systems / Grant Malcolm // *Electronic Notes in Theoretical Computer Science*. — 2009. — Vol. 225. — Pp. 3–19. — <https://doi.org/10.1016/j.entcs.2008.12.063>.
13. Marchesi L. ABCDE – agile block chain DApp engineering / Lodovica Marchesi, Michele Marchesi, Roberto Tonelli // *Blockchain: Research and Applications*. — 2020. — Vol. 1, no. 1–2. — Pp. 100002. — <https://doi.org/10.1016/j.bcr.2020.100002>.
14. Meldman-Floch W. Blockchain Cohomology / W. Meldman-Floch. — [S. l.], 2018. — <https://doi.org/10.48550/arXiv.1805.07047>.
15. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System / S. Nakamoto // *SSRN Electronic Journal*. — 2008. — <https://doi.org/10.2139/ssrn.3440802>.
16. Nowak T. Topology in Distributed Computing [Electronic resource] : Master's Thesis / T. Nowak. — Vienna, Austria, 2010. — Mode of access: https://publik.tuwien.ac.at/files/PubDat_194085.pdf (date of access: 22.07.2024). — Title from screen.
17. Sagar P. V. Sheaf Representation of an Information System / Pyla Vamsi Sagar, M. Phani Krishna Kishore // *International Journal of Rough Sets and Data Analysis*. — 2019. — Vol. 6, no. 2. — Pp. 73–83. — <https://doi.org/10.4018/ijrds.2019040106>.
18. Sapirshtein A. Optimal selfish mining strategies in bitcoin / A. Sapirshtein, Y. Sompolinsky, A. Zohar // *International Conference on Financial Cryptography and Data Security*. — [S. l.], 2016. — <https://doi.org/10.48550/arXiv.1507.06183>.
19. Selfish Behavior in the Tezos Proof-of-Stake Protocol / M. Neuder [et al.]. — [S. l.], 2020. — <https://doi.org/10.48550/arXiv.1912.02954>.
20. SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Blockchain Sharding System / Jianting Zhang [et al.] // *ICPP '20: 49th International Conference on Parallel Processing*, Edmonton AB Canada. — New York, NY, USA, 2020. — <https://doi.org/10.1145/3404397.3404460>.
21. SquirRL: Automating Attack Discovery on Blockchain Incentive Mechanisms with Deep Reinforcement Learning / C. Hou [et al.] // *In Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, 21–24 February 2021. — [S. l.]. — <https://doi.org/10.48550/arXiv.1912.01798>.
22. Towards a Performance Evaluation of Private Blockchain Frameworks using a Realistic Workload / Marcela T. Oliveira [et al.] // *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, France, 19–21 February 2019. — [S. l.], 2019. — <https://doi.org/10.1109/icin.2019.8685888>.
23. Wolfram D. A Sheaf Semantics for FOOPS Expressions [Electronic resource] / D. Wolfram, J. Goguen // *In Proceedings of the Object-Based Concurrent Computing, ECOOP'91 Workshop*, Geneva, 15–16 July 1991. — [S. l.], 1991. — Mode of access: https://www.researchgate.net/publication/220842390_A_Sheaf_Semantics_for_FOOPS_Expressions (date of access: 22.07.2024). — Title from screen.
24. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger [Electronic resource] / Gavin Wood. — 2014. — Mode of access: <http://cryptochainuni.com/wp-content/uploads/Ethereum-A-Secure-Decentralised-Generalised-Transaction-Ledger-Yellow-Paper.pdf> (date of access: 22.07.2024). — Title from screen.
25. Wooldridge M. An Introduction to MultiAgent Systems / M. Wooldridge. — 2nd ed. — Hoboken, NJ, USA : John Wiley & Sons, 2009. — 488 p.

References

- Alharby, M., & van Moorsel, A. (2020). BlockSim: An extensible simulation tool for blockchain systems. *Frontiers in Blockchain*, 3. <https://doi.org/10.3389/fbloc.2020.00028>.
- Androulaki, E., et al. (2018). *Hyperledger fabric*. EuroSys '18: Thirteenth EuroSys Conference, Porto, Portugal. <https://doi.org/10.1145/3190508.3190538>.
- Buchman, E., Kwon, J., & Milosevic, Z. (2018). *The latest gossip on BFT consensus*. arXiv. <https://doi.org/10.48550/arXiv.1807.04938>.
- Buterin, V. (2014). *A next-generation smart contract and decentralized application platform*. [White paper]. https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.
- Cachin, C. (2011). *Introduction to Reliable and Secure Distributed Programming* (2nd ed.). Springer Publishing Company.
- Eyal, I., & Sirer, E. G. (2014). *Majority is not enough: Bitcoin mining is vulnerable*. International Conference on Financial Cryptography and Data Security, Christ Church. <https://doi.org/10.48550/arXiv.1311.0243>.
- Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Reading.
- Goodman, L. M. (2014). *Tezos—A self-amending crypto-ledger white paper*. <https://www.tezos.com/static/papers/whitepaper.pdf>.
- Gürçan, Ö. (2019). Multi-agent modelling of fairness for users and miners in blockchains. *Communications in Computer and Information Science*, 92–99. https://doi.org/10.1007/978-3-030-24299-2_8.
- Gürçan, Ö. (2020). *On using agent-based modeling and simulation for studying blockchain systems*. JFMS 2020 — Les Journées Francophones de la Modélisation et de la Simulation. <https://doi.org/10.48550/arXiv.2405.01574>.
- Hou, C., et al. (2021). *Squirrel: Automating attack discovery on blockchain incentive mechanisms with deep reinforcement learning*. Proceedings of the Network and Distributed System Security Symposium, San Diego, CA. <https://doi.org/10.48550/arXiv.1912.01798>.
- Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall PTR.
- Malcolm, G. (2009). Sheaves, objects, and distributed systems. *Electronic Notes in Theoretical Computer Science*, 225, 3–19. <https://doi.org/10.1016/j.entcs.2008.12.063>.
- Marchesi, L., Marchesi, M., & Tonelli, R. (2020). ABCDE — Agile blockchain DApp engineering. *Blockchain: Research and Applications*, 1 (1–2), 100002. <https://doi.org/10.1016/j.bcr.2020.100002>.
- Meldman-Floch, W. (2018). *Blockchain cohomology*. arXiv. <https://doi.org/10.48550/arXiv.1805.07047>.
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.3440802>.
- Neuder, M., et al. (2020). *Selfish behavior in the Tezos proof-of-stake protocol*. arXiv. <https://doi.org/10.48550/arXiv.1912.02954>.
- Nowak, T. (2010). *Topology in distributed computing*. [Master's thesis]. Vienna, Austria. https://publik.tuwien.ac.at/files/PubDat_194085.pdf.
- Sagar, P. V., & Kishore, M. P. K. (2019). Sheaf representation of an information system. *International Journal of Rough Sets and Data Analysis*, 6 (2), 73–83. <https://doi.org/10.4018/ijrds.2019040106>.
- Sapirshstein, A., Sompolsky, Y., & Zohar, A. (2016). *Optimal selfish mining strategies in bitcoin*. International Conference on Financial Cryptography and Data Security. <https://doi.org/10.48550/arXiv.1507.06183>.
- Oliveira, M. T., et al. (2019). *Towards a performance evaluation of private blockchain frameworks using a realistic workload*. 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France. <https://doi.org/10.1109/icin.2019.8685888>.
- Wolfram, D., & Goguen, J. (1991). *A sheaf semantics for FOOPS expressions*. Proceedings of the Object-Based Concurrent Computing, ECOOP'91 Workshop, Geneva. https://www.researchgate.net/publication/220842390_A_Sheaf_Semantics_for_FOOPS_Expressions.
- Wood, G. (2014). *Ethereum: A secure decentralised generalised transaction ledger*. [White paper]. <http://cryptochainuni.com/wp-content/uploads/Ethereum-A-Secure-Decentralised-Generalised-Transaction-Ledger-Yellow-Paper.pdf>.
- Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). John Wiley & Sons.
- Zhang, J., et al. (2020). *SkyChain: A deep reinforcement learning-empowered dynamic blockchain sharding system*. ICPP '20: 49th International Conference on Parallel Processing, Edmonton, AB, Canada. <https://doi.org/10.1145/3404397.3404460>.

M. Yeshchenko

MODELING BLOCKCHAIN TECHNOLOGY

Blockchain is a very attractive technology because it provides a public, append-only, immutable, and ordered transaction log. Blockchain systems are inherently interdisciplinary as they combine different fields such as cryptography, multi-agent systems, distributed systems, social systems, economics, and finance. Additionally, they have a highly active and dynamic ecosystem where new blockchain platforms and algorithms are constantly being developed due to public and industry interest in the technology. Given the complexity and multifaceted nature of the blockchain, its presentation – modeling – via other, more well-known means should contribute to a better understanding of the capabilities and features of this technology.

First, the peculiarities and challenges of blockchain modeling are considered, which are mainly based on blockchain data structure, transactions, and consensus mechanism usage.

Later, a comparative analysis of four different modeling paradigms is carried out. The process-oriented approach is discussed first and provides an understanding of how blockchain nodes may be represented by components, while their behavior may be constructed as an algorithm executed by individual components. The graph-theoretic paradigm offers a more visual representation of the subject matter while providing

distributed algorithm capabilities. The object-oriented way of modeling offers more convenient encapsulation possibilities, as well as a well-adopted UML-based graphical accompaniment of the hierarchy and links between modeled blockchain nodes. Finally, the agent-oriented approach provides possibly the best overall approach to modeling blockchain technology by offering an object capable of representing the flexible behavioral nature of blockchain nodes, an agent, while continuing to allow a visual depiction of the modeled entity.

This article aims to explore the main methods of modeling blockchain technology and to determine the most promising one for further in-depth research into the possibilities of modeling all aspects of blockchain technology in the most efficient manner. As part of it, it was confirmed that agent-oriented approach to blockchain modeling is the most suitable one. Even more specifically, the organization-oriented approach, as a subdivision of the agent-oriented approach, is advised for experimentation to potentially provide a new perspective on blockchain technology representation with the help of multi-agent systems.

Keywords: blockchain, multi-agent systems, modeling.

Матеріал надійшов 22.07.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Давиденко А. М.

ЗАБЕЗПЕЧЕННЯ ПОРЯДКУ ОБРОБЛЕННЯ ПОВІДОМЛЕНЬ У РОЗПОДІЛЕНИХ СИСТЕМАХ

У статті проаналізовано основні виклики, які є актуальними для розробників розподілених систем, зокрема стосовно забезпечення порядку оброблення подій, що відбуваються у системі, та його впливу на загальний стан системи. Відповідно до цього, запропоновано класифікацію розподілених систем. Наведено приклади використання можливостей сучасних брокерів повідомлень RabbitMQ та Apache Kafka, які дають змогу забезпечити оброблення повідомлень лише одним споживачем у розподіленій системі, та порівняно їхні можливості.

Ключові слова: розподілені системи, програмування, алгоритми, класифікація, брокери повідомлень.

Вступ

У розподіленій системі (РС) комп'ютери (вузли або процесори) з'єднані та взаємодіють один з одним через комунікаційну мережу. На відміну від централізованої системи, де весь процес оброблення даних керується одним комп'ютером або сервером, у РС завдання і дані розподіляються між декількома вузлами. Забезпечуючи масштабованість і стійкість, РС вносять нові виклики — неминучий паралелізм і складність впорядкованих комунікацій між вузлами всередині системи. Тому проблема підтримки впорядкованості повідомлень (також відомих як події) є актуальною і широко досліджується.

У цій статті проаналізовано актуальні виклики у розробленні РС і наведено типові сценарії, за яких виникає необхідність упорядкування повідомлень для їх оброблення, а також запропоновано класифікацію РС відповідно до вимог щодо збереження поточного стану та обчислень.

1. Виклики у розробленні розподілених систем

Лампорт у своїй статті вводить часову логіку відношення під назвою «відбулося раніше», яке допомагає описати концепцію часткового впорядкування (логічний годинник) [3]. Це поняття далі використовують для опису розподіленого алгоритму, спрямованого на визначення часткового порядку подій у системі, а також проблем із синхронізацією. Крім того, Лампорт уперше формалізував важливе поняття причиновості (causality) в контексті РС. Підхід до визначення порядку подій, який він запропонував, базується на часових мітках. Кожному повідомленню в джерелі надають часову мітку, яку використовують для доставлення і оброблення повідомлення в правильному порядку. Безсумнівно, ця робота заклала основу для впорядкування повідомлень у РС і дала поштовх для багатьох інших досліджень у контексті РС.

Нині існує велике різноманіття запропонованих алгоритмів, які забезпечують контрольований порядок доставлення повідомлень, однак кожен з них має свої переваги і недоліки, що вкотре підкреслює актуальність цієї проблеми [1].

Попри те, що проблеми РС уже досить добре досліджені, певні питання досі залишаються без відповіді [7]. Наприклад, чи має впорядкування повідомлень бути вирішено за допомогою проміжного програмного забезпечення (middleware) або все ж таки кожним прикладним програмним забезпеченням? Порядок оброблення повідомлень є критично важливим для коректного стану РС. Часто проблеми оброблення повідомлень не помічають на етапі розроблення через недостатнє тестування або незначні навантаження. Проблема довгий час може лишатися непоміченою, якщо час оброблення події є меншим за час між тим, як нові події стаються у системі. Саме тому це важливо враховувати на етапі проєктування РС.

2. Класифікація розподілених систем

Ми пропонуємо класифікувати розподілені системи (РС) за такими ознаками стосовно вимог щодо поточного стану системи.

Перший клас — це РС, у яких необхідно забезпечити коректність поточного стану. В цьому разі можливе паралельне оброблення повідомлень системами, адже послідовне оброблення повідомлень не є критичним. Однак необхідно відстежувати порядок повідомлень задля збереження наміру повідомлень. Розглянемо приклад, у якому система може мати три стани: *Off*, *Initializing* та *On*. Припустимо, що процес p_1 надсилає подію $E_1 = \text{Initializing}$ і подію $E_2 = \text{On}$ із мінімальною часовою різницею. У РС може виникнути ситуація, коли подію E_2 буде оброблено першою, а подію E_1 — другою. В такому разі, подію E_1 може бути відкинута програмною логікою під час її оброблення як неактуальну. Використовуючи лінійні темпоральні логіки [4], очікуваний стан системи можна представити як $F(G\text{ On})$. Тоді порядок повідомлень можливо встановити, використовуючи логічний годинник, наприклад годинник Лампорта або векторний годинник.

Другий клас — це РС, у яких наявні причинно-наслідкові залежності відповідно до поточного стану системи. В цьому разі необхідно забезпечити строгий порядок оброблення кожного повідомлення. Розглядаючи приклад із попереднього пункту, припустимо, що у РС існують умови на перехід із попереднього стану у наступний (наприклад, $\text{Off} \rightarrow \text{Initializing}$). Пропустивши стан *Initializing*, як у першому прикладі, ми можемо порушити логіку роботи РС. Враховуючи ці вимоги, систему можна представити як $G(\text{Off} \cup (\text{Initializing} \cup G\text{ On}))$. У цьому випадку навіть у разі масштабування сервісів необхідно передбачити оброблення повідомлень лише одним споживачем в один момент часу. Деякі брокери повідомлень мають вбудовані для цього засоби (як-от RabbitMQ чи Apache Kafka). Варто зауважити, що в разі застосування цього підходу найбільше обмежується пропускну здатність системи.

Третій клас — це РС, у яких наявні причинно-наслідкові залежності відповідно до поточного стану системи в рамках певного контексту. Цей варіант накладає ті самі обмеження, що й попередній, але вони є незалежними для кожного контексту. Завдяки цій умові можливо покращити пропускну здатність системи. Тому цей варіант підходить для тих систем, в яких зберігається поточний стан і можливо виділити об'єкти у стані системи, які є незалежними в обробленні. В такому разі одночасне оброблення подій для кількох незалежних об'єктів не вплине на коректний стан системи.

За допомогою такої класифікації на етапі проєктування розподіленої системи можливо визначити необхідний технічний підхід для взаємодії між компонентами РС відповідно до поставлених вимог.

3. Забезпечення впорядкованого оброблення повідомлень за допомогою брокерів повідомлень

Брокери повідомлень — це проміжне програмне забезпечення, яке дає змогу різним системам, службам і програмам обмінюватися інформацією та взаємодіяти між собою. Простіше кажучи, брокери повідомлень діють як посередники для роботи різних програмних систем, наприклад вебдодатків. Окрім базових функцій, брокери повідомлень за допомогою вбудованих механізмів дають можливість досягти бажаних властивостей системи, як-от забезпечити впорядковане оброблення повідомлень.

3.1. RabbitMQ

RabbitMQ — це брокер повідомлень, який імплементує систему асинхронного обміну повідомленнями між компонентами програмної системи через відкритий стандарт протоколу AMQP (Advanced Message Queuing Protocol). За допомогою RabbitMQ виробники публікують повідомлення до обмінників (exchanges), які потім направляють це повідомлення до підв'язаних до нього черг. Черга є основним компонентом RabbitMQ, який зберігає повідомлення доти, поки їх не буде доставлено одержувачам. Черга є абстракцією, де повідомлення надходять після їх публікації від виробників і чекають на оброблення споживачами. Важливо, що черга дає змогу обробляти повідомлення незалежно від часу їх надсилання та отримання, що дозволяє розподіляти навантаження між кількома споживачами.

RabbitMQ дає можливість використовувати властивість SAC (Single Active Consumer) для певної черги та є корисною, якщо повідомлення мають бути оброблені у тому самому порядку, у якому вони

надійшли до черги [5]. SAC дозволяє мати лише одного активного споживача повідомлень з черги і перемикається на іншого доступного у разі, якщо активний споживач виходить із ладу. Щоб покращити пропускну здатність системи та забезпечити послідовне оброблення повідомлень у рамках контексту, можна використовувати плагіни, що підтримують механізм узгодженого хешування (consistent hashing) [6]. Слід зазначити, що проблема впорядкованого оброблення повідомлень може виникнути в разі створення не тільки РС, а й багатопотокових додатків.

3.2. Apache Kafka

Apache Kafka — це розподілене сховище подій і платформа для їх багатопотокового оброблення з високою пропускну здатністю та низькою затримкою, яке базується на власному протоколі. На відміну від RabbitMQ, Kafka ґрунтується на моделі витягування повідомлень (pull-based). Основою Kafka є розподілений журнал (log). Kafka використовує концепцію тем (topics) і розділів (partitions). Якщо порівнювати з RabbitMQ, тему можна уявити як чергу, що поділяється на розділи, базуючись на ключі повідомлення. Наприклад, у прикладній системі таким розділом може бути назва країни. Виробник додає своє повідомлення до теми, а споживач самостійно вичитує повідомлення, оновлюючи зміщення (offset), що фіксує індекс останнього прочитаного повідомлення. Таким чином, повідомлення зберігаються до планового очищення, що дає змогу за необхідності перечитувати повідомлення кілька разів, а брокер повідомлень не відстежує вичитування повідомлень споживачами.

Задля досягнення впорядкованого оброблення повідомлень Apache Kafka надає вбудовані засоби. Наприклад, щоб забезпечити строгий порядок оброблення повідомлень у рамках теми, необхідно використовувати єдиний розділ для всіх повідомлень і одну групу споживачів (consumer group) [2]. За ідеологією Kafka один розділ може мати лише одного споживача в рамках кожної групи споживачів (рис. 1). При цьому в одній групі споживачів може бути більша кількість споживачів, ніж розділів. У такому разі не призначені до розділів споживачі будуть неактивними, і у разі виходу з ладу одного з активних споживачів іншого буде під’єднано автоматично. Щоб досягти послідовного оброблення повідомлень у рамках контексту, тему може бути поділено на розділи. При цьому послідовне оброблення повідомлень здійснюватиметься в рамках кожного з розділів.

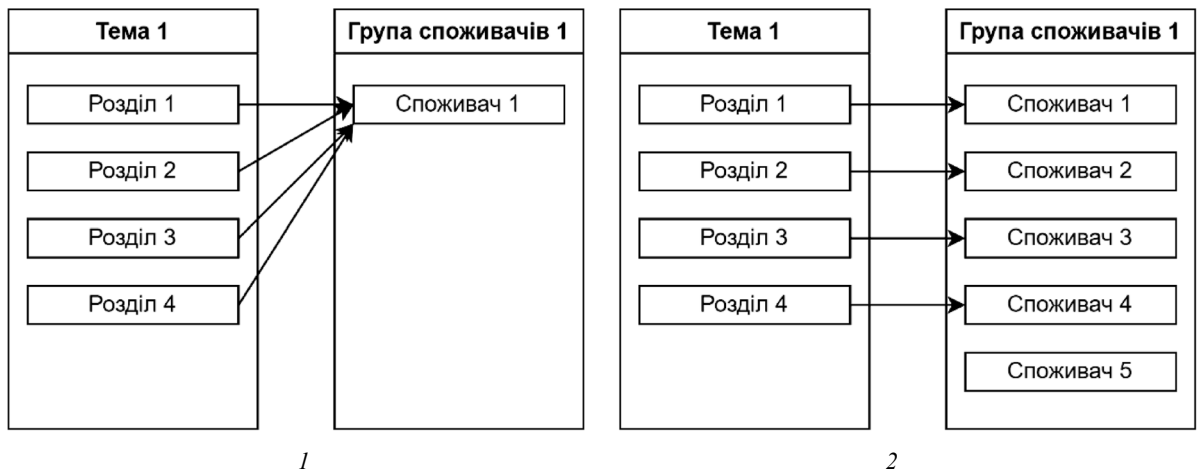


Рис. 1. Можливі сценарії розподілу розділів і споживачів у Apache Kafka: 1 — один споживач оброблює повідомлення з кількох розділів; 2 — один зі споживачів не є активним, інші споживачі підключені до розділів у відношенні 1:1

Застосування брокерів повідомлень до класифікованих РС

Таблиця 1

Брокер	RabbitMQ	Apache Kafka
Клас РС		
Клас 1	Логічний годинник	Логічний годинник
Клас 2	SAC	Єдиний розділ у темі, одна група споживачів
Клас 3	SAC + плагін з узгодженим хешуванням	Окремі розділи у темі, одна група споживачів

Отже, сучасні брокери повідомлень надають ефективні засоби, що дають змогу забезпечити контрольований порядок оброблення повідомлень для РС другого і третього класу. Apache Kafka надає ці засоби «з коробки», а для деяких сценаріїв RabbitMQ потрібно встановити додаткові плагіни. Для підтримки РС першого класу брокери не надають вбудованих засобів. Це можна пояснити необхідністю тісної інтеграції з бізнес-логікою прикладного програмного забезпечення, адже рішення про відкидання повідомлення чи події може бути здійснено лише на рівні прикладного програмного забезпечення.

Висновки

У рамках цієї роботи висвітлено деякі актуальні виклики, з якими стикаються розробники РС. Незважаючи на те, що зазначені проблеми вже доволі добре досліджені, підходи до їх вирішення досі не є уніфікованими.

У статті запропоновано класифікацію РС відповідно до бажаних характеристик стосовно збереження поточного стану. Наведено приклади використання засобів сучасних брокерів повідомлень RabbitMQ та Apache Kafka, які дають змогу забезпечити оброблення повідомлень лише одним споживачем, порівняно можливості їх застосування до класифікованих РС.

Подальшим розвитком цієї роботи є створення дизайн-патернів для описаних типів РС. Також цікавим може бути застосування елементів штучного інтелекту для визначення контексту поточного стану систем і визначення подій, які не впливають на нього задля можливості паралельного оброблення повідомлень без порушення стану системи.

Список літератури

1. Défago X. Total order broadcast and multicast algorithms / X. Défago, A. Schiper, P. Urbán // *ACM Computing Surveys*. — 2004. — Vol. 36, no. 4. — Pp. 372–421. — <https://doi.org/10.1145/1041680.1041682>.
2. Kafka 2.0 Documentation [Electronic resource]. Apache Kafka. — Mode of access: <https://kafka.apache.org/20/documentation.html> (date of access: 22.09.2024). — Title from screen.
3. Lamport L. Time, clocks, and the ordering of events in a distributed system / L. Lamport // *Communications of the ACM*. — 1978. — Vol. 21, no. 7. — Pp. 558–565. — <https://doi.org/10.1145/359545.359563>.
4. Pnueli A. The temporal logic of programs / A. Pnueli // *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*. — 1977. — Pp. 46–57. — <https://doi.org/10.1109/SFCS.1977.32>.
5. RabbitMQ Documentation [Electronic resource]. *RabbitMQ: One broker to queue them all | RabbitMQ*. — Mode of access: <https://www.rabbitmq.com/docs> (date of access: 22.09.2024). — Title from screen.
6. Vahab M. Consistent hashing with bounded loads / M. Vahab, M. Thorup, M. Zadimoghaddam // *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. — 2017. — Pp. 587–604. — <https://doi.org/10.1137/1.9781611975031.39>.
7. van Steen M. Distributed Systems 4th edition [Electronic resource] / M. van Steen, A. S. Tanenbaum // *DISTRIBUTED-SYSTEMS.NET*. — Mode of access: <https://www.distributed-systems.net/index.php/books/ds4/> (date of access: 22.09.2024). — Title from screen.

References

- Défago, X., Schiper, A., & Urbán, P. (2004). Total order broadcast and multicast algorithms. *ACM Computing Surveys*, 36 (4), 372–421. <https://doi.org/10.1145/1041680.1041682>.
- Kafka 2.0 Documentation. (n.d.). Apache Kafka. <https://kafka.apache.org/20/documentation.html>.
- Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21 (7), 558–565. <https://doi.org/10.1145/359545.359563>.
- Pnueli, A. (1977). The temporal logic of programs. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, 46–57. <https://doi.org/10.1109/SFCS.1977.32>.
- RabbitMQ Documentation. (n.d.). RabbitMQ: One broker to queue them all | RabbitMQ. <https://www.rabbitmq.com/docs>.
- Vahab, M., Thorup, M., & Zadimoghaddam, M. (2017). Consistent hashing with bounded loads. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 587–604. <https://doi.org/10.1137/1.9781611975031.39>.
- van Steen, M., & Tanenbaum, A. S. (n.d.). *Distributed Systems 4th edition*. *DISTRIBUTED-SYSTEMS.NET*. <https://www.distributed-systems.net/index.php/books/ds4/>.

A. Davydenko

ENSURING THE ORDER OF MESSAGE PROCESSING IN DISTRIBUTED SYSTEMS

In a distributed system, computers, also known as nodes or processors, are connected and communicate with each other through a communication network. Unlike a centralized system, where a single computer or server handles all processing tasks, in a distributed system, tasks and data are distributed among

multiple nodes. While providing great scalability and resilience, distributed systems introduce new challenges – unavoidable concurrency and the difficulty of maintaining ordered communication between nodes within a system. That is why distributed systems and applications are especially difficult to create and maintain. Since the problem of preserving message ordering (also known as event ordering) is crucial, it has been widely studied. Some of the key findings are covered in this paper.

Many proposed algorithms ensure a strict order of message delivery, but each has its advantages and disadvantages, which once again emphasizes the urgency of this problem.

This paper highlights some of the current challenges faced by developers of distributed systems. Despite the fact that these problems are already well-researched, approaches to their resolution are still not settled.

The article proposes a classification of distributed systems based on the desired characteristics in terms of maintaining and evaluating the current state. An example of using modern message broker tools (such as RabbitMQ and Apache Kafka), which ensure the processing of messages by only one consumer, is given.

Further development of this work involves creating design patterns for the described types of distributed systems. Additionally, incorporating artificial intelligence elements to determine the context of the current state of systems and identify events that do not affect it could enable parallel processing of messages without disrupting the system state.

Keywords: distributed systems, programming, algorithms, classification, message brokers.

Матеріал надійшов 27.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

СТРУКТУРОВАНІЙ ОПТИМІЗОВАНИЙ ПОШУК У НЕСТРУКТУРОВАНІХ ДАНИХ ДЛЯ ЗАДАЧІ АНАЛІЗУ МЕНЮ

У статті представлено розроблення пошукової системи для цифрових меню закладів харчування Києва українською мовою. Проєкт реалізовано з використанням сучасних методів обробки природної української мови, зокрема для завдань лематизації, класифікації текстів і фільтрації даних. Основну увагу приділено розробленню алгоритмів, які забезпечують пошук інформації про складники страв, харчові обмеження, алергени та інші ключові характеристики.

У рамках дослідження здійснено аналіз цифрових меню київських закладів, що надало змогу сформувати цілісну картину сучасного стану ресторанного бізнесу в Україні. Результати роботи демонструють можливість структурованого та оптимізованого пошуку в неструктурованих даних меню українською мовою, що дає змогу користувачам легко обирати страви, які розміщено ресторанами.

Ключові слова: обробка природної мови, цифрове меню, заклад харчування, місто Київ, пошуковик, ресторанний бізнес, класифікація текстів, видобування інформації, фільтрація даних, структурований пошук, база даних, база знань, онтологія, Protégé, OWL, SWRL, Pellet, ризонер, Retrieval-Augmented Generation, інформаційні технології, інформаційна система, великі мовні моделі.

Вступ

У сучасному світі зростає обсяг неструктурованої текстової інформації, тому здатність ефективно обробляти та аналізувати великі масиви даних є важливою для різноманітних галузей. Індустрія ресторанного бізнесу в Києві стрімко розширюється, кожен рік відкриваються нові заклади, які розміщують меню українською мовою в цифровому форматі. З огляду на це, автоматизація процесу пошуку страв стає ключовою умовою підвищення якості обслуговування клієнтів. Такий підхід дає змогу враховувати харчові вподобання, дієтичні обмеження та алергічні реакції, що робить аналіз меню більш персоналізованим і зручним для споживачів.

Сучасні технології обробки природної мови (NLP) надають нові можливості для роботи з текстами цифрових меню, зокрема щодо їхньої класифікації і фільтрації. У розробленій системі для аналізу та стандартизації текстів запитів користувачів використано великі мовні моделі (LLM), а також для автоматичного розширення запитів технологію RAG (Retrieval-Augmented Generation) [6], що дає змогу підвищити точність і релевантність пошуку.

Актуальність статті обґрунтовано необхідністю удосконалення українськомовних рішень у ресторанному бізнесі. Сучасні застосунки здебільшого орієнтовано на англійськомовних користувачів, тому впровадження методів для аналізу і структурування цифрових меню українською мовою сприяє оптимізації систем обслуговування для користувачів.

Зважаючи на сформовану збірку даних оцінено популярності харчових пропонуваль, їхніх складників і ціноутворення. На основі цих даних ресторани можуть визначати актуальність страв і враховувати поточні кулінарні тренди, покращувати клієнтоорієнтованість і конкурентоспроможність у галузі громадського харчування.

Використання NLP-моделей для оброблення текстів

NLP-модель (модель обробки природної мови) — це комп'ютерна система, яку створено для аналізу людської мови. Такі моделі використовують у застосунках для розпізнавання мовлення, машинного перекладу, автоматичного підсумовування текстів, розуміння природної мови тощо.

Зазвичай на сайтах закладів можна помітити, що в описі страв написано «Гарного дня», «Смачного», подано недоречний текст або ж опису бракує. Також використовують молодіжний сленг і популярні фрази, наприклад «Трушна паста», «Топчик», «Хелсі страва» (див. рис. 1). Помічено, що наявність зазначених фраз ускладнює точне оброблення і класифікацію страв, тому що створює «шум» у даних. Такі випадки оброблено з використанням NLP-моделі, видалено недоречні слова і виділено складники в описах. Використано такі моделі: GPT від OpenAI та LLaMA від Meta AI.

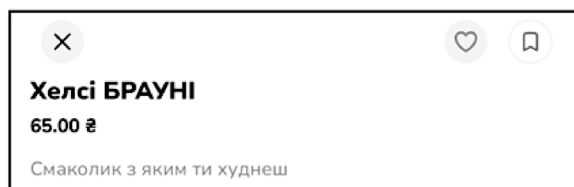


Рис. 1. Приклад недоречного опису

Формування бази даних

Для реалізації застосунку пошуку за стравами розглянуто меню закладів харчування в місті Київ, які розміщено в інтернеті в цифровому форматі. Для пошуку й завантаження використано сайт Choice.qr [2]. Таке рішення зумовлено тим, що на цьому сайті міститься понад шість тисяч меню закладів харчування і всі вони мають однакову структуру.

У роботі застосунку для реалізації функціоналу зберігання, оброблення та швидкого доступу до даних використано систему керування базами даних MySQL [7]. У неї попередньо завантажено опис страв, їхні ціну, вагу та особливі характеристики, а саме: гостра; вегетаріанська — без м'яса, риби чи інших морських істот; веганська — без продуктів тваринного походження; без глютену.

Для створення застосунку, у якому можна здійснювати пошук страв, потрібен швидкий доступ до структурованих меню закладів. Для користувача вагомою є інформація про назву страви, її складники, ціну, вагу, можливі алергени та доступність страви, тобто розташування закладу, де її можна замовити, тому використано базу даних для зберігання потрібної інформації.

restaurant_name
id : integer * dish_name : varchar (255) * description : text o price: decimal (10,2) o weight : varchar (10) * location: varchar (255) o image: varchar (255) o spicy: boolean o vegetarian: boolean o vegan: boolean o gluten_free: boolean

Рис. 2. Схема бази даних цифрових меню закладів харчування

Окрім реляційної бази даних MySQL, для зберігання та оброблення семантичної інформації про страви використано онтологічну базу знань, яку розроблено в Protégé [4], для виявлення прихованих залежностей, а саме: логічний висновок за допомогою Pellet [3] дає змогу ідентифікувати страви, які відповідають складним критеріям, навіть якщо ці залежності не явно зазначені в базі даних.

Оброблення запиту від користувача

У тексті запиту можливі слова в різних граматичних формах, тому їх оброблено і приведено до основної форми (леми) за допомогою UDPipe. Цей інструмент містить функції для аналізу текстів, а саме: токенизацію, лематизацію, розмітку частинами мови (POS tagging) тощо [8]. Отримані лемми

проаналізовано, і слова, які позначено іменниками (NOUN), використано для здійснення пошуку. Текст запиту очищено від зайвих слів і символів, виділено ключові слова та приведено запити до єдиного формату за допомогою великих мовних моделей (LLM). Для забезпечення коректності пошуку та фільтрації страв у застосунку також сформовано спеціалізовані словники, які містять синонімічні назви інгредієнтів, зокрема сирів, м'яса, овочів тощо (див. рис. 3).

Для покращення формування необроблений текст запиту передано до LLM, наприклад, «Хелсі смачненький круасан із сирною начинкою недорогого». Також до моделі надіслано заздалегідь підготовлені словники, як-от словник для сирів, що містить варіативні назви: «маскарпоне», «брі», «cheder», і додано семантичні зв'язки між інгредієнтами із застосуванням бази знань, яку створено у Protégé, наприклад автоматично визначено, що складник із класу «Молочні продукти» містить алерген «лактоза». Крім того, у запропонованій онтологічній базі знань встановлено логічні залежності між об'єктами за допомогою правил SWRL (Semantic Web Rule Language) [9]. Для реалізації логічного висновку використано рушій Pellet, який інтегровано в Protégé.

Після цього запит приведено до єдиного формату, видалено нерелевантні слова та збережено ключові терміни для пошуку за допомогою мовної моделі. Для розширення запиту користувача використано технологію RAG (Retrieval-Augmented Generation). Наприклад, запит «паста» автоматично розширено на терміни «макарони», «спагеті», «італійська кухня», що дає змогу знайти всі відповідні страви.

У застосунку реалізовано комбінований підхід, який поєднує аналіз ключових слів, фільтрацію та семантичний пошук, так створено високий рівень точності пошукових запитів і надано можливість користувачам легко знаходити страви, що задовольняють їхні особливі вимоги й фінансові можливості. Потрібно зазначити, що в розробленій системі використано не лише методи машинного навчання, а й rule-based підхід, зокрема регулярні вирази застосовано для реалізації фільтрації.

Щоби здійснювати пошук, створено список алергенів [1], які можна вибрати за допомогою порця, наприклад горіхи, гриби, глютен, лактоза, яйця, риба, соя, селера. Окрім алергій, люди можуть мати певні харчові звички, наприклад веганство, вегетаріанство, безглютенова дієта.

```
"Сир": ["Брі", "Камамбер", "Чеддер", "Гауда",  
        "Горгонзола", "Рокфор", "Пармезан", "Моцарела",  
        "Фета", "Бринза", "Грюйер", "Емменталь", "Рікотта",  
        "Козячий сир", "Блакитний сир", "Пекоріно", "Качокавалло",  
        "Халлумі", "Чечіль", "Шевр", "Горгонзола", "Горгондзола",  
        "Філадельфія", "Плавлений сир", "Маскарпоне", "Рікотта",  
        "Сулугуні", "Грюер", "Фетакса", "Маасдам", "Едам",  
        "Проволоне", "Буррата", "Манчего", "Стілсон", "Радамер",  
        "Панір", "Скаморца", "Асьяго", "Раклет", "Шевр", "Ансьяго"]
```

Рис. 3. Приклад словника для сортів сиру

Дослідження на основі вхідних даних

Зважаючи на сформовану збірку даних цифрових меню закладів харчування Києва з усіма назвами, складниками, ціною та вагою страв, проведено кількісні аналізи. Спершу проведено аналіз для оцінювання популярності страв у меню ресторанів українською мовою. Результати візуалізовано у вигляді графіка для наочності (рис. 4). За отриманими результатами найпопулярнішою стравою є салат, який трапляється 899 разів. Це дослідження також є прикладом використання підходів комп'ютерної соціальної науки (computational social science), що дає змогу вивчати гастрономічні переваги споживачів та взаємозв'язок між географічним розташуванням закладів харчування і ціноутворенням. Проаналізовано великі обсяги даних і отримано соціально значущу інформацію для вдосконалення ресторанного бізнесу з використанням такого підходу.

Також проведено кількісний аналіз складників страв (див. рис. 5). Найчастішим інгредієнтом виявлено соус, що є досить неоднозначно, оскільки є прості соуси, наприклад майонез або кетчуп, проте знайдено й такі, що містять багато складників, наприклад болоньезе або бешамель. Другим за популярністю інгредієнтом є сир, звідси можна емпірично припустити, що найчастіший алерген, який трапляється у стравах, це лактоза, що підтверджено далі в роботі.

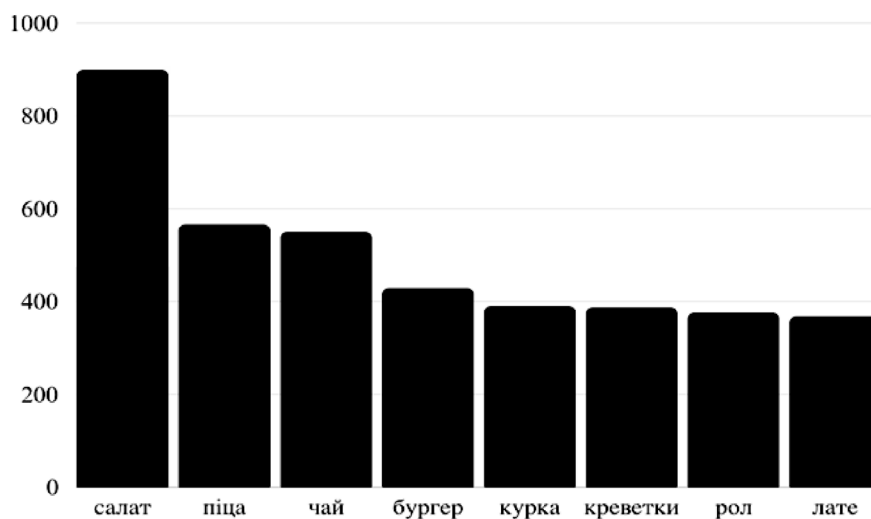


Рис. 4. Перелік найпопулярніших страв

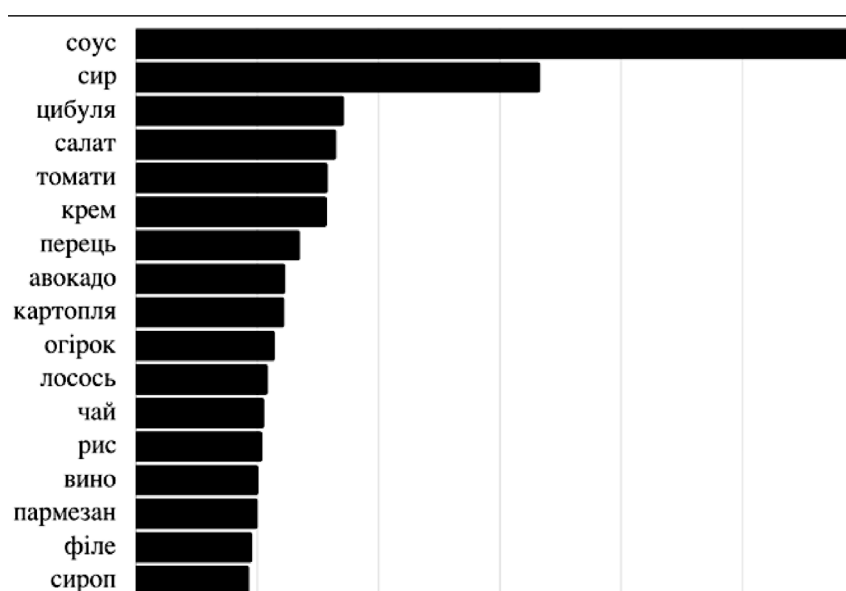


Рис. 5. Перелік найчастіших інгредієнтів у стравах

Додатково проведено аналіз найчастіших алергенів, де безперечним лідером є лактоза, яка міститься у 7652 стравах, що становить 21 % усіх страв. Кількість решти алергенів у меню показано на рис. 6.

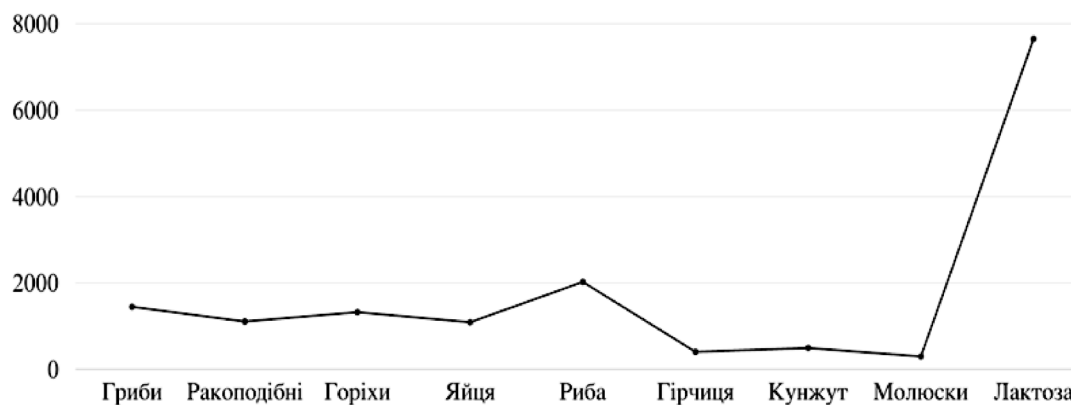


Рис. 6. Перелік найчастіших алергенів

Оскільки в базі даних описаного пошуковика страв міститься окремий стовпець, де вказано ціни всіх страв, то їх також проаналізовано. За отриманими результатами середня ціна страви в Києві становить 291,25 грн. Щоби переконатися, що такий результат не виник через можливу появу одного ресторану з великою середньою ціною, виведено п'ять найдорожчих закладів і п'ять найдешевших (див. рис. 7).

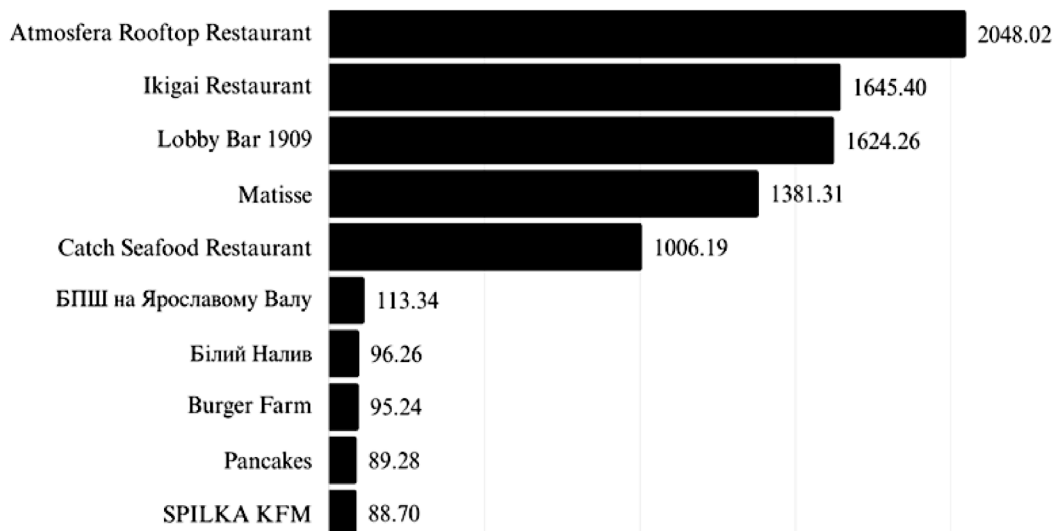


Рис. 7. Перелік п'яти найдорожчих і п'яти найдешевших ресторанів

Наступним кроком проведено дослідження варіацій у ціноутворенні страв залежно від географічного розташування ресторанів у місті Києві. Гіпотеза полягає в тому, що заклади з розташуванням ближче до центру міста мають вищі ціни порівняно з тими, які розташовуються на периферії. На рис. 8 показано розташування на мапі закладів харчування, що використано при створенні пошуковика. Щоби продемонструвати зміну цін залежно від розташування, використано теплокарту: теплі кольори (червоний, оранжевий) показують високі значення, холодні кольори (синій, зелений) — низькі значення.

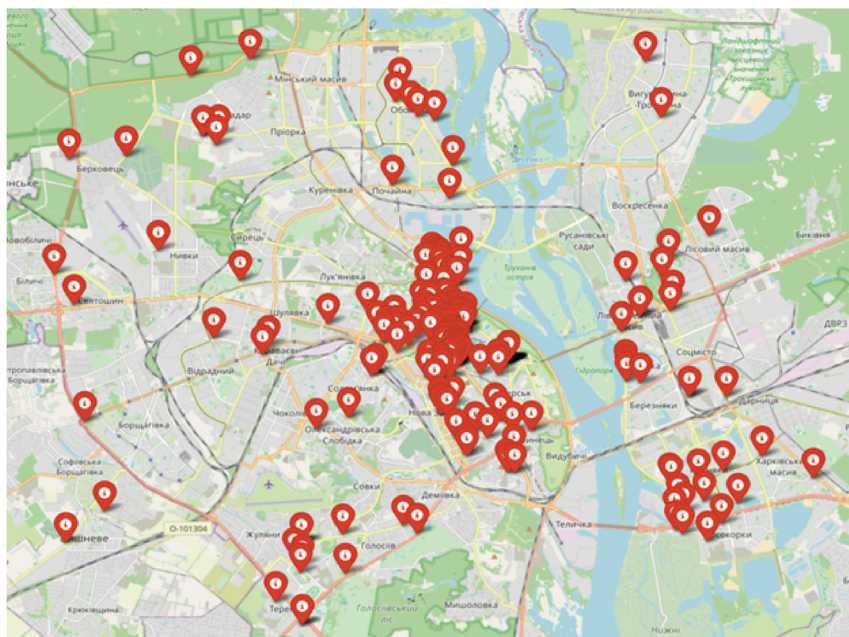


Рис. 8. Розташування всіх проаналізованих закладів харчування в місті Київ

На рисунку 9 можна побачити, що гіпотезу підтверджено. Географічне розташування має значний вплив на ціноутворення в ресторанах. Найдорожчі заклади харчування розміщено в центрі міста, а заклади на периферії мають нижчу цінову політику.

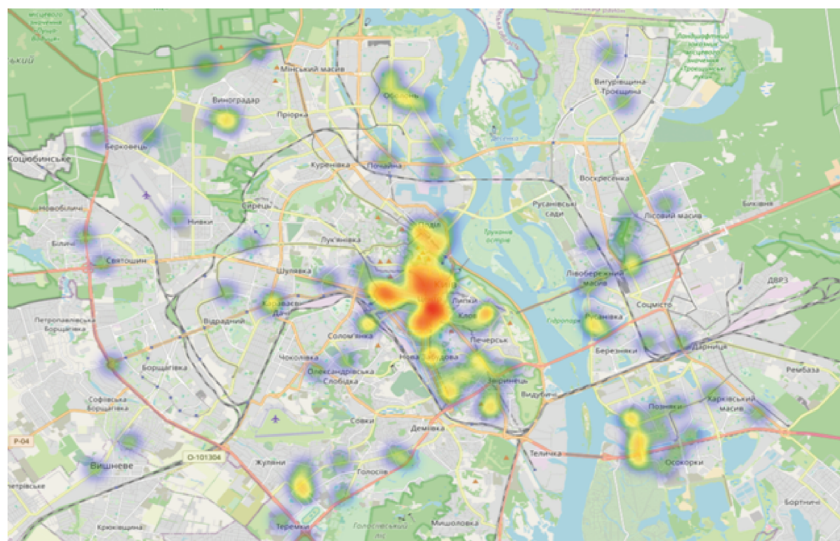


Рис. 9. Тенденція зміни середніх цін у закладах харчування Києва залежно від розташування

Висновки

У статті описано та продемонстровано роботу пошукової системи для цифрових меню закладів харчування Києва. Розроблену систему інтегровано із сучасними методами обробки природної мови (NLP), зокрема з використанням морфосинтаксичного аналізу, семантичного аналізу із застосуванням онтологічної бази знань, розширення запитів за допомогою технології RAG (Retrieval-Augmented Generation), логічних висновків із правилами SWRL та рушієм Pellet. У системі використано реляційну базу даних MySQL для зберігання інформації про страви та онтологічну базу знань, створену в Protégé, для збагачення семантичної інформації та підвищення точності пошуку. Це надає змогу здійснювати структурований пошук страв з урахуванням складників, дієтичних обмежень, алергенів, цінкових параметрів та географічного розташування закладів.

Система поєднує результати роботи великих мовних моделей (LLM), з використанням яких очищено і стандартизовано текст запиту, а також його розширено за допомогою заздалегідь сформованих словників та правил, для забезпечення повноти пошукових результатів. Для точного виділення інгредієнтів та фільтрації страв за заданими критеріями застосовано rule-based підхід, зокрема регулярні вирази, і підвищено точність і релевантність пошуку. Інтегровано онтологічну базу знань із правилами SWRL і рушієм Pellet, автоматично визначено логічні залежності між інгредієнтами та стравами, а також виявлено нові знання.

На прикладах цифрових меню київських закладів харчування проілюстровано можливості системи щодо автоматичного оброблення даних, фільтрації страв за алергенами та дієтичними обмеженнями, а також кількісного аналізу популярності страв і їхніх складників, оцінено вплив географічного розташування закладів на цінову політику, що є прикладом використання підходів комп'ютерної соціальної науки (computational social science) і може слугувати основою для оптимізації меню ресторанів.

Отже, запропоновані рішення є значущим кроком для вдосконалення клієнтоорієнтованих послуг у галузі ресторанного бізнесу.

Список літератури

1. Харчові алергени в меню ресторану: як зробити ваш заклад безпечним? — Блог ChoiceQR [Електронний ресурс] // ChoiceQR — Smart solutions for modern restaurants. — Режим доступу: <https://choiceqr.com/uk/news/harchovi-alergeny-v-menyu-restoranu-yak-zrobyty-vash-zaklad-bezpechnym/>.
2. ChoiceQR — онлайн QR меню для закладів [Електронний ресурс] // ChoiceQR — Smart solutions for modern restaurants. — Режим доступу: <https://choiceqr.com/uk/>.
3. Pellet — Semantic Web Standards [Electronic resource] // W3C. — Mode of access: <https://www.w3.org/2001/sw/wiki/Pellet>.
4. protégé [Electronic resource] // protégé. — <https://protege.stanford.edu/>.
5. Sabry F. Information Extraction [Electronic resource] / Fouad Sabry. — [S. l.]: One Billion Knowledgeable. — 78 p.
6. Siddharth L. Retrieval Augmented Generation using Engineering Design Knowledge [Electronic resource] / L. Siddharth, Jianxi Luo // Knowledge-Based Systems. — 2024. — P. 112410. — <https://doi.org/10.1016/j.knosys.2024.112410>.
7. SQL: Learn SQL (using MySQL) in One Day and Learn It Well. SQL for Beginners with Hands-on Project. (Learn Coding Fast with Hands-On Project): LCF Publishing, Chan, Jamie: 9781731039668.

8. Straka Milan. UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task / Milan Straka // *Proceedings of CoNLL 2018: The SIGNLL Conference on Computational Natural Language Learning*. — Association for Computational Linguistics, Stroudsburg, PA, USA, 2018. — Pp. 197–207.
9. SWRL: A Semantic Web Rule Language Combining OWL and RuleML [Electronic resource] // W3C. — Mode of access: <https://www.w3.org/submissions/SWRL/>.

References

- Chan, J. (2018). *SQL: Learn SQL (using MySQL) in One Day and Learn It Well. SQL for Beginners with Hands-on Project. (Learn Coding Fast with Hands-On Project)*. LCF Publishing.
- ChoiceQR — online QR menu dlia zakladiv. (n.d.). ChoiceQR — Smart solutions for modern restaurants. <https://choiceqr.com/uk/>.
- Kharchovi alerheny v menu restoranu: yak zrobyty vash zaklad bezpechnym? — Bloh ChoiceQR. (2023, 23 lystopada). ChoiceQR — Smart solutions for modern restaurants. <https://choiceqr.com/uk/news/harchovi-alergeny-v-menu-restoranu-yak-zrobyty-vash-zaklad-bezpechnym>.
- Pellet — Semantic Web Standards. (2011, 9 sichnia). W3C. <https://www.w3.org/2001/sw/wiki/Pellet>.
- protégé. (n.d.). protégé. <https://protege.stanford.edu/>.
- Sabry, F. (2023). *Information Extraction*. One Billion Knowledgeable.
- Siddharth, L., & Jianxi, L. (2024). *Retrieval augmented generation using engineering design knowledge*. Knowledge-Based Systems. <https://doi.org/10.1016/j.knosys.2024.112410>.
- Straka, M. (2018). UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task. *Proceedings of CoNLL 2018: The SIGNLL Conference on Computational Natural Language Learning*, 197–207.
- SWRL: A Semantic Web Rule Language Combining OWL and RuleML. (2004, 21 травня). W3C. <https://www.w3.org/submissions/SWRL/>.

O. Smysh, A. Chyzhova

STRUCTURED OPTIMIZED SEARCH IN UNSTRUCTURED DATA FOR MENU ANALYSIS TASKS

The article describes the development of a dish search engine for digital restaurant menus in Kyiv, focusing on Ukrainian-speaking users. The system integrates modern Natural Language Processing (NLP) methods such as lemmatization, text classification, and data filtering, alongside Retrieval-Augmented Generation (RAG), specialized ingredient dictionaries, a database, and an ontological knowledge base designed in Protégé. Using rules from the Semantic Web Rule Language (SWRL) and logical inference through the Pellet reasoning engine, the system performs semantic analysis of user queries, automatically identifying relationships between dish components, and improving search relevance.

The search algorithm utilizes a multi-layered approach that combines machine learning, logical reasoning, and rule-based filtering. User queries, often containing informal or varied phrasing, are first processed by a large language model (LLM) to identify and standardize key terms. The LLM is enhanced with pre-defined dictionaries (e.g., for cheese types like “mascarpone”, “brie,” or “cheddar”) and connected to an ontological knowledge base, which enriches the query with semantic relationships. RAG extends this functionality by automatically expanding search terms to include synonyms or related concepts, such as linking “pasta” to “macaroni” or “spaghetti.”

The study incorporates principles of computational social science to analyze semi-structured data from digital restaurant menus, such as the popularity of dishes and their ingredients, as well as the impact of restaurant location on pricing. The data highlights trends in customer preferences and provides actionable insights for optimizing restaurant menus.

The developed system successfully integrates NLP techniques, logical reasoning, and structured data storage, achieving high accuracy and relevance in search results. By incorporating an LLM, RAG, and ontological reasoning, the system demonstrates the potential for significantly enhancing customer-oriented services in the restaurant industry through advanced data analysis and semantic search tools.

Keywords: natural language processing, digital menu, restaurant, Kyiv city, search engine, restaurant business, text classification, information extraction, data filtration, structured search, database, knowledge base, ontology, Protégé, OWL, SWRL, Pellet, reasoner, Retrieval-Augmented Generation, information technologies, information system, large language models.

Матеріал надійшов 02.12.2024



УЗАГАЛЬНЕНИЙ АГЕНТ ШТУЧНОГО ІНТЕЛЕКТУ SIMA

Стаття аналізує проєкт SIMA (Scalable, Instructable, Multiworld Agent) від Google DeepMind, спрямований на створення узагальненого ШІ-агента, здатного виконувати завдання у різноманітних тривимірних середовищах за мовними інструкціями. Розглянуто методи навчання, оцінювання та особливості роботи агента, а також проведено порівняння з іншими проєктами, такими як OpenAI Five і AlphaStar. Попри досягнуті результати, висвітлено ключові виклики, зокрема технічні та етичні аспекти, що залишаються на шляху до створення загального штучного інтелекту.

Ключові слова: штучний інтелект, агент, SIMA, віртуальне середовище.

Вступ

Створення узагальненого агента штучного інтелекту як часткового випадку загального ШІ (AGI, універсального вирішувача задач) є однією з найскладніших задач сучасної науки. Створення такого агента обов'язково передбачає здатність засвоювати нові навички без спеціального програмування, адаптуватися до незнайомих ситуацій і ухвалювати ефективні рішення. Задоволення цих вимог потребує значних технічних проривів та обчислювальних ресурсів.

Розроблення універсального вирішувача супроводжується етичними викликами, зокрема йдеться про безпеку, ризики використання та відповідальне впровадження. Перспективи його застосування охоплюють наукові дослідження в різноманітних галузях (медицина, освітні інновації, оптимізація виробництва).

Серед важливих сучасних досягнень у цій царині — проєкт **SIMA** (Scalable, Instructable, Multiworld Agent) від Google DeepMind [3]. Основною складовою проєкта є розроблення спеціального агента, який наділений здатністю переносити знання між віртуальними середовищами. В майбутньому апробовані технології можуть бути інтегровані у фізичні системи для застосування у реальному світі.

Коротко охарактеризуємо історію попередніх досліджень у сфері створення універсального агента, зокрема розглянемо деякі агенти та проєкти, які були розроблені в останні роки та показали цікаві результати.

Відомо, що відеоігри є важливим середовищем для дослідження та розроблення універсальних ШІ-агентів. Вони дають змогу тестувати алгоритми, здатні адаптуватися до різноманітних завдань без необхідності спеціального програмування для кожної задачі. Багатство ігрових світів забезпечує умови для вивчення адаптивності, навчання з досвідом та розвиток стратегічного мислення агентів.

У 2019 році компанія DeepMind представила AlphaStar [2] — агента штучного інтелекту, який досяг рівня професійних гравців у складній стратегії в реальному часі StarCraft II. У цій грі гравці мають керувати ресурсами, будувати бази, створювати армії та стратегічно планувати дії проти суперника. AlphaStar спочатку навчався на записах ігор професійних гравців, щоб засвоїти базові стратегії та тактики. Потім він грав мільйони ігор проти себе та інших версій агентів, покращуючи свої навички через досвід.

AlphaStar продемонстрував здатність до довгострокового планування, адаптації до стилю гри опонента та управління багатьма юнітами одночасно, що є важливими кроками до створення AGI.

У 2019 році OpenAI розробила OpenAI Five [1], команду з п'яти агентів штучного інтелекту, які успішно змагалися проти професійних гравців у грі Dota 2 — складній багатокористувацькій онлайн-баталії. Агенти навчалися, граючи проти себе у швидко симульованому середовищі, що дало їм змогу накопичити тисячі років ігрового досвіду за короткий час.

Ключові аспекти цього досягнення такі. OpenAI Five демонстрували здатність до командної роботи, координуючи дії між п'ятьма агентами без явного обміну інформацією. Вони могли швидко реагувати на стратегії суперників і змінювати свою тактику в реальному часі.

Цей проєкт підкреслив можливість створення агентів, здатних до складної співпраці та стратегії в динамічному середовищі.

Дослідницькі середовища, такі як *AI2-THOR*, *CARLA*, *MuJoCo*, *AI Habitat* та *ALFRED*, слугують платформами для тестування та імплементації агентів. Вони дають змогу моделювати конкретні життєві ситуації та взаємодію з об'єктами, забезпечуючи зручний інтерфейс для навчання агентів. Однак їхні можливості обмежені специфічними сценаріями, що створює виклики для генералізації навичок агентів до реальних умов.

Кінцевою метою розроблення узагальненого агента є його інтеграція у роботизовані системи для роботи у реальному світі. Попри складність цього завдання, вже існують роботи, які демонструють успішний перенос моделей, навчених у симульованих середовищах, до фізичних систем. Це підтверджує потенціал застосування подібних агентів у робототехніці, зокрема для виконання складних завдань у неконтрольованих умовах реального світу.

Google DeepMind, спираючись на багаторічний досвід, представила нового агента — Scalable, Instructable, Multiworld Agent (SIMA) [3]. Цей проєкт є важливим етапом у напрямі розвитку загального штучного інтелекту, відображаючи прогрес, досягнутий у галузі за останні роки.

У цій статті викладено висновки аналізу універсального агента SIMA, який продемонстрував значний прогрес у перенесенні навичок між різними середовищами та zero-shot навички, що дають змогу виконувати базові дії в нових середовищах без попереднього навчання.

SIMA

Основною метою проєкту SIMA є створення агента, здатного виконувати довільні мовні інструкції для здійснення дій у будь-якому віртуальному тривимірному середовищі, використовуючи клавіатуру та мишку. До таких середовищ належать як спеціально розроблені дослідницькі платформи, так і широкий спектр комерційних відеоігор. SIMA прагне подолати розрив між символами мови та їхніми зовнішніми референтами, об'єднуючи абстрактне розуміння мови зі втіленим сприйняттям і дією в масштабованих середовищах.

Серед досягнень SIMA — створення агента, який може виконувати короткострокові завдання на основі мовних інструкцій у більш ніж десяти різноманітних 3D-середовищах. Для оцінювання продуктивності SIMA розроблено кілька методів, зокрема оптичне розпізнавання символів (OCR) для виявлення тексту, що сигналізує про завершення завдань, а також людське оцінювання записаних відео з діями агента. Використання єдиного інтерфейсу для різних середовищ дає агенту змогу працювати без додаткового налаштування під нові умови, що є кроком у напрямі створення універсального агента, здатного виконувати широкий спектр дій, доступних людині, у симульованих 3D-середовищах.

Агент SIMA вирізняється серед інших тим, що сфокусований на роботі з природною мовою у різноманітних середовищах, які відрізняються як візуальним оформленням, так і доступними діями. Це допомагає агенту демонструвати високу адаптивність та здатність до генералізації знань у різних контекстах.

Для забезпечення різноманітності досвіду агента було використано набір віртуальних середовищ, що передбачають як симуляції реального світу, так і міфічні чи науково-фантастичні сценарії [3, р. 6]. Такі середовища створюють умови, які хоча і є уявними, але мають зв'язок із реальними задачами. У дослідженні застосовували як комерційні відеоігри, так і спеціалізовані дослідницькі платформи.

Комерційні відеоігри забезпечують візуально насичені світи з широкими можливостями для складних взаємодій. У рамках проєкту вибирали ігри без насильства, із завданнями, які наближені до реальних умов: *Goat Simulator 3*, *Hydroner*, *No Man's Sky*, *Satisfactory*, *Teardown*, *Valheim*, *Wobbly Life*. У цих іграх агентам необхідно було виконувати різноманітні завдання, як-от збирання ресурсів, будівництво споруд, планування дій, орієнтація у просторі та створення складних структур (наприклад, фабрик або транспортних систем).

Дослідницькі платформи створені для цілеспрямованого навчання ШІ та пропонують контрольовані умови. Вони забезпечують більш точну та швидко валідацію результатів, а також підтримують

реалістичну фізичну взаємодію. Були використані середовища *Construction Lab*, *Playhouse*, *Proctor*, *WorldLab*, які моделюють буденні сценарії, наприклад орієнтацію в приміщенні, пошук об'єктів, прибирання або приготування їжі.

Завдання, які виконує SIMA в зазначених середовищах, мають такі характеристики: висока невизначеність і відсутність чітких критеріїв оптимального рішення; велика розмірність простору можливих дій; відсутність алгоритмів, які б гарантували завжди правильний результат. Ці задачі вважають інтелектуальними, оскільки вони потребують від агента здатності до адаптації, аналізу і прийняття рішень у складних та змінних умовах.

Тренування агента SIMA здійснювали за допомогою навчання з учителем, використовуючи записи ігрових сесій реальних людей, доповнені метаданими. При цьому агент отримував лише зображення, рендерене (візуалізує середовище через віртуальну камеру) грою (таке саме, яке бачить звичайний гравець), а також інструкцію, задану природною мовою, яку необхідно виконати. Для забезпечення рівномірного навчання команди були вручну кластеризовані за логічними категоріями дій.

Агент SIMA перетворює візуальні спостереження та інструкції, задані природною мовою, на дії за допомогою клавіатури та миші. Зважаючи на високу розмірність вхідних і вихідних даних, агента було спроектовано так, щоб обчислювати відповідь менш ніж за 10 секунд.

У системі використовують попередньо навчені моделі, які були донавчені для специфічних завдань: для оброблення пар «зображення — текст», які створюють загальні представлення вхідних даних; здатні прогнозувати послідовності зображень, що допомагає в аналізі динамічних сценаріїв.

Окрім цього, було розроблено з нуля моделі-трансформери. Їхні результати передаються до іншої нейронної мережі, яка відповідає за перетворення вихідних даних у конкретні дії клавіатури та миші. Основним методом навчання було повторення поведінки на основі записів дій людини.

Для підвищення відповідності виконаних дій інструкціям було застосовано метод Classifier-Free Guidance, який посилює мовну умовність і забезпечує більшу точність виконання завдань. Маючи функцію полісі π , нове значення обчислюватиметься за такою формулою:

$$\pi_{CFG} = \pi(image, language) + \lambda (\pi(image, language) - \pi(image, \cdot)).$$

Складність оцінювання агента SIMA полягає у відсутності загальних критеріїв успіху для виконання мовних інструкцій, складному доступі до даних середовища у комерційних іграх та високій вартості завантаження завдань. Для забезпечення надійності використовують різноманітні методи, які перевіряють здатність агента дотримуватися мовних інструкцій, а не лише взаємодіяти із середовищем.

Основні методи оцінювання ефективності агента такі: логарифмічні ймовірності дій, статичне візуальне введення, завдання з використанням реальних середовищ, оптичне розпізнавання символів (OCR), оцінювання людиною [3, р. 11].

Логарифмічні ймовірності дій базуються на основі прогнозів агента за новими даними. Така оцінка слабо корелює з реальними навичками, тому потрібне онлайн-оцінювання. Статичне візуальне введення використовує зображення для оцінювання простих дій без запуску гри. Воно швидко надає базову оцінку, але не підходить для складних завдань. Завдання з використанням реальних середовищ ґрунтуються на точному тестуванні виконання мовних інструкцій із контролем точності та реакції на зміну інструкцій. Оптичне розпізнавання символів (OCR) використовує текстові повідомлення у грі для автоматизованого оцінювання, особливо у комерційних іграх (наприклад, *No Man's Sky*, *Valheim*). Воно обмежено завданнями з текстовими сповіщеннями. Експертний аналіз відеозаписів дій агента людиною є надійним, але ресурсозатратним. Завдання маркується як провалене, якщо агент виконував нерелевантні дії.

Ці методи забезпечують усебічне оцінювання здатності агента виконувати завдання за мовними інструкціями в різноманітних віртуальних середовищах, сприяючи досягненню довгострокової мети створення універсального ШІ.

Оскільки в останні роки з'являється все більше побоювань та обмежень щодо використання ШІ, то SIMA теж було оцінено на відповідність Google AI Principles (<https://ai.google/responsibility/principles/>) та визначено, що потенційний виграш перевищує ризики.

Порівняльні результати

Найперше розглянемо продуктивність агента SIMA у семи середовищах, запозичену із [3, р. 16].

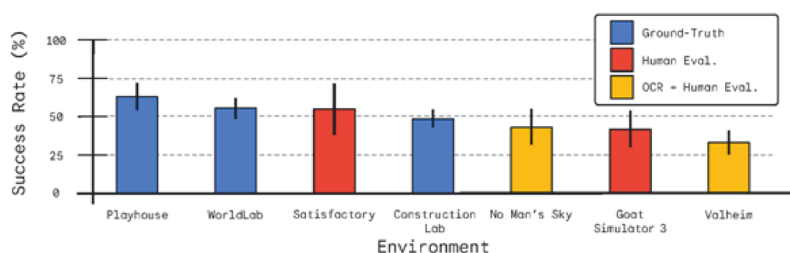


Рис. 1. Середня продуктивність агента SIMA у семи середовищах, для яких визначено кількісну оцінку результату

Результати (рис. 1) [3, р. 16] показують, що агент SIMA здатен виконувати широкий спектр завдань у різноманітних середовищах, хоча і залишається чималий простір для покращень. Варто зазначити, що продуктивність у середовищах Playhouse та WorldLab значно вища, бо самі середовища є порівняно простішими.

Зрозуміло, що важливим є і оцінювання вміння агента узагальнювати. Спочатку наведемо порівняння різних моделей, описане в [3, р. 16, 18] (рис. 2). Столпчики відображають продуктивність агента SIMA, базових моделей та їхніх модифікацій порівняно з продуктивністю спеціалізованих агентів у середовищах (нормалізовано до 100 %). Суцільна лінія показує нормалізовану продуктивність спеціалізованих агентів.

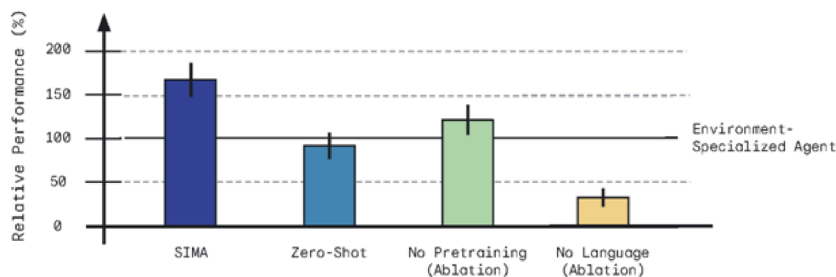


Рис. 2. Порівняння на різних моделях

Порівняння основного агента SIMA здійснено з кількома базовими моделями та модифікаціями, як сукупно, так і в різних середовищах. Результати за кожним з оцінюваних агентів такі:

- *SIMA* — основний агент, навчений на всіх середовищах, окрім Hydroneer та Wobbly Life, які використовують для якісного оцінювання без додаткового навчання (zero-shot);
- *Zero-shot* — окремі агенти SIMA, кожен з яких навчений на $N - 1$ середовищах і оцінюється на тому, на якому не відбувалося навчання, щоб оцінити здатність до перенесення знань. Ці агенти використовують ті самі попередньо навчені енкодери, що й основний SIMA, окрім випадку з Goat Simulator 3, де результати переносу є незалежними;
- *без попереднього навчання* — агент без попередньо навчених енкодерів, замінений на ResNet-модель для оброблення зображень, яка навчається з нуля. Це дає змогу перевірити, як попередньо навчені моделі впливають на продуктивність агента;
- *без мовного компонента* — агент без мовних вхідних даних під час навчання й оцінювання, що дає змогу оцінити, наскільки продуктивність агента пояснюється не мовою, а іншими поведінковими принципами;
- *спеціалізований для середовища* — експертний агент, навчений виключно для кожного середовища з використанням доступних даних і попередньо навчених енкодерів. Продуктивність інших агентів порівнюють із експертним агентом, щоб оцінити максимальні можливості в кожному середовищі.

Основний агент SIMA демонстрував на 67 % кращу продуктивність, ніж спеціалізовані агенти, що свідчить про позитивний перенос навичок між середовищами. Пермутаційні тести підтверджують, що SIMA значно перевершує спеціалізованих агентів у кожному середовищі (p -рівні: 0.001, 0.002, 0.036, 0.0002, 0.008, 0.004, 0.0002).

SIMA також перевищує базові моделі. Агент без попереднього навчання показав значно гірші результати, підтверджуючи, що знання на великій базі підтримують навчання (хоча різниця варіюється за середовищами). Модель без мовного компонента показала низьку продуктивність ($p < 0.001$), що свідчить про те, що агент використовує мовні інструкції, а не лише ймовірні поведінкові шаблони.

Оцінки zero-shot показали обнадійливі результати. Навіть без тренування в конкретному середовищі агент демонстрував високу продуктивність на загальних завданнях, хоча поступався в середовищно-специфічних навичках. Агенти zero-shot здатні виконувати загальні навігаційні завдання, що трапляються в багатьох іграх (наприклад, «спустиись із пагорба»), та демонстрували складніші вміння, як-от взаємодія з об'єктом за кольором, використовуючи постійність кольорів між іграми та загальні шаблони взаємодії (наприклад, використання лівої кнопки миші для взаємодії з об'єктами).

Важливо, що навіть у середовищі Goat Simulator 3, де агенти не отримували візуального донання, агент zero-shot працював на рівні зі спеціалізованим агентом. Це вказує на те, що перенос знань не обмежується лише візуальними компонентами.

Обнадійливою є і така оцінка. В роботі [3, р. 19] наведено порівняння результатів основної моделі SIMA із її варіаціями без CFG та без мовного компонента відповідно. Результати порівняння ілюструє рис. 3 [3, р. 19].

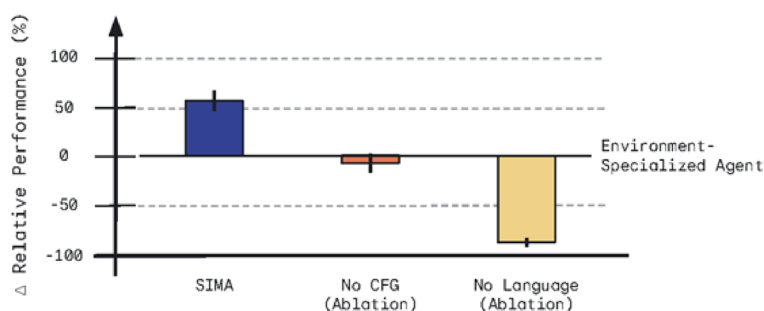


Рис. 3. Порівняння результатів основної моделі SIMA із її варіаціями без CFG та без мовного компонента відповідно

Порівняно продуктивність агентів із використанням і без використання classifier-free guidance на підмножині дослідницьких середовищ: Construction Lab, Playhouse та WorldLab. Без CFG ($\lambda = 0$) агент SIMA працює помітно гірше. Проте навіть без CFG агент демонструє високий рівень мовної умовності, значно перевершуючи базову модель без мовного компонента. Ці результати підкреслюють переваги CFG та вплив втручань під час інференсу на керованість агента.

Зрозуміло, що варто порівняти агента і з продуктивністю людини. У роботі [3, р. 2] наведено результат такого порівняння (рис. 4). Хоча люди і показують кращий результат, ніж агенти, проте і їхня точність становить лише 60 %.

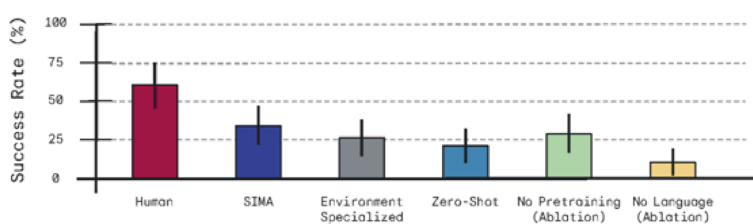


Рис. 4. Порівняння з людиною

Ми вважаємо цей показник дуже важливим, адже завдання, які виконувалися, це не є щось механічне, де машина легко може перевершити людину. Це досить комплексні й нетривіальні завдання (інтелектуальні задачі). Порівняння показало, наскільки насправді агент наблизився до людської продуктивності, тобто за цим критерієм результати можна вважати кращими, ніж за абсолютної оцінки, адже продуктивність агента становила приблизно 50 % продуктивності людини.

Далі розглянемо, як проводили цей тест і якого висновку дійшла команда DeepMind [3, р. 20].

Для додаткового порівняння агентів SIMA з людською продуктивністю було проведено оцінювання на додатковому наборі завдань у грі No Man's Sky. Завдання варіювалися за складністю: від простих інструкцій на кшталт «іди вперед» до складніших, як-от «використай аналізатор для ідентифікації нових тварин». Людські учасники були досвідченими гравцями, які брали участь у збиранні даних. Їхню продуктивність оцінювали ті самі судді й за тією самою методикою, що й для агентів; судді не знали, чиї дії оцінюють — людей чи агента.

Результати, подані на рис. 4, показують, що люди успішно виконали лише 60 % завдань, що свідчить про їхню складність і суворість критеріїв оцінювання. Деякі помилки гравців виникали через зайві дії перед виконанням завдання, наприклад вони відкривали меню корабля за інструкції «заряди лазер для видобутку». Незважаючи на складність, агент SIMA досяг 34 % успіху, значно перевершуючи базового агента без мовного компонента (11 % успіху). Зазначено, що 100 % успіху може бути недосяжним через розбіжності в оцінках суддів щодо неоднозначних завдань. Однак залишається значний обсяг роботи, необхідний для досягнення рівня людської продуктивності. Це підкреслює корисність системи SIMA як складного, але інформативного методу оцінки мовної взаємодії в агентів із втіленням.

OpenAI Five та AlphaStar досягли значного успіху, перемігши професійних гравців у іграх-стратегіях Dota 2 та Starcraft II відповідно. Хоча SIMA ще не досяг такого рівня, але все ж цей проєкт зосереджений на універсальності та роботі з природною мовою, і за цими параметрами SIMA показує наразі найкращі результати.

Подальші кроки

SIMA — проєкт, що перебуває в процесі розвитку. У роботі [3, р. 20] описано цілі та філософію SIMA, а також представлено попередні результати, які демонструють здатність агента пов'язувати мовні інструкції з поведінкою в різних насичених 3D-середовищах. Спостерігається помітна продуктивність і ранні ознаки переносу навичок між середовищами, зокрема zero-shot перенесення базових навичок на невідомі середовища. Проте багато навичок і завдань залишаються недосяжними.

Майбутні дослідження передбачають:

- масштабування до більшої кількості середовищ і наборів даних, розширюючи портфоліо ігор та середовищ;
- підвищення стійкості та керованості агентів;
- використання все якісніших попередньо навчених моделей;
- розроблення більш комплексних і ретельно контрольованих оцінювань.

Очікують, що це зробить SIMA ідеальною платформою для проведення передових досліджень із безпечного поєднання мови та попередньо навчених моделей у складних середовищах і допоможе вирішити фундаментальну проблему AGI (Artificial General Intelligence). Дослідження також має потенціал збагачення навчального досвіду та середовищ розгортання майбутніх моделей; одна з цілей — пов'язати абстрактні можливості великих мовних моделей зі втіленими середовищами. Є надія, що SIMA допоможе подолати фундаментальну проблему зв'язку мови зі сприйняттям і дією в масштабі.

Висновок

SIMA продемонстрував значний прогрес у перенесенні навичок між різними середовищами та zero-shot навички, що дають змогу виконувати базові дії в нових середовищах без попереднього навчання. Незважаючи на досягнуті успіхи, багато завдань залишаються недоступними для агента, що підкреслює необхідність подальших досліджень. У підсумку, SIMA прокладає шлях до створення більш універсального та безпечного інтелектуального агента, що може розв'язувати складні завдання в симульованих середовищах і відкриває перспективи для майбутнього розвитку загального штучного інтелекту.

На прикладі SIMA ми бачимо, що тема універсального агента III актуальна, над нею працюють і є певні здобутки. Хоча до досягнення людського рівня навіть у обмежених віртуальних середовищах ще треба провести чимало досліджень, і також не забуваймо про величезні обчислювальні потужності, які для цього необхідні.

Список літератури

1. Berner C. Dota 2 with Large Scale Deep Reinforcement Learning / C. Berner et al. — 2019. — <https://doi.org/10.48550/arXiv.1912.06680>.
2. Mathieu M. AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning / M. Mathieu et al. — 2023. — <https://doi.org/10.48550/arXiv.2308.03526>.
3. Team S. Scaling Instructable Agents Across Many Simulated Worlds / S. Team et al. — 2024. — <https://doi.org/10.48550/arXiv.2404.10179>.

References

- Berner, C., et al. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. <https://doi.org/10.48550/arXiv.1912.06680>.
Mathieu, M., et al. (2023). *AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning*. <https://doi.org/10.48550/arXiv.2308.03526>.
Team, S., et al. (2024). *Scaling Instructable Agents Across Many Simulated Worlds*. <https://doi.org/10.48550/arXiv.2404.10179>.

M. Glybovets, N. Bachynskyi

A GENERALIST AI AGENT SIMA

Developing a universal artificial intelligence agent, a subset of Artificial General Intelligence (AGI), is one of the most complex challenges in modern science. Such an agent must generalize knowledge, learn new skills without explicit programming, adapt to unfamiliar environments, and make effective decisions. Addressing this challenge requires advancements across technical domains while also navigating ethical and computational constraints.

This paper examines SIMA (Scalable, Instructable, Multiworld Agent), a project by Google DeepMind aimed at creating an agent capable of executing natural language instructions across diverse 3D environments. SIMA operates through a single keyboard-and-mouse interface in both commercial video games and research platforms, making it distinct from task-specific AI systems like OpenAI Five or AlphaStar. It processes visual input akin to what a human player sees and executes commands categorized for balanced skill training. Techniques like Classifier-Free Guidance enhance the agent's ability to align actions with instructions.

SIMA's evaluation combines methods such as OCR for task verification in games, static visual input tests for simple actions, and human evaluations for more nuanced performance metrics. These methods demonstrate SIMA's ability to transfer knowledge and perform tasks across environments, though challenges remain in long-term planning and complex physical interactions. Despite limitations, SIMA represents a foundational step toward AGI by integrating language understanding with embodied actions.

The findings underline SIMA's potential as a scalable platform for autonomous operation in both virtual and real-world settings, offering key insights into bridging language, perception, and action. Future research will focus on expanding its environmental adaptability, improving robustness, and addressing ethical deployment concerns.

Keywords: artificial intelligence, agent, SIMA, virtual environments.

Матеріал надійшов 29.11.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

ФРЕЙМВОРК УПРАВЛІННЯ ЗАГРОЗАМИ ТА РЕАГУВАННЯ НА ІНЦИДЕНТИ ІОТ СИСТЕМИ

У статті розглянуто розроблення та впровадження фреймворку для управління загрозами та реагування на інциденти в системах Інтернету речей (IoT). Запропонований фреймворк поєднує елементи розподіленої архітектури, зокрема Nginx як розподільник навантаження, MQTT брокер HiveMQ, сервер авторизації та збірку сервісів ELK Stack. Це рішення забезпечує безпечну комунікацію IoT пристроїв за допомогою протоколу TLS та механізмів шифрування, автентифікації і авторизації. Особливу увагу надано використанню машинного навчання для виявлення аномалій у реальному часі, що дає змогу ефективно реагувати на потенційні загрози в різних доменах IoT. Фреймворк розроблено з урахуванням обмежених обчислювальних ресурсів IoT пристроїв і вимог до їхньої безпеки.

Ключові слова: Інтернет речей, вразливості безпеки, архітектура IoT, розподільник навантаження Nginx, брокер повідомлень MQTT HiveMQ, збірка сервісів ELK Stack, TLS протокол, протокол MQTT, Logstash, Elasticsearch, Kibana.

Вступ

Інтернет речей розпочав трансформацію нашого цифрового простору, в якому повсякденні об'єкти є взаємопов'язаними та здатні до комунікації між собою. Ця трансформація не тільки спрощує нам життя, а й відкриває дорогу до небачених раніше можливостей та ефективності у сферах розумного будинку, медицини, індустриального виробництва, міського управління тощо [2]. Проте, як і з будь-яким технологічним проривом, IoT (Internet of Things) потребує зваженого та обережного впровадження. Насамперед, масштабне та неконтрольоване впровадження IoT пристроїв провокує багато проблем з безпекою, які необхідно вирішувати.

Основою IoT пристроїв є здатність збирати, оброблювати й передавати інформацію автономно, без людського втручання. Хоча саме це і є головною інновацією, така автономність створює багато вразливостей у безпеці. Наприклад, IoT пристрої часто використовують як приватні, так і публічні мережі, в такий спосіб розширюючи потенційний вектор атаки зловмисника на конфіденційність і цілісність даних. Але це не єдине, про що потрібно турбуватися. Не менш важливими є надійність і доступність критично важливого функціоналу, що його такі прилади надають. Від термостатів і годинників до автономних транспортних засобів та інфраструктури міста, вплив скомпрометованих IoT пристроїв може варіюватися від простої незручності до катастрофічних збоїв, що вплинуть на життя мільйонів людей.

IoT прилади зазвичай мають мало обчислювальної потужності та оперативної пам'яті. Такі обмеження призводять до того, що часто традиційні підходи до безпеки як складні алгоритми шифрування є неможливими у впровадженні. Зважаючи на це і на вибухове зростання кількості IoT приладів у використанні, також збільшується і потенціал для несанкціонованого втручання в роботу систем. За прогнозом Cybersecurity Ventures, кількість IoT пристроїв у світі зросте до 25,1 мільярда до 2025 р. Кожний із них є потенційною точкою входу для зловмисника. Таке значне поширення створює велику, часто погано захищену мережу взаємопов'язаних приладів, яку можуть використати зловмисники для власних цілей: крадіжки даних, DDoS атак тощо.

Більше того, non-patchable (не підлягають виправленню) і forever-day (вічні) вразливості тільки ускладнюють проблему. Багато IoT пристроїв нечасто отримують оновлення прошивки через логістичні проблеми доставки цих оновлень або ж тому, що виробник про них забув і більше не підтримує. Такі вразливості являють собою постійно відкриті двері для зловмисників і тому потребують інноваційного підходу до поточних рішень.

Інфраструктуру навколо IoT пристроїв можливо розділити на чотири рівні [1]: сприйняття (захоплення вузла, імперсоніфікація вузла, атака відтворення, часова атака, атака позбавлення сну), мережі (підслуховування, DoS/DDoS атака, Main-in-The-Middle, оброблення даних (виснаження ресурсів, експлойти), застосунок (експлойти). Отже, основними проблемами для безпеки в IoT є несанкціонований доступ до пристроїв і даних, підслуховування комунікації, маніпуляція комунікації, виснаження ресурсів систем і пристроїв.

Метою цієї статті є опис запропонованого комплексного рішення для захисту IoT систем, представленого у формі спеціального фреймворка. Існує нагальна потреба в надійних і здатних до масштабування рішень. Наш підхід враховує унікальні характеристики IoT екосистем, а саме різноманітність, обмеженість в обчислювальних ресурсах і фізичну природу приладів.

Неоднорідність і велика різноманітність пристроїв інтернету речей створюють значні проблеми для моніторингу, виявлення та реагування. Це означає, що для різних груп пристроїв потенційно може бути потрібен різний підхід для вищезгаданих процесів. Отримання повного логу операцій пристроїв і відкритого мережевого трафіку є складним завданням, що ускладнює виявлення та розуміння масштабу інцидентів. Обмежена потужність обчислювальних ресурсів і кількість постійної пам'яті багатьох пристроїв інтернету речей обмежують обсяг даних, які можна зібрати після інциденту. Крім того, потенційна фізична недоступність деяких пристроїв ускладнює аналіз, якщо є підозра на фізичний характер інциденту порушення безпеки.

Усі IoT системи є суттєво залежними від власної доменної області. Наприклад, методи та підходи до управління приладами в системах розумного міста і системах домашнього будинку будуть суттєво відрізнятися. Ця різниця залежатиме від багатьох причин. Спершу, обчислювальна здатність приладів, кількість оперативної та постійної пам'яті прямо диктують обмеження для всіх інших частин системи. Від цих параметрів залежить як спосіб внутрішньої комунікації між приладами, так і спосіб комунікації з централізованими серверами: чи кожний прилад спілкується напряму один з одним та з сервером, чи існують проміжні вузли, чи прилади покладаються на сервер для комунікації один з одним. А це вже своєю чергою диктує методи та підходи до шифрування [3]. До того ж, останнє, але не за значенням — авторизація та автентифікація [4]. Вони можуть набувати різних форм залежно від обраного підходу до інтеркомунікації та шифрування даних.

Отже, було прийнято рішення розробити систему, що може слугувати фундаментом для конкретних рішень. Запропонований фреймворк надає базовий функціонал, що є легко розширюваним, і відображає наше бачення реалізації безпеки в кінцевій системі.

Архітектура фреймворку

Система складається з таких компонентів (рис. 1): розподільник навантаження Nginx [5], брокер повідомлень MQTT HiveMQ [6], сервер авторизації, сервер збору інформації, збірка сервісів ELK Stack [7].

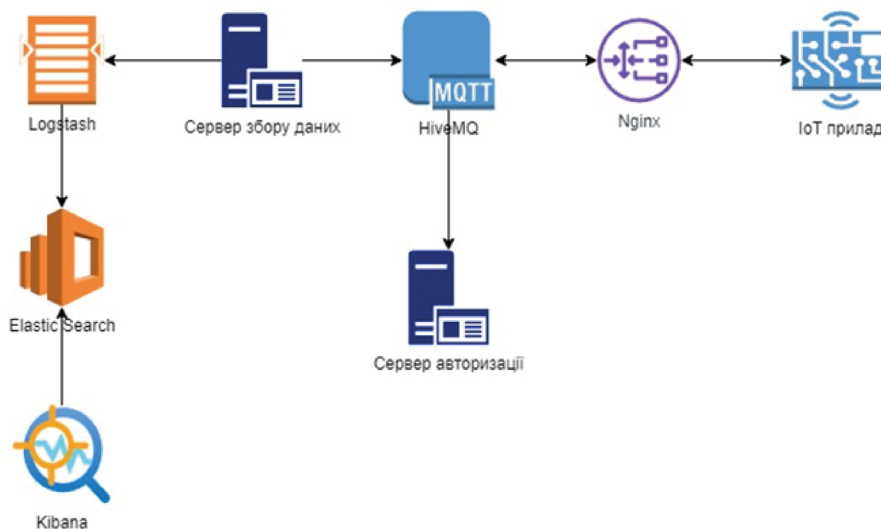


Рис. 1. Архітектура розробленої системи

Nginx

Nginx — це open-source програмне забезпечення для вебсервера, яке використовують для зворотного проксі-сервера, балансування навантаження та кешування. Воно надає можливості HTTPS сервера і розроблене для максимальної продуктивності й стабільності. Nginx також функціонує як проксі-сервер для протоколів електронної пошти, таких як IMAP, POP3 і SMTP.

Nginx використовує master-slave архітектуру, шляхом імплементації асинхронного, неблокуючого та керованого подіями підходу. Воно використовує мультиплексування та сповіщення про події і призначає окремі завдання окремим процесам.

Nginx не створює новий потік виконання для кожного нового з'єднання. Натомість спеціальний майстер-процес обробляє прийом нових запитів із загального сокету прийому даних і запускає високоефективні цикли виконання всередині окремих робочих процесів. Це дає змогу обробляти тисячі з'єднань у межах одного такого процесу. Спеціальних механізмів розподілу з'єднань між різними робочими процесами в Nginx не існує, ця робота виконується на рівні ядра ОС.

Майстер-процес розподіляє запити для робочих процесів відповідно до вимог клієнта. Після того, як роботу буде розподілено між процесами, майстер очікуватиме наступний запит від клієнта, тобто не чекатиме на відповідь від робочого процесу. Після отримання асинхронної відповіді від останнього майстер-процес надішле відповідь клієнту. Також майстер читає і перевіряє конфігурацію сервера шляхом створення, приписування та з'єднання сокетів. Він також обробляє запуск, завершення та підтримку кількості налаштованих робочих процесів. До того ж, він може переналаштовувати робочий процес без переривання оброблення запитів.

Проксі-кеші — це спеціальні процеси. Вони мають завантажувач кешу та менеджер. Завантажувач кешу перевіряє елемент дискового кешу та заповнює in-memory базу даних метаданими кешу. Він готує Nginx до роботи з файлами, які вже зберігаються на диску. Менеджер кешу відповідальний за час життя елементів та інвалідацію елементів у кеші.

У розробленій системі Nginx виконує роль реверс-проксі та слугує точкою входу для IoT приладів до MQTT брокера та точкою входу користувачам до інших частин системи. Такий підхід спрощує роботу з та підтримку TLS сертифікатів [6].

Сервер авторизації

У запропонованій системі цей сервер виконує функції авторизації та автентифікації IoT пристроїв. Сервер розроблено у формі окремого модуля мовою Java для Spring фреймворку.

Spring вважають безпечною, недорогою і гнучкою платформою, яка покращує ефективність написання коду та скорочує загальний час розроблення програми за рахунок ефективного використання системних ресурсів. Spring усуває виснажливу роботу з налаштування, щоб розробники могли зосередитися на написанні бізнес-логіки [139]. Крім того, Spring керує інфраструктурою.

Також Spring надає численні слабко зв'язані модулі для покращення функціональності вебдодатка. Ці модулі, такі як Spring AOP, Spring Security, Spring Session, Spring MVC, Spring Data та інші, можна використовувати окремо або разом, забезпечуючи більшу гнучкість під час розроблення.

Розроблена бібліотека виконує такі дві функції: постачання облікових даних від MQTT брокера для IoT пристроїв, автентифікація та авторизація IoT пристроїв на MQTT брокері.

Облікові дані

Запропонований підхід до роботи з обліковими даними має на меті усунути проблему доставки та ротації їх на IoT пристроях. Для досягнення такого результату потрібно, щоби попередньо виконувалися такі умови:

1. Сервер авторизації має інформацію про серійний номер усіх випущених IoT пристроїв, які потрібно підтримувати.
2. Всі IoT пристрої мають убудованими в прошивку домен та публічні TLS ключі сервера.

Задля виконання першого пункту розробникам потрібно імплементувати та використовувати такий інтерфейс:

```
public interface IoTDeviceCrudService {  
    boolean save(IoTDevice device);  
    boolean deleteBySerialNumber(String serialNumber);  
    Optional<IoTDevice> findBySerialNumber(String serialNumber);  
    Optional<IoTDevice> findByMqttLogin(String mqttLogin);  
}
```

Надається стандартна імплементація, що зберігає дані в оперативній пам'яті. Це можливо використовувати для тестування. Такий підхід дає змогу розробникам зберігати дані про пристрої в будь-якому зручному для них форматі, наприклад у SQL/NoSQL базах даних тощо.

Дані потрібно зберігати в такому форматі:

```
public class IoTDevice {  
    private String serialNumber;  
    private String mqttLogin;  
    private String mqttPassword;  
    private Date mqttCredentialsLastRotation;  
    private boolean blacklisted = false;  
    private Set<String> topics = new HashSet<>();  
}
```

Вважають, що сутність IoTDevice з'явиться в системі до продажу пристроїв клієнтам. Спосіб генерації логіну та паролю залишають на розсуд розробників. Ознака blacklisted відповідає за те, чи дозволено тому чи тому пристрою отримувати нові облікові дані. Список topics відповідає за дозволені для зчитування топіки цього пристрою.

Для IoT пристроїв надають два HTTPS ендпоінти:

- 1) /api/iot/device/credentials – прилад надсилає свій серійний номер, і якщо всі перевірки проходять успішно, сервер відповідає актуальними обліковими даними для цього пристрою;
- 2) /api/iot/device/certificate – прилад запитує актуальний публічний ключ TLS для Nginx/HiveMQ.

Для HiveMQ теж надають два HTTPS ендпоінти, що є захищеними наперед вибраним паролем:

- 1) /api/iot/mqtt/authenticate – брокер надсилає логін та пароль, отриманий від IoT пристрою, та очікує на відповідь про дозвіл на автентифікацію;
- 2) /api/iot/mqtt/authorize – брокер надсилає логін автентифікованого пристрою і топик, із яким пристрій хоче взаємодіяти (писати або читати).

Важливим компонентом безпеки в запропонованій системі є механізм certificate(public key) pinning. Його основна мета — підвищити безпеку з'єднання шляхом зниження ризику атак типу «людина посередині» (MITM) і забезпечення зв'язку клієнта лише з нашим автентичним сервером.

У системі реалізується стандартна перевірка сертифіката. У типовому TLS handshake, коли клієнт під'єднується до сервера, сервер представляє клієнту свій цифровий сертифікат. Потім клієнт перевіряє автентичність сертифіката, зокрема те, що він підписаний довіреним центром сертифікації (CA) і що його термін дії не закінчився або він не був відкликаний. Якщо перевірки проходять успішно, клієнт продовжує безпечне з'єднання.

Ми пропонуємо і закріплення сертифіката. Цей механізм покращує процес встановлення довіри. Замість того, щоб покладатися виключно на систему CA, клієнт має попередньо налаштований список відкритих ключів або сертифікатів, яким він явно довіряє.

Таким чином, у мікропрограми пристроїв має бути перелік попередньо згенерованих публічних ключів серверу авторизації для запобігання атаки MITM. Публічний ключ точки входу до MQTT брокера буде отримано вже від серверу авторизації.

HiveMQ

HiveMQ є брокером повідомлень протоколу MQTT [1410].

Протокол має два типи об'єктів: брокер повідомлень і клієнт. Брокер MQTT функціонує як сервер, який приймає повідомлення від клієнтів і поширює їх для інших клієнтів. Клієнтом MQTT може бути будь-який пристрій, який використовує бібліотеку MQTT і під'єднується до брокера через мережу.

Інформацію в MQTT структуровано як ієрархію топіків. Коли видавець (publisher) має нові дані, він надсилає повідомлення брокеру на відповідний топік. Далі це повідомлення вчитують підписники (subscriber) цієї теми. Видавцеві не потрібно знати, хто саме підписаний на його повідомлення, а підписникам не потрібно знати, звідки приходять повідомлення.

Якщо брокер отримує повідомлення на тему без підписників, він його відхиляє. Однак видавець може позначити повідомлення як збережене, щоб брокер зберіг його для майбутніх підписників. Нові підписники отримують збережене повідомлення відразу після підписки. Збережене повідомлення містить спеціальний прапорець, що вказує про його статус. Підписникам надсилають тільки останнє таке повідомлення, всі попередні ігноруються. До того ж, можливо налаштувати повідомлення за замовчуванням, яке буде надіслано підписникам у разі його несподіваного від'єднання від брокера.

MQTT підтримує передавання даних через TCP, але існує також MQTT-SN [11], що працює через інші транспортні протоколи, такі як UDP або Bluetooth. MQTT надсилає облікові дані під'єднання як звичайний текст, що означає відсутність убудованих заходів безпеки або автентифікації. Для гарантування безпеки можна використовувати TLS.

Існує чотирнадцять типів повідомлень MQTT, які відповідають за різні функції, зокрема під'єднання і від'єднання клієнтів, публікацію та підтвердження отримання даних, а також управління з'єднанням між клієнтом і сервером. Обсяг даних в одному повідомленні може варіюватися від двох байтів до 256 мегабайтів.

Кожний клієнт має вказати QoS – Quality of Service (якість з'єднання). Цей показник класифікується в порядку збільшення накладних витрат:

- щонайбільше один раз – повідомлення надсилається лише один раз, і клієнт і брокер не роблять жодних додаткових кроків для підтвердження доставки;
- принаймні один раз – відправник повторює повідомлення кілька разів, доки не буде отримано підтвердження;
- рівно один раз – відправник і одержувач беруть участь у дворівневому рукоштованні, щоб забезпечити отримання лише однієї копії повідомлення.

QoS не впливає на порядок передавання даних у TCP. Він є надбудовою протоколу MQTT.

Існує чотири основні типи повідомлень в MQTT: CONNECT використовують для надсилання клієнтами запитів на під'єднання до брокера; PUBLISH використовують видавці для надсилання повідомлень брокеру; SUBSCRIBE використовують підписники для отримання повідомлень від брокера; DISCONNECT використовують клієнти для припинення комунікації.

Протокол MQTT має декілька версій, найновішою з яких є MQTT 5.0. Вона вводить декілька нових змін, найбільші з яких: ACK повідомлення повертають коди помилок, об'єднані підписки на топіки, час життя для недоставлених повідомлень і псевдоніми для топіків.

HiveMQ — MQTT брокер і платформа обміну повідомленнями, розроблена мовою Java, яка використовує протокол MQTT для швидкого, надійного та ефективного двонаправленого передання даних до і з пристроїв IoT. HiveMQ надає власну клієнтську бібліотеку, але її можна використовувати з будь-якою клієнтською бібліотекою, сумісною з протоколом MQTT. Він може бути розгорнутий у приватній, гібридній або публічній хмарі. Є можливість інтегрувати HiveMQ з наявними корпоративними системами завдяки його відкритому API та гнучкому розширенню. Також розробники HiveMQ створили інструмент під назвою MQTT CLI, який надає інтерфейс командного рядка для взаємодії з брокерами MQTT, у нашому випадку з HiveMQ. Така утиліта є корисною, бо дає змогу відкрити декілька інстансів одночасно. А це є зручним для тестування.

HiveMQ було обрано через наявність кластеризації та зручні можливості розширення функціоналу брокеру шляхом написання плагінів.

Кластер MQTT брокерів — це розподілена система, яка діє як єдиний логічний брокер MQTT для під'єднаних клієнтів. Як правило, вузли кластера MQTT встановлюють на окремих фізичних або віртуальних машинах і під'єднують через мережу. Такий підхід гарантує, що розгорнута система не матиме єдиної точки збою. В інфраструктурі, де доступність є надзвичайно важливою, кластеризація є необхідним компонентом. Якщо один MQTT брокер у кластері більше не доступний, решта вузлів можуть продовжувати обробляти трафік. У HiveMQ реалізовано архітектуру без майстра, що дозволяє мати необмежену кількість нод.

HiveMQ пропонує Extension SDK для розроблення власних розширень, що можуть виконувати такі функції: перехоплення та маніпуляція повідомленнями MQTT, інтеграція з іншими сервісами,

збір статистики, розширення управління доступом. У запропонованій системі увагу приділяють першій і четвертій функціям.

Імплементуючи інтерфейс SimpleAuthenticator, ми реалізуємо метод onConnect (SimpleAuthInput simpleAuthInput, SimpleAuthOutput simpleAuthOutput). У цьому методі замість простої перевірки ACL файлу, що містить дані про доступи, виконується запит до сервера авторизації для отримання актуальної інформації про статус клієнта, що намагається під'єднатися.

Імплементуючи інтерфейс PublishAuthorizer, ми реалізуємо метод authorizePublish(PublishAuthorizerInput input, PublishAuthorizerOutput output). Знову, замість простої перевірки ACL файлу, виконується запит до сервера авторизації для отримання актуальної інформації про доступні топіки для автентифікованого клієнта.

Дані, отримані за обома методами, можливо кешувати для збільшення швидкодії. Це можливо реалізувати як установленням часу життя на отримані дані, так і більш складними підходами, такими як інвалідація кешу, що буде ініційована сервером авторизації.

Додатково ми імплементуємо інтерфейс PublishInboundInterceptor для дублювання трафіку у визначений топік. Це потрібно для зручного передавання даних до Elastic Search та подальшого їхнього оброблення.

Для забезпечення конфіденційності та цілісності даних використовують TLS.

Сервер збору інформації

Сервер написано мовою Java на платформі Spring з використанням бібліотеки spring-integration-mqtt. Основним завданням сервісу є вичитка даних із MQTT брокера і надсилання їх до Logstash. Додатково, тут можливо робити всі необхідні трансформації та фільтрування.

У поточній реалізації Logstash підіймає TCP сервер і очікує під'єднання від сервера збору інформації. Проте варто зазначити, що таку функцію можливо було б реалізувати інакше. Logstash, як і HiveMQ, надає широкий інструментарій для розширення системи. А саме як інформація надходить, як оброблюється і як надсилається далі. Більше того, вже існує три розширення, які мали б дозволити зчитувати повідомлення напряму з MQTT брокера. Проте останнє оновлення найновішого з них датовано 2018 роком. Після багатьох невдалих спроб інтеграції цих розширень стало зрозуміло, що потрібно розробити щось нове. Розширення для Logstash пишуть мовою Ruby. Задля спрощення розроблення і тестування було прийнято рішення не розроблювати нове розширення для Logstash чи намагатися виправити старі версії, а написати власний сервіс на Spring, що виконуватиме ті самі функції.

ELK Stack

Logstash — це потужний інструмент, який відіграє важливу роль в ELK стеку [1512]. Він дає змогу користувачам збирати, обробляти та аналізувати дані з різних джерел. За своєю суттю Logstash — це пайплайн для даних, який допомагає організаціям отримувати цінну інформацію зі своїх логів шляхом перетворення необроблених даних на формат, який можна легко використовувати та аналізувати. Сильною стороною Logstash є його здатність збирати дані з різноманітних джерел та в різних довільних форматах: файли, журнали подій додатків чи пристроїв, інші джерела даних. Така універсальність дозволяє організаціям збирати всі свої логи чи інші дані в одному місці, що є корисним для визначення тенденцій, виявлення аномалій та розбору проблем.

Logstash також надає зручний механізм фільтрації та конвертації, які дають змогу користувачам видаляти небажані дані, перетворювати дані у відповідний формат і агрегувати їх на основі певних критеріїв. Такий функціонал надає можливість розробникам знаходити корисну інформацію серед великої кількості даних: виявляти вузькі місця чи загрози безпеці тощо.

Гнучкість Logstash також є однією з його найбільших переваг. Завдяки широкому спектру плагінів введення, фільтрування та виводу користувачі можуть налаштувати інструмент відповідно до своїх потреб. Незалежно від того, чи йдеться про інтеграцію Logstash з іншими інструментами стеку ELK, як-от Elasticsearch чи Kibana, чи використання його для надсилання даних до інших місць призначення, як Kafka чи Redis, Logstash забезпечує високий ступінь гнучкості.

У розробленій системі Logstash виконує роль пайплайну до Elasticsearch. Він отримує дані від сервера збору інформації та надсилає їх далі. Також тут можливо скористатися всіма потрібними

плагінами для попереднього оброблення даних, якщо щодо цього є потреба. Використовують такий конфіг:

```
input {
  beats {
    port => 5044
  }

  tcp {
    port => 50000
    codec => json
  }
}

filter {
}

output {
  elasticsearch {
    index => "iot-index"
    hosts => "elasticsearch:9200"
    user => "logstash_internal"
    password => "${LOGSTASH_INTERNAL_PASSWORD}"
  }
}
```

Elasticsearch є потужною пошуковою системою, яка пропонує масштабовані рішення для повнотекстового пошуку [1512]. Інструмент побудовано на принципах розподілених систем, призначених для оброблення великих обсягів даних на кількох вузлах у кластері. Це досягається за допомогою механізму шардингу, коли проіндексовані дані поділяються на менші, більш керовані частини, кожна з яких зберігається на різних вузлах. Це не тільки забезпечує масштабованість, а й підвищує відмовостійкість і доступність завдяки реплікації цих сегментів на різних вузлах у кластері.

Однією з найважливіших особливостей Elasticsearch є його здатність пошуку в реальному часі. На відміну від традиційних систем пошуку, які індексують дані пакетами, Elasticsearch може обробляти потоки даних у міру їх надходження. Тож користувачі можуть виконувати пошук у контексті найновішої інформації, що є особливо корисним для таких застосувань, як аналіз журналів подій, де актуальність даних є важливою для негайного аналізу та прийняття рішень. Мова запитів Elasticsearch одночасно потужна та гнучка. Інструмент надає можливість робити запити у багатьох форматах: пошук за ключовими словами, складними фразами і регулярними виразами. До того ж, існує можливість пошуку за повнотекстовими запитами, які аналізують контекст і семантику термінів. Ще одною великою перевагою використання Elasticsearch є можливість інтеграції з Kibana [1512], що є інструментом візуалізації. Він дає змогу аналізувати отримані дані від пошукової системи в зручному візуальному форматі.

Система підрахунку релевантності є ще одним важливим компонентом. Вона використовує частоту термінів, зворотну частоту документа та довжину поля для ранжування документів. Це робиться для того, щоби користувачі спершу отримували найрелевантніші результати пошуку.

Elasticsearch постачають разом із широким набором API, що дозволяє інтегруватися з будь-якими іншими системами. Це робить його зручним для розробників, які працюють у різних технологічних стеках.

Машинне навчання

Однією з найголовніших причин для приєднання Elasticsearch до розробленої системи є широкі можливості та простота використання інструменту в сфері машинного навчання.

Виявлення аномалій є одним із найпотужніших застосувань машинного навчання в Elasticsearch. Такий механізм дає змогу користувачам знаходити незвичні закономірності або просто незвичності

у своїх даних, які можуть вказувати на потенційні проблеми. Це особливо корисно для виявлення моніторингу безпеки мережі, моніторингу здоров'я життєздатності вузлів чи пристроїв.

Система виявлення аномалій в Elasticsearch працює шляхом постійного аналізу вхідних даних у порівнянні з історичними закономірностями. Задачі машинного навчання використовують методи неконтрольованого навчання на історичних даних, щоб створити базову статистичну модель (z-оцінки / XGBoost) того, що вважається «нормальним». Коли модель навчена, вона може відстежувати нові дані в режимі реального часу, щоб виявляти відхилення. Користувачі можуть налаштувати чутливість системи виявлення, тобто наскільки суворим чи м'яким, на їхню думку, має бути виявлення аномалій.

Найважливішим компонентом виявлення аномалій в Elasticsearch є навчання в режимі реального часу, тобто здатність ML моделей поступово оновлювати свою базу знань у міру появи нових даних без необхідності перенавчання з нуля. Це означає, що з розвитком поведінки користувачів або системних показників ML моделі адаптуються і продовжують забезпечувати точне виявлення аномалій. Навчання в режимі реального часу має вирішальне значення для підтримки актуальності та ефективності системи виявлення аномалій, особливо в динамічних середовищах, де тенденції можуть швидко змінюватися.

Kibana

Kibana є важливою частиною ELK стеку. Вона пропонує набір інструментів, розроблених, щоб допомогти користувачам візуалізувати, досліджувати та взаємодіяти зі своїми даними, що зберігаються в Elasticsearch. Kibana діє як інтерфейс, за допомогою якого величезний набір даних можна перетворити на практичну інформацію, що допомагає компаніям і аналітикам приймати обґрунтовані рішення на основі інформації в реальному часі.

Kibana є потужним інструментом візуалізації даних, який дає можливість користувачам створювати діаграми та графіки, формат яких найкраще відповідає тим чи тим даним (рис. 2). Вона підтримує розширені візуалізації, такі як криві часових рядів, діаграми розсіювання, теплові та геопросторові карти. Візуалізації не статичні, а динамічні та інтерактивні. Поєднуючи все вищезгадане, Kibana може задовольнити будь-які варіанти використання.

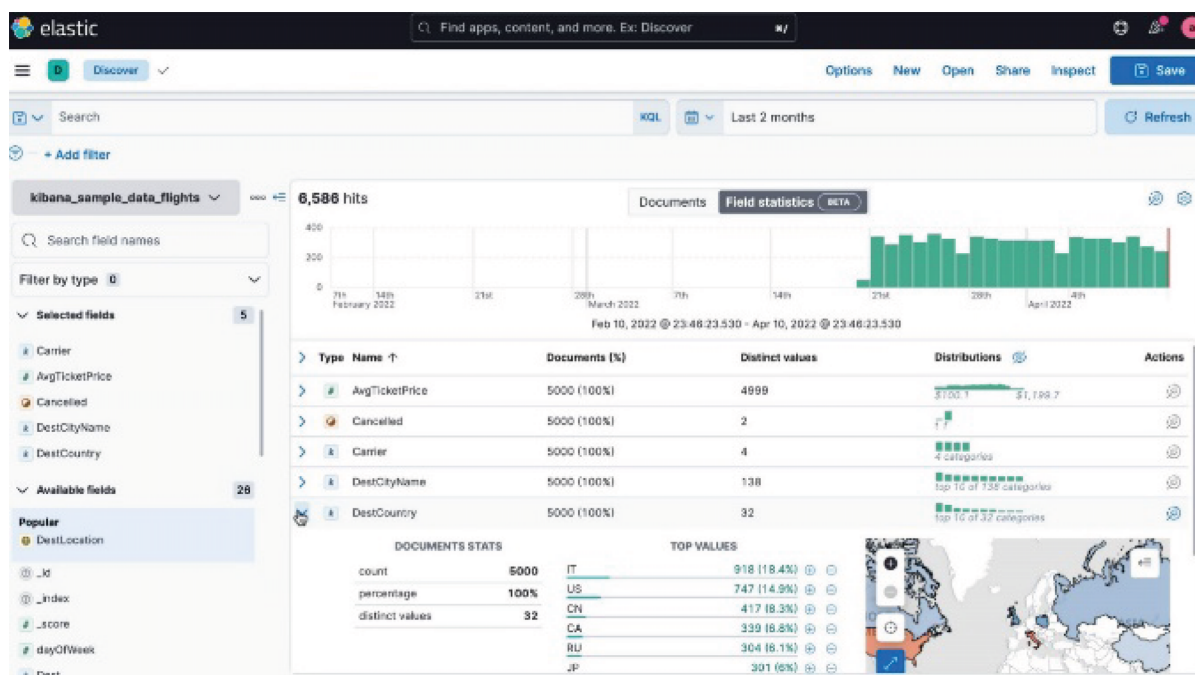


Рис. 2. Приклад сконфігурованого інтерфейсу Kibana

Однією з видатних особливостей Kibana є її здатність подавати складні дані інтуїтивно зрозумілим способом. Використовуючи пошукові можливості Elasticsearch, Kibana може миттєво фільтрувати мільйони записів для відображення найбільш релевантних результатів. Ця інтерактивність у ре-

жимі реального часу має вирішальне значення для моніторингу систем, відстеження логів подій або аналізу моделей поведінки користувачів у міру їх виникнення. Більше того, Kibana має бездоганну інтеграцію з Elasticsearch.

Науковці та дата-аналітики можуть використовувати Kibana для дослідження індексованих даних безпосередньо з Elasticsearch, застосовуючи фільтри, запити та агрегації, не виходячи з інтерфейсу візуалізації. Такий тісний зв'язок гарантує, що користувачі завжди працюють з актуальною інформацією, оскільки запити Kibana виконуються з живими даними в режимі реального часу.

Гнучкість платформи є ще одним ключовим активом. Kibana дає змогу користувачам налаштовувати свої інформаційні панелі, а саме: обирати візуалізації та впорядковувати їх у макеті. Такий підхід дозволяє користувачам підлаштовувати інструмент під свою доменну область та робочий процес. Цими інформаційними панелями можна ділитися з членами команди або взагалі будь з ким, якщо на це є потреба.

На додаток до всього згаданого вище, Kibana надає набір інструментів для керування кластерами Elasticsearch. Користувачі можуть контролювати працездатність систем, керувати кластерами та оптимізувати розподіл ресурсів. Цей аспект Kibana особливо цінний для DevOps команд, яким необхідно забезпечити безперебійну роботу своєї інфраструктури.

Наприклад, на рис. 3 наведено екран огляду завдань з виявлення аномалій.

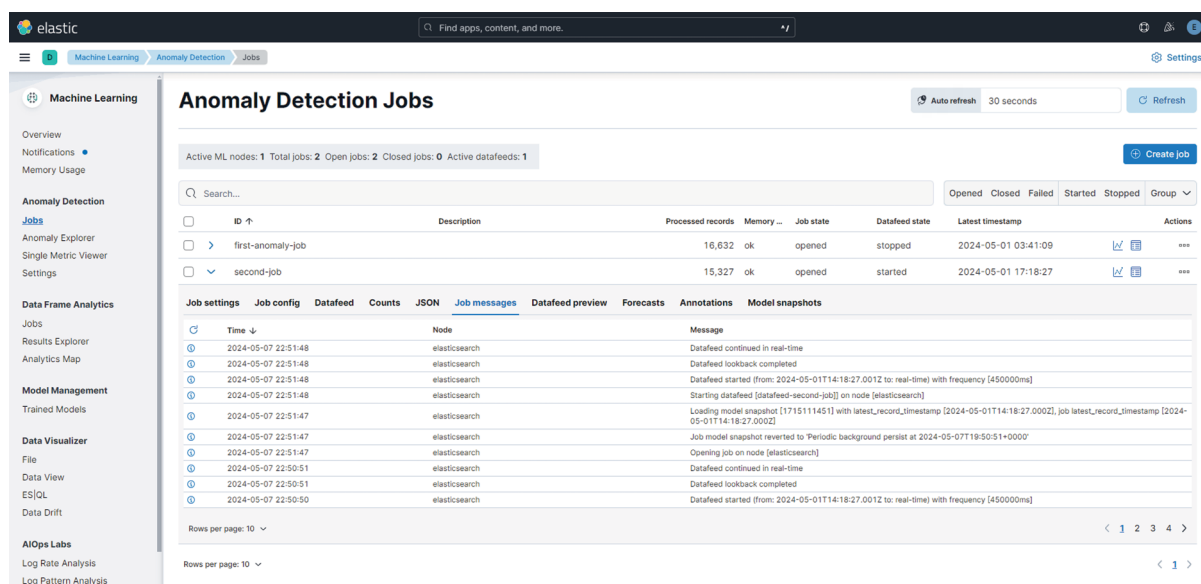


Рис. 3. Екран огляду завдань із виявлення аномалій

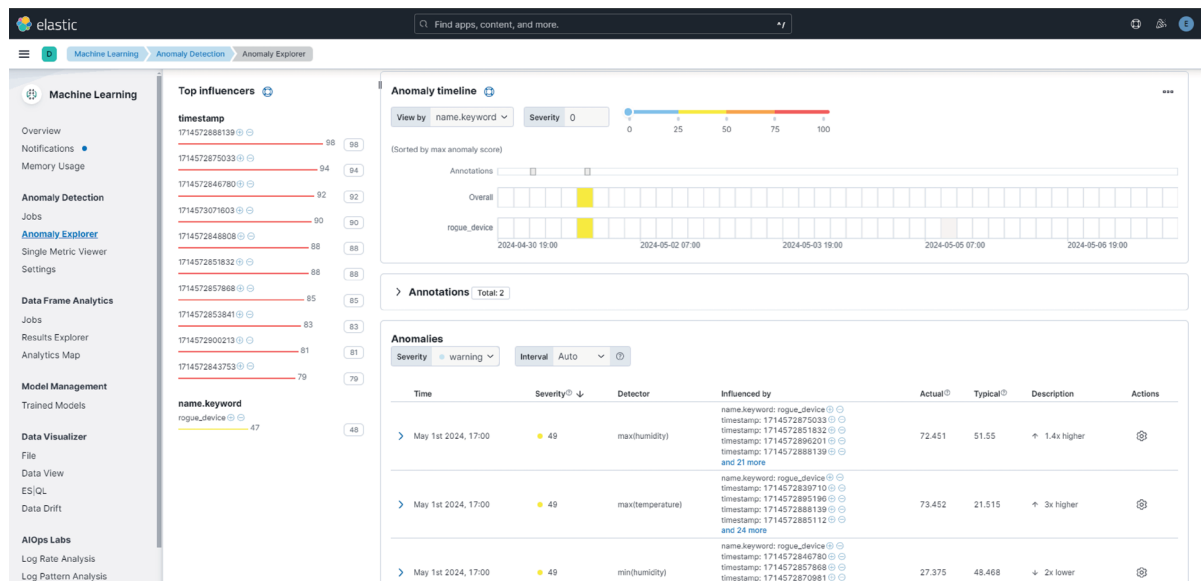


Рис. 4. Екран огляду виявлених аномалій

Як було вже згадано, Kibana пропонує зручний інтерфейс для взаємодії з Elasticsearch, в тому числі роботу з ML моделями.

Для демонстрації було налаштовано і натреновано задачу виявлення аномалій (рис. 4). В цьому випадку у роботі брали участь два змодельовані пристрої, які надсилали інформацію зі своїх зчитувачів, а саме: температуру навколишнього середовища та вологість повітря. Спочатку робочий пристрій надіслав 15 000 пакетів з консистентними та очікуваними даними, на яких і було натреновано модель. Далі, скомпрометований пристрій почав надсилати дані, які не відповідали заданому тренду, а саме набагато завищені показниками температури та вологості. Elasticsearch відразу позначив такі пакети як аномальні.

Рис. 5. Встановлення дій, які потрібно виконати після виявлення аномалій

Крім простої візуалізації можливо додати правила сповіщення на випадок виявлення аномалій. У межах нашої системи пропонується налаштовувати вебхуки, які будуть сповіщати про скомпрометовані пристрої відповідний сервер. Той, своєю чергою, робитиме відповідні кроки, наприклад, відмовлятиме приладу в комунікації до з'ясування проблеми тощо.

Висновок

Запропонована система забезпечує фундамент для побудови різноманітних рішень в сфері IoT, враховуючи унікальність IoT систем та їхніх доменних областей. Розроблений фреймворк пропонує базовий функціонал, який можна легко розширювати для задоволення конкретних вимог.

Основні компоненти системи – розподільник навантаження Nginx, брокер повідомлень MQTT HiveMQ, сервер авторизації, сервер збору інформації та збірка сервісів ELK Stack. Разом вони формують надійну та масштабовану архітектуру. Ці елементи об'єднуються для забезпечення ефективної та безпечної комунікації. TLS протокол пропонує широку гнучкість для вибору конкретних криптографічних бандлів, що найбільше підійдуть для конкретних ситуацій. Такий підхід дає змогу пристроям IoT безпечно взаємодіяти між собою та із сервером, використовуючи ефективні методи шифрування, автентифікації та авторизації.

Додатково, в системі використовується машинне навчання для навчання моделей і виявлення аномалій у реальному часі. Система може стати основою для розроблення кінцевих рішень, що враховують унікальні потреби та обмеження кожного домену.

Розроблена система використовує TLS як основний протокол шифрування. Рекомендують застосовувати ECDHE, проте це все ще залишається на розсуд кінцевих розробників, які будуть використовувати систему. На додаток, між приладами та сервером рекомендують використовувати certificate pinning, що дасть змогу убезпечитися від компрометації центрів довіри.

Унікальні характеристики IoT екосистем потребують спеціального підходу до управління загрозами та реагування на інциденти. Розуміючи конкретні загрози, з якими стикаються пристрої та мережі IoT, і запроваджуючи систему реагування на інциденти, адаптовану до цих проблем, організації можуть підвищити свою стійкість проти кібератак. Це передбачає не лише технічні заходи, а й організаційну готовність, зокрема навчання, планування та аналіз після інциденту.

Список літератури

1. Глибовець А. М. Агентно-базовані програмні системи пошуку та аналізу інформації / А. М. Глибовець. — Київ : Видавничий дім «Києво-Могилянська академія», 2019. — 281 с. : іл.
2. Burhan M. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey / M. Burhan, R. Rehman, B. Khan, B.-S. Kim // *Sensors*. — 2018. — Vol. 18, no. 9. — P. 2796. — <https://doi.org/10.3390/s18092796>.
3. Computationally simple, lightweight replacement for SSL/TLS [Electronic resource]. Information Security Stack Exchange, Apr. 2011. Mode of access: <https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls>.
4. Difference TLS Vs DTLS protocol TLS [Electronic resource]. The Network DNA, Nov. 2022. Mode of access: <https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html>.
5. Echeverría S. Authentication and Authorization for IoT Devices in Disadvantaged Environments / S. Echeverría, G. A. Lewis, D. Klinedinst, L. Seitz // 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). — Limerick, Ireland, 2019. — Pp. 368–373. — <https://doi.org/10.1109/WF-IoT.2019.8767192>.
6. Grigorik I. 4. Transport Layer Security (TLS) — High Performance Browser Networking [Electronic resource] / I. Grigorik. — Mode of access: <https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html> (date of access: May 26, 2024).
7. HiveMQ — Enterprise ready MQTT to move your IoT data [Electronic resource]. — Mode of access: <https://www.hivemq.com/>.
8. Horovits D. The Complete Guide to the ELK Stack [Electronic resource] / D. Horovits // Logz.io, Feb. 08, 2023. — Mode of access: <https://logz.io/learn/complete-guide-elk-stack/#what-elk-stack>.
9. Kim H. Authentication and Authorization for the Internet of Things / H. Kim, E. A. Lee // *IT Professional*. — 2017. — Vol. 19, no. 5. — Pp. 27–33. — <https://doi.org/10.1109/mitp.2017.3680960>.
10. Roy S. A Lightweight Cellular Automata Based Encryption Technique for IoT Applications / S. Roy, U. Rawat, J. Karjee // *IEEE Access*. — 2019. — Vol. 7. — Pp. 39782–39793. — <https://doi.org/10.1109/ACCESS.2019.2906326>.
11. Shahzad M. Continuous Authentication and Authorization for the Internet of Things / M. Shahzad, M. P. Singh // *IEEE Internet Computing*. — 2017. — Vol. 21, no. 2. — Pp. 86–90. — <https://doi.org/10.1109/MIC.2017.33>.
12. Spring Framework Overview: Spring Framework [Electronic resource]. — Mode of access: <https://docs.spring.io/spring-framework/reference/overview.html>.

References

- Burhan, M., Rehman, R., Khan, B., & Kim, B.-S. (2018). IoT Elements, Layered Architectures, and Security Issues: A Comprehensive Survey. *Sensors*, 18 (9), 2796. <https://doi.org/10.3390/s18092796>.
- Computationally simple lightweight replacement for SSL/TLS. (2011). *Information Security Stack Exchange*. <https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls>.
- Difference TLS Vs DTLS protocol. (2022). *The Network DNA*. <https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html>.
- Echeverría, S., Lewis, G. A., Klinedinst, D., & Seitz, L. (2019). Authentication and Authorization for IoT Devices in Disadvantaged Environments. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, 368–373. <https://doi.org/10.1109/WF-IoT.2019.8767192>.
- Grigorik, I. (2013). *Transport Layer Security (TLS) — High Performance Browser Networking*. O'Reilly. <https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html>.
- HiveMQ — Enterprise ready MQTT to move your IoT data. (n.d.). <https://www.hivemq.com/>.
- Hlybovets, A. M. (2019). *Agent-based software systems for information search and analysis*. Publishing House “Kyiv-Mohyla Academy”.
- Horovits, D. (2023). The Complete Guide to the ELK Stack. *Logz.io*. <https://logz.io/learn/complete-guide-elk-stack/>.
- Kim, H., & Lee, E. A. (2017). Authentication and Authorization for the Internet of Things. *IT Professional*, 19 (5), 27–33. <https://doi.org/10.1109/mitp.2017.3680960>.
- Roy, S., Rawat, U., & Karjee, J. (2019). A Lightweight Cellular Automata Based Encryption Technique for IoT Applications. *IEEE Access*, 7, 39782–39793. <https://doi.org/10.1109/ACCESS.2019.2906326>.
- Shahzad, M., & Singh, M. P. (2017). Continuous Authentication and Authorization for the Internet of Things. *IEEE Internet Computing*, 21 (2), 86–90. <https://doi.org/10.1109/MIC.2017.33>.
- Spring Framework Overview: Spring Framework. (n.d.). <https://docs.spring.io/spring-framework/reference/overview.html>.

S. Shcherbyna, T. Babych

FRAMEWORK FOR THREAT MANAGEMENT AND INCIDENT RESPONSE IN IOT SYSTEMS

The article presents the development and implementation of a framework for managing threats and responding to incidents in Internet of Things (IoT) systems. The proposed framework integrates elements of a distributed architecture, including Nginx as a load balancer; the MQTT HiveMQ broker; an authorization server; and the ELK Stack for data analysis and visualization. This solution ensures secure communication

between IoT devices using the TLS protocol and employs advanced mechanisms for encryption, authentication, and authorization. Particular attention is paid to leveraging machine learning for real-time anomaly detection, which enables effective responses to potential threats in various IoT domains. The framework is designed to accommodate the computational constraints of IoT devices while meeting stringent security requirements.

The importance of IoT lies in its ability to autonomously collect, process, and transmit information without human intervention. However, this autonomy introduces several security vulnerabilities. IoT devices, often operating within public and private networks, increase the attack surface for malicious actors targeting data confidentiality, integrity, and availability. With an estimated 25.1 billion IoT devices expected by 2025, each device represents a potential entry point for cyber threats. Issues like unpatchable vulnerabilities and outdated firmware exacerbate security risks, highlighting the need for innovative solutions.

The proposed framework addresses these challenges by establishing a modular and scalable architecture tailored to the diverse and resource-constrained nature of IoT ecosystems. Components such as Nginx, HiveMQ, and the ELK Stack enable reliable communication and data analysis. Nginx serves as a reverse proxy and an entry point, simplifying TLS certificate management and load balancing. HiveMQ, selected for its extensibility and clustering capabilities, acts as a message broker that facilitates efficient and secure data exchange. The ELK Stack, comprising Logstash, Elasticsearch, and Kibana, provides a comprehensive pipeline for real-time data ingestion, processing, and visualization.

A key feature of the framework is its integration of machine learning models for anomaly detection. These models, trained on historical data, monitor real-time metrics to identify deviations from normal patterns. This capability is crucial for detecting potential security breaches and irregular operations. Moreover, the system employs certificate pinning and other cryptographic measures to protect against Man-in-the-Middle (MITM) attacks and ensure secure device-server interactions.

The framework's modularity allows for customization across specific IoT domains, such as smart cities, healthcare, and industrial IoT. By providing foundational functionality, the framework facilitates the development of domain-specific solutions that address unique challenges while ensuring scalability and security.

In conclusion, the proposed framework represents a comprehensive approach to managing IoT threats and responding to incidents. By integrating secure communication protocols, machine learning-driven anomaly detection, and a modular architecture, it lays the groundwork for reliable and adaptive IoT security solutions.

Keywords: IoT security vulnerabilities, IoT architecture, Nginx load balancer, MQTT HiveMQ broker, ELK Stack services, TLS protocol, MQTT protocol, Logstash, Elasticsearch, Kibana, anomaly detection, threat management, incident response.

Матеріал надійшов 22.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Глибовець А. М., Щербина С. С., Кирієнко О. В.

ВРАЗЛИВОСТІ БЕЗПЕКИ ТА РІШЕННЯ ДЛЯ ЗАХИСТУ В СИСТЕМАХ ІНТЕРНЕТУ РЕЧЕЙ

У статті представлено аналіз розробки комплексного рішення для захисту IoT систем та відомих і новітніх рішень у цій сфері.

Спочатку було окреслено шари представлення архітектури IoT систем, а саме рівні сприйняття, мережі, оброблення даних і застосунок. Кожному з цих рівнів притаманні як спільні вразливості, так і унікальні. Ми уточнили критичні точки вразливості, охарактеризували основні проблеми автентифікації та авторизації. Зазначили, що стандартні облікові дані визначають як найпоширенішу та найпростішу складову вразливості, якою користуються зловмисники.

Проаналізовано наукові роботи, присвячені вирішенню проблем у сфері контролю доступу: централізовані центри довіри в TLS протоколі і пропозицію переходу на розподілені центри; випадки застосування IoT пристроїв без традиційних способів контролю доступу.

Значну увагу приділено шифруванню. Досліджені такі шифрувальні протоколи та методи, як TLS, DTLS, Novel Tiny Symmetric Encryption Algorithm, Lightweight CA Cipher (LCC) та Functional Encryption (FE), а також їхнє оптимальне застосування в IoT.

Ключові слова: Інтернет речей, вразливості безпеки, архітектура IoT, центр довіри, TLS протокол, DTLS, протокол, Novel Tiny Symmetric Encryption Algorithm, Lightweight CA Cipher (LCC) та Functional Encryption (FE).

Вступ

Інтернет речей розпочав трансформацію нашого цифрового простору, в якому повсякденні об'єкти є взаємопов'язаними та здатні до комунікації між собою. Ця трансформація не тільки спрощує нам життя, а й відкриває дорогу до небачених раніше можливостей та ефективності у сферах розумного будинку, медицини, індустріального виробництва, міського управління тощо [1]. Проте, як і з будь-яким технологічним проривом, IoT (Internet of Things) потребує зваженого та обережного впровадження. Насамперед, масштабне та неконтрольоване впровадження IoT пристроїв провокує багато проблем з безпекою, які необхідно вирішувати.

Основою IoT пристроїв є здатність збирати, оброблювати і передавати інформацію автономно, без людського втручання. Хоча саме це і є головною інновацією, така автономність створює багато вразливостей у безпеці. Наприклад, IoT пристрої часто використовують як приватні, так і публічні мережі, таким чином розширюючи потенційний вектор атаки зловмисника на конфіденційність і цілісність даних. Але це не єдине, на що потрібно зважати. Не менш важливими є надійність і доступність критично важливого функціоналу, який такі прилади надають. Від термостатів і годинників до автономних транспортних засобів та інфраструктури міста, вплив скомпрометованих IoT пристроїв може варіюватися від простої незручності до катастрофічних збоїв, що вплинуть на життя мільйонів людей.

IoT прилади зазвичай мають мало обчислювальної потужності та оперативної пам'яті. Такі обмеження призводять до того, що часто такі традиційні підходи до безпеки, як складні алгоритми шифрування, неможливо впровадити. Зважаючи на це і на вибухове зростання кількості IoT приладів у використанні, також збільшується і потенціал для несанкціонованого втручання в роботу систем. За прогнозом Cybersecurity Ventures, кількість IoT пристроїв у світі зросте до 25,1 мільярда до 2025 року. Кожний із них є потенційною точкою входу для зловмисника. Таке значне поширення створює велику, часто погано захищену мережу взаємопов'язаних приладів, яку можуть використати зловмисники для власних цілей: крадіжки даних, DDoS атак тощо.

Більше того, non-patchable (що не підлягають виправленню) та forever-day (вічні) вразливості тільки ускладнюють проблему. Багато IoT пристроїв нечасто отримують оновлення прошивки через логістичні проблеми доставлення цих оновлень або ж тому, що виробник про них забув і більше не підтримує. Такі вразливості являють собою постійно відкриті двері для злоумисників, і тому потрібен інноваційний підхід до поточних рішень.

Інфраструктуру навколо IoT пристроїв можливо розділити на чотири рівні [2]: сприйняття (захоплення вузла, імперсоніфікація вузла, атака відтворення, часова атака, атака позбавлення сну), мережі (підслуховування, DoS/DDoS атака, Main-in-The-Middle, оброблення даних (виснаження ресурсів, експлойти), застосунок (експлойти). Отже, основними проблемами для безпеки в IoT є несанкціонований доступ до пристроїв і даних, підслуховування комунікації, маніпуляція комунікації, виснаження ресурсів систем і пристроїв.

Метою цієї статті є опис запропонованого комплексного рішення для захисту IoT систем. Існує нагальна потреба в надійних і здатних до масштабування рішень. Наш підхід враховує унікальні характеристики IoT екосистем, а саме різноманітність, обмеженість в обчислювальних ресурсах і фізичну природу приладів.

Проблеми автентифікації та авторизації

IoT прилади мають багато вразливостей безпеки. Серед них можливо виокремити використання слабких паролів, паролів за замовчуванням і hardcoded (вписаних у прошивку пристроїв) паролів. Ця вразливість вирізняється поміж інших через свою поширеність і простоту. Автентифікація та авторизація є критично важливими для забезпечення безпеки IoT приладів. Автентифікація відповідає за те, щоби тільки дозволені прилади та користувачі мали змогу отримати доступ до сервісу. Авторизація ж — контролює дії, які може робити авторизована сутність. У контексті IoT, традиційних підходів може бути недостатньо,

Важливими тут є поняття довіри та центрів довіри. У роботі [7] запропоновано використовувати локальну розподілену систему управління довірою. Завдяки децентралізації процесів автентифікації та авторизації і розміщенню їх ближче до місця, де генеруються та обробляються дані, IoT може досягти вищого рівня безпеки. Цей метод вирішує проблеми масштабу та різноманітності в системах IoT, зменшує затримку, підвищує ефективність і потенційно пропонує більшу стійкість проти атак.

Зрозуміло, що потрібно окрему увагу надавати IoT пристроям, у яких немає традиційних інтерфейсів користувача, таких як клавіатура і сенсорний екран [13]. Як приклад вразливостей, пов'язаних із нетрадиційними способами автентифікації та авторизації таких пристроїв, наведено Apple Watch, із його функціоналом розблокування за допомогою розблокування зв'язаного з ним телефону. Якщо такий годинник було втрачено, власник, нічого не підозрюючи, може розблокувати свій телефон, а з ним і годинник. Далі злоумисник може отримати доступ до Mac жертви за допомогою годинника. Тож автори стверджують про потребу так званої постійної автентифікації та авторизації.

Динамічний характер середовищ IoT, де пристрої можуть непостійно перебувати під контролем користувача, потребує переходу від одноразових механізмів автентифікації до рішень, які можуть постійно перевіряти особу користувача. Також розглядають варіанти з використанням унікальних біологічних властивостей людини, таких як ритм серцебиття, кількість світла чи радіохвиль, що відбиває людське тіло тощо.

За допомогою машинного навчання, гіпотетично, це можливо перетворити на системи автентифікації та авторизації.

Стаття [5] пропонує вирішення проблеми захисту IoT пристроїв у середовищах, де традиційні заходи безпеки можуть бути неможливими, наприклад у зонах з обмеженими каналами під'єднання і ресурсами. Найперше, це запис унікальних облікових даних у прошивку на стадії виробництва. Такий метод є серйозною перепорою для імперсоніфікації пристрою чи підслуховування комунікації, проте потребує більше зусиль на стадії виробництва та підтримки. Дієвим є і варіант, коли облікові дані пристроїв зберігаються у вигляді фізичних QR кодів і додаються до сервера авторизації перед тим, як їх віддадуть користувачам. QR код залишається в безпечному місці. В цьому і подальшому варіантах використовують так звану довіру через фізичну близькість, а QR код застосовують для процесу ACE обміну. Так, облікові дані пристроїв можуть зберігатися у вигляді фізичних QR кодів і додаватися до сервера авторизації напряду перед використанням. У цьому варіанті існує проблема втрати QR коду та подальшої компрометації. Також, якщо QR було скомпрометовано, його

неможливо змінити без зміни прошивки. Життєвим є варіант, коли випадково згенерований QR код, що надано з пристроєм, використовують для початкового шифрування та отримання нового ключа. Цей процес можливо виконати тільки після фізичного перезавантаження пристрою, щоби уникнути використання «довіри через фізичну близькість» як вразливості.

Для забезпечення безпечної комунікації використовують CoAP через IPv6 із DTLS. Також важливою частиною всієї системи є можливість відкликати токени авторизації (token revocation). Пристрої та ресурс сервер постійно запитують у сервера авторизації інформацію про актуальність токенів та доступів.

Шифрування

У величезному та взаємопов'язаному світі IoT шифрування даних є гарантом конфіденційності та цілісності. Оскільки пристрої IoT збирають, передають і обробляють щораз більший обсяг конфіденційних даних, брак надійного шифрування даних стає явною вразливістю. Це не тільки наражає користувачів на ризик витоку даних і конфіденційності, а й підриває довіру, необхідну для ширшого впровадження IoT. Зважаючи на це й унікальну природу IoT приладів, а саме — часту обмеженість у ресурсах, виникає потреба в обчислювально легких і достатньо надійних методах та парадигмах шифрування.

Протокол TLS можна вважати стандартним вибором для впровадження захисту комунікації. Наприклад, TLS використовують в інфраструктурах, що містять брокер повідомлень MQTT. TLS (transport layer security) у своїй основі має протокол SSL (Secure Sockets Layer), розроблений компанією Netscape [6]. Протокол було реалізовано над транспортним рівнем, у цьому випадку — над TCP. Свого часу правильно сконфігурований SSL надавав хороші гарантії безпеки. Потенційний зловмисник міг дізнатися лише параметри з'єднання, такі як конкретний тип шифрування, частота обміну та кількість даних.

Протокол TLS виконує такі функції: шифрування, автентифікація і гарантія цілісності даних. Задля встановлення безпечного зв'язку учасники мають вибрати методи шифрування та ключі. В протоколі цей процес має назву TLS Handshake. Протокол використовує парадигму криптографії з відкритим ключем. Це означає, що клієнти можуть встановити безпечний зв'язок без попередніх знань один про одного, проте це залежить від обраного криптографічного набору (наприклад, для TLS-SRP потрібно попередньо знати про спільний секрет). Також у процедурі TLS Handshake підтверджується автентичність презентованих сторін, зазвичай клієнт перевіряє сервер, проте існує можливість взаємної перевірки, відома як mTLS.

Кожне надіслане повідомлення містить MAC (Message Authentication Code) код, що створюється за допомогою односторонньої криптографічної функції хешування та виконує функцію контрольної суми. Цю контрольну суму також генерує і отримує для визначення цілісності інформації.

Ще у версії SSL 2.0 було перебачено скорочений процес TLS Handshake, що має назву abbreviated handshake. В пакеті ServerHello сервер може надіслати 32-байтний ідентифікатор сесії. Цей ідентифікатор генерується на сервері та зберігається в кеш для подальшого використання. Клієнт, отримавши ідентифікатор сесії, також зберігає його в себе. Тепер, якщо з'єднання було розірвано, клієнт може в пакеті ClientHello надіслати вищезгаданий ідентифікатор. Якщо учасники з'єднання мають однакові ідентифікатори, процес TLS Handshake буде скорочено. Якщо ні — виконається повна версія.

Abbreviated handshake дає змогу пропустити генерацію симетричного ключа, що суттєво скорочує час встановлення з'єднання. Проте за такого підходу сервер має зберігати дані про всі минулі сесії, що на великих обсягах може впливати на його швидкодію. Для вирішення цієї проблеми існує механізм Session Ticket. Якщо клієнт підтримує такий механізм, то під час першого з'єднання замість ідентифікатора сесії сервер надішле Session Ticket — параметри сесії, зашифровані приватним ключем сервера. В такому варіанті при відновленні з'єднання клієнт замість ідентифікатора сесії надсилає вищезгаданий Ticket, що дозволяє серверу не зберігати дані про сесію.

Механізм Abbreviated Handshake пришвидшує процес відновлення сесії, проте він ніяк не впливає ні на найперший етап встановлення зв'язку, TLS Handshake, ні на випадок, коли сесія вже не є дійсною. Щоби пришвидшити роботу і тут, існує опціональне розширення TLS False Start. Воно дає змогу надсилати дані ще до завершення процесу TLS Handshake. В результаті використання такого механізму стає на одну ітерацію обміну повідомленнями менше.

Із різних причин раніше частіше за все обмін ключами відбувався за допомогою алгоритму RSA [9]. Цей алгоритм має один недолік: якщо зловмисник заволодіє приватним ключем сервера, він зможе розшифровувати всі сеанси зв'язку з цим сервером. Навіть гірше, зловмисник може роками записувати шифрований трафік, маючи на меті розшифрувати дані, коли він отримає приватний ключ сервера. На відміну від RSA, процес обміну ключами Діффі-Геллмана (D-H) [11] є більш захищеним, бо вибраний симетричний ключ шифрування ніколи не виходить за межі ні клієнта, ні сервера, і тому не може бути перехоплений по мережі. DH зменшує ризики, пов'язані з компрометацією попередніх сеансів зв'язку, оскільки для кожного нового сеансу створюється новий тимчасовий симетричний ключ.

Варто ще раз зауважити, що шифрування з публічним ключем використовується тільки в процесі TLS Handshake. Після встановлення зв'язку дані шифруються і розшифровуються за допомогою симетричної криптографії. Такий підхід вибрано для збільшення швидкодії, оскільки шифрування з публічним ключем є набагато більш вимогливим у розрізі обчислювальних ресурсів.

У протоколі TLS важливим концептом є ланцюг довіри. Коли клієнт звертається до сервера, він має мати змогу підтвердити автентичність наданого публічного ключа / сертифіката. Зробити це він може декількома способами, а саме: *certiticate/public key pinning* — клієнт має збережений сертифікат або публічний ключ сервера, який він порівнює з наданим; перевірка наданого сертифіката за допомогою сертифіката центру довіри (CA certificate).

Ланцюг довіри оснований на сертифікатах автентичності, що їх надають спеціальні установи — центри сертифікації (CA, *certificate authorities*). З виданих сертифікатів складається певний ланцюг. Причиною використання ланцюга є розподілення ризиків безпеки. Сертифікати містять інструкції з перевірки їхньої актуальності, тому під час перевірки сертифіката потрібно перевіряти весь ланцюг довіри. В основі процедури перевірки лежить *Certificate Revocation List (CRL)* — список відкликаних сертифікатів. Кожний центр сертифікації має такий список. Якщо сертифікат там згадано, то це означає, що його було відкликано і з'єднання встановлювати не можна. Проте запит на отримання всього списку відкликаних сертифікатів для перевірки одного є нераціональним із погляду ресурсів. До того ж, такі списки публікують періодично, а не постійно. Тому було розроблено механізм *Online Certificate Status Protocol (OCSP)*, що дає змогу перевіряти статус сертифіката динамічно й селективно.

Найбільшою проблемою використання TLS для деяких приладів буде обмеженість обчислювальних ресурсів. Проте існує велика кількість полегшених бібліотек та імплементацій [3], які залишають лише найнеобхідніші компоненти протоколу, позбавляючись всього іншого. Наприклад, можливо створити TLS бібліотеку, що займатиме лише 20 Кб оперативної пам'яті під час роботи. Більшість складнощів при розробці таких рішень полягає у великій кількості підтримуваних шифрувальних комплектів. Якщо взяти за основу TLS 1.2 комплект *TLS_RSA_WITH_AES_128_CBC_SHA256*, то доведеться імплементувати тільки RSA, AES та SHA-256. Також процес Handshake можливо зробити синхронним і позбавитися всіх додаткових механізмів, як *False Start* та *Abbreviated Handshake*. Найскладнішою частиною всієї імплементації є робота з X.509 сертифікатами, але і тут можливо спростити, використавши механізм *certificate pinning*. Інший варіант — використати бандл *TLS_SRP_SHA_WITH_AES_128_CBC_SHA*, але тоді треба мати наперед відомий спільний секрет. Варто ще раз зауважити, що процес обміну RSA не надає *forward secrecy* захисту, тому, все ж таки, рекомендують використовувати ECDHE шифр.

Datagram Transport Layer Security (DTLS) — протокол, побудований над UDP, що загалом надає ті самі гарантії безпеки і має схожу структуру, що і TLS. Використовують у таких протоколах, як CoAP. Через використання UDP як транспортного протоколу DTLS має як плюси, так і мінуси порівняно з TLS. Найчастіше застосовують у системах із великим обсягом стримінгових даних. DTLS дає змогу уникати повільної комунікації, що характерна для TLS. Проте, водночас, він не змінює порядок пакетів і не гарантує їхньої унікальності (*non-replayability*). Всі відмінності між TLS та DTLS пов'язані з використанням протоколу UDP [4].

Крім шифрування, DTLS також використовують для запобігання втраті або надходженню пакетів даних у неправильному порядку. DTLS використовує для цього простий таймер повторного передання, при цьому кожний учасник зв'язку продовжує повторно передавати своє останнє повідомлення, доки не буде отримано відповідь. TLS ділить довгу послідовність даних на кілька фрагментів. Протокол є відповідальним за розбір цих фрагментів на кінці отримувача, і тому клієнтський код не потребує жодних додаткових модифікацій. Натомість DTLS використовує записи, які можна надсилати повністю чи ні. Цими записами повинні керувати самі програми, тобто протокол не є відповідальним

за це. Так само, як і UDP, у DTLS немає сигналу завершення передавання. У DTLS передавання даних просто припиняється. Це означає, що клієнт, який отримує дані від сервера, не знає, чи було доставлено всі дані, чи сталася помилка під час зв'язку. TLS сигналізує про це сповіщенням.

DTLS використовує cookie для запобігання підміні IP-адреси. Це дає змогу спілкуватися із сервером, не розкриваючи клієнтську IP-адресу. З іншого боку, TLS здійснює зв'язок лише після встановлення TCP-handshake, що ускладнює підміну IP-адреси.

У дослідженні SuHyun Kim і ImYeong Lee [8] запропоновано використовувати Proxy Re-Encryption (PRE) — перешифрування на проксі. Цей спосіб шифрування вирішує критичну проблему захисту передавання даних в екосистемах IoT, що характеризується безліччю взаємопов'язаних пристроїв з обмеженими обчислювальними можливостями.

PRE — це криптографічна парадигма, яка дає змогу делегувати права на дешифрування. Вона надає проксі ноді можливість перетворювати дані, шифровані ключем А, на дані, які можливо розшифрувати ключем В. Під час цієї операції дані у відкритому вигляді не з'являються.

Цей механізм особливо підходить для середовища IoT з кількох причин. Пристрої IoT часто мають обмежену обчислювальну потужність і пам'ять, що в цих випадках робить традиційні схеми шифрування непрактичними. PRE пропонує полегшену альтернативу, яка враховує ці обмеження. Інфраструктура IoT часто вимагає безпечного обміну даними між кількома пристроями або користувачами. PRE полегшує це, дозволяючи безпечно та безпосередньо ділитися зашифрованими даними без необхідності розшифровки та повторного шифрування, заощаджуючи таким чином обчислювальні ресурси. IoT-ландшафт дуже динамічний, де пристрої часто приєднуються до мережі та виходять з неї. PRE підтримує цей динамізм, забезпечуючи гнучке та безпечне керування ключами та механізми контролю доступу до даних.

У контексті IoT PRE можна використовувати так. У шифруванні кожний пристрій генерує собі пару ключів: публічний (pk) і приватний (sk). Потім він генерує ключ перешифрування, щоби обмінюватися даними з іншими пристроями. Щоби пристрій А міг поділитися даними з пристроєм В, йому потрібно згенерувати ключ повторного шифрування $rk(A \rightarrow B)$. Він генерується за допомогою sk(A) та pk(B) і далі надсилається на проксі ноду. У цьому сценарії кожен пристрій створює ключ перешифрування для всіх пристроїв, крім себе. Проксі ноду перешифровує дані від пристрою А за допомогою ключа $rk(A \rightarrow B)$ та надсилає на пристрій В. Пристрій В розшифровує дані за допомогою sk(B).

У статті [10] запропоновано новий підхід до TEA шифрування. Цей метод спрямований на підвищення безпеки та ефективності передавання текстових файлів між пристроями IoT, що відповідає критичній потребі в легких криптографічних рішеннях.

NTSA розроблено для подолання обмежень алгоритму Tiny Encryption Algorithm (TEA) і його варіантів. Вони, не зважаючи на невелике використання пам'яті та простоту імплементації, страждають від вразливостей безпеки через використання постійного ключа протягом усього процесу шифрування. NTSA вводить динамічну плутанину ключів (Dynamic Key Confusions) для кожного раунду шифрування, значно покращуючи безпеку зашифрованих даних. Цей підхід особливо корисний для середовищ Інтернету речей, де пристрої часто працюють із суворими обмеженнями ресурсів.

На відміну від TEA, який використовує той самий ключ для всіх раундів шифрування, NTSA динамічно генерує додаткові ключі для кожного раунду. Динамічні ключі генеруються на основі основного ключа та даних, що планується зашифрувати. Цей метод значно підвищує безпеку процесу шифрування, збільшуючи складність ключа та роблячи його більш стійким до атак. Окрім того, NTSA спеціально оптимізовано для безпечної передачі текстових файлів у системах IoT. Хоча NTSA і пропонує розширені гарантії безпеки, цей метод шифрування залишається легким і ефективним. Це робить його придатним для використання у пристроях IoT, де кількість обчислювальних можливостей є обмеженою.

У статті [12] представлено шифрування за назвою Lightweight Cellular Automata Based Encryption Technique — полегшене шифрування на основі клітинних автоматів (CA).

LCC використовує принципи клітинного автомата — математичної моделі, яка складається з сітки комірок, кожна з яких перебуває в одному зі скінченної кількості станів, наприклад «увімкнено» або «вимкнено». Стан комірки на наступному кроці часу визначається набором правил на основі поточного стану комірки і станів сусідніх комірок. LCC використовує простоту та ефективність скінченного автомата, щоб забезпечити надійний, але легкий метод шифрування, який підходить для обмежених середовищ пристроїв IoT. Цей метод шифрування є симетричним.

Алгоритм LCC розроблено так, щоб бути ефективним як із погляду обчислювальних вимог, так і з точки зору споживання енергії. Це, своєю чергою, робить його особливо корисним для використання в IoT пристроях, бо вони часто живляться від батарей і мають обмежені ресурси для оброблення даних.

У методі LCC ключем шифрування слугує наперед обраний набір правил RVList512 і кількість ітерацій (1–8). Вводом слугує 512 біт початкових даних.

Правила клітинних автоматів можуть бути згенеровані залежно від радіуса та кількості можливих значень, які клітина може мати як значення стану. У випадку одновимірного клітинного автомату з радіусом $r = 1$ (три можливі сусіди (лівий, центральний, правий)), можна створити 256 можливих правил СА, але усі ці комбінації не утворюють правила GCA (груповий клітинний автомат). Для генерації GCA правил використовують алгоритм, який випадковим чином бере вісім правил СА(RV8) із доступних 256 правил СА, а потім генерує GCA вектор (RVList512) шляхом конкатенації. Такий вектор, що містить 512 правил СА, можна використовувати в шифруванні. Зловмисник має витратити майже експоненційний час, щоби вгадати 512 правил, що зводить нанівець можливість брутфорс атаки.

Функціональне шифрування (Functional Encryption, FE) є узагальненням наявних технологій шифрування за допомогою відкритого ключа, такі як шифрування на основі ідентифікації (IBE), на основі атрибутів (ABE), гомоморфне шифрування (HE), шифрування предикатів (PE), шифрування з можливістю пошуку (SE) тощо [14].

FE являє собою зміну парадигми технології шифрування, що дає змогу більш детально контролювати доступ до зашифрованих даних. У системі FE ключі дешифрування видаються на основі певних функцій; власники цих ключів можуть обчислювати лише певні функції над зашифрованими даними, не вивчаючи нічого іншого про дані. У застосуванні в IoT FE пропонує новий підхід до захисту конфіденційних даних клієнтів, водночас дозволяючи постачальникам послуг виконувати необхідну аналітику та оброблення даних.

Функціональне шифрування дає змогу точно контролювати, які дані можна розшифрувати й обробити, залежно від дозволів, пов'язаних із ключем розшифрування. Воно гарантує, що конкретні актори можуть отримати доступ лише до тих даних, на які вони мають право. Кожен фахівець може отримати доступ лише до даних, що стосуються його галузі, забезпечуючи конфіденційність клієнта.

Стратегії реагування на інциденти безпеки

Безпека IoT інфраструктури є критичним аспектом, що потребує комплексного підходу. З огляду на різноманітність атак і вразливостей потрібно впроваджувати багаторівневий підхід до безпеки, що передбачає як технічні, так і організаційні рішення. Основними проблемами для безпеки в IoT є несанкціонований доступ до пристроїв і даних, підслуховування комунікації, маніпуляція комунікації, виснаження ресурсів систем і пристроїв.

Також необхідно враховувати загрози, що є специфічними для кожного окремо рівня інфраструктури, та впроваджувати відповідні заходи безпеки, наприклад: використання надійних методів шифрування, регулярне оновлення програмного забезпечення, моніторинг мережевої активності, впровадження горизонтальної архітектури для систем, фізичний захист IoT пристроїв тощо.

Проблему, яку становлять вразливі облікові дані в IoT приладах, неможливо переоцінити. Усунення цієї вразливості потребує зусиль від виробників, користувачів і навіть державних установ.

Забезпечення захисту IoT пристроїв і мереж, у яких вони оперують, потребує імплементації надійних механізмів автентифікації та авторизації. Враховуючи часто конфіденційний характер даних і потенційні наслідки дір у безпеці, розроблення і впровадження таких механізмів мають першочергове значення. Рішення мають бути ефективними та масштабованими, здатними адаптуватися до унікальних вимог IoT екосистем.

У світі IoT шифрування даних відіграє ключову роль у захисті конфіденційності та цілісності інформації. Через зростання обсягу чутливих даних, які збирають і обробляють пристрої IoT, надійне шифрування є критично важливим для запобігання витокам даних і порушенням конфіденційності. Це особливо актуально у зв'язку з обмеженістю ресурсів IoT пристроїв, що підвищує необхідність у легких і водночас ефективних методах шифрування. У статті ми розглянули як перевірені часом, так і нові підходи, які можуть бути використані для підвищення безпеки IoT систем. Викла-

дені методи та парадигми забезпечують міцну основу для створення надійних, ефективних і безпечних систем, здатних адаптуватися до різних вимог.

Враховуючи складність і різноманітність середовищ Інтернету речей, традиційні стратегії реагування на інциденти часто виявляються неефективними. Для ефективної та правильної реакції на випадки компрометації систем Інтернету речей потрібен спеціальний підхід.

Неоднорідність і велика різноманітність пристроїв Інтернету речей створюють значні проблеми для моніторингу, виявлення та реагування. Це означає, що різні групи пристроїв потенційно можуть потребувати різного підходу для вищезгаданих процесів. Отримання повного логу операцій пристроїв і відкритого мережевого трафіку є складним завданням, що ускладнює виявлення та розуміння масштабу інцидентів. Обмежена потужність обчислювальних ресурсів і кількість постійної пам'яті багатьох пристроїв Інтернету речей обмежують обсяг даних, які можна зібрати після інциденту. Крім того, потенційна фізична недоступність деяких пристроїв ускладнює аналіз, якщо є підозра на фізичний характер інциденту порушення безпеки.

Висновок

Метою цієї роботи був аналіз розробки комплексного рішення для захисту IoT систем і відомих і новітніх рішень у цій сфері.

Спочатку було окреслено шарове представлення архітектури IoT систем, а саме рівні сприйняття, мережі, оброблення даних та застосунок. Кожному з цих рівнів притаманні як спільні вразливості, так і унікальні. Ми уточнили критичні точки, які можуть використати зловмисники для отримання доступу до систем чи перешкоджання роботи тих чи тих вузлів.

Потім ми сфокусували увагу на проблемах автентифікації та авторизації. Зауважили, що стандартні облікові дані визначаються як найпоширеніша та найпростіша складова вразливості, якою користуються зловмисники. Ця вразливість існує в великому переліку систем і вже не один раз призводила до широких успішних кібератак, як-от Mirai ботнет.

Далі аналізували наукові роботи, присвячені менш поширеним проблемам у сфері контролю доступу: проблемі централізованих центрів довіри в TLS протоколі і пропозиції переходу на розподілені центри; випадкам застосування IoT пристроїв без традиційних способів контролю доступу чи з потребою в продовжуваному контролі доступу; використанню IoT у несприятливих умовах (зоні бойових дій), де існує ризик фізичної втрати пристроїв чи серверів і компрометації систем, та способам захисту в таких сценаріях.

Значну увагу приділено шифруванню. Тут досліджено такі шифрувальні протоколи та методи, як TLS, DTLS, Novel Tiny Symmetric Encryption Algorithm, Lightweight CA Cipher (LCC) та Functional Encryption (FE), а також їхнє оптимальне застосування в IoT.

Унікальні характеристики IoT екосистем потребують спеціального підходу до управління загрозами та реагування на інциденти. Розуміючи конкретні загрози, з якими стикаються пристрої та мережі IoT, і запроваджуючи систему реагування на інциденти, адаптовану до цих проблем, організації можуть підвищити свою стійкість проти кібератак. Це передбачає не лише технічні заходи, а й організаційну готовність, зокрема навчання, планування та аналіз після інциденту. Завдяки старанням зусиллям і стратегічному плануванню можна значно посилити безпеку всієї інфраструктури, захищаючи як пристрої, так і дані, які оброблюються пристроями.

Список літератури

1. Глибовець А. М. Агентно-базовані програмні системи пошуку та аналізу інформації / А. М. Глибовець. — Київ : Видавничий дім «Києво-Могилянська академія», 2019. — 281 с. : іл.
2. Burhan M. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey / M. Burhan, R. Rehman, B. Khan, B.-S. Kim // *Sensors*. — 2018. — Vol. 18, no. 9. — P. 2796. — <https://doi.org/10.3390/s18092796>.
3. Computationally simple, lightweight replacement for SSL/TLS Information Security Stack Exchange, Apr. 2011 [Electronic resource]. — Mode of access: <https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls> (date of access: 18.09.2024).
4. Difference TLS Vs DTLS protocol [Electronic resource] // The Network DNA. — 2022. — Mode of access: <https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html> (date of access: 18.9.2024).
5. Echeverría S. Authentication and Authorization for IoT Devices in Disadvantaged Environments / S. Echeverría, G. A. Lewis, D. Klin-edinst, L. Seitz // 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). — Limerick, Ireland, 2019. — Pp. 368–373. — <https://doi.org/10.1109/WF-IoT.2019.8767192>.
6. Grigorik I. 4. Transport Layer Security (TLS) — High Performance Browser Networking [Electronic resource] / I. Grigorik. — Mode of access: <https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html> (date of access: May 26, 2024).

7. Kim H. Authentication and Authorization for the Internet of Things / H. Kim, E. A. Lee // *IT Professional*. — 2017. — Vol. 19, no. 5. — Pp. 27–33. — <https://doi.org/10.1109/mitp.2017.3680960>.
8. Kim S. IoT device security based on proxy re-encryption / S. Kim, I. Lee // *Journal of Ambient Intelligence and Humanized Computing*. — 2017. — Vol. 9, no. 4. — Pp. 1267–1273. — <https://doi.org/10.1007/s12652-017-0602-5>.
9. Nemec M. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli / M. Nemec, M. Sys, P. Svenda, D. Klinec, V. Matyas // *24th ACM Conference on Computer and Communications Security (CCS'2017)*. ACM: 1631–1648. — <https://doi.org/10.1145/3133956.3133969>.
10. Rajesh S. A Secure and Efficient Lightweight Symmetric Encryption Scheme for Transfer of Text Files between Embedded IoT Devices / S. Rajesh, V. Paul, V. Menon, M. Khosravi // *Symmetry*. — 2019. — Vol. 11, no. 2. — P. 293. — <https://doi.org/10.3390/sym11020293>.
11. Rescorla E. Diffie-Hellman Key Agreement Method [Electronic resource] / E. Rescorla // RFC 2631 — 1999. — Mode of access: <https://datatracker.ietf.org/doc/html/rfc2631>.
12. Roy S. A Lightweight Cellular Automata Based Encryption Technique for IoT Applications / S. Roy, U. Rawat, J. Karjee // *IEEE Access*. — 2019. — Vol. 7. — Pp. 39782–39793. — <https://doi.org/10.1109/ACCESS.2019.2906326>.
13. Shahzad M. Continuous Authentication and Authorization for the Internet of Things / M. Shahzad, M. P. Singh // *IEEE Internet Computing*. — 2017. — Vol. 21, no. 2. — Pp. 86–90. — <https://doi.org/10.1109/MIC.2017.33>.
14. Sharma D. Functional Encryption in IoT E-Health Care System / D. Sharma, Devesh Jinwala // *Lecture notes in computer science*. — 2015. — Pp. 345–363. — https://doi.org/10.1007/978-3-319-26961-0_21.

References

- Burhan, M., Rehman, R., Khan, B., & Kim, B.-S. (2018). IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. *Sensors*, 18 (9), 2796. <https://doi.org/10.3390/s18092796>.
- Computationally simple, lightweight replacement for SSL/TLS. (April, 2011). Information Security Stack Exchange. <https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls>.
- Difference TLS Vs DTLS protocol. (2022, 4 November). *The Network DNA*. <https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html>.
- Echeverría S., Lewis G. A., Klinedinst, D., and Seitz, L. (2019). Authentication and Authorization for IoT Devices Disadvantaged Environments. In *IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland* (pp. 368–373). <https://doi.org/10.1109/WF-IoT.2019.8767192>.
- Grigorik, I. (2013). 4. Transport Layer Security (TLS). In *High Performance Browser Networking*. O'Reilly Online Learning. <https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html>.
- Hlybovets, A. (2019). *Ahento-bazovani prohramni systemy poshuku ta analizu informatsii*. Vydavnychiy dim “Kyievo-Mohylianska akademiiia”.
- Kim, H., & Lee, E. A. (2017a). Authentication and Authorization for the Internet of Things. *IT Professional*, 19 (5), 27–33. <https://doi.org/10.1109/mitp.2017.3680960>.
- Kim, S., & Lee, I. (2017b). IoT device security based on proxy re-encryption. *Journal of Ambient Intelligence and Humanized Computing*, 9 (4), 1267–1273. <https://doi.org/10.1007/s12652-017-0602-5>.
- Nemec, M., Sys, M., Svenda, P., Klinec, D., & Matyas, V. (2017). The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *24th ACM SIGSAC Conference on Computer and Communications Security*. ACM. <https://doi.org/10.1145/3133956.3133969>.
- Rajesh, S., Paul, V., Menon, V. G., Khosravi, M. R. (February, 2019). A Secure and Efficient Lightweight Symmetric Encryption Scheme for Transfer of Text Files between Embedded IoT Devices. *Symmetry*, 11 (2), 293. <https://doi.org/10.3390/sym11020293>.
- Rescorla, E. (June, 1999). Diffie-Hellman Key Agreement Method. *RFC 2631* <https://datatracker.ietf.org/doc/html/rfc2631>.
- Roy, S., Rawat, U., & Karjee, J. (2019). A Lightweight Cellular Automata Based Encryption Technique for IoT Applications. *IEEE Access*, 7, 39782–39793. <https://doi.org/10.1109/ACCESS.2019.2906326>.
- Shahzad, M., & Singh, M. P. (2017). Continuous Authentication and Authorization for the Internet of Things. *IEEE Internet Computing*, 21 (2), 86–90. <https://doi.org/10.1109/MIC.2017.33>.
- Sharma, D., & Jinwala Devesh. (January, 2015). Functional Encryption in IoT E-Health Care System. *Lecture notes in computer science*, 345–363. https://doi.org/10.1007/978-3-319-26961-0_21.

A. Hlybovets, S. Shcherbyna, O. Kyriienko

SECURITY VULNERABILITIES AND PROTECTION SOLUTIONS IN INTERNET OF THINGS SYSTEMS

The Internet of Things (IoT) has begun transforming our digital space, in which everyday objects are interconnected and capable of communicating with each other. This transformation not only simplifies our lives but also creates unprecedented opportunities and enhances efficiency in areas such as smart homes, healthcare, industrial manufacturing, and urban management. However, as with any technological breakthrough, IoT requires a careful and well-planned implementation. The large-scale and unregulated deployment of IoT devices raises significant security concerns that must be mitigated.

At the core of IoT devices lies the ability to autonomously collect, process, and transmit information independently of human intervention. While this autonomy is the primary innovation, it also introduces numerous security vulnerabilities. For instance, IoT devices often operate on private and public networks, increasing the attack surface for malicious actors capable of compromising data confidentiality and

integrity. However, security is not the sole concern. The reliability and availability of the critical functions these devices provide are equally crucial. From thermostats and smartwatches to autonomous vehicles and urban infrastructure, compromised IoT devices can cause anything from minor inconveniences to catastrophic failures affecting millions.

IoT devices typically have limited computational power and memory. These constraints often render the implementation of traditional security measures, such as complex encryption algorithms, impractical. Given these limitations and the explosive growth in the number of IoT devices, the potential for unauthorized interference with systems also increases. According to Cybersecurity Ventures, the number of IoT devices worldwide is expected to reach 25.1 billion by 2025. Each device represents a potential entry point for attackers. This widespread proliferation creates a vast, often poorly secured, network of interconnected devices susceptible to exploitation by malicious actors for data theft, DDoS attacks, and other cyber threats.

The infrastructure surrounding IoT devices can be divided into four levels: perception (node capture, node impersonation, replay attack, timing attack, sleep deprivation attack), network (eavesdropping, DoS/DDoS attacks, Man-in-the-Middle attacks), data processing (resource exhaustion, exploits), and application (exploits). Therefore, key security challenges in IoT include unauthorized access to devices and data, communication interception, data manipulation, and depletion of system and device resources.

The purpose of this article is to describe a comprehensive proposed solution for securing IoT systems. There is an urgent need for scalable and reliable solutions. Our approach considers the unique characteristics of IoT ecosystems, particularly their diversity, limited computational resources, and the physical nature of the devices.

Keywords: Internet of Things, security vulnerabilities, IoT architecture, trust center, TLS protocol, DTLS, protocol, Novel Tiny Symmetric Encryption Algorithm, Lightweight CA Cipher (LCC), Functional Encryption (FE).

Матеріал надійшов 27.09.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Андрощук М. В.

СУЧАСНІ ПІДХОДИ ДО ВИКОРИСТАННЯ БАЗ ЗНАНЬ ДЛЯ ВИРІШЕННЯ ПРОБЛЕМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Розглянуто можливості інтеграції великих мовних моделей (ВММ) із базами знань для підвищення точності й надійності їхніх відповідей. Визначено переваги такого поєднання, зокрема зменшення ризику галюцинацій — генерування неправильної або вигаданої інформації. Проаналізовано різні підходи до об'єднання ВММ із базами знань, їхні сильні та слабкі сторони. Обговорено перспективи та виклики у застосуванні цієї технології в різних галузях, зокрема інформаційний пошук, підтримку прийняття рішень та автоматизоване створення контенту. Стаття надає огляд сучасного стану досліджень у цій сфері та окреслює напрями для подальшого вивчення.

Ключові слова: ВММ, база знань, галюцинації, RAG.

Вступ

Великі мовні моделі (ВММ) пройшли значну еволюцію від простих статистичних підходів до складних нейронних мереж, здатних розуміти й генерувати людську мову на високому рівні. Незважаючи на їхній потенціал у різних сферах застосування, ВММ стикаються з низкою викликів, як-от проблеми з точністю та достовірністю генерованої інформації, потенційні упередження, етичні питання та технічні обмеження. Особливу увагу привертає проблема «галюцинацій» — генерування неправдивої, але правдоподібної інформації. Для подолання цих викликів розробляють різні підходи, зокрема техніки самоперевірки, навчання на якісніших даних і використання баз знань як достовірних джерел інформації.

Розвиток ВММ

Великі мовні моделі мають довгу історію розвитку, яка сягає кількох десятиліть. Їхня еволюція тісно пов'язана з прогресом у галузі оброблення природної мови та штучного інтелекту. Ці моделі пройшли шлях від простих статистичних підходів до складних нейронних мереж, що здатні розуміти та генерувати людську мову на високому рівні.

На початку свого шляху ці моделі базувалися на простих статистичних підходах, які давали змогу аналізувати текст на базовому рівні. Вони могли виконувати такі завдання, як підрахунок частоти слів або визначення найбільш імовірного наступного слова в реченні. Однак ці ранні моделі мали обмежені можливості й не могли впоратися зі складними лінгвістичними структурами. З розвитком комп'ютерних технологій і появою більш потужних алгоритмів машинного навчання мовні моделі почали ставати все більш складними. Вони перейшли від простих статистичних методів до використання нейронних мереж, що дало їм змогу краще розуміти контекст та семантику мови.

Сучасні великі мовні моделі являють собою складні нейронні мережі, які навчаються на величезних обсягах текстових даних. Вони здатні не лише розуміти й аналізувати людську мову, а й генерувати власний текст, який часто важко відрізнити від написаного людиною. Ці моделі можуть виконувати широкий спектр завдань, зокрема переклад, відповіді на запитання, написання есе та навіть створення програмного коду [1].

Важливо зазначити, що розвиток великих мовних моделей триває. Дослідники постійно працюють над удосконаленням цих технологій, прагнучи зробити їх ще більш ефективними, точними та етичними у використанні. Майбутнє цих моделей обіцяє ще більш вражальні можливості у розумінні та генерації людської мови, що може мати значний вплив на різні сфери нашого життя — від освіти до бізнесу та наукових досліджень. Попри переваги і перспективи великі мовні моделі мають і проблеми.

Отже, великі мовні моделі демонструють значний потенціал і переваги, які можуть революціонізувати багато галузей та аспектів нашого життя. Ці передові технології здатні обробляти та генерувати людську мову на рівні, який раніше вважали неможливим. Їх застосовують у різноманітних сферах, як-от автоматизований переклад, створення контенту, аналіз даних та підтримка клієнтів. Крім того, великі мовні моделі відкривають нові можливості для наукових досліджень, освіти і творчості.

Однак, поряд із цими надзвичайними можливостями, великі мовні моделі стикаються з певними викликами та обмеженнями, які не можна ігнорувати. Серед них — проблеми з точністю та достовірністю генерованої інформації, потенційні упередження в навчальних даних, які можуть призвести до несправедливих або дискримінаційних результатів, а також етичні питання, пов'язані з конфіденційністю даних і можливим зловживанням технологією. Крім того, існують технічні обмеження, як-от високі обчислювальні вимоги та енергоспоживання, які можуть обмежувати широке впровадження цих моделей.

Хоча ці технології відкривають нові можливості в різних галузях, від медицини до фінансів, від освіти до розваг, важливо критично оцінювати їх застосування та продовжувати інтенсивні дослідження для подолання наявних проблем. Це передбачає розробку більш прозорих та інтерпретованих моделей, удосконалення методів навчання для зменшення упереджень, а також створення надійних механізмів контролю та верифікації генерованого контенту.

Подальший розвиток і вдосконалення великих мовних моделей потребуватиме збалансованого підходу, який враховуватиме як їхні незаперечні переваги, так і потенційні ризики. Це вимагатиме тісної співпраці між дослідниками, розробниками, політиками та громадськістю для створення етичних рамок і регуляторних механізмів, які забезпечать відповідальне використання цих потужних технологій. Важливо також інвестувати в освіту та підвищувати обізнаність суспільства щодо можливостей та обмежень великих мовних моделей, щоб сприяти їх ефективному та безпечному використанню. У кінцевому підсумку, майбутнє великих мовних моделей залежатиме від здатності максимізувати їхній потенціал, одночасно мінімізуючи пов'язані з ними ризики.

Проблеми LLM

Попри досягнення, актуальними залишаються й проблеми: галюцинації, етичність, прозорість, обмеженість предметних областей. Причина більшості проблем лежить у самій природі великих мовних моделей і їхнього оброблення знань [6].

Проблема галюцинацій у великих мовних моделей є однією з найбільш актуальних. Окрім цього, упередження даних призводять до аналогічної упередженості з боку великих мовних моделей [4].

Це явище виникає, коли модель генерує неправдиву або неточну інформацію, яка виглядає правдоподібно. Галюцинації можуть призвести до поширення дезінформації та зниження довіри до AI-систем. Дослідники активно працюють над розробленням методів для виявлення та зменшення галюцинацій у мовних моделях.

Про що знають мовні моделі, важливо розвивати розуміння для можливості гарантування точності замість повноти [7]. Надійність їхніх знань повністю залежить від навчальних даних. Якщо таких даних не вистачає для відповіді, потрібно залучати інструменти, що дають можливість отримувати актуальні дані, незалежні від тренувальних наборів. Наприклад, для медицини це є принциповим моментом, щоб забезпечити правильність лікування і прийняття рішень [8].

Одним із перспективних підходів є використання техніки «самоперевірки», коли модель оцінює власні відповіді на достовірність. Інший метод полягає у навчанні моделей на більш якісних і перевірених даних, щоб зменшити ймовірність генерації неправдивої інформації. Крім того, розробляють системи зовнішньої верифікації, які можуть автоматично перевіряти вихідні дані моделі на відповідність фактам із надійних джерел.

Пропозиції щодо покращення через бази знань

Для розгляду частини проблем, пов'язаних із точністю відповідей, запропоновано підхід із використанням баз знань як достовірних джерел інформації.

Графи знань є потужним інструментом для представлення та аналізу складних взаємозв'язків між різними концепціями та фактами [2]. Вони дозволяють візуалізувати інформацію у вигляді мережі взаємопов'язаних вузлів, що полегшує розуміння складних систем і процесів. Завдяки наявності

інфраструктури для роботи зі знаннями в такому випадку з'являється передбачуваність, визначеність і актуальність даних при інтеграції з ВММ. У галузі охорони здоров'я бази знань відіграють критичну роль у забезпеченні точності діагностики та лікування [5]. Вони містять структуровану інформацію про симптоми, захворювання, методи лікування та взаємодії лікарських засобів, що допомагає медичним працівникам приймати обґрунтовані рішення.

Інтеграція великих мовних моделей із базами знань відкриває нові можливості для автоматичного збагачення та оновлення інформації. Дослідження демонструють потенціал таких рішень для отримання точних і повних результатів [3]. Це дає змогу підтримувати актуальність баз знань і розширювати їх охоплення, використовуючи останні досягнення у сфері оброблення природної мови.

Підхід із використанням баз знань як достовірних джерел інформації є перспективним напрямом у галузі оброблення та аналізу даних. Цей метод передбачає створення та використання структурованих сховищ інформації, які містять верифіковані факти, концепції та взаємозв'язки між ними. Бази знань можуть охоплювати різноманітні галузі, від медицини до інженерії, і слугувати надійним фундаментом для прийняття рішень і проведення досліджень.

Використання баз знань дозволяє підвищити точність і надійність інформаційних систем, мінімізуючи ризик помилок та неточностей, які можуть виникнути при використанні неперифікованих джерел. Крім того, цей підхід сприяє ефективному обміну знаннями між експертами та полегшує процес навчання для новачків у певній галузі. Інтеграція баз знань із сучасними технологіями штучного інтелекту та машинного навчання відкриває нові можливості для створення інтелектуальних систем, здатних аналізувати складні ситуації та надавати обґрунтовані рекомендації на основі достовірної інформації.

Особливо корисними такі рішення є для RAG (Retrieval Augmented Generation) систем, бо поєднує в собі потужність великих мовних моделей і точність фактуальної інформації. Прикладами таких систем є: питально-відповідальні, пошукові, підтримки прийняття рішень та інші.

Висновки

Великі мовні моделі демонструють значний прогрес у розумінні та генеруванні людської мови, але їхній розвиток супроводжується низкою викликів. Ці моделі здатні виконувати широкий спектр завдань, від перекладу та узагальнення тексту до створення оригінального контенту, що робить їх потужним інструментом у різних галузях. Однак, незважаючи на можливості, ВММ стикаються з серйозними проблемами, такими як упередженість у даних, на яких вони навчаються, що може призвести до відтворення та посилення соціальних стереотипів.

Крім того, ВММ часто мають труднощі в забезпеченні послідовності та достовірності генерованої інформації, особливо при роботі зі складними або спеціалізованими темами. Етичні питання, пов'язані з конфіденційністю даних та потенційним зловживанням технологією, також залишаються актуальними. Дослідники і розробники активно працюють над вирішенням цих проблем, прагнучи створити більш надійні, прозорі та етично відповідальні моделі. Незважаючи на ці виклики, потенціал ВММ для революціонізації взаємодії людини з машиною та розширення можливостей у галузі оброблення природної мови залишається величезним.

Використання баз знань дає змогу моделям отримувати доступ до актуальної та перевіреної інформації. Це допомагає зменшити кількість помилок і неточностей у відповідях моделей. Крім того, бази знань можуть регулярно оновлюватися, що забезпечує моделям доступ до найсвіжіших даних.

Список літератури

1. Cooper N. Perplexed: Understanding When Large Language Models are Confused [Electronic resource] / N. Cooper, T. Scholak // arXiv.org. — Mode of access: <https://doi.org/10.48550/arxiv.2404.06634> (date of access: 21.09.2024).
2. Min B. Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey ACM Computing Surveys [Electronic resource] / B. Min et al. — 2023. — Mode of access: <https://doi.org/10.1145/3605943> (date of access: 21.09.2024).
3. Nayak A., Timmapathini H. P. LLM2KB: Constructing Knowledge Bases using instruction tuned context aware Large Language Models [Electronic resource] / A. Nayak, H. P. Timmapathini // arXiv.org. — Mode of access: <https://doi.org/10.48550/arxiv.2308.13207> (date of access: 21.09.2024).
4. Ranaldi L. A Trip Towards Fairness: Bias and De-Biasing in Large Language Models [Electronic resource] / L. Ranaldi et al. // arXiv.org. — Mode of access: <https://doi.org/10.48550/arxiv.2305.13862> (date of access: 21.09.2024).
5. Wang H. Knowledge-tuning Large Language Models with Structured Medical Knowledge Bases for Trustworthy Response Generation in Chinese ACM Transactions on Knowledge Discovery from Data [Electronic resource] / H. Wang et al. — 2024. — Mode of access: <https://doi.org/10.1145/3686807> (date of access: 21.09.2024).

6. Yang L. Give Us the Facts: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling IEEE Transactions on Knowledge and Data Engineering [Electronic resource] / L. Yang et al. — 2024. — Pp. 1–20. — Mode of access: <https://doi.org/10.1109/tkde.2024.3360454> (date of access: 21.09.2024).
7. Yin Z. Do Large Language Models Know What They Don't Know? [Electronic resource] / Z. Yin et al. // arXiv.org. — Mode of access: <https://doi.org/10.48550/arxiv.2305.18153> (date of access: 21.09.2024).
8. Zheng D. Large Language Models as Reliable Knowledge Bases? [Electronic resource] / D. Zheng, J. Z. Pan, M. Lapata // arXiv.org. — Mode of access: <https://doi.org/10.48550/arxiv.2407.13578> (date of access: 21.09.2024).

References

- Cooper, N., & Scholak, T. (n. d.). Perplexed: Understanding When Large Language Models are Confused. arXiv.org. <https://doi.org/10.48550/arxiv.2404.06634>.
- Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heintz, I., & Roth, D. (2023). Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey. ACM Computing Surveys. <https://doi.org/10.1145/3605943>.
- Nayak, A., & Timmapathini, H. P. (2023). LLM2KB: Constructing Knowledge Bases using instruction tuned context aware Large Language Models. arXiv.org. <https://doi.org/10.48550/arxiv.2308.13207>.
- Ranaldi, L., Zanzotto, F. M., Onorati, D., Venditti, D., & Ruzzetti, E. S. (2023). A Trip Towards Fairness: Bias and De-Biasing in Large Language Models. arXiv.org. <https://doi.org/10.48550/arxiv.2305.13862>.
- Wang, H., Zhao, S., Qiang, Z., Li, Z., Liu, C., Xi, N., Du, Y., Qin, B., & Liu, T. (2024). Knowledge-tuning Large Language Models with Structured Medical Knowledge Bases for Trustworthy Response Generation in Chinese. ACM Transactions on Knowledge Discovery from Data. <https://doi.org/10.1145/3686807>.
- Yang, L., Chen, H., Li, Z., Ding, X., & Wu, X. (2024). Give Us the Facts: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling. IEEE Transactions on Knowledge and Data Engineering, 1–20. <https://doi.org/10.1109/tkde.2024.3360454>.
- Yin, Z., Huang, X., Qiu, X., Wu, J., Guo, Q., & Sun, Q. (2023). Do Large Language Models Know What They Don't Know? arXiv.org. <https://doi.org/10.48550/arxiv.2305.18153>.
- Zheng, D., Pan, J. Z., & Lapata, M. (2024). Large Language Models as Reliable Knowledge Bases? arXiv.org. <https://doi.org/10.48550/arxiv.2407.13578>.

M. Androshchuk

MODERN APPROACHES TO USING KNOWLEDGE BASES TO ADDRESS THE CHALLENGES OF LARGE LANGUAGE MODELS

This paper examines the potential of integrating Large Language Models (LLMs) with knowledge bases to improve the accuracy and reliability of their responses. The advantages of such a combination are evaluated, particularly in reducing the risk of hallucinations – the phenomenon where models generate erroneous or fabricated information. Various methodologies for combining LLMs with knowledge bases are analyzed, along with their respective advantages and limitations. The prospects and challenges of implementing this technology in diverse fields—such as information retrieval, decision support, and automated content creation—are discussed. The paper presents an overview of the current state of research in this domain and delineates directions for future investigation. The integration of LLMs with knowledge bases represents a significant advancement in artificial intelligence, aiming one of the key concerns regarding LLMs—their tendency to generate inaccurate or fabricated information, commonly referred to as hallucinations. This approach leverages the vast language understanding and generation capabilities of LLMs while grounding their outputs in structured and verified information from knowledge bases. The synergy between these two technologies has the potential to significantly enhance the reliability and factual accuracy of AI-generated responses across a wide range of applications. The methodologies for combining LLMs with knowledge bases differ in their implementation and effectiveness. Some approaches involve pre-training LLMs on curated knowledge bases, while others reference knowledge bases externally during the inference process. Each method presents its own set of advantages and challenges, such as balancing computational efficiency against accuracy and maintaining the fluency of LLM outputs while adhering strictly to factual information. The application of this integrated technology extends beyond mere information retrieval, showing promise in complex decision support systems, automated content creation for specialized domains, and contributing to the advancement of explainable AI by providing traceable sources for generated information. As research in this area progresses, it is expected to open new avenues for developing more trustworthy and capable AI systems across various industries and academic disciplines.

Keywords: LLM, knowledge base, hallucinations, RAG.

Матеріал надійшов 23.09.2024



Глибовець М. М., Задохін Д. В., Дехтяр Б.-Я., Печкурова О. М.

ОБРОБЛЕННЯ ПРИРОДНОЇ МОВИ ЗА ДОПОМОГИ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ І МЕТОДІВ МАШИННОГО НАВЧАННЯ

У статті представлено аналіз можливостей великих мовних моделей для вирішення задач NLP. Описано особливості архітектури Transformer, що є основою для сучасних моделей з оброблення природної мови. Розглянуто окремі компоненти архітектури, їхню роль і важливість для роботи з людською мовою. Проведено порівняльний аналіз Transformer та інших наявних моделей для завдання машинного перекладу.

Проаналізовано фактори, що дали змогу створювати моделі з мільярдами параметрів — великі мовні моделі. Розглянуто сім'ю моделей Llama від Meta як приклад такої моделі. Особливу увагу було приділено моделям порівняно невеликого розміру, що можуть бути потужним і водночас доступним інструментом для оброблення природної мови.

Наразі глибинне машинне навчання і згорткові нейронні мережі (CNN) посідають важливе місце у сфері оброблення природної мови (NLP). Тому в статті оцінено ефективність використання його алгоритмів, моделей і методів для вирішення основних задач на прикладі задачі розпізнавання іменованих сутностей (NER).

Наведено методи глибинного навчання, які зробили революцію в NER, надавши можливість багаторазово краще розуміти контекст, фіксувати залежності на великих відстанях і ефективно використовувати великі обсяги даних. Проведено класифікацію моделей на основі трансформерів, що дають найкращі результати на цей момент. Зараз існує багато моделей, розроблених на основі трансформера.

Описано результати порівняння двох із найпоширеніших моделей — BERT (гарні результати у широкому спектрі завдань NLP, зокрема відповіді на запитання, класифікація тексту, висновки природною мовою, передбачення лівого і правого контексту слова) і GPT-3 (великі успіхи, як-от мовне моделювання, генерування тексту й відповіді на запитання). Ці моделі проходять попереднє навчання на великих текстових наборах даних, щоб вивчити фундаментальні мовні уявлення. Обидві моделі активно використовують потенціал тонкого налаштування.

Ключові слова: NLP, NER, CNN, машинне навчання, архітектура нейронних мереж, архітектура Transformer, машинний переклад, великі мовні моделі (Llama, BERT, GPT).

Вступ

Оброблення природної мови (NLP) — це один із напрямів досліджень у сфері штучного інтелекту, що полягає у створенні інструментів для оброблення, аналізу і генерування людської мови. Вирішення проблем NLP можуть потенційно стерти бар'єр у взаємодії між людиною і комп'ютером, адже для людей основний засіб комунікації — це мова, натомість як комп'ютерні системи в основному сприймають інформацію за строгими протоколами.

Можна виділити два основні підходи до розв'язання проблем NLP — статистичний і лінгвістичний. Лінгвістичний використовує лінгвістичні знання про граматику, морфологію, синтаксис і семантику [17]. Цей підхід намагається знайти структурні й логічні правила, які описують, як формується і використовується мова. За статистичним підходом моделі можуть вивчати закономірності в текстах, не маючи знань про граматику чи правила мови, просто аналізуючи велику кількість текстових даних [14].

Алгоритми машинного навчання належать до другого — статистичного підходу. Розвиток цих алгоритмів, зокрема поява глибоких нейронних мереж, дав сильний поштовх для розвитку NLP. До 2017 року рекурентні нейронні мережі [10], довготривала короткочасна пам'ять (LSTM) [4] і реку-

рентні мережі з воротами (GRU) [7] міцно закріпилися як передові підходи до задач моделювання послідовностей і трандукції, таких як моделювання мови та машинний переклад.

Основним недоліком рекурентних нейронних мереж є послідовна природа їхньої роботи, що не дає можливості повною мірою використати паралельні обчислення для тренування та інференції [12]. Вирішити цю проблему намагались за допомогою згорткових нейронних мереж [6; 9], однак зі збільшенням вікна уваги сильно збільшувалась кількість параметрів та обчислювальна вартість роботи таких мереж.

Архітектуру Transformer було розроблено для вирішення наведених проблем в 2017 р. [22]. Відтоді нейронні мережі, побудовані на цій архітектурі, стали state-of-the-art інструментом для вирішення задач з оброблення природної мови.

Архітектура Transformer і механізм уваги

В основі архітектури Transformer лежить механізм уваги. Він встановлює зв'язки між різними словами однієї послідовності і використовує ці зв'язки для обчислення їх представлень. Самоувагу (self-attention) успішно застосовували в різних задачах, як-от розуміння прочитаного, абстрактне узагальнення, текстові висновки та навчання незалежних від задач представлень речень, і до появи Transformer [15; 16].

Особливість архітектури полягає у тому, що вона **цілком** покладається на цей механізм, не використовуючи рекурентну структуру чи згорткові фільтри.

Для кращого розуміння важливості уваги для NLP розглянемо такий приклад у задачі машинного перекладу з англійської на французьку. Є два речення, що відрізняються лише останнім словом: "The animal didn't cross the street because it was too tired.", "The animal didn't cross the street because it was too wide.". У цих реченнях слово "it" стосується різних понять: тварина у першому випадку і вулиця в другому. Ці слова мають різні роди в французькій мові, і тому для правильного перекладу другої половини речення моделі необхідно проаналізувати зв'язки між словом "it" та всіма іншими словами у реченні.

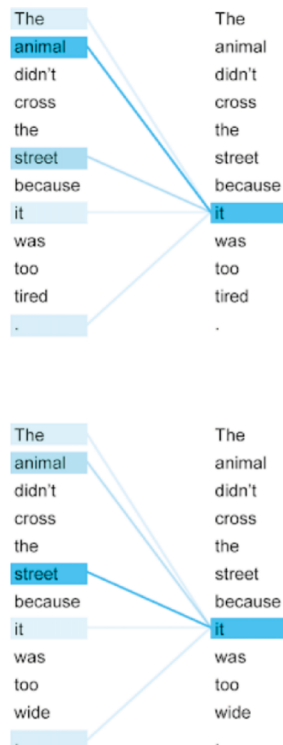


Рис. 1. Коефіцієнти уваги між словом it та іншими словами в реченнях

На рис. 1 зображено значення коефіцієнтів уваги між словами в обох реченнях, що були обраховані на одному з шарів мережі Transformer. Завдяки цій інформації модель змогла правильно перекласти обидва речення.

Оригінальна модель Transformer має структуру encoder-decoder [5]: складається із двох складових енкодера і декодера. Енкодер (зліва на рис. 2) перетворює вхідну символічну послідовність на послідовність векторних представлень такої самої розмірності.

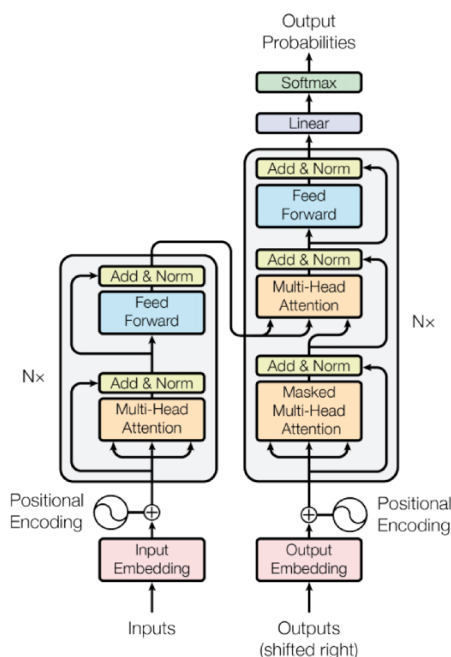


Рис. 2. Структура мережі Transformer

Роль декодера (справа на рис. 2) — із послідовності Z , згенерованої енкодером для вхідної послідовності tokenів, послідовно генерувати послідовність tokenів $Y = \{y_1, y_2, \dots, y_m\}$. Цей процес відбувається, поки не згенерований спеціальний token EOS (end of sequence) або не досягнуто граничної довжини відповіді.

Оброблення символічних вхідних даних починається з процесу word embedding — перетворення tokenів на багатовимірні (d_{model}) числові вектори. Це відбувається завдяки окремо навченим embedding моделям, що натреновані переводити семантичні зв'язки між словами у просторові між багатовимірними векторами.

Також застосовують позиційне кодування. Позиція слова у реченні може сильно впливати на його значення, особливо в деяких мовах. Тому інформація про позицію кожного токена у послідовності необхідна для правильного розуміння тексту.

У кінці декодера міститься додаткове лінійне перетворення, а також застосовується функція *soft max*:

$$\text{soft max}_i(X) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}.$$

На рис. 2 видно, що основною частиною енкодера і декодера є повторювані блоки. Кількість цих блоків N — гіперпараметр моделі, що напряду впливає на глибину мережі і кількість параметрів.

Блок енкодера складається з двох частин: Multi-Head Attention і Feed Forward. Перший буде розглянуто детальніше в наступних абзацах, а другий являє собою два лінійні перетворення із функцією активації Relu між ними. Для обох наявні залишкові з'єднання [11] і нормалізація [13].

Блок декодера має подібну структуру. Відмінністю є наявність двох Multi-Head Attention компонентів: перший відповідає за увагу над уже згенерованими токенами відповіді, другий на вхід також отримує представлення, створені енкодером.

Multi-Head Attention (досл. «багатоголовна увага») — підхід до уваги, вперше використаний саме в архітектурі Transformer. Ідея полягає в тому, щоб порівнювати між собою не повнорозмірні вектори, а їхні проєкції на простори менших розмірностей (d_k). При цьому, розмірність цих просторів має бути кратно меншою від розмірності векторних представлень tokenів (ембедингів): $d_{model} = d_k \cdot h$. Параметр h — це кількість центрів уваги.

У кожному «центрі уваги» незалежно тренуються матриці параметрів: $W_Q, W_K, W_V \in \mathbb{R}^{d_{model} \times d_k}$. Три матриці використовуються для трьох лінійних проєкцій векторів із простору $\mathbb{R}^{d_{model}}$ в простір $\mathbb{R}^{d_k}$. Таким чином, для кожного вхідного представлення токена розмірності d_{model} буде отримано три вектори розмірності d_k : Q, K, V .

Для обчислення коефіцієнтів уваги для кожного слова рахується скалярний добуток між вектором Q і векторами K всіх елементів послідовності. Скалярні добутки діляться на коефіцієнт $\sqrt{d_k}$ а також застосовується функція *soft max*, — таким чином отримуються коефіцієнти між 0 та 1. Далі ці коефіцієнти перемножуються з векторами V для отримання наступних представлень нейронної мережі.

Результати кожної голови уваги мають розмірність d_k , склеюються для отримання повнорозмірного представлення, а також зазнають ще одного лінійного перетворення:

$$MultiHead(X) = \text{concat}(head_1, \dots, head_h)W_0$$

$$head_i = \text{soft max} \left(\frac{XW_i^Q (XW_i^K)^T}{\sqrt{d_k}} \right) XW_i^V.$$

Усі матриці W — це треновані параметри.

У таблиці 1 наведено порівняльну характеристику різних типів шарів у нейронних мережах, що використовуються в NLP.

Таблиця 1

Порівняльна характеристика різних типів шарів у глибоких нейронних мережах для NLP

Тип шару	Обчислювальна складність	Послідовні операції
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(\log_k(n))$

Примітка: n — довжина послідовності, d — розмірність представлень токенів, k — розмір згорткового фільтра.

Із таблиці 1 видно, що обчислення представлень в шарі уваги може бути виконане за константну кількість послідовних операцій. Також обчислювальна складність лінійно залежить від розмірності моделі, на відміну від квадратичної залежності у інших моделях.

Це означає, що для обчислення уваги можна дуже ефективно застосовувати паралельні обчислення, а також що шари уваги менше чутливі до збільшення розмірності моделі.

Розробники архітектури Transformer показали, що, використовуючи тільки механізм уваги, можна досягти кращих результатів у обробленні природної мови за меншої (на декілька порядків) складності навчання та інференції. Також, використовуючи кластери графічних процесорів, можна порівняно швидко тренувати моделі на великих обсягах текстових даних.

Великі мовні моделі. Огляд моделей Llama

Оптимізованість архітектури Transformer для паралельних обчислень, а також висока ефективність архітектури для задач оброблення природної мови дали змогу тренувати моделі з величезною кількістю параметрів на терабайтах текстових даних.

Оригінальна модель Transformer мала приблизно 50 мільйонів параметрів і показала найкращі на той час результати з машинного перекладу. Розмір деяких сучасних мовних моделей уже перевищує трильйон параметрів.

Враховуючи величезні ресурси, що необхідні для тренування таких мереж, розробляти такі моделі мають можливість лише великі технологічні корпорації з майже необмеженими ресурсами. Розробляють додаткові оптимізації та нові підходи: моделі, створені декілька років тому, вже вважають застарілими і неактуальними.

Водночас з'являються відкриті моделі, що доступні для безкоштовного скачування, тренування на власних даних та використання. Прикладом можуть слугувати моделі Llama від компанії Meta [20; 21].

Llama — це група моделей різного розміру і версій. Першу версію випустила Meta AI у лютому 2023 р., вона тренувана на 1,4 трильйона tokenів. Модель було навчено на широкому спектрі джерел даних, із підтримкою 20 мов із латинським і кириличним алфавітами. Навчальні дані також брали із відкритих репозиторіїв на платформі Github, із книжок і наукових публікацій.

У найновішій версії Llama 3.2 моделі навчались вже на 15 трильйонах tokenів. Серед моделей останньої версії доступні маленькі моделі розміром 1 і 3 мільярди, які можна запускати навіть на мобільних пристроях і які можуть виконувати нескладні задачі.

Більші моделі — 7B, 13B, 33B і 70B (B — від англ. billion). На кінець 2024 р. кожна із цих моделей набирає найвищі бенчмарки серед моделей своєї розмірної категорії. Найбільша модель має 405 мільярдів параметрів і становить конкуренцію проприетарним моделям більшого розміру. Моделі від 33B також треновані обробляти графічні зображення.

Оскільки моделі відкриті, будь-хто може завантажити, дотренувати та публікувати свої версії моделей. У мережі вже доступні сотні кастомізацій мережі, кожна з яких спеціалізується для вирішення конкретної задачі.

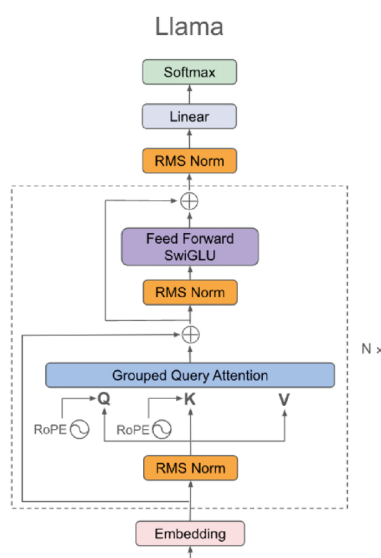


Рис. 3. Структура моделі Llama

На рис. 3 зображено структурну діаграму Llama. Найпомітніша відмінність від попередньо розглянутої архітектури — немає окремого енкодера. Моделі, що спеціалізуються саме на генерації тексту, часто використовують таку структуру — decoder only Transformer. Інша принципова відмінність полягає у механізмі уваги.

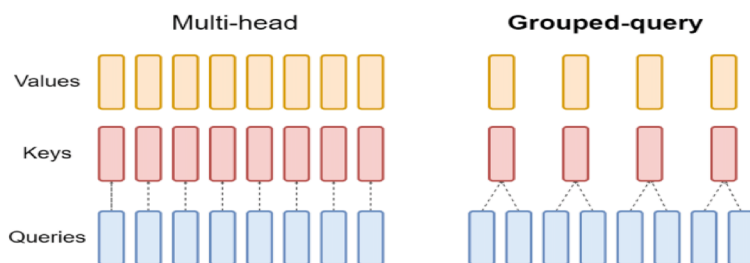


Рис. 4. Grouped-query Attention vs Multi-head Attention

Із другої версії в Llama застосовується механізм уваги Grouped-query Attention (рис. 4) [3]. Multi-head Attention, що застосовувався в оригінальній мережі Transformer, передбачав унікальні лінійні перетворення для всіх трьох векторів Q, K і V для кожної голови уваги. Grouped-query Attention пропонує таку оптимізацію: голови уваги згруповані, перетворення Q залишаються унікальними для кожної із них, а K і V — спільні у кожній групі. Така оптимізація дещо негативно впливає на точність моделі, але дуже сильно зменшує кількість тренуваних параметрів.

Окрім механізму уваги, змінились також методи позиційного кодування (rotary position embedding [19]), нормування (RMS normalisation [24]) та функції активації в лінійних шарах мережі (SwiGLU [18]).

Ефективність використання моделей глибинного навчання в NLP на прикладі NER

Нині глибинне машинне навчання посідає важливе місце у сфері оброблення природної мови (NLP). Важливо оцінити ефективність використання його алгоритмів, моделей і методів для вирішення основних задач, наприклад задачі розпізнавання іменованих сутностей (NER) [1].

Розпізнавання іменованих сутностей (NER) — це підгалузь оброблення природної мови (NLP), яка зосереджена на ідентифікації та класифікації конкретних сутностей із текстового вмісту. NER працює з важливими деталями тексту, відомими як іменовані сутності — окремі слова, фрази або послідовності слів — шляхом ідентифікації та категоризації їх у заздалегідь визначені групи. Категорії охоплюють різноманітне коло тем, згаданих у тексті, зокрема імена людей, географічне розташування, назви організацій, дати, події та навіть конкретні кількісні значення, такі як гроші та відсотки.

Задача розпізнавання іменованих сутностей (ЗРІС) має довгу історію спроб розв'язання (ведення великої колекції імен, застосування регулярних виразів, оснований на правилах і списках шаблонів на основі токенів слів та POS-тегів), що робить її більш придатною для ілюстрації використання машинного навчання. Ця задача залишається актуальною і зараз. Підходи до її розв'язку з використанням глибинного машинного навчання активно розвиваються [2]. Серед сфер застосування традиційно виділяють пошук інформації, рекомендаційні системи. У сфері машинного навчання ЗРІС зазвичай називають «маркуванням (розміткою) послідовностей».

Пояснимо цю концепцію. Розглянемо речення: «Я ... кошу». Якщо класифікатор аналізує слово «кошу», тоді для вирішення, чи воно стосується зачіски людини, берега річки або знаряддя праці, потрібно розглядати контекст речення (враховувати слова навколо неї). Починають працювати класифікатори послідовностей. Вони враховують слова, які передують поточному слову і йдуть за ним. Наприклад, якщо попереднє слово вказує на річку, існує більша ймовірність того, що це слово є іменником, що належить до типу берега річки. Такий підхід доповнюється інженерією функцій, де конкретні атрибути та інформація про дані витягуються вручну для покращення продуктивності моделі. До поширених функцій належать: характеристика слова, контекст, синтаксична інформація, шаблони слів, морфологічні подробиці.

Серед базових методів реалізації тут традиційно виділяють машини опорних векторів (SVM), дерева рішень та умовні випадкові поля (CRF).

Машинне навчання використовують, коли сутності бувають різних типів і не завжди дотримуються послідовної схеми. Якщо існує великий набір анотованих даних, тоді можна навчити модель, яка зможе добре узагальнювати різні шаблони сутностей. Цей підхід проблемний на етапі розроблення функцій, значно залежить від якості та розміру набору даних, потребує значних обчислювальних ресурсів і додаткового оброблення зашумлених даних. Звідси впливає неідеальна продуктивність. Безсумнівною перевагою цього підходу є значна адаптивність, добре усвідомлення контексту та покращене запам'ятовування.

У цьому розділі ми зосередимось на уточненні ефективності використання моделей глибинного навчання у розв'язанні ЗРІС.

Рекурентні нейронні мережі (РНМ) мають спеціальний механізм пам'яті, який дає їм змогу розглядати слова, що з'являються до і після аналізованого слова. Однак у них виникає проблема зникаючих і вибухових градієнтів, коли аналізуються довгі послідовності. Це впливає на ефективність фіксації контекстної інформації. Усувають ці проблеми мережі довгої короткочасної пам'яті, або LSTM (long short-term memory).

Архітектура LSTM — це тип РНМ зі складнішою коміркою пам'яті. Така комірка вирішує проблему зникаючого градієнта, явно керуючи потоком інформації. Комірка має спеціальний клапан забуття для прийняття рішення про те, яку частину інформації з попередніх комірок слід зберігати, вхідний клапан для контролю кількості нової інформації, що має зберігатися, та вихідний клапан для визначення ступеня впливу поточного вмісту на вихід.

Використання такої пам'яті дає їм можливість ефективно фіксувати далекосяжні залежності в тексті. Така архітектура добре застосовується до вхідних даних, які охоплюють одне або два речення. Проблеми з залежностями на великій відстані тут виникають, коли справа доходить до оброблення великої контекстної семантики. Тоді ці мережі можуть неефективно вловлювати нюанси дуже

довгих текстових послідовностей. Недоліком архітектури LSTM є обмежена можливість розпаралелювання і виявлення довготривалих залежностей.

Однією з ключових переваг трансформерної архітектури є те, що її можна навчати на великих обсягах даних паралельним способом, що робить її дуже масштабованою. Багато випадків використання NER включають вхідний текст, який охоплює кілька абзаців і сторінок. Трансформер ідеально підходить для цього.

Зараз існує багато моделей, розроблених на основі трансформера. Розглянемо одні з найпоширеніших — BERT і GPT-3.

BERT — це модель, розроблена Google, яка досягла гарних результатів у широкому спектрі завдань NLP, як-от відповіді на запитання, класифікація тексту і висновок природною мовою. Вона складається з двонаправленого кодера-трансформера, що означає, що модель навчена передбачати лівий і правий контекст цього слова. BERT попередньо навчається на великих обсягах текстових даних за допомогою двох завдань: моделювання замаскованої мови та прогнозування наступного речення.

GPT-3 — це мовна модель, розроблена OpenAI, яка досягла вражаючих результатів у широкому спектрі завдань NLP, як-от мовне моделювання, генерація тексту та відповіді на запитання. Вона була (до недавнього часу) перемагала Мегатрон-Тюрінг NLG з параметрами 530B [23]) найбільшою мовною моделлю з 175 мільярдами параметрів, навченою на різноманітному діапазоні текстових даних. GPT-3 — це генеративна модель, яка генерує новий текст на основі заданого запиту. На відміну від BERT, вона не така багатоцільова, навчена генерувати текст, і, отже, не такий хороша, як спеціалізовані моделі NER. Спроби використовувати GPT для NER [8] існують, але результати не є значними.

Спробуємо порівняти BERT і GPT.

Ні GPT, ні BERT безпосередньо не виконують конкретні завдання у своїй базовій формі. Вони проходять попереднє навчання на масивних текстових наборах даних, щоб вивчити фундаментальні мовні уявлення. Обидві моделі активно використовують потенціал тонкого налаштування: їх можна точно налаштувати для різних подальших завдань, таких як NER, аналіз настроїв, відповіді на запитання та узагальнення тексту.

Зрозуміло, що є і відмінності. Насамперед це архітектура: GPT оснований на архітектурі авторегресійного трансформера, де він передбачає наступне слово на основі попередніх; BERT — на двонаправленій архітектурі трансформера, що дає можливість аналізувати все речення одночасно. Окрім цього, у них різна мета попередньої підготовки: GPT — передусім навчений для генерації мови; BERT — для розуміння мовного контексту. У таблиці 2 узагальнено відмінності між GPT і BERT.

Таблиця 2

Відмінності між GPT і BERT

	GPT	BERT
Архітектура	Авторегресійний трансформер	Двонаправлений трансформер
Мета попередньої підготовки	Передусім підготовлений для генерації мови	Навчений розуміти мовний контекст
Для чого добре підходить	Творче письмо, генерація коду та діалог	Аналіз настроїв, відповіді на запитання та NER

Згорткові нейронні мережі (CNN) спочатку було розроблено для оброблення зображень. Тим не менш, їх також успішно застосовують для завдань оброблення природної мови, таких як класифікація тексту (насправді, включно з NER), аналіз настроїв і мовне моделювання.

У NLP CNN зазвичай використовують для оброблення послідовностей вкладень слів змінної довжини, де кожне слово представлено у вигляді щільного вектора у високовимірному просторі. Вхідна послідовність спочатку пропускається через один або кілька згорткових шарів, які застосовують фільтри фіксованої ширини до вхідної послідовності для вилучення локальних об'єктів. Вихідні дані згорткових шарів потім пропускаються через один або кілька шарів об'єднання, які зменшують дискретизацію карт об'єктів, щоб зменшити їхню розмірність.

У NLP використовується кілька варіацій архітектури CNN, залежно від конкретного завдання і характеру вхідних даних. Одним із поширених підходів є використання кількох фільтрів різного розміру для фіксації різних ознак n-грам, таких як уніграми, біграми і триграми. Вихідні дані кожного фільтра потім об'єднуються в один вектор ознак і пропускаються через один або кілька повністю пов'язаних шарів для класифікації або регресії.

Іншим підходом є ієрархічна архітектура CNN, де вхідна послідовність спочатку розбивається на менші підпослідовності, такі як речення або абзаци. Кожна підпослідовність обробляється незалеж-

но CNN нижчого рівня. Вихідні дані CNN нижчого рівня потім об'єднуються і обробляються CNN вищого рівня, щоб зафіксувати глобальні особливості та залежності між підпоследовностями.

CNN мають низку переваг для завдань NLP. Вони є обчислювально ефективними та можуть паралельно обробляти вхідні последовності, що робить їх придатними для великомасштабних наборів даних. Вони також можуть фіксувати локальні особливості та закономірності в последовності введення, що може бути корисним для таких завдань, як аналіз настроїв і класифікація тексту. Однак CNN менш ефективні в моделюванні довгострокових залежностей і последовних зв'язків у текстовій последовності, ніж рекурентні нейронні мережі (PHM) і трансформерні моделі, які краще підходять для мовного моделювання і завдань машинного перекладу.

Загалом, CNN у NER насамперед цікаві з історичної точки зору; за таблицею лідерів paperswithcode.com, найефективніша архітектура на основі CNN розташована на 58-й позиції; це гібридна модель, що використовує комбіновану потужність CNN і LSTM, яка була представлена ще в 2015 р. Отже, за останні 9 років не було зроблено жодних покращень у NER з архітектурами на основі CNN.

Для прикладу розглянемо хронологію використання NER в Electronic Health Records (див. рис. 5, запозичений із [21]). Як бачимо, глибоке навчання з'являється у 2017 р. До 2019 р. панівною архітектурою стала двонаправлена довготривала короткочасна пам'ять (BiLSTM). Однак з 2021 р. архітектура BERT та її варіанти стали основною моделлю NER, що застосовується до ЕМК, і ця тенденція зберігається донині.

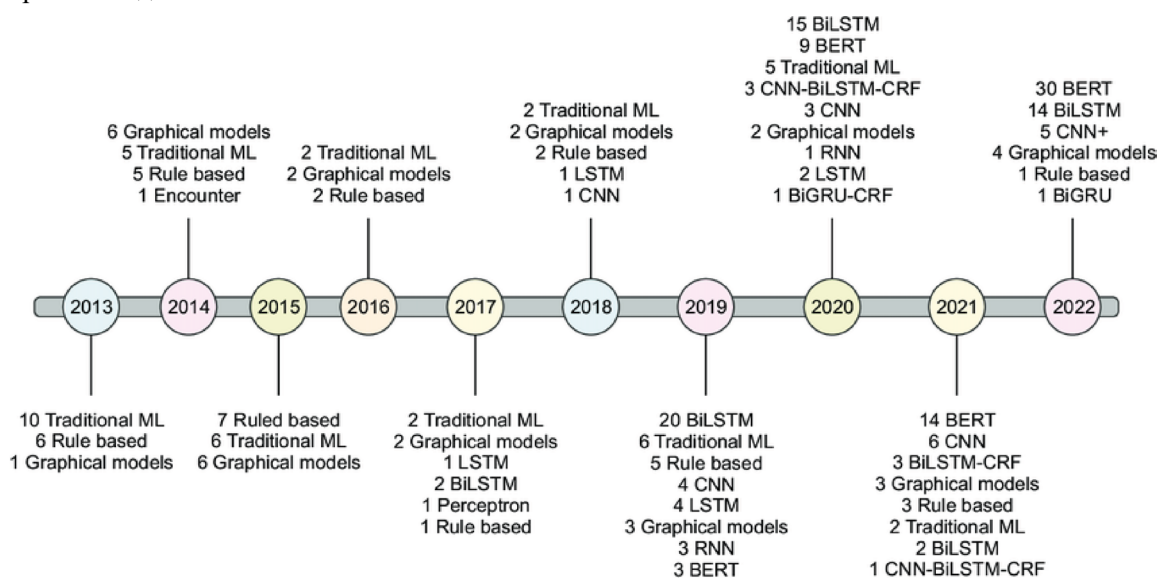


Рис. 5. Еволюція підходів

Висновки

Поява архітектури Transformer відкрила можливості у вирішенні задач оброблення природної мови, що ще зовсім недавно були недоступними.

Моделі з багатьма мільярдами параметрів можуть виконувати різноманітні задачі над текстом. Додаткові оптимізації, а також розвиток технічних засобів для паралельних обчислень призвели до появи ефективніших, більших і розумніших моделей. Навчання та використання гігантських мереж є складним і дорогим процесом, однак існують альтернативні, доступніші моделі. Моделі Llama від компанії Meta є хорошим прикладом такого доступного рішення з високим відношенням розмір / ефективність.

Поява такого доступного NLP засобу відкриває нові горизонти в розробленні комп'ютерних систем, що здатні взаємодіяти з людиною за допомогою природної мови.

З іншого боку, не варто цілком покладатись на великі мовні моделі при вирішенні задач. У моделей є ціла низка недоліків, такі як можливість галюцинацій, неможливість знайти джерело інформації та причин тієї чи тієї відповіді.

Оптимальним сценарієм використання є гібридні системи, в яких велика мовна модель (або декілька) є лише одним з компонентів, що виконує вузькоспеціалізовану задачу з оброблення природної мови.

Методи глибокого навчання зробили революцію в NER, надавши методи з набагато кращим розумінням контексту, здатністю фіксувати залежності на великих відстанях і ефективно використовувати великі обсяги даних. Моделі на основі трансформерів дають найкращі результати на цей момент, переважною архітектурою є BERT. Однак дослідження в цій галузі все ще тривають, і нові методи (наприклад, підказки) незабаром можуть витіснити наявні. Незважаючи на те, що трансформери є найбільш поширеними, все ще використовуються інші, більш базові моделі, наприклад LSTM, CNN, і навіть методи, основані на правилах.

Список літератури

1. Глибовець А. М. Автоматизований пошук іменованих сутностей у нерозмічених текстах українською мовою / А. М. Глибовець // Штучний інтелект. — 2017. — № 2. — С. 45–51.
2. Глибовець А. М. Архітектура системи моделей глибоких нейронних мереж для знаходження подібності об'єктів в гетерогенному середовищі / А. М. Глибовець, Т. І. Легіневич // V Міжнародно науково-практична конференція «Обчислювальний інтелект». — Ужгород, 2019. — С. 249–250.
3. Ainslie Joshua. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints [Electronic resource] / Joshua Ainslie et al. — 2023. — Mode of access: arXiv:2305.13245.
4. Cheng Jianpeng. Long short-term memory-networks for machine reading [Electronic resource] / Jianpeng Cheng, Li Dong, and Mirella Lapata. — 2016. — Mode of access: arXiv:1601.06733.
5. Cho Kyunghyun. Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs [Electronic resource] / Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 1406.1078, 2014. — Mode of access: <https://aclanthology.org/D14-1179/>.
6. Chollet Francois. Xception: Deep learning with depthwise separable convolutions [Electronic resource] / Francois Chollet. — 2016. — Mode of access: arXiv:1610.02357.
7. Chung Junyoung. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs [Electronic resource] / Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, Yoshua Bengio. — Mode of access: <https://arxiv.org/abs/1412.3555>.
8. Durango Maria. Named Entity Recognition in Electronic Health Records: A Methodological Review / Maria Durango, Silva Ever Torres, Andres Orozco-Duque // Healthcare Informatics Research. — 2023. — Vol. 29. — Pp. 286–300. — <https://doi.org/10.4258/hir.2023.29.4.286>.
9. Gehring Jonas. Convolutional sequence to sequence learning [Electronic resource] / Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, Yann N. Dauphin. — 2017. — Mode of access: arXiv:1705.03122v2.
10. Graves Alex. Generating sequences with recurrent neural networks [Electronic resource] / Alex Graves. — 2013. — Mode of access: arXiv:1308.0850.
11. He Kaiming. Deep residual learning for image recognition / Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. — 2016. — Pp. 770–778.
12. Hochreiter Sepp. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies [Electronic resource] / Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber. — 2001. — Mode of access: https://www.researchgate.net/publication/2839938_Gradient_Flow_in_Recurrent_Nets_the_Difficulty_of_Learning_Long-Term_Dependencies.
13. Lei Ba Jimmy. Layer normalization [Electronic resource] / Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton. — 2016. — Mode of access: arXiv:1607.06450.
14. Manning Christopher. Foundations of statistical natural language processing / Christopher Manning, Schütze Hinrich. — MIT press, 1999.
15. Parikh Ankur. A decomposable attention model. In Empirical Methods in Natural Language Processing [Electronic resource] / Ankur Parikh, Oscar Täckström, Dipanjan Das, Jakob Uszkoreit // Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. — 2016. — Pp. 2249–2255. — Mode of access: <https://aclanthology.org/D16-1244/>.
16. Paulus Romain. A deep reinforced model for abstractive summarization [Electronic resource] / Romain Paulus, Caiming Xiong, Richard Socher. — 2017. — Mode of access: arXiv:1705.04304.
17. Sag Ivan A. Linguistic theory and natural language processing / Ivan A. Sag // Natural Language and Speech: Symposium Proceedings Brussels. — November 26/27, 1991. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1991.
18. Shazeer Noam. Glu variants improve transformer [Electronic resource] / Noam Shazeer. — 2020. — Mode of access: arXiv:2002.05202.
19. Su Jianlin. Roformer: Enhanced transformer with rotary position embedding / Su Jianlin et al. // Neurocomputing. — 2024. — 568: 127063.
20. Touvron Hugo. Llama 2: Open foundation and fine-tuned chat models [Electronic resource] / Hugo Touvron et al. — 2023. — Mode of access: arXiv:2307.09288.
21. Touvron Hugo. Llama: Open and efficient foundation language models [Electronic resource] / Hugo Touvron et al. — 2023. — Mode of access: arXiv:2302.13971.
22. Vaswani A. Attention is all you need / A. Vaswani // Advances in Neural Information Processing Systems. — 2017.
23. Wang S. Named entity recognition via large language models [Electronic resource] / S. Wang, X. Sun, X. Li, R. Ouyang, F. Wu, T. Zhang, ... & G. Wang. — 2023. — Mode of access: arXiv:2304.10428.
24. Zhang Biao. Root mean square layer normalization / Biao Zhang, Rico Sennrich // Advances in Neural Information Processing Systems. — 2019. — Vol. 32.

References

- Ainslie, Joshua, et al. (2023). GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245.
- Cho, Kyunghyun, Merriënboer, Bart van, Gulçehre, Çağlar, Bougares, Fethi, Schwenk, Holger, & Bengio, Yoshua. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs, 1406.1078. <https://aclanthology.org/D14-1179/>.
- Chollet, Chung, Junyoung, Gülçehre, Çağlar, Cho, Kyunghyun, & Bengio, Yoshua. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs. 1412.3555. <https://arxiv.org/abs/1412.3555>.
- Durango, Maria, Torres, Silva Ever, & Orozco-Duque, Andres. (2023) Named Entity Recognition in Electronic Health Records: A Methodological Review. *Healthcare Informatics Research*, 29, 286–300. <https://doi.org/10.4258/hir.2023.29.4.286>.
- Francois. (2016). Xception: Deep learning with depthwise separable convolutions. arXiv:1610.02357.

- Gehring, Jonas, Auli, Michael, Grangier, David, Yarats, Denis, & Dauphin, Yann N. (2017). Convolutional sequence to sequence learning. arXiv:1705.03122v2.
- Graves, Alex. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850.
- Hlybovets, A. M. (2017). Avtomatyzovanyi poshuk imenovanykh sutnostei u nerozmichenykh tekstakh ukrainskoiu movoiu. *Shtuchnyi intelekt*, 2, 45–51 [in Ukrainian].
- Hlybovets, A. M., Lehinevych, T. I. (2019). Arkhitektura systemy modelei hlybokykh neironnykh merezh dlia znakhodzhennia podobnosti ob'ektiv v heteroennomu seredovyshchi. In *Mizhnarodno naukovopraktychna konferentsiia "Obchysliuvnyi intelekt"* (pp. 249–250) [in Ukrainian].
- Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, & Jürgen Schmidhuber. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. https://www.researchgate.net/publication/2839938_Gradient_Flow_in_Recurrent_Nets_the_Difficulty_of_Learning_Long-Term_Dependencies.
- Jianpeng, Cheng, Li Dong, & Mirella, Lapata. (2016). Long short-term memory-networks for machine reading. arXiv:1601.06733.
- Kaiming, He, Zhang, Xiangyu, Ren, Shaoqing, & Sun, Jian. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778).
- Lei Ba, Jimmy, Kiros, Jamie Ryan, & Hinton, Geoffrey E. (2016). Layer normalization. arXiv:1607.06450.
- Manning, Christopher, & Hinrich, Schütze. (1999). *Foundations of statistical natural language processing*. MIT press.
- Parikh Ankur, Täckström, Oscar, Das, Dipanjan, & Uszkoreit, Jakob. (2016). A decomposable attention model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 2249–2255). <https://aclanthology.org/D16-1244/>.
- Paulus, Romain, Xiong, Caiming, & Socher, Richard. (2017). A deep reinforced model for abstractive summarization. arXiv:1705.04304.
- Sag, Ivan A. (1991). Linguistic theory and natural language processing. *Natural Language and Speech: Symposium Proceedings Brussels*. Springer Berlin Heidelberg.
- Shazeer, Noam. (2020). Glu variants improve transformer. arXiv:2002.05202.
- Su, Jianlin, et al. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568: 127063.
- Touvron, Hugo, et al. (2023a). Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.
- Touvron, Hugo, et al. (2023b). Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.
- Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.
- Wang, S., Sun, X., Li, X., Ouyang, R., Wu, F., Zhang, T., ... & Wang, G. (2023). Gpt-ner: Named entity recognition via large language models. arXiv preprint arXiv:2304.10428.
- Zhang, Biao, & Rico Sennrich. (2019) Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

M. Glybovets, D. Zadokhin, B. Dekhtiar, O. Pyechkurova

NATURAL LANGUAGE PROCESSING USING LARGE LANGUAGE MODELS AND MACHINE LEARNING METHODS

The article analyzes the capabilities of large language models in solving NLP tasks. It describes the features of the Transformer architecture, which serves as the foundation for modern natural language processing models. The individual components of the architecture, their roles, and their significance for working with human language are discussed. A comparative analysis of the Transformer and other existing models in the context of machine translation task is provided.

Factors that have enabled the development of models with billions of parameters—known as large language models—are analyzed. The Llama model family from Meta is reviewed as an example of such models. Special attention is given to smaller-scale models, which can be powerful yet accessible tools for natural language processing.

Currently, deep machine learning and convolutional neural networks (CNN) hold an important place in the field of natural language processing (NLP). Therefore, the article evaluates the effectiveness of these algorithms, models, and methods for solving key tasks, using the named entity recognition (NER) task as an example.

Deep learning methods have revolutionized NER, providing a significantly better understanding of context, capturing dependencies over long distances, and enabling the effective use of large datasets. A classification of Transformer-based models that currently yield the best results is provided. Currently, many models have been developed based on the Transformer architecture.

We describe the results of comparing two of the largest BERT models (which have achieved strong results across a wide range of NLP tasks, including question answering, text classification, natural language interference, and context prediction) with GPT-3 (which has demonstrated impressive successes in language modeling, text generation, and question answering). These models are pre-trained on large-scale textual datasets to learn fundamental linguistic representations. Both models leverage fine-tuning to enhance their performance.

Keywords: NLP, NER, CNN, machine learning, neural network architecture, Transformer architecture, machine translation, large language models (Llama, BERT, GPT).

Матеріал надійшов 23.09.2024



Волинець Є. А., Хмель С. М., Печкурова О. М.

ВІЗУАЛІЗАЦІЯ КОГНІТИВНИХ СТАНІВ НА ОСНОВІ RASPBERRY PI ДЛЯ БІОЛОГІЧНОГО ЗВОРОТНОГО ЗВ'ЯЗКУ В РЕАЛЬНОМУ ЧАСІ ЗА ДОПОМОГОЮ НЕЙРО-КОМП'ЮТЕРНОГО ІНТЕРФЕЙСУ

Нейрокомп'ютерний інтерфейс на основі ЕЕГ (Electroencephalography, електроенцефалографія) — це технологія, яка дає змогу встановлювати взаємодію між мозком людини і зовнішніми пристроями, такими як комп'ютери чи робототехніка. Основні принципи такого інтерфейсу:

1. Зчитування електричної активності мозку за допомогою електродів, розміщених на поверхні скальпа. Ці сигнали ЕЕГ відображають коливання потенціалів нейронів.
2. Оброблення та аналіз ЕЕГ-сигналів за допомогою алгоритмів машинного навчання, щоб визначити певні патерни, пов'язані з конкретними розумовими станами чи намірами користувача.
3. Переклад цих розпізнаних патернів у команди для управління зовнішніми пристроями, такими як комп'ютер, протези чи інтерфейси віртуальної реальності.

Ключові переваги нейрокомп'ютерних інтерфейсів на основі ЕЕГ — це можливість безпосереднього керування пристроями за допомогою думок та намірів, не вдаючись до традиційних методів введення, наприклад клавіатури чи миші. Це відкриває нові перспективи для людей з обмеженими можливостями, а також для інноваційних застосувань у сферах нейрореабілітації, ігор, віртуальної реальності тощо.

У статті описано дослідження можливостей використання ЕЕГ пристрою для збору та візуалізації інформації про когнітивний стан користувача пристрою. Дослідження проводили на електроенцефалографічному приладі Emotiv INSIGHT (5-канальна система електроенцефалографії (ЕЕГ) із напісхими полімерними датчиками), та Raspberry Pi 4B. В рамках дослідження було розроблено прототип, що дає можливість візуалізовувати показники про емоційний і ментальний стан користувача за допомогою LED, що під'єднані до Raspberry Pi.

Ключові слова: нейрокомп'ютерні інтерфейси, BCI, Raspberry Pi 4B.

Вступ

У сучасному світі стрімкого технологічного розвитку спостерігається тенденція до все тіснішого поєднання таких сфер, як штучний інтелект (AI), інтерфейси мозок-комп'ютер (BCI) та апаратне забезпечення. Ці технології мають величезний потенціал, коли використовуються у взаємодоповнювальний спосіб, відкриваючи нові горизонти для інновацій та вирішення складних проблем.

Завданням цієї статті є продемонструвати приклад інтеграції BCI та апаратних рішень на базі одноплатного комп'ютера Raspberry Pi. У статті розглянуто можливості використовувати дані про когнітивні стани людини, отримані з BCI-пристрою, для динамічного керування LED. Такий підхід може бути застосований у широкому спектрі сценаріїв, зокрема у системах «розумного будинку», освітніх і тренувальних застосунках.

Розроблення архітектури системи візуалізації когнітивних станів

Система має кілька компонентів, які працюють разом, щоб забезпечити збір, оброблення і відображення даних, зібраних з ЕЕГ-пристрою (рис. 1).

Компоненти системи: Emotiv INSIGHT, Raspberry Pi 4B, 7 LED, 7 резисторів 10 кОм, дроти для з'єднання.

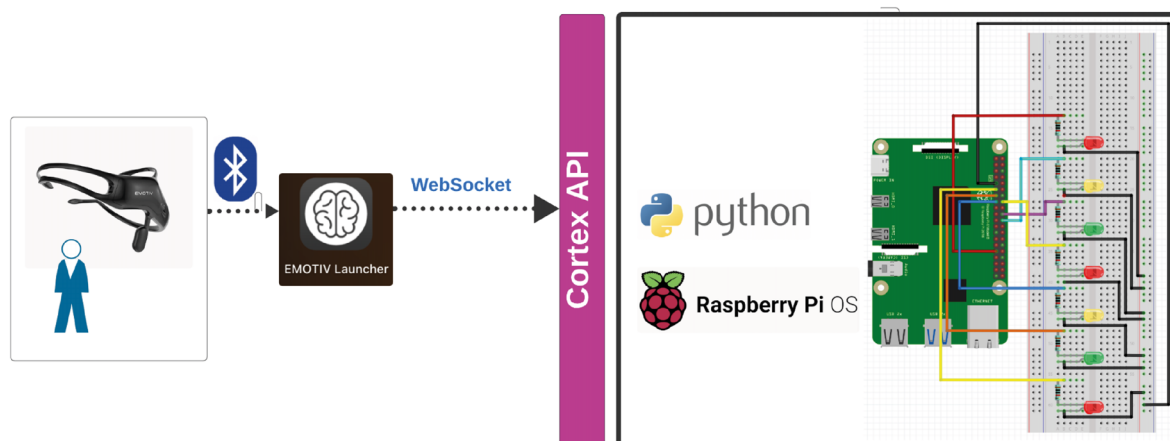


Рис. 1. Загальне архітектурне рішення

Засоби комунікації: EEG прилад з'єднується з RaspberryPi за допомогою BlueTooth, сигнали на Cortex API передаються за допомогою WebSocket.

Візуалізація сигналів відбувається за допомогою LED, що приєднані до GPIO портів RaspberryPi.

Реалізація прототипу

Для збору даних було використано бібліотеку для роботи з пристроєм Cortex, що надає виробник пристрою.

Для роботи з пристроєм має бути встановлено Launcher (програмне забезпечення для приєднання пристрою), що також надає виробник пристрою. Для реалізації прототипу було вибрано Emotiv INSIGHT, оскільки виробник Emotiv надає програмне забезпечення для приєднання пристрою до Raspberry Pi (OS – Debian version 116 32-bit).

Попередньою умовою для реалізації проєкту є придбання ліцензії, яка дає змогу отримати доступ до електроенцефалографічних (EEG) даних і Performance Metrics — показників когнітивної діяльності.

Performance Metrics — це показники продуктивності EMOTIV, як-от увага, стрес, зацікавленість, хвилювання, розроблені за допомогою власних алгоритмів компанії-виробника, машинного навчання, ІІІ та набору анонімних даних EEG. Розроблені власною командою неврологів і психологів EMOTIV, показники ефективності EMOTIV побудовані на основі даних, зібраних за допомогою встановлених парадигм нейронауки, які викликають бажаний когнітивний стан, наприклад увагу на основі наборів даних EEG від учасників контрольованих нейро наукових експериментів за допомогою гарнітур EMOTIV EPOC, Insight і MN8 [3].

Для взаємодії з гарнітурою використовують Cortex API [1]. Для розроблення проєкту попередньо створено client_id та client_secret на ресурсі для розробників EMOTIV, що є необхідною умовою.

Реалізовано робочий процес API [2] (рис. 2), створено підписку на отримання даних.

Для відображення отриманих даних за допомогою LED створено схему ланцюга та під'єднання (рис. 3).

Взаємодія з LED відбувається шляхом передавання даних на GPIO порти Raspberry Pi. GPIO (General-Purpose Input/Output) означає введення / виведення загального призначення. Ці контакти дають змогу під'єднувати електричні компоненти до Raspberry Pi і взаємодіяти з ними за допомогою програмного забезпечення.

Використано такі порти:

- GPIO 17 (Physical Pin 11)
- GPIO 27 (Physical Pin 13)
- GPIO 22 (Physical Pin 15)
- GPIO 23 (Physical Pin 16)
- GPIO 24 (Physical Pin 18)
- GPIO 25 (Physical Pin 22)
- GPIO 6 (Physical Pin 31)

Note

Before user could do record or mental command following steps should be done

- login to emotivapp manually once
- use client_id / client_secret in third party app
- request access for third party app with code
- accept third party app manually on emotivapp
- connect with headset
- authorize to get cortex token
- create a new session

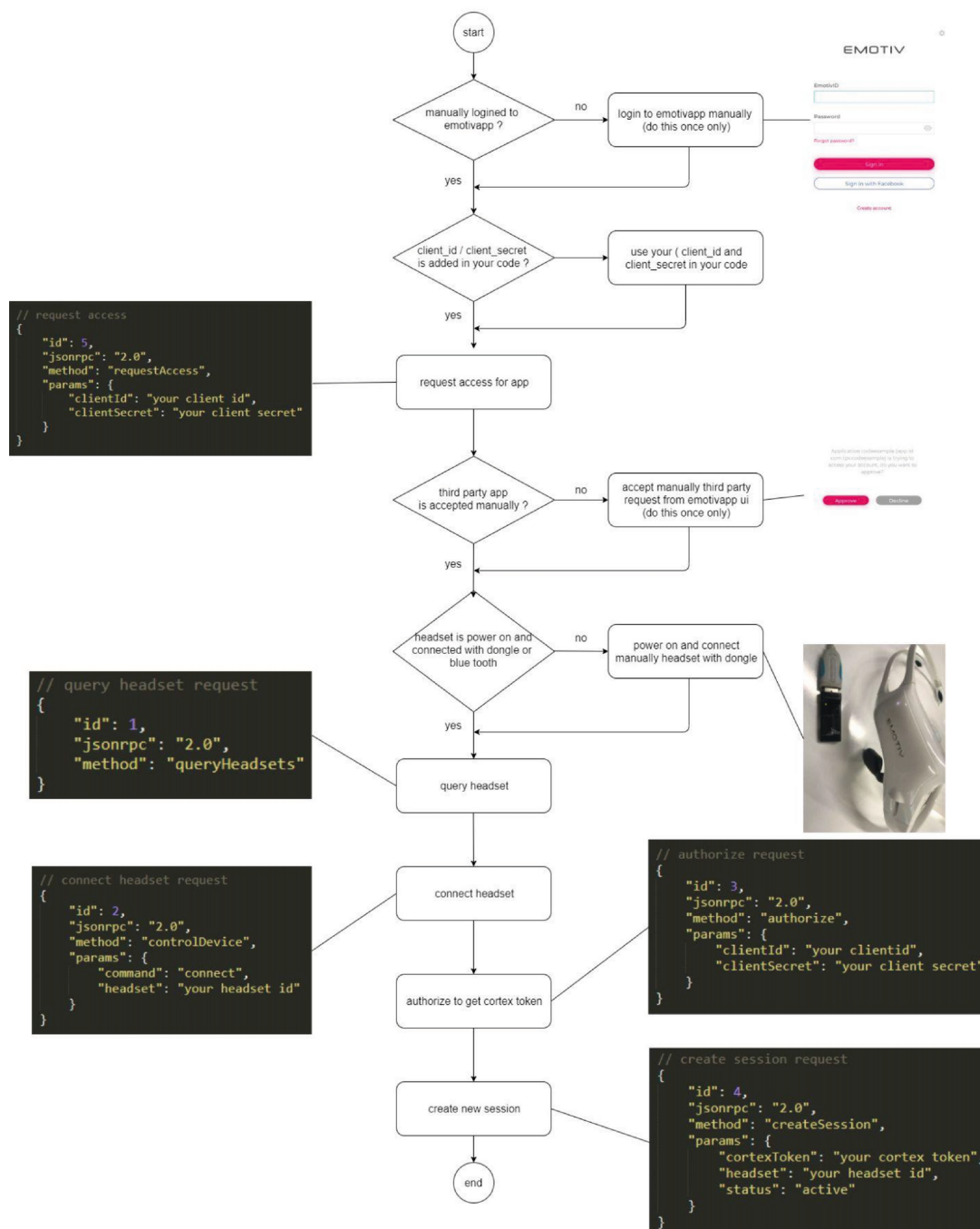


Рис. 2. Схема процесів Cortex API

Інтенсивність сигналу змінюється за допомогою використання методу `led_change_duty_cycle`.

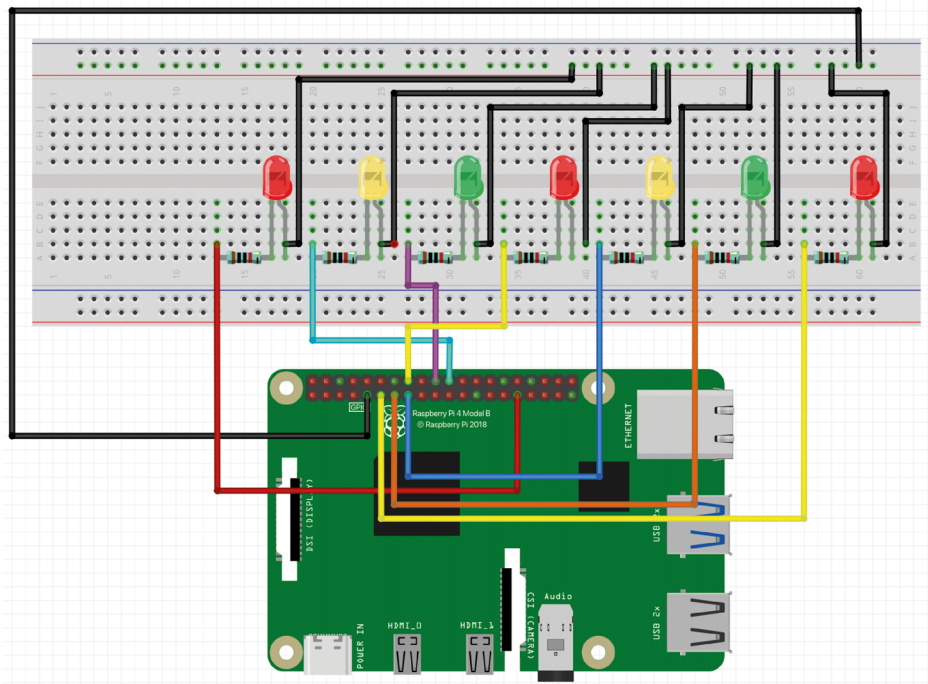


Рис. 3. Схема під'єднання ланцюга до Raspberry Pi

```
import RPi.GPIO as GPIO
import time
from cortex import Cortex
class Subscribe():
    def __init__(self, app_client_id, app_client_secret, **kwargs):
        «»»
        Constructs cortex client and bind a function to handle subscribed data streams
        If you do not want to log request and response message , set debug_mode =
        False. The default is True
        «»»
        print("Subscribe __init__")
        self.c = Cortex(app_client_id, app_client_secret, debug_mode=True, **kwargs)
        self.c.bind(create_session_done=self.on_create_session_done)
        self.c.bind(new_data_labels=self.on_new_data_labels)
        self.c.bind(new_met_data=self.on_new_met_data)
        self.c.bind(inform_error=self.on_inform_error)
        # Set the GPIO mode to BCM
        GPIO.setmode(GPIO.BCM)
        # Set up the LED pins
        self.led_pins = [17, 27, 22, 23, 24, 25, 6]
        self.led_pwm = []
        # Set up each pin as a PWM output
        for pin in self.led_pins:
            GPIO.setup(pin, GPIO.OUT)
            globals()[f"pwm_{pin}"] = GPIO.PWM(pin, 100) # 100 Hz PWM
            globals()[f"pwm_{pin}"].start(0) # Start the PWM at 0% duty cycle
        def start(self, streams, headsetId=''):
            self.streams = streams
            if headsetId != '':
                self.c.set_wanted_headset(headsetId)
            self.c.open()
        def sub(self, streams):
            self.c.sub_request(streams)
        def on_new_data_labels(self, *args, **kwargs):
```

```
data = kwargs.get('data')
stream_name = data['streamName']
stream_labels = data['labels']
print('{} labels are : {}'.format(stream_name, stream_labels))
def led_change_duty_cycle(self, pin, value):
    pwm = globals()[f"pwm_{pin}"]
    pwm.ChangeDutyCycle(value)
def on_new_met_data(self, *args, **kwargs):
    «»»
    To handle performance metrics data emitted from Cortex and control LED brightness.
    «»»
    data = kwargs.get('data')
    met = data['met']
    print("MET data:", met)
    print("MET engagement:", met[1])
    print("MET lex:", met[4])
    print("MET ex:", met[3])
    print("MET str:", met[6])
    print("MET relax:", met[8])
    print("MET interest:", met[10])
    print("MET foc:", met[12])

    engagement = 0
    excitement = 0
    lex = 0
    stress = 0
    relax = 0
    interest = 0
    focus = 0
    try:
        engagement = met[1]
    except ValueError:
        print("'eng' not found in met data. Using default value.")
    try:
        excitement = met[3]
    except ValueError:
        print("'exc.isActive' not found in met data. Using default value.")
    try:
        lex = met[4]
    except ValueError:
        print("'lex' not found in met data. Using default value.")
    try:
        stress = met[6]
    except ValueError:
        print("'str.isActive' not found in met data. Using default value.")
    try:
        relax = met[8]
    except ValueError:
        print("'relax' not found in met data. Using default value.")

    try:
        interest = met[10]
    except ValueError:
        print("'foc.isActive' not found in met data. Using default value.")
    try:
        focus = met[12]
    except ValueError:
        print("'foc.isActive' not found in met data. Using default value.")
```

```

    # Control LED brightness based on performance metrics
    self.set_led_brightness(engagement, excitement, lex, stress, relax, interest, focus)
    def set_led_brightness(self, engagement, excitement, lex, stress, relax, interest, focus):
        «»»
        Set the brightness of the LEDs based on the performance metrics.
        «»»
        # engagement
        if isinstance(engagement, (int, float)):
            self.led_change_duty_cycle(self.led_pins[0], engagement)
        # excitement
        if isinstance(excitement, (int, float)):
            self.led_change_duty_cycle(self.led_pins[1], excitement)
        # lex
        if isinstance(lex, (int, float)):
            self.led_change_duty_cycle(self.led_pins[2], lex)
        # stress
        if isinstance(stress, (int, float)):
            self.led_change_duty_cycle(self.led_pins[3], stress)
        # relax
        if isinstance(relax, (int, float)):
            self.led_change_duty_cycle(self.led_pins[4], 100)
        # interest
        if isinstance(interest, (int, float)):
            self.led_change_duty_cycle(self.led_pins[5], 100)
        # focus
        if isinstance(focus, (int, float)):
            self.led_change_duty_cycle(self.led_pins[6], 100)
    def on_create_session_done(self, *args, **kwargs):
        print('on_create_session_done')
        # subscribe data
        self.sub(self.streams)
    def on_inform_error(self, *args, **kwargs):
        error_data = kwargs.get('error_data')
        print(error_data)
def main():
    your_app_client_id = 'YOUR_CLIENT_ID'
    your_app_client_secret = 'YOUR_CLIENT_SECRET'
    s = Subscribe(your_app_client_id, your_app_client_secret)
    streams = ['met']
    s.start(streams)
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        pass
    finally:
        for pin in led_pins:
            pwm = globals()[f'pwm_{pin}']
            pwm.stop()
        GPIO.cleanup() # Ensure GPIO cleanup occurs
if __name__ == '__main__':
    main()

```

requirements.txt:

```
certif==2024.8.30
charset-normalizer==3.4.0
idna==3.10
python-dispatch==0.2.2
requests==2.32.3
RPi.GPIO==0.7.1
urllib3==2.2.3
websocket-client==1.8.0
```

Приклад зібраних даних:

```
met labels are : ['eng.isActive', 'eng', 'exc.isActive', 'exc', 'lex', 'str.
isActive', 'str', 'rel.isActive', 'rel', 'int.isActive', 'int', 'foc.isActive',
'foc']
```

```
MET data: [True, 0.070344, True, 0.633367, 0.0, True, 0.224075, True, 0.052446,
True, 0.252325, True, 0.094797]
```

Висновки

У цій роботі було продемонстровано прототип системи, що інтегрує нейрокомп'ютерний інтерфейс на основі ЕЕГ-пристрою, та апаратну платформу Raspberry Pi для динамічної візуалізації когнітивних станів користувача. Цей підхід має низку важливих переваг:

1. Можливість безпосереднього контролю зовнішніх пристроїв, таких як світлодіоди, за допомогою мозкових сигналів, що дає змогу створювати системи біологічного зворотного зв'язку.
2. Гнучкість і масштабованість рішення на базі Raspberry Pi, що робить його придатним для широкого кола застосувань, від розумних будинків до нейроосвітніх тренажерів.
3. Можливість інтеграції з іншими технологіями, як-от штучний інтелект, для глибшого аналізу та класифікації когнітивних станів.

Результати проведеного дослідження демонструють, що поєднання BCI-технологій та апаратних рішень на базі Raspberry Pi відкриває перспективи для створення інтелектуальних систем, орієнтованих на потреби людини. Подальший розвиток цього напрямку може суттєво вплинути на якість життя, продуктивність праці та розширення можливостей людини.

Серед недоліків виявлено такі: сигнал слабкий і швидко втрачається у міру висихання сенсорів. Досягти 100 % з'єднання вдається рідко, максимальне отримане значення було приблизно 60 %.

Для розроблення та користування прототипом потрібно попереднє отримання clientId.

Для отримання EEG Performance metrics потрібна ліцензія. Для усунення недоліків потрібно замінити прилад.

Загалом представлений прототип демонструє перспективність поєднання BCI-технологій та апаратних рішень на базі Raspberry Pi як основи для створення систем людино-комп'ютерної взаємодії.

Список літератури

1. Cortex API Documentation [Electronic resource] // Emotiv. — Mode of access: <https://emotiv.gitbook.io/cortex-api/> (date of access: 2.11.2024).
2. Overview of API [Electronic resource] // Emotiv. — Mode of access: <https://emotiv.gitbook.io/cortex-api/overview-of-api-flow> (date of access: 20.11.2024).
3. Performance metrics [Electronic resource] // Emotiv. — Mode of access: https://www.emotiv.com/pages/performance-metrics?_pos=1&_sid=dbfbb5e33&_ss=r (date of access: 22.11.2024).

References

Emotiv. *Cortex API documentation*. <https://emotiv.gitbook.io/cortex-api/>.

Emotiv. *Overview of API flow*. <https://emotiv.gitbook.io/cortex-api/overview-of-api-flow>.

Emotiv. *Performance metrics*. https://www.emotiv.com/pages/performance-metrics?_pos=1&_sid=dbfbb5e33&_ss=r.

Y. Volynets, S. Khmel, O. Pyechkurova

VISUALIZING COGNITIVE STATES ON A RASPBERRY PI PLATFORM FOR REAL-TIME BIOFEEDBACK USING A BRAIN-COMPUTER INTERFACE

This research article presents a prototype system that integrates an EEG-based brain-computer interface (BCI) and the Raspberry Pi hardware platform for the real-time visualization of a user's cognitive states. The key advantages of this approach include:

- 1. The ability to directly control external devices, such as LEDs, using brain signals, which enables the creation of biofeedback systems.*
- 2. The flexibility and scalability of the Raspberry Pi-based solution, making it suitable for various applications, from smart home systems to educational training.*

The study utilized the Emotiv INSIGHT EEG system, which is a 5-channel EEG devices with semi-dry polymer sensors, alongside a Raspberry Pi 4B microcontroller. As part of the research, a prototype was developed to visualize the user's emotional and mental state indicators through LEDs connected to the Raspberry Pi.

The results of this work demonstrate the potential of combining BCI technologies and Raspberry Pi-based hardware solutions as a foundation for developing human-centered intelligent systems. Further development in this area can significantly impact the quality of life, productivity, and human capability expansion.

One identified limitation in the study is the quality of the EEG signal, which is relatively weak and deteriorates over time as the sensors dry out. Achieving a stable and reliable connection was challenging, with the maximum observed value reaching approximately 60%. Additionally, developing and using the prototype requires prior acquisition of a client ID, and access to EEG data and "Performance Metrics" depends on obtaining a license. Overcoming these limitations may require replacing the EEG device with a more reliable alternative.

Overall, the prototype showcases the potential of integrating BCI technologies and Raspberry Pi-based hardware as a foundation for developing human-computer interaction systems. Future research and enhancements in this field could lead to significant progress in several domains, including neurorehabilitation, smart home applications, and educational training scenarios.

Keywords: *neuro-computer interfaces, BCI, Raspberry Pi.*

Матеріал надійшов 01.12.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

Чорней Р. К.

ЛОКАЛЬНЕ КЕРУВАННЯ В МЕРЕЖАХ ГОРДОНА — НЬЮЕЛЛА

Запропоновано модифікацію мережі Гордона — Ньюелла з локальною та синхронною взаємодією, яка обслуговує клієнтів у замкнутому режимі. Система околів задається за допомогою деякого скінченного графа вузлів системи. Запропоновано процедуру знаходження оптимальних нерандомізованих стратегій керування для систем із критерієм усереднених в одиницю часу витрат.

Ключові слова: мережа Гордона — Ньюелла, локальна та синхронна взаємодія, керовані випадкові процеси, процеси прийняття рішень.

1. Вступ

У цій роботі конструюються моделі просторово розподілених систем із неперервним часом, які керуються особами, що приймають рішення, які діють на локальному рівні та мають лише інформацію про стан системи навколо свого положення. При цьому особливу увагу приділяємо марковським системам.

Це узгоджується з основними напрямками досліджень у літературі: оптимальне (послідовне) керування стохастичними системами в більшості випадків описано моделями процесу Маркова. Однак існує багато літератури з керування процесами, які не використовують властивість пам'яті марковських систем. Введення непоказникових, відповідно геометричних, термінів перебування процесів призводить до розгляду напівмарковських процесів і марковських процесів відновлення.

Найдавнішими роботами зі стохастичної динамічної оптимізації для напівмарковських процесів (які іноді називають процесами відновлення з винагородою) є роботи [7; 9; 10]. Більш сучасні роботи з напівмарковських процесів прийняття рішень можна знайти, наприклад, в [1; 12; 14–16] і літературних джерелах, на які в них посилаються. Загальну конструкцію напіввідновлювальних керованих процесів і напівмарковських керованих процесів із застосуванням до систем черг можна знайти в роботі Китаєва і Рикова [13].

Метою цієї роботи є визначення доповненої мережі Гордона — Ньюелла, для якої виконуються умови синхронної та локальної взаємодії вузлів, і знаходження оптимальних режимів її функціонування.

2. Локальність і синхронність доповненої мережі Гордона — Ньюелла

Розглядаємо стохастичні процеси з неперервним часом із загальним компактним простором значень, який структурований деяким основним графом, як це було визначено в роботі [3]. Кожна вершина графа перебуває у деякому локальному стані системи, такому, що глобальний простір станів процесу є добутком (індексованим графом) локальних просторів станів. Таким чином, граф визначає околову структуру станів описаних систем. Ці околи визначають локальні взаємодії координат просторового процесу. Тоді для фіксованого моменту часу випадковий стан системи, відповідно до описання випадкового процесу, є випадковим полем відносно графа околів.

Припускаємо, що процес має напівмарковську властивість часу і що ядра переходу процесу мають просторову марковську властивість відносно основного графа. Додатково створюємо структуру керування для оптимізації розвитку системи в часі. Оптимальність тут розуміється відносно критерію асимптотично усередненої в часі винагороди.

Розглядатимемо тільки ступінчасті керування, як це визначено в розділі 5 роботи [2]. Необхідні означення стохастичних процесів, які розвиваються в просторі і часі, подані в роботі [3]. Припускаємо, що час буде неперервним, однак прийняття рішень дозволяється лише в моменти стрибків про-

цесу (детальніше див. [1; 17]). Випадкові політики представляються, як звичайно, умовною незалежною структурою. Припускаємо також, що таку структуру може бути знайдено в ядрах переходу стрибкоподібних ланцюгів (синхронізовані ядра). В роботах [2; 3] введено абстрактні умови існування стаціонарних детермінованих стратегій у класі локальних стратегій, і потім показується, що за деяких припущень слабкої гладкості ці умови виконуються.

Для систем із локально взаємодіючими координатами формальне означення структури взаємодії та простору станів визначається за допомогою ненаправленого скінченного графа околів. Припустимо, що локальний простір станів X_i — компактний метричний простір зі зліченною базою, наділений борелівською σ -алгеброю $\mathcal{X}_i$. Тоді $X := \prod_{i \in V} X_i$ — глобальний простір станів системи, наділений добутковою σ -алгеброю $\mathcal{X}_V = \sigma\{\prod_{i \in V} \mathcal{X}_i\} =: \mathcal{X}$.

Еволюція в часі системи описується стохастичним процесом $\eta = (\eta^t)$ з простором станів $(X, \mathcal{X}) = (\prod_{i \in V} X_i, \sigma\{\prod_{i \in V} \mathcal{X}_i\})$ з випадковими полями як одновимірними маргіналами в часі.

Припускаємо, що час неперервний, $t \in \mathbb{R}_+ = [0, +\infty)$. Індекс k при η_k^t позначатиме належність вершині k , η_k^t таким чином позначає маргінальний розподіл у момент часу t вершини k деякого векторнозначного процесу $\eta = (\eta^t : t \geq 0)$. Такі процеси, що змінюються в просторі та часі, причому просторова змінна структурується деяким графом, були досліджені в роботах [2; 3] за допомогою використання методів ланцюгів Маркова.

У роботі [3] усувається припущення, що в часі процеси змінюються без запам'ятовування (експоненціально чи геометрично розподіленого); тобто досліджуються напівмарковські процеси та марковські процеси відновлення. Наведемо їхні означення.

Означення 1 [марковський процес відновлення і напівмарковський процес]. Розглянемо стохастичний стрибкоподібний процес $\eta = (\eta^t : t \geq 0)$ з простором станів X і неперервною праворуч траєкторією, яка має лівосторонні границі (cadlag шляхи), визначений послідовністю $(\xi, \tau) = \{(\xi^n, \tau^n), n = 0, 1, \dots\}$. $\mathbb{R}_+$ -значні випадкові величини τ^n є тривалостями часу між стрибками, а послідовність $\{\xi^n, n = 0, 1, \dots\}$ — послідовність станів процесу, отриманих одразу після моментів стрибків. Далі більш точно (детально).

Нехай $\sigma = \{\sigma^n : n = 0, 1, \dots\}$ — зростаюча послідовність моментів стрибків, визначена співвідношеннями $\sigma^0 = 0$ і $\sigma^n = \sum_{i=0}^{n-1} \tau^i$, $n > 0$. Тоді, якщо час $t \in [\sigma^n, \sigma^{n+1})$, то $\eta^t = \xi^n$, $n \in \mathbb{N}$.

Якщо (для будь-якого n) $P(\xi^k, k=0,1,\dots,n-1, \sigma^0, \tau^k, k=0,1,\dots,n-2)$ — майже всюди визначених умовних розподілів послідовності $(\xi, \tau) = \{(\xi^n, \tau^n), n = 0, 1, \dots\}$ виконується

$$\begin{aligned} P\{\xi^{n+1} \in C, \tau^n \leq s \mid \xi^k = x^k, k = 0, 1, \dots, n, \dots, \sigma^0 = 0, \tau^k = s^k, k = 0, 1, \dots, n-1\} \\ = P\{\xi^{n+1} \in C, \tau^n \leq s \mid \xi^n = x^n\}, \quad n > 0, \end{aligned}$$

$P(\xi^k, k=0,1,\dots,n-1, \sigma^0, \tau^k, k=0,1,\dots,n-2)$ — майже всюди, і вони незалежні від n , то (ξ, τ) називається однорідним марковським процесом відновлення.

Процес $\eta = (\eta^t : 0 \leq t < \sum_{i=0}^{\infty} \tau^i)$ називають однорідним напівмарковським процесом. Перехідну функцію

$$\tilde{Q}(C, s \mid x) = P\{\xi^{n+1} \in C, \tau^n \leq s \mid \xi^n = x\}$$

називають напівмарковським ядром обидвох: марковського процесу відновлення і напівмарковського процесу.

Припускаємо, що задано наступне ядро переходу, яке визначає умовний час перебування процесу з урахуванням теперішнього та наступного станів системи:

$$T(\cdot \mid x, y) = P\{\tau^n \leq \cdot \mid \xi^n = x, \xi^{n+1} = y\}, \quad x, y \in X,$$

де з правого боку умовний розподіл часу τ^n перебування процесу в стані x , якщо наступним станом буде y , де термін τ^n визначається $P^{\xi^n, \xi^{n+1}}$ — майже напевно.

Для систем із локально взаємодіючими координатами в роботах [2; 3] було введено формальне означення структури взаємодії і простору станів, тобто спеціальну структуру на графі взаємодії.

Припускаємо, що локальні простори станів X_i — компактні метричні простори зі зліченною базою, наділені борелівською σ -алгеброю $\mathcal{X}_i$.

Еволюція в часі такої системи з глобальним простором станів

$$(X, \mathcal{X}) = \left(\prod_{i \in V} X_i, \sigma \left\{ \prod_{i \in V} \mathcal{X}_i \right\} \right)$$

описується стохастичним процесом $\eta = (\eta^t : t \in [0, +\infty))$ з випадковими полями як одномірними маргіналями в часі. Таким чином, $\eta^t = (\eta_k^t : k \in V)$, де індекс k в η_k^t позначає належність вершині k , і отже, η_k^t позначає маргінальний стан в момент часу t в вершині k векторозначного процесу $\eta = (\eta^t : t \geq 0)$.

Спочатку розглянемо марковські стрибкоподібні процеси, а згодом спеціальні напівмарковські процеси з неперервним часом, описані в розділі 5 роботи [2]. Важливою особливістю марковських процесів є те, що час затримки у відповідних станах процесів залишається без пам'яті, тобто експоненційно розподіляється з параметрами, що залежать від фактичного стану. Маємо

$$\begin{aligned} T(s | x, y) &= P\{\tau^n \leq s \mid \xi^n = x, \xi^{n+1} = y\} \\ &= P\{\tau^n \leq s \mid \xi^n = x\} \\ &= 1 - e^{-\lambda(x)s}, \quad x, y \in X, s \geq 0. \end{aligned}$$

Означення 2 [марковський стрибкоподібний процес]. Однорідний марковський стрибкоподібний процес із неперервним часом є напівмарковським процесом $\eta = (\eta^t : t \geq 0)$ з простором станів $(X, \mathcal{X})$ і неперервними праворуч шляхами, що мають лівосторонні границі (cadlag шляхи), який визначається послідовністю $(\xi, \tau) = \{(\xi^n, \tau^n), n = 0, 1, \dots\}$. Припускаємо, що процес $\eta = (\eta^t : t \geq 0)$ має стаціонарні перехідні ймовірності p :

$$p(t; x, C) := P\{\eta^{t+s} \in C \mid \eta^s = x\} \quad \forall s \in \mathbb{R}_+ = [0; \infty), x \in X, C \in \mathcal{X}.$$

Зазначимо, що права частина не залежить від s . X -значна послідовність $\{\xi^n, n = 0, 1, \dots\}$ — послідовність станів процесу, що виникає одразу після моментів стрибків. $\mathbb{R}_+$ — значна випадкова величина $\{\tau^n, n = 1, 2, \dots\}$ — міжстрибкові інтервали часу. Якщо $\{\xi^n = x_n, n = 0, 1, \dots\}$ задано, то $\{\tau^n, n = 1, 2, \dots\}$ — послідовність незалежних експоненційно розподілених випадкових величин із параметрами $\lambda(x_n)$. Якщо бути точнішим щодо конструкції, то:

- послідовність моментів стрибків $\eta - \sigma = \{\sigma^n : n = 0, 1, \dots\}$, що задається як $\sigma^0 = 0$ і $\sigma^n = \sum_{i=1}^n \tau^i$, $n \in \mathbb{N}$. Тоді для $t \in [\sigma^n, \sigma^{n+1})$ маємо $\eta^t = \xi^n$, $n \in \mathbb{N}$;
- однокрокові перехідні ймовірності вкладеного ланцюга стрибків — марковське ядро $(Q(C | x) : C \in \mathcal{X}, x \in X)$ послідовності $\{\xi^n = \eta^{\sigma^n}, n = 0, 1, \dots\}$;
- функція інтенсивності $\lambda : X \rightarrow \mathbb{R}_+$ — обмежена вимірна функція від x з $\lambda(x) < \Lambda < \infty$ для всіх $x \in X$.

Перехідні ймовірності p однорідного марковського стрибкоподібного процесу $\eta = (\eta^t : t \geq 0)$ задовольняють для $x \notin C$ рівняння

$$p(t; x, C) = \lambda(x) \int_0^t e^{-\lambda(x)s} ds \int_X Q(dy | x) p(t-s; y, C) \quad (1)$$

і для $x \in C$

$$p(t; x, C) = e^{-\lambda(x)t} + \lambda(x) \int_0^t e^{-\lambda(x)s} ds \int_X Q(dy | x) p(t-s; y, C). \quad (2)$$

Звідси слідує, що

$$\frac{\partial p(t; x, C)}{\partial t} = \lambda(x) \left[-p(t; x, C) + \int_X Q(dy | x) p(t; y, C) \right].$$

Однорідний марковський стрибкоподібний процес із неперервним часом $\eta = (\eta^t : t \in \mathbb{R}_+)$ можна визначити як марковський процес, для якого слабкий інфінітезимальний оператор

$$[\mathfrak{A}f(\cdot)](x) = \lambda(x) \left[-f(x) + \int_X Q(dy | x) f(y) \right]$$

визначений на його області визначення, тобто на підмножині множини $M(X)$ вимірних обмежених функцій на X .

У некерованому випадку основний інтерес зосередимо на стрибкоподібних процесах взаємодії, для яких марковська умова визначається так само, як у випадку загальних напівмарковських процесів через вбудовані ланцюги стрибків. Наступне означення просто нагадує означення для марковської системи.

Означення 3 [синронні і локальні перехідні ядра]. Марковський стрибкоподібний процес із неперервним часом $\eta = (\eta^t : t \geq 0)$ з пов'язаним процесом $(\xi, \tau) = \{(\xi^n, \tau^n), n = 0, 1, \dots\}$ і простором станів $(X, \mathcal{X}) = (\prod_{i \in V} X_i, \sigma\{\prod_{i \in V} \mathcal{X}_i\})$ є локальним, якщо виконується

$$P\{\xi_k^{n+1} \in C_k \mid \xi^n = x^n, \dots, \xi^0 = x^0\} = P\{\xi_k^{n+1} \in C_k \mid \xi_{N(k)}^n = x_{N(k)}^n\}, \quad (3)$$

і синхронним, якщо виконується

$$P\{\xi_k^{n+1} \in C_k \mid \xi^n = x^n\} = \prod_{k \in K} P\{\xi_k^{n+1} \in C_k \mid \xi^n = x^n\}, \forall K \subset V, x^n \in X. \quad (4)$$

Тоді η називають марковським стрибкоподібним процесом із неперервним часом і локально взаємодіючими синхронними компонентами на (Γ, X) , або, коротко, марковським стрибкоподібним випадковим полем.

Далі опишемо приклад із теорії масового обслуговування, де застосовують локальне керування. Детальніше про мережеву теорію див. [4; 11]. У роботі [3] показано, що мережеві процеси з чергами, хоча і демонструють жорстку локальну поведінку щодо природної базової структури графів, не будуть повністю відповідати вказаній властивості (3) і наступній (4). Як і в [3], припустимо, що описані марковські процеси мають неперервні праворуч шляхи з лівосторонніми границями (cadlag шляхи) і що динаміка процесів повністю задана відповідними Q -матрицями $Q = (q(m, n) : m, n \in \mathbb{N})$. Наведемо опис ізольованої станції технічного обслуговування, яка згодом визначатиме локальну структуру вузлів мережі.

Означення 4 [залежний від стану сервер із чергою]. Розглянемо сервер типу $M/M/1/\infty$, куди клієнти (що не розрізняються) потрапляють по одному. Клієнти, які потрапляють на незайнятий сервер, негайно обслуговуються, тоді як клієнти, які прибувають на зайнятий сервер, заходять у зал очікування нескінченної ємності. Якщо не вказано інше, черга організована відповідно до режиму «першим прийшов, першим обслужили» (FCFS): якщо час обслуговування закінчується, клієнт негайно залишає систему, а клієнт на чолі черги, якщо такий є, йде на обслуговування, інші клієнти в черзі пересуваються на одну позицію вперед. Завжди вважатимемо, що ці зсуви тривають нульовий час.

Якщо в системі є n клієнтів, тоді час до наступного надходження клієнта проходить з інтенсивністю $\lambda(n)$, і (якщо $n > 0$) клієнт, що обслуговується, обслуговується з інтенсивністю $\mu(n)$. Враховуючи кількість клієнтів у системі, яку далі називатимемо довжиною черги, фактичний залишковий час обслуговування і залишковий час очікування надходження клієнта не залежать від минулого та не залежать один від одного.

Позначимо $\eta = (\eta^t : t \geq 0)$ процес випадкової довжини черги системи.

Зауваження 1. З означення випливає, що для опису залежної від стану еволюції черги одного сервера процесом Маркова достатньо записати довжину черги, тобто η , визначена у відповідному ймовірнісному просторі $(\Omega, \mathcal{F}, P)$ з простором станів $\mathbb{N}$, — це строго марковський процес.

Робимо висновок: час перебування в стані n експоненціально розподілений з параметром $\lambda(n) + \mu(n)$, і подальше рішення про стрибок не залежить від часу перебування, а з імовірністю $\frac{\lambda(n)}{\lambda(n) + \mu(n)}$ наступний стрибок, спричинений переходом до стану $n + 1$ через надходження ще одного клієнта, з імовірністю $\frac{\mu(n)}{\lambda(n) + \mu(n)}$ наступний стрибок, викликаний закінченням терміну обслуговування (подія відправлення клієнта) і переходом до стану $n - 1$.

Виявляється, клас моделей, описаний в означенні 1, збігається з класом (залежних від стану) процесів народження — загибелі.

Теорема 1 [4]. Процес η , залежної від стану сервера довжини черги, є процесом народження — загибелі. Процес η ергодичний тоді й тільки тоді, коли

$$G = \sum_{n=0}^{\infty} \prod_{i=1}^n \frac{\lambda(i-1)}{\mu(i)} < \infty.$$

Якщо η ергодичний, то його граничний розподіл $\pi = (\pi(n): n \in \mathbb{N})$ з

$$\pi(n) = G^{-1} \prod_{i=1}^n \frac{\lambda(i-1)}{\mu(i)}, \quad n \in \mathbb{N},$$

де $G < \infty$ — нормуюча константа.

Для подальшого обговорення структури класичних мереж масового обслуговування з урахуванням вимог до локальної та синхронізованої поведінки взаємодії марковських випадкових процесів і випадкових полів Маркова наведемо означення з [6].

Означення 5 [мережа Гордона — Ньюелла]. Мережа Гордона — Ньюелла є мережею зі станціями обслуговування (вузлами), пронумерованими множиною $\{1, 2, \dots, J\} := V$. Вузол j — система, залежна від стану сервера, з нескінченним залом очікування і режимом FCFS (див. означення 1 щодо деталей механізму обслуговування). Є $I > 0$ клієнтів, що обслуговуються циклічно згідно з незвідною матрицею Маркова $R = (r(i, j): i, j = 1, \dots, J)$.

Клієнт на виході з вузла i вибирає з імовірністю $r(i, j) \geq 0$ відвідування наступного вузла j і негайно туди потрапляє на обслуговування, якщо вузол j вільний; інакше, стає у хвіст черги вузла j .

Якщо вузол відправлення i задано, то рішення про маршрутизацію клієнта приймається незалежно від історії мережі. Клієнти, що прибувають у вузол j , вимагають певного обсягу роботи (часу обслуговування), що експоненціально розподілений із математичним сподіванням 1. Всі тривалості обслуговування — сімейство незалежних випадкових величин.

Нехай η_j^t позначає кількість клієнтів, що перебувають у вузлі j в момент часу $t \geq 0$ в очікуванні чи на обслуговуванні (локальна довжина черги у вузлі j). Тоді $\eta^t := (\eta_j^t: j = 1, \dots, J)$ — спільний вектор довжини черги мережі в момент часу t . Позначимо як $\eta = (\eta^t: t \geq 0)$ процес довжини спільної черги мережі Гордона — Ньюелла.

Теорема 2 [6, 8]. Позначимо як

$$S(I, J) = \{(n_1, \dots, n_J) \in \mathbb{N}^J: n_1 + \dots + n_J = I\}$$

простір станів процесу довжини спільної черги

$$\eta = (\eta_t: (\Omega, \mathcal{F}, P) \rightarrow (S(I, J), \mathcal{P}(S(I, J)))) ; t \in \mathbb{R}_+$$

мережі Гордона — Ньюелла, означеного вище.

Тоді η — марковський процес з Q -матрицею $Q = (q(y, x): y, x \in S(I, J))$, заданою так: для $i, j \in V$ і $x = (n_1, \dots, n_J) \in S(I, J)$

$$\begin{aligned} q(n_1, \dots, n_i, \dots, n_j, \dots, n_J; n_1, \dots, n_i - 1, \dots, n_j + 1, \dots, n_J) &= \\ &= \mu_i(n_i) r(i, j), \text{ якщо } n_i > 0, \\ i q(x, x) &= - \sum_{y \in S(I, J) - \{x\}} q(x, y), \\ q(x, y) &= 0, \text{ інакше.} \end{aligned}$$

η незвідний, консервативний, нескінченний (non-explosive) та ергодичний.

Нехай $\eta = (\eta_1, \dots, \eta_J)$ позначає єдиний імовірнісний розв'язок рівняння руху $\eta = \eta R$.

Єдиним стаціонарним і граничним розподілом $\pi = \pi(I, J)$ процесу η на $S(I, J)$ є

$$\pi(I, J)(n_1, \dots, n_J) = G(I, J)^{-1} \prod_{j=1}^J \left\{ \prod_{k=1}^{n_j} \frac{\eta_j}{\mu_j(k)} \right\}$$

$$(n_1, \dots, n_J) \in S(I, J),$$

де $G(I, J)$ — нормуюча константа.

Зауваження 2. Означення мережі Гордона — Ньюелла стосується тільки множини вузлів, що представляють вузли базового графу. Для структурованого графу $\Gamma = (V, B)$ ребра

$$\{i, j\} \in B \Leftrightarrow [r(i, j) > 0 \vee r(j, i) > 0]. \quad (5)$$

визначаються через матрицю R маршрутизації клієнтів.

Хоча мережеві процеси діють у очевидному сенсі локально щодо цієї структури сусідства, ймовірності переходів вкладеного ланцюга стрибків не визначаються локально та не синхронізуються у сенсі (3) та (4). Це результат досить простого розвитку мережі, що дозволяє завершити лише одне обслуговування за один момент часу. Але можемо прямо узагальнити поведінку переходу у напрямку, що робить його більш відповідним властивості синхронізації (див. [3]).

Означення 6 [узагальнена мережа Гордона — Ньюелла]. Узагальнена мережа Гордона — Ньюелла — це мережа вузлів $\{1, 2, \dots, J\} := V$ із залежними від стану окремими серверами з означення 4. $I > 0$ клієнтів циркулюють згідно з незвідною матрицею Маркова $R = (r(i, j): i, j = 1, \dots, J)$ (див. означення 5).

Нехай η_j^t позначає кількість клієнтів, що перебувають у вузлі j в момент часу $t \geq 0$, що чекають чи обслуговуються (локальна довжина черги у вузлі j). Тоді $\eta^t := (\eta_j^t: j = 1, \dots, J)$ — спільний вектор довжини черги мережі в момент часу t . Позначимо $\eta = (\eta^t: t \geq 0)$ процес спільної довжини черги узагальненої мережі Гордона — Ньюелла.

Можна запропонувати декілька варіантів еволюції мережі в часі, якщо $\eta^t = (\eta_j^t: j = 1, \dots, J) = (n_1, \dots, n_J)$. Система залишається в цьому стані протягом часу, експоненціально розподіленого з параметром $\lambda(n_1, \dots, n_J) > 0$, $\lambda(\max_{i \in V} n_i) > 0$ або $\lambda(\frac{1}{J} \sum_{i \in V} n_i) > 0$. Якщо цей час утримання закінчується, то в кожному вузлі приймається рішення (для кожного вузла незалежно від історії мережі) щодо завершення (з імовірністю $\gamma(i) > 0$) поточного обслуговування (якщо таке є, тобто, якщо $n_i > 0$) або його продовження (з імовірністю $1 - \gamma(i) > 0$). Якщо термін обслуговування у вузлі i закінчується, то клієнт, що обслуговувався, при виході з вузла i з імовірністю $r(i, j) \geq 0$ обирає для переходу вузол j наступним, і потім негайно входить у вузол j для обслуговування, якщо сервер вільний; в іншому випадку він приєднується до хвоста черги вузла j . Якщо вузол відправлення i задано, рішення про маршрутизацію клієнта не залежить від історії мережі.

Зауваження 3. Звідси випливає, що η є незвідним, консервативним, нескінченним та ергодичним, тому існує єдиний стаціонарний і граничний розподіл.

У виробничому контексті рішення про те, завершено чи продовжено обслуговування (прийняття наприкінці часу затримки), можемо інтерпретувати як діяльність локальних контролерів, які перебувають у вузлах. α_j^n — це рішення контролера у вузлі j в момент n -го переходу процесу і вказує, чи був виробничий процес успішним (з імовірністю $\gamma(i) > 0$) чи ні (з імовірністю $1 - \gamma(i) > 0$).

Імовірність переходу вбудованого ланцюга стрибків тепер локальна, і за побудовою рішення про стрибок у вузлах приймаються завжди одночасно в один момент миттєво і паралельно, але легко бачити, що загалом властивість синхронізації, яка передбачає умовну незалежність переходів, виконується не повністю, оскільки в імовірностях переходу стрибків ланцюга містяться параметри часу затримки $\lambda(x)$.

З останнього зауваження випливає, що, хоча певною мірою узагальнена мережа Гордона — Ньюелла виглядає як марковське стрибкоподібне випадкове поле, воно таким не є.

Отже, тепер додатково опишемо техніку внесення змін із використанням розширеного простору станів, що робить розвиток системи в часі таким, що у формальному розумінні виконує властивості марковського стрибкоподібного випадкового поля. Як і в роботі [2], основна ідея полягає в тому, щоб включити для кожного вузла j до опису локального стану його локальну довжину черги n_j , значення рішення щодо того, чи в момент наступного стрибка системи закінчується поточне обслуговування ($=1$), чи продовжується принаймні ще один період ($=0$), а також напрямок маршруту (якщо такий є), який може виникнути після можливого виходу з вузла j у наступний момент стрибку.

Означення 7 [доповнена мережа Гордона — Ньюелла]. Узагальнена доповнена мережа Гордона — Ньюелла — це мережа вузлів $V := \{1, 2, \dots, J\}$ із залежними від стану окремими серверами з означен-

ня 4. $I > 0$ клієнтів циркулюють згідно з незвідною матрицею Маркова $R = (r(i, j): i, j = 1, \dots, J)$ (див. означення 5).

Основний граф маршрутизації подано в (5).

Нехай для моменту часу $t \geq 0$ $\eta_j^t = (v_j^t, \theta_j^t, q_j^t)$ позначає кількість v_j^t клієнтів у вузлі j , що очікують або обслуговуються (локальна довжина черги у вузлі j), рішення θ_j^t , що приймає в момент наступного стрибка споживач сервісу (якщо такий є) щодо того, чи покидати вузол j , і напрямом $q_j^t \in V$, в якому цей споживач потім прибуде. Рішення θ_j^t споживач приймає на підставі інформації про стан повного околу вузла $\tilde{N}(j)$. Тоді $\eta^t := ((v_j^t, \theta_j^t, q_j^t): j = 1, \dots, J)$ спільний вектор — довжини черги доповненої мережі у момент часу t . Позначимо як $\eta = (\eta^t: t \geq 0)$ спільний доповнений процес довжини черги доповненої узагальненої мережі Гордона — Ньюелла.

Еволюція в часі цієї мережі така: якщо

$$\eta^t = ((v_j^t, \theta_j^t, q_j^t): j = 1, \dots, J) \\ = (n_1, t_1, r_1; \dots; n_i, t_i, r_i; \dots; n_j, t_j, r_j; \dots; n_J, t_J, r_J),$$

то система залишається в цьому стані протягом часу, розподіленого експоненціально з параметром, наприклад, $\lambda(n_1, \dots, n_i, \dots, n_j, \dots, n_J) > 0$, $\lambda(\max_{i \in V} n_i) > 0$ або $\lambda(\frac{1}{J} \sum_{i \in V} n_i) > 0$. Протягом цього часу затримки особи, що приймають рішення у вузлах, визначають значення $\alpha_k^t \in \{0, 1\}$, $k \in V$, на підставі інформації про стан повного околу вузла $\tilde{N}(j)$.

Незалежно від історії мережі вирішується, чи є поточне обслуговування (якщо воно здійснюється, тобто, якщо $n_i > 0$) завершеним (з імовірністю $P(\alpha_i^t = 1/v_{\tilde{N}(i)}^t) = \gamma(\tilde{N}(i)) > 0$), чи ні (з імовірністю $P(\alpha_i^t = 0/v_{\tilde{N}(i)}^t) = 1 - \gamma(\tilde{N}(i)) > 0$) для всіх $i \in V$.

Якщо цей час затримки закінчується, то у кожному вузлі j довжина черги оновлюється відповідно до значень $\{(\eta_i^t, \theta_i^t, q_i^t) = (x_i, t_i, r_i): i \in \tilde{N}(j)\}$. Це дає v_j^{t+1} .

Тоді для кожного вузла j значення θ_j^t оновлюється до $\theta_j^{t+1} = \alpha_j^t$.

Насамкінець значення q_j^t скидається до $q_j^{t+1} = k$ з імовірністю $r(j, k) \geq 0$, де з урахуванням вузла відправлення j рішення про маршрутизацію клієнта приймається незалежно від історії мережі.

Наслідок. Вкладений ланцюг переходів процесу η доповненої узагальненої мережі Гордона — Ньюелла з означення 8 задовольняє властивість локальності (3) та властивість синхронізації (4) з означення 3, і тому це — марковське стрибкоподібне випадкове поле.

Крім того, ланцюг стрибків керованого мережевого процесу (η, α) задовольняє умову (5.24) з означення 5.40 роботи [2], і тому (η, α) є керованим випадковим стрибкоподібним марковським полем.

Доведення цілком аналогічне до доведення наслідку 5.30 в роботі [2].

3. Знаходження оптимальних стратегій керування доповненою мережею Гордона — Ньюелла

Простори локальних рішень $A_i = \{t^1, t^2, \dots, t^{n_i}\}$, $i = 1, \dots, J$, скінченні і не залежать від t . З [2, теорема 3.39] знаємо, що оптимальну стратегію Δ^* (згідно з асимптотично усередненими очікуваними витратами) можна знайти в класі детермінованих стаціонарних марковських стратегій, тобто $\Delta^* = (\Delta_i^*(x_{\tilde{N}(i)}): i = 1, 2, \dots, J)$. Тому далі розглянемо для моменту часу t і вузла j рішення $\Delta_j^t(x_{\tilde{N}(j)})$, тобто залежне тільки від станів свого околу. Звідси випливає, що наприкінці часового слота у вузлі j (якщо присутні h клієнтів) обслуговування клієнтів закінчується з імовірністю $p_j(h, u) \in (0, 1)$ і з імовірністю $q_j(h, u) := 1 - p_j(h, u)$ замовник залишиться там щонайменше ще один проміжок часу, $h \geq 1$, $u \in A_j$. Отже, отримуємо локальну марковську стратегію $\Delta = (\Delta_i: i = 1, 2, \dots, J)$ і керований процес (ξ, Δ) .

Для будь-якої детермінованої стратегії Δ , що призводить до ймовірності обслуговування $p_j(h, u) \in (0, 1)$ у вузлах, керований процес (ξ, Δ) ергодичний у просторі станів $S(I, J)$ з деяким стаціонарним розподілом $\pi^{I, J, \Delta} =: \pi^\Delta = (\pi^\Delta(x): x \in S(I, J))$. Витрати, що виникають протягом очікуваного часу перебування між стрибками t^n , якщо система перебуває у стані $\xi^t = x^t$ і приймається рішення u^t , становлять $r(x^t, u^t, t^n \geq 0)$. Для визначення оптимальної стратегії будемо використовувати процедуру покращення стратегії [5, розділ 6], яка полягає у такому.

Вибираємо довільну стратегію Δ і розглядаємо для деякої (невідомої) функції $v = (v(x): x \in S(I, J))$ рівняння [5, сс. 56, 66]

$$R_y^\Delta + v(y) = r(y, \Delta(y), t^n) + \sum_{x \in S(I, J)} Q(x | y, \Delta(y)) v(x), \quad y \in S(I, J),$$

$$\sum_{x \in S(I, J)} \pi^\Delta(x) v(x) = 0.$$

Розв'язок $\{(v(x), R_y^\Delta): x, y \in S(I, J)\}$ дає витрати R_y^Δ при використанні стратегії Δ , що виявляються незалежними від початкового стану y . Алгоритм удосконалення політики в [5, с. 70] полягає у такому.

Для кожного $y \in S(I, J)$ визначимо A^y як множину рішень a в стані y , для яких

$$\sum_{x \in S(I, J)} Q(x | y, a) R_x^\Delta < R_y^\Delta,$$

або, якщо таких рішень немає, множину рішень, що задовольняє

$$\sum_{x \in S(I, J)} Q(x | y, a) R_x^\Delta = R_y^\Delta$$

$$r(y, a, t^n) + \sum_{x \in S(I, J)} Q(x | y, a) v^\Delta(x) < r(y, \Delta(y), t^n) + \sum_{x \in S(I, J)} Q(x | y, \Delta(y)) v^\Delta(x) = R_y^\Delta + v^\Delta(y).$$

Починаючи із заданої стратегії Δ , визначимо деяку локальну стратегію Δ' , яка приймає деякі рішення $a \in A^y$ у принаймні одному стані y , для якого A^y непорожня; інакше приймається рішення згідно зі стратегією Δ .

Теорема 3 [5, с. 70, 71]. 1. Якщо $\Delta' \neq \Delta$, то $R_y^{\Delta'} \leq R_y^\Delta, y \in S(I, J)$.

2. Процедура вдосконалення стратегії призводить до оптимальної стратегії за обмежену кількість ітерацій. Якщо Δ' актуальна стратегія і якщо A^y порожня для всіх y , то актуальна політика оптимальна: $\Delta' =: \Delta^*$ і

$$R_x^{\Delta^*} = \rho(x, \Delta^*) = \inf_{\Delta} \rho(x, \Delta).$$

4. Висновки

У статті запропоновано модифікацію замкненої мережі Гордона — Ньюелла, яка враховує локальну та синхронну взаємодію вузлів системи масового обслуговування. Припускається, що у вузлах перебувають особи, що приймають рішення щодо обслуговування клієнтів на підставі інформації про стан сусідніх вузлів. Критерій керування — усереднені витрати (доходи) в одиницю часу. Для таким чином доповненої мережі Гордона — Ньюелла пропонується процедура знаходження оптимальної детермінованої стратегії керування.

Список літератури

1. Cantaluppi L. Optimality of piecewise-constant policies in semi-Markov decision chains / L. Cantaluppi // *SIAM Journal of Control and Optimization*. — 1984. — Vol. 22. — Pp. 723–739.
2. Chorney R. K. Control of Spatially Structured Random Processes and Random Fields with Applications / R. K. Chorney, H. Daduna, P. S. Knopov. — New York : Springer Science + Business Media, Inc., 2006.
3. Daduna H. Controlled semi-markov fields with graph-structured compact state space / H. Daduna, P. S. Knopov, R. K. Chorney // *Teor. Ymovirnost. Matem. Statist.* — 2003. — Vol. 69. — Pp. 38–51. English translation in: *Controlled semi-Markov fields with graph-structured compact state space. Theory of Probability and Mathematical Statistics*, 69: 39–53, 2004.
4. Daduna H. Stochastic networks with product form equilibrium // *Stochastic Processes: Theory and Methods* / ed. by D. N. Shanbhag, C. R. Rao. — Amsterdam : Elsevier Science, 2001. — Vol. 19 of *Handbook of Statistics*. — Pp. 309–364.
5. Derman C. *Finite State Markovian Decision Processes* / C. Derman. — New York, London : Academic Press, 1970.
6. Gordon W. J. Closed queueing networks with exponential servers / W. J. Gordon, G. F. Newell // *Operations Research*. — 1967. — Vol. 15. — Pp. 254–265.
7. Howard R. A. Research in semi-Markov decision structures / R. A. Howard // *Journal of the Operational Research Society of Japan*. — 1964. — Vol. 6. — Pp. 163–199.
8. Jackson J. R. Jobshop-like queueing systems / J. R. Jackson // *Management Science*. — 1963. — Vol. 10. — Pp. 131–142.
9. Jewell W. S. Markov renewal programming. I. Formulation, finite return models / W. S. Jewell // *Operations Research*. — 1963. — Vol. 11. — Pp. 938–948.
10. Jewell W. S. Markov renewal programming. II. Infinite return models, example / W. S. Jewell // *Operations Research*. — 1963. — Vol. 11. — Pp. 948–971.
11. Kelly F. P. *Reversibility and Stochastic Networks* / F. P. Kelly. — Chichester — New York — Brisbane — Toronto : John Wiley and Sons, 1979.
12. Kitaev M. Elimination of randomization in semi-Markov decision models with average cost criterion / M. Kitaev // *Optimization*. — 1987. — Vol. 18 (3). — Pp. 439–446.
13. Kitaev M. Y. *Controlled Queueing Systems* / M. Y. Kitaev, V. V. Rykov. — Boca Raton : CRC Press, 1995.
14. Luque Vasquez F. Controlled semi-Markov models with discounted unbounded costs / Vasquez F. Luque, Alcaraz M. T. Robles // *Bol. Soc. Math. Mex.* — 1994. — Vol. 39 (1–2). — Pp. 51–68.
15. Vega-Amaya O. Average optimality in semi-Markov control models on Borel spaces: unbounded cost and controls / O. Vega-Amaya // *Bol. Soc. Math. Mex.* — 1993. — Vol. 38 (1–2). — Pp. 47–60.
16. Wakuta K. Arbitrary state semi-Markov decision processes with unbounded rewards / K. Wakuta // *Optimization*. — 1987. — Vol. 18 (3). — Pp. 447–454.
17. Yushkevich A. A. On homogeneous markov model with continuous time and finite or countable state space / A. A. Yushkevich, E. A. Fainberg // *Theor. Veroyatnost. Primen.* — 1979. — Vol. 24 (1). — Pp. 155–160. English translation in: *On homogeneous Markov model with continuous time and finite or countable state space. Theory Probab. Appl.*, 24 (1): 156–161, 1979.

References

- Cantaluppi, L. (1984). Optimality of piecewise-constant policies in semi-Markov decision chains. *SIAM Journal of Control and Optimization*, 22, 723–739.
- Chorney, R. K., Daduna, H. & Knopov, P. S. (2006). *Control of Spatially Structured Random Processes and Random Fields with Applications*. Springer Science + Business Media, Inc., New York.
- Daduna, H., Knopov, P. S. & Chorney, R. K. (2003). Controlled semi-markov fields with graph-structured compact state space. *Teor. Ymovirnost. Matem. Statist.*, 69, 38–51 [in Ukrainian]. English translation in: Daduna, H., Knopov, P. S. & Chorney, R. K. (2004). Controlled semi-Markov fields with graph-structured compact state space. *Theory of Probability and Mathematical Statistics*, 69, 39–53.
- Daduna, H. (2001). Stochastic networks with product form equilibrium. *Stochastic Processes: Theory and Methods*, 19, 309–364.
- Derman, C. (1970). *Finite State Markovian Decision Processes*. Academic Press, New York, London.
- Gordon, W. J. & Newell, G. F. (1967). Closed queueing networks with exponential servers. *Operations Research*, 15, 254–265.
- Howard, R. A. (1964). Research in semi-Markov decision structures. *Journal of the Operational Research Society of Japan*, 6, 163–199.
- Jackson, J. R. (1963). Jobshop-like queueing systems. *Management Science*, 10, 131–142.
- Jewell, W. S. (1963a). Markov renewal programming. I. Formulation, finite return models. *Operations Research*, 11, 938–948.
- Jewell, W. S. (1963b). Markov renewal programming. II. Infinite return models, example. *Operations Research*, 11, 948–971.
- Kelly, F. P. (1979). *Reversibility and Stochastic Networks*. John Wiley and Sons, Chichester, New York, Brisbane, Toronto.
- Kitaev, M. (1987). Elimination of randomization in semi-Markov decision models with average cost criterion. *Optimization*, 18 (3), 439–446.
- Kitaev, M. Y. & Rykov, V. V. (1995). *Controlled Queueing Systems*. CRC Press, Boca Raton.
- Luque, Vasquez F. & Robles, Alcaraz M. T. (1994). Controlled semi-Markov models with discounted unbounded costs. *Bol. Soc. Math. Mex.*, 39 (1–2), 51–68.
- Vega-Amaya, O. (1993). Average optimality in semi-Markov control models on Borel spaces: unbounded cost and controls. *Bol. Soc. Math. Mex.*, 38 (1–2), 47–60.
- Wakuta, K. (1987). Arbitrary state semi-Markov decision processes with unbounded rewards. *Optimization*, 18 (3), 447–454.
- Yushkevich, A. A. & Fainberg E. A. (1979). On homogeneous markov model with continuous time and finite or countable state space. *Theor. Veroyatnost. Primen.*, 24 (1), 155–160 [in Russian]. English translation in: Yushkevich, A. A. & Fainberg E. A. (1979). On homogeneous Markov model with continuous time and finite or countable state space. *Theory Probab. Appl.*, 24 (1), 156–161.

R. Chornei

LOCAL CONTROL IN GORDON-NEWELL NETWORKS

We examine continuous-time stochastic processes with a general compact state space, which is organized by a fundamental graph defining a neighborhood structure of states. These neighborhoods establish local interactions among the coordinates of the spatial process. At any given moment, the random state of the system, as described by the stochastic process, forms a random field concerning the neighborhood graph.

The process is assumed to have a semi-Markov temporal property, and its transition kernels exhibit a spatial Markov property relative to the basic graph. Additionally, a local control structure is introduced to optimize the evolution of the system over time. Here, optimality is defined in terms of the criterion of asymptotic average reward over time. Only discrete stepwise control is considered, meaning decisions are made exclusively at process jump moments. As is customary, random policies are represented by a conditionally independent structure. It is also assumed that this structure exists within the transition kernels of jump-like chains (synchronized kernels). The controlled random fields described, featuring local and synchronous component interactions, are applied to queueing systems—specifically, to the extended and generalized closed Gordon-Newell network. The modification primarily involves synchronizing customer service times at nodes. Based on the queue length at the node and its vicinity, a decision is made regarding serving the customer. If service is provided, a decision is also made regarding the customer's next direction. Consequently, the enhanced Gordon-Newell network satisfies the conditions for synchronous and local system node interactions. A procedure is outlined for determining optimal non-randomized control strategies in the enhanced Gordon-Newell network.

Keywords: Gordon-Newell network, local and synchronous interaction, controlled stochastic processes, decision-making processes.

Матеріал надійшов 17.12.2024



Creative Commons Attribution 4.0 International License (CC BY 4.0)

ВІДОМОСТІ ПРО АВТОРІВ

Андрощук Максим Віталійович — здобувач доктора філософії, програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», тахум. androshchuk@ukma.edu.ua

Артюшенко Богдан Андрійович — кандидат технічних наук, консультант ТОВ «Інститут інформаційних технологій “Інтелліас”», bogdan.artyuishenko@intellias.com

Бабич Трохим Анатолійович — студент Ph.D. програми «Комп'ютерні науки», старший викладач кафедри інформатики, заступник декана факультету інформатики Національного університету «Києво-Могилянська академія», t.babych@ukma.edu.ua

Бачинський Назарій Петрович — студент магістерської програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», n.bachynskiy@ukma.edu.ua

Верета Владислав Віталійович — студент магістерської програми «Інженерія програмного забезпечення» факультету інформатики Національного університету «Києво-Могилянська академія», v.vereta@ukma.edu.ua

Волинець Євген Анатолійович — кандидат фізико-математичних наук, старший викладач кафедри мережних технологій факультету інформатики Національного університету «Києво-Могилянська академія», ye.volynets@ukma.edu.ua

Глибовець Андрій Миколайович — професор, доктор технічних наук, декан факультету інформатики Національного університету «Києво-Могилянська академія», a.glybovets@ukma.edu.ua

Глибовець Микола Миколайович — доктор фізико-математичних наук, професор кафедри інформатики факультету інформатики Національного університету «Києво-Могилянська академія», glib@ukma.edu.ua

Давиденко Андрій Миколайович — студент Ph.D. програми «Інженерія програмного забезпечення», старший викладач кафедри інформатики факультету інформатики Національного університету «Києво-Могилянська академія», andrii.davydenko@ukma.edu.ua

Дехтяр Богдан-Ярема — студент магістерської програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», b.dekhtiar@ukma.edu.ua

Задохін Дмитро Володимирович — студент магістерської програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», d.zadokhin@ukma.edu.ua

Єщенко Микола Сергійович — студент PhD програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», m.yeshchenko@ukma.edu.ua

Касьяненко Максим Володимирович — магістр магістерської програми «Інженерія програмного забезпечення» факультету інформатики Національного університету «Києво-Могилянська академія», omnix.try@gmail.com

Кириєнко Оксана Валентинівна — старший викладач кафедри інформатики факультету інформатики Національного університету «Києво-Могилянська академія», o.kyriienko@ukma.edu.ua

Колінко Павло Володимирович — студент Ph.D. програми «Комп'ютерні науки», факультету інформатики Національного університету «Києво-Могилянська академія», p.kolinko@ukma.edu.ua

Моголівський Віталій Олегович — студент Ph.D. програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», v.moholivskiy@ukma.edu.ua

Нагірна Алла Миколаївна — кандидат фізико-математичних наук факультету інформатики Національного університету «Києво-Могилянська академія», a.nahirna@ukma.edu.ua

Новак Володимир Ігорович — студент бакалаврської програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», volodymyr.novak@ukma.edu.ua

Олецкий Олексій Віталійович — кандидат технічних наук, доцент, доцент кафедри мультимедійних систем факультету інформатики Національного університету «Києво-Могилянська академія», oletsky@ukma.edu.ua

Печкурова Олена Миколаївна — старший викладач кафедри мультимедійних технологій факультету інформатики Національного університету «Києво-Могилянська Академія», ryechkurova@ukma.edu.ua

Смиш Олег Русланович — доктор філософії з комп'ютерних наук, старший викладач кафедри мультимедійних систем факультету інформатики Національного університету «Києво-Могилянська академія», o.smysh@ukma.edu.ua, <https://orcid.org/0000-0002-8074-9745>

Ткаченко Владислав Олександрович — студент PhD програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», vo.tkachenko@ukma.edu.ua

Хмель Світлана Миколаївна — магістр програми «Інженерія програмного забезпечення» факультету інформатики Національного університету «Києво-Могилянська академія», svitlana.khmel@ukma.edu.ua

Чижова Анастасія Олексіївна — студентка 4 року бакалаврської програми «Комп'ютерні науки» факультету інформатики Національного університету «Києво-Могилянська академія», anastasiia.chyzhova@ukma.edu.ua

Чорней Руслан Костянтинівич — кандидат фізико-математичних наук, доцент, завідувач кафедри математики факультету інформатики Національного університету «Києво-Могилянська академія», r.chornei@ukma.edu.ua

Щербина Сергій Сергійович — студент магістерської програми «Інженерія програмного забезпечення» факультету інформатики Національного університету «Києво-Могилянська академія», serhii.shcherbina@ukma.edu.ua

Яремко Соломія Андріївна — магістр за спеціальністю «Комп'ютерні науки», освітньо-наукова програма «Комп'ютерні науки» Національного університету «Києво-Могилянська академія», старший викладач кафедри інформатики факультету інформатики Національного університету «Києво-Могилянська академія», s.yaremko@ukma.edu.ua

NOTES ABOUT AUTHORS

Maksym Androshchuk — Ph.D. Student in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, maxym.androshchuk@ukma.edu.ua

Bohdan Artiushenko — Ph.D. (Candidate of Technical Sciences), consultant at “Institute of Information Technologies “Intellias LLC””, bogdan.artiushenko@intellias.com

Trokhym Babych — Ph.D. Candidate, Deputy Dean, Faculty of Informatics, National University of Kyiv-Mohyla Academy, t.babych@ukma.edu.ua

Nazarii Bachynskyi — Student of the Master’s Degree Program Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, n.bachynskyi@ukma.edu.ua

Ruslan Chornei — Candidate of Physical and Mathematical Sciences, Associate Professor, Head of the the Department of Mathematics, Faculty of Informatics, National University of Kyiv-Mohyla Academy, r.chornei@ukma.edu.ua

Anastasiia Chyzhova — Senior student in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, anastasiia.chyzhova@ukma.edu.ua

Andrii Davydenko — Ph.D. student in Software Engineering, Senior Lecturer in Department of Informatics, Faculty of Informatics, National University of Kyiv-Mohyla Academy, andrii.davydenko@ukma.edu.ua

Bohdan Dekhtiar — Student of the Master’s Degree Program in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, b.dekhtiar@ukma.edu.ua

Mykola Glybovets — Doctor of Physical and Mathematical Sciences, Professor, Faculty of Informatics, National University of Kyiv-Mohyla Academy, glib@ukma.edu.ua

Andrii Hlybovets — Doctor of Technical Sciences and Dean, Faculty of Informatics, National University of Kyiv-Mohyla Academy, a.glybovets@ukma.edu.ua

Maksym Kasianenko — Master of Master’s Degree Program in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, omnix.try@gmail.com

Svitlana Khmel — Student of Master’s Degree Program Software engineering, Faculty of Informatics, National University of Kyiv-Mohyla Academy Master of Software Engineering, Faculty of Informatics, National University of Kyiv-Mohyla Academy, svitlana.khmel@ukma.edu.ua

Pavlo Kolinko — Student of Master’s Degree Program in Software Engineering, Faculty of Informatics, National University of Kyiv-Mohyla Academy, p.kolinko@ukma.edu.ua

Oksana Kyriienko — Senior Lecturer in the Department of Informatics, National University of Kyiv-Mohyla Academy, o.kyriienko@ukma.edu.ua

Vitalii Moholivskyi — Ph.D. student in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, v.moholivskyi@ukma.edu.ua

Alla Nahirna — Candidate of Physical and Mathematical Sciences, Faculty of Informatics, National University of Kyiv-Mohyla Academy, a.nahirna@ukma.edu.ua

Volodymyr Novak — Student of the Bachelor’s program in Computer Sciences, Faculty of Informatics, National University of Kyiv-Mohyla Academy, volodymyr.novak@ukma.edu.ua

Oleksii Oletsky — Ph.D. (Candidate of Technical Sciences), Associate Professor, Department of Multimedia Systems, Faculty of Informatics, National University of Kyiv-Mohyla Academy, oletsky@ukma.edu.ua

Olena Pyechkurova — Senior Lecturer in the Faculty of Informatics, National University of Kyiv-Mohyla Academy, pyechkurova@ukma.edu.ua

Oleh Smysh — Ph.D. in Computer Science, Senior Lecturer, Department of Multimedia Systems, Faculty of Informatics, National University of Kyiv-Mohyla Academy, o.smysh@ukma.edu.ua, <https://orcid.org/0000-0002-8074-9745>

Serhii Shcherbyna — Student of the Master's Degree Program in Software Engineering, Faculty of Informatics, National University of Kyiv-Mohyla Academy, serhii.shcherbyna@ukma.edu.ua

Vladyslav Tkachenko — Student of the Ph.D. Program in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, vo.tkachenko@ukma.edu.ua

Vladyslav Vereta — Student of the Master's Degree Program in Software Engineering, Faculty of Informatics, National University of Kyiv-Mohyla Academy, v.vereta@ukma.edu.ua

Yevhen Volynets — Candidate of Physical and Mathematical Sciences, Senior Lecturer in the Department of Network Technologies, Faculty of Informatics, National University of Kyiv-Mohyla Academy, ye.volynets@ukma.edu.ua

Solomiya Yaremko — Master's Degree in Computer Science, Computer Science Educational Program, National University of Kyiv-Mohyla Academy, Senior Lecturer in the Department of Informatics, Faculty of Informatics, National University of Kyiv-Mohyla Academy, s.yaremko@ukma.edu.ua

Mykola Yeshchenko — Student of the Ph.D. Program in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, m.yeshchenko@ukma.edu.ua

Dmytro Zadokhin — Student of the Master's Degree Program in Computer Science, Faculty of Informatics, National University of Kyiv-Mohyla Academy, d.zadokhin@ukma.edu.ua

ЗМІСТ

<i>Олецький О. В., Моголівський В. О.</i> Координація мікросервісів із використанням машин станів.....	4
<i>Яремко С. А., Новак В. І.</i> Система для автоматизації процесу нормалізації баз даних.....	11
<i>Касьяненко М. В., Нагірна А. М.</i> Алгоритмічний підхід до аналізу метрик у галузі розроблення відеоігор з відкритим світом	18
<i>Артюшенко Б. А.</i> Розподіл хмарних ресурсів на базі нуклеолуса.....	25
<i>Колінько П. В.</i> Автоматизована генерація і налаштування мікросервісів для спрощення процесу розроблення	31
<i>Верета В. В., Ткаченко В. О.</i> Автоматизована локалізація застосунків у мікросервісній архітектурі	40
<i>Єщенко М. С.</i> Моделювання технології блокчейн	51
<i>Давиденко А. М.</i> Забезпечення порядку оброблення повідомлень у розподілених системах.....	58
<i>Смиш О. Р., Чижова А. О.</i> Структурований оптимізований пошук у неструктурованих даних для задачі аналізу меню.....	63
<i>Глибовець М. М., Бачинський Н. П.</i> Узагальнений агент штучного інтелекту SIMA	70
<i>Щербина С. С., Бабич Т. А.</i> Фреймворк управління загрозами та реагування на інциденти IoT системи.....	77
<i>Глибовець А. М., Щербина С. С., Кирієнко О. В.</i> Вразливості безпеки та рішення для захисту в системах Інтернету речей	89
<i>Андрощук М. В.</i> Сучасні підходи до використання баз знань для вирішення проблем великих мовних моделей	98
<i>Глибовець М. М., Задохін Д. В., Дехтяр Б.-Я., Печкурова О. М.</i> Оброблення природної мови за допомоги великих мовних моделей і методів машинного навчання.....	102
<i>Волинець Є. А., Хмель С. М., Печкурова О. М.</i> Візуалізація когнітивних станів на основі Raspberry Pi для біологічного зворотного зв'язку в реальному часі за допомогою нейро-комп'ютерного інтерфейсу.....	112
<i>Чорней Р. К.</i> Локальне керування в мережах Гордона — Ньюелла.....	120
Відомості про авторів	130

CONTENTS

<i>O. Oletsky, V. Moholivskyi.</i> Coordination of Microservices Using State Machines	4
<i>S. Yaremko, V. Novak.</i> System for Automating the Database Normalization Process	11
<i>M. Kasianenko, A. Nahirna.</i> Metric Quality Analysis in Open-World Video Games	18
<i>B. Artiushenko.</i> A Nucleolus-Based Approach for Cloud Resource Allocation	25
<i>P. Kolinko.</i> Automated Generation and Configuration of Microservices to Simplify the Development Process.....	31
<i>V. Vereta, V. Tkachenko.</i> Automated Localization of Applications in Microservice Architecture.....	40
<i>M. Yeshchenko.</i> Modeling Blockchain Technology	51
<i>A. Davydenko.</i> Ensuring the Order of Message Processing in Distributed Systems	58
<i>O. Smysh, A. Chyzhova.</i> Structured Optimized Search in Unstructured Data for Menu Analysis Tasks.....	63
<i>M. Glybovets, N. Bachynskyi.</i> A Generalist AI Agent SIMA.....	70
<i>S. Shcherbyna, T. Babych.</i> Framework for Threat Management and Incident Response in IoT Systems	77
<i>A. Hlybovets, S. Shcherbyna, O. Kyriienko.</i> Security Vulnerabilities and Protection Solutions in Internet of Things Systems.....	89
<i>M. Androshchuk.</i> Modern Approaches to Using Knowledge Bases to Address the Challenges of Large Language Models.....	98
<i>M. Glybovets, D. Zadokhin, B. Dekhtiar, O. Pyechkurova.</i> Natural Language Processing Using Large Language Models and Machine Learning Methods	102
<i>Y. Volynets, S. Khmel, O. Pyechkurova.</i> Visualizing Cognitive States on a Raspberry Pi Platform for Real-Time Biofeedback Using a Brain-Computer Interface	112
<i>R. Chornei.</i> Local Control in Gordon-Newell Networks.....	120
Notes about authors.....	130

**НАУКОВІ ЗАПИСКИ НаУКМА
КОМП'ЮТЕРНІ НАУКИ**

**NaUKMA RESEARCH PAPERS
COMPUTER SCIENCE**

Том 7

Редакторка і коректорка *О. Пазюк*
Комп'ютерна верстка *М. Кулікова*

**Повну електронну версію наукового журналу
«Наукові записки НаУКМА. Комп'ютерні науки»
представлено на вебсайті
nrcsmp.ukma.edu.ua**

Підписано до друку 28.12.2024.
Формат 60 × 84 ¹/₈. Гарнітура Times New Roman.
Папір офсетний № 1. Ум. друк. арк. 15,58.

Адреса редакції:
вул. Г. Сковороди, 2, м. Київ, 04070,
тел.: (044) 463-66-68

Свідоцтво про внесення до Державного реєстру видавців, виготівників і розповсюджувачів
книжкової продукції серія ДК № 3631 від 23.11.2009