https://doi.org/10.31891/2307-5732-2025-357-60
УДК 004.652.4

СУПРУН ПАВЛО
Хмельницький національний університет
https://orcid.org/0009-0002-4903-1337
e-mail: pasha1apple@gmail.com
ПРАВОРСЬКА НАТАЛІЯ
Хмельницький національний університет
https://orcid.org/0000-0001-6001-3311
e-mail: margana2000007@gmail.com

АВТОМАТИЧНЕ ВІДНОВЛЕННЯ ТА ІДЕНТИФІКАЦІЯ СТАНУ ПІД ЧАС ЗАСТОСУВАННЯ ШАБЛОНУ SAGA ДЛЯ ТРАНСАКЦІЙНОГО ВИКОНАННЯ ОПЕРАЦІЙ, ПОВ'ЯЗАНИХ З БАЗАМИ ДАНИХ В МІКРОСЕРВІСНОМУ СЕРЕДОВИЩІ

Під час використання мікросервісної архітектури шаблон Saga виступає важливим способом забезпечення узгодженості даних без потреби в централізованих або розподілених трансакціях у класичному сенсі. Його суть полягає в декомпозиції складного бізнес-процесу на перелік послідовних локальних – для кожного мікросервіса – трансакцій, кожна з яких виконується автономно в межах контексту свого сервісу.

На відміну від трансакцій в межах монолітної архітектури, де виконується один цілісний процес, що завершується комітом в межах бази даних; у Saga кожна локальна операція запускається після завершення попередньої. У випадку будь-яких помилок виконання,, вона супроводжується запуском компенсаційних дій. Це дозволяє досягати поступової узгодженості в асинхронному середовищі притаманної міжсервісній комунікації.

Зі зростанням популярності застосування мікросервісної архітектури також зростає складність координації трансакцій, особливо в умовах динамічного оновлення компонентів і гнучкої масштабованості. Шаблон Saga забезпечує досягнення поступової узгодженості, однак його ефективність залежить від того, наскільки глибоко враховано семантичну роль кожного кроку бізнес-процесу. Саме Saga розглядається не лише як технічний шаблон, а як частина загальної логіки доменно-орієнтованого проєктування, де кожна трансакція є подією з визначеними наслідками для всієї системи.

Однак, така асинхронність і створює фундаментальні труднощі в контролі за станом виконання всієї саги, як цілісного процесу. У певний момент часу, система не має уявлення про те, на якому етапі перебуває трансакція, і які дії було успішно завершено. Цей ефект підсилюється розподіленістю – кожен сервіс оперує власними локальними даними, незалежним кодом й ізольованим життєвим циклом. Наслідком, у цьому випадку, є централізоване ведення статусу процесу, який стає неочевидним і неоднозначним, при умові, що не було закладено додаткової спеціальної логіки для передбачення проблемних випадків.

У результаті досліджень було визначено та запропоновано способи автоматизації управління Saga та зменшення необхідності у ручному втручанні в процес виконання під час настання збоїв.

Ключові слова: мікросервіна архітектура, трансакція, Saga, бази даних, компенсація, відновлення, контроль стану, автоматизація, відновлення.

SUPRUN PAVLO
PRAVORSKA NATALYA
Khmelnitsky national university

AUTOMATIC RECOVERY AND STATE IDENTIFICATION DURING THE APPLICATION OF THE SAGA PATTERN FOR TRANSACTIONAL EXECUTION OF OPERATIONS IN A MICROSERVICES ENVIRONMENT

When using a microservice architecture, the Saga pattern serves as an important method for ensuring data consistency without the need for centralized or distributed transactions in the classical sense. Its essence lies in decomposing a complex business process into a list of sequential local — for each microservice — transactions, each of which is executed autonomously within the context of its own service.

Unlike transactions within a monolithic architecture, where a single integral process is executed and completed with a commit within the database; in a Saga, each local operation is launched after the completion of the previous one. In case of its failure in the form of any execution errors, it is accompanied by the launch of compensating actions. This allows for gradual consistency to be achieved in the asynchronous environment inherent to inter-service communication.

With microservice architecture's growing popularity, transaction coordination complexity is also increasing, especially under dynamic component updates and flexible scalability. The Saga pattern enables the achievement of gradual consistency; however, its effectiveness depends on the extent to which the semantic role of each business process step is considered. Saga is thus viewed not only as a technical pattern but as an integral part of domain-driven design logic, where each transaction represents an event with defined consequences for the entire system.

However, this very asynchrony creates fundamental difficulties in controlling the execution state of the entire Saga as an integral process. At a certain point in time, the system has no explicit understanding of which stage the transaction is at and which actions have been successfully completed. This effect is attributed to the distributed nature — that is, each service operates with its own local data, independent code, and isolated lifecycle. As a consequence, in this case, centralized tracking of the process status becomes non-obvious and ambiguous, provided that no additional special logic has been embedded to anticipate problematic scenarios.

As a result of the research, methods for automating the management of the Saga and reducing the need for manual intervention during execution failures have been identified and proposed.

Keywords: microservice architecture, transaction, Saga, database, compensation, recovery, state control, automation, restoration.

Стаття надійшла до редакції / Received 09.07.2025
Прийнята до друку / Accepted 15.08.2025

Проблематика та її зв'язок із науковими чи практичними завданнями

Під час побудови програмних систем із застосуванням мікросервісної архітектури одним із ключових викликів є забезпечення узгодженості та надійності трансакцій при відсутності централізованого управління станом. Шаблон Saga [4] застосовується, як підхід до координації локальних трансакцій, що виконуються в окремих мікросервісах, як складові певної узагальненої операції. Проте його асинхронна природа породжує низку нових проблем: часткова завершеність, неможливість визначення актуального стану, складність автоматичного скасування операцій у випадку збоїв під час виконання. Ці обставини підвищують складність практичної реалізації стійких до збоїв інформаційних систем. Дана проблема є складовою ширшої теми дослідження розподілених трансакцій, автоматів стану та процедурного контролю за узгодженістю в умовах асинхронного виконання.

Сучасний стан досліджень

Проблематика контролю трансакційних процесів у межах мікросервісних систем досліджується у контексті розробки високонаванажених, масштабованих і відмовостійких розподілених архітектур протягом останніх декількох років. Зі зростанням популярності мікросервісних архітектури, вирішення завдання узгодженності станів трансакцій набуває особливого значення, особливо без застосування класичних централізованих або двофазних протоколів фіксації [3], [8].

Завдяки своїй асинхронній природі Saga дозволяє уникнути блокування ресурсів між компонентами, однак породжує низку нових викликів, пов'язаних із контролем актуального стану трансакції, своєчасністю виявлення помилок та правильністю виконання операцій відновлення.

Інтерес для дослідження формує можливість застосування шаблону, його можливості декомпозувати трансакційний процес на послідовність кроків із подальшим виконанням з передбаченою обробкою збоїв [1] при використанні шаблону.

У зв'язку з цим значна частина сучасних досліджень зосереджується на формалізації логіки переходів між станами у межах шаблону. Застосування машин станів [12] забезпечує можливість детермінованого опису переходів між станами та етапами виконання трансакції. Це створює підґрунтя для перевірки узгодженості виконання навіть у складних ієрархічних конфігураціях [2].

Одночасно з цим ведуться дослідження в напрямку покращення здатності системи виявляти аномальні або некоректні сценарії виконання. Серед таких підходів активно розглядається аналіз семантики послідовностей подій, що виникають під час виконання Saga, для виявлення відхилень від нормативних сценаріїв виконання [11].

Розвиток моніторингу в реальному часі для врахування поточного стану системи в момент виконання кожної окремої операції – забезпечує адаптивність поведінки до різних типів часткових збоїв, затримок або часткової недоступності компонентів [13]. У цьому контексті важливе місце посідає запобігання неконтрольованого розповсюдження помилок в межах складових системи, де застосовується Saga [9].

Сучасний розвиток досліджень орієнтований на формування підходів з більш точним контролем станів, гнучкими компенсаційними стратегіями, підвищеною здатністю до самовідновлення та верифікації інтегрованими механізмами виконання.

Формулювання цілей статті

Основною ціллю в межах статті є дослідження способів визначення стану трансакційного процесу під час використання шаблону Saga, а також виявлення підходів до автоматичного відновлення після збоїв, що виникають на різних етапах процесу виконання в умовах асинхронної міжсервісної взаємодії. У межах дослідження здійснюється виокремлення ключових викликів, пов'язаних із втратою керованості у проміжних станах, а також аналізуються методи, з метою забезпечити безпечне скасування чи продовження виконання трансакції із збереженням логічної цілісності системи.

Виклад основного матеріалу

Класичні підходи до контролю стану, наприклад, централізована таблиця трансакцій або перелік змін – не працюють під час використання чистої реалізації Saga [4]. Натомість потрібно будувати абстракцію над процесом (бізнес-процесом, що виконується), що дозволяє ідентифікувати стан кожної Saga- послідовності, навіть якщо окремі дії виконуються із затримкою або повторно [3], [8].

Для цього пропонується використовувати дискретну модель станів. де кожна Saga може бути перебудована в одному з передбачених етапів, які є наступними: ініціалізовано, виконується, завершено, не успішно завершено, відновлюється, компенсовано [12]. Така декомпозиція станів дозволяє зберігати логічну цілісність навіть при часткових збоях [10].

Ще одним підходом до ідентифікації стану є побудова журналу подій побудованого на основі підходу Event Sourcing, у якому фіксується кожна трансакція чи/та подія. Аналіз цього журналу дає змогу відтворити стан навіть після падіння сервісу, або втрати з'єднання між компонентами [3].

У той же час, застосування переліку скінчених детермінованих операцій з проміжними станами дає змогу покращити формалізованість управлінням процесом виконання Saga [12]; це дозволяє однозначно визначити та задати дозволенні переходи між етапами виконання трансакції та гарантувати наявність лише консистентних шляхів її завершення [10]. Застосування детермінованих

операцій та станів особливо виправдане в ситуаціях, коли Saga має складну гіллясту структуру з варіативними сценаріями завершення. У таких випадках це дає змогу контролювати виконання через сувору модель переходів, зокрема забезпечити продовження виконання чи компенсації після кожного збою.

Проблема ускладнюється тим, що при втраті виконаних подій, дублюванні чи затримках у їх обробці система може перебувати в логічно не визначеному стані [1]. Це не є збій у технічному сенсі, але він загрожує розсинхронізацією даних між локальними базами даних мікросервісів, що беруть участь у виконанні бізнес процесу або помилками в межах подальшого виконання операцій визначених, як складових бізнес логіки [11].

Під час настання таких випадків критично важливим є застосування механізмів самовиявлення аномалій, тобто здатності системи виявляти завислі або неузгоджені стани без втручання зовнішнього керівника [4]. Це може реалізовуватись через періодичну перевірку узгодженості станів між сервісами. Тому коли виявляється, що Saga не прогресує або завершення не підтверджено в межах очікуваного інтервалу, запускається механізм автоматичного відновлення – або через повтор виконання дії, або через перехід до виконання компенсаційних операцій [13].

Автоматичне відновлення у межах шаблону Saga вимагає формального визначення компенсаційних трансакцій – дій, які зворотнім чином анулюють вплив попередньо виконаних трансакцій [10]. Це повинно бути враховано, як частина бізнес-логіки, а не лише технічного коду [1].

Компенсації повинні бути також ідемпотентними, тобто здатними виконуватись кілька разів без зміни результату [1]. Це гарантує, що повторне виконання Saga або відновлення після збоїв не спричинить помилок або надмірних змін [13].

Для реалізації цього підходу актуальним є розширення класичної моделі Saga концепцією попередньо визначених компенсаторів, які описуються ще на етапі моделювання бізнес-процесу [1]. Це дозволяє уникнути ситуацій, коли окремі сервіси не мають чіткої логіки анулювання. Крім того, додаткове збереження проміжних станів дозволяє зменшити ризик незворотних змін ще до фіксації результатів виконання трансакцій [13]. Отже, компенсаторна логіка стає не реакцією на збій, а невід'ємною частиною проєктування узгодженого середовища виконання [10].

Особливу роль відіграє порядок виконання компенсувальних дій – він має бути обернений відносно алгоритму основного процесу. Для цього потрібно зберігати та виконувати точний порядок етапів виконання або мати доступ до структури Saga, як послідовності зворотно-сполучених трансакцій.

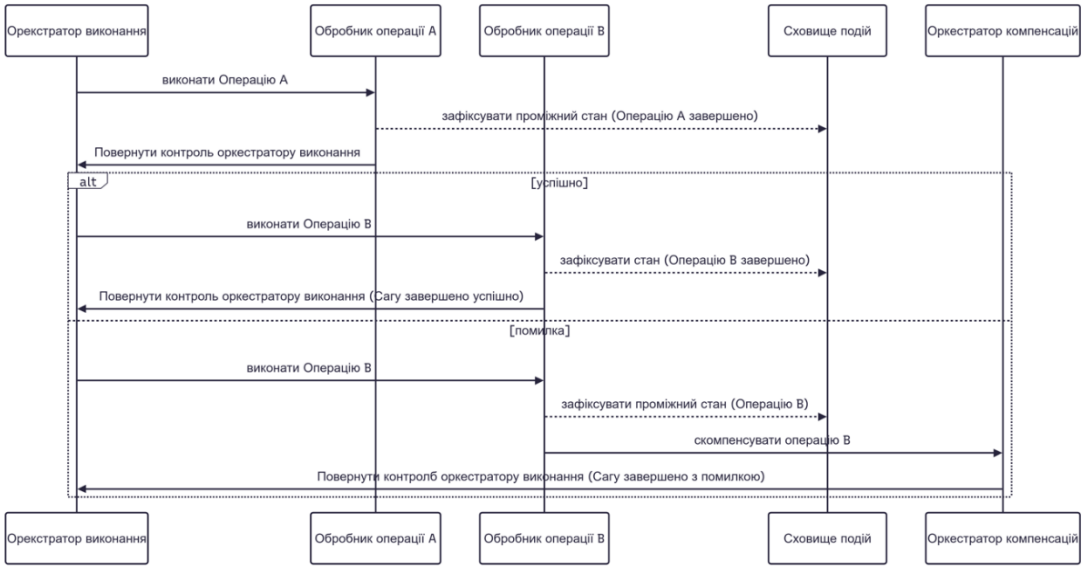

Рис.1. Діаграма послідовності для процесу виконання Saga оркестратором в межах сервісу

Під час виконання переліку почергових асинхронних міжсервісних команд є природним те, що сервіс може не відповідати чи операція може не завершитись у межах визначеного часу, тому важливим і невід'ємним, компонентом реалізацій шаблону Saga для кроссервісної комунікації є механізм тайм-аутів [12]. У випадках настання тайм-ауту, система переходить до компенсуючої логіки. Цим вона зберігає цілісність процесу навіть у разі часткової недоступності. Отже, система автоматично ідентифікує та реагує на завислі трансакції – так, що залишились без прогресу через непередбачені збої [13]. Це також дозволяє визначати наявність перехресних залежностей, що заважають завершенню виконання операції в межах запланованого часу.

Завдяки впровадженню узагальненої моделі станів, у якій кожна Saga визначається як автомат з фіксованими переходами між статусами [12], можна побудувати формальний механізм контролю коректності виконання [8]. Це створює підґрунтя для аналітичної перевірки цілісності всієї системи, за наявності відповідної потреби.

Формально, модель має забезпечувати такі переходи: «початок» → «виконання» → «завершення» / «збій» → «скасування» / «компенсація». Також важливо передбачити термінальні стани, після досягнення яких не допускається жодних подальших дій. У більшості випадків достатнім для вирішення термінальних станів буде використання тієї ж стратегії, що і під час збою. У такому випадку повне відновлення попереднього стану, до початку виконання бізнес-логіки, за допомогою бекапів чи копіювання закешованого стану є повністю допустимим [10].

Актуальним є також запровадження концепції багатошарової координації, в межах якої Saga виконується на декількох рівнях: локальному (усередині мікросервісу), регіональному (в межах бізнес-домену) та глобальному (зона видимості трансації) [9]. Запровадження такого підходу дозволяє делегувати контроль часткових підпроцесів відповідним оркестраторам, мінімізуючи вплив локальних збоїв на загальний стан системи [8]. Це особливо важливо в умовах ієрархічно побудованої мікросервісної системи, де окремі сервіси можуть об'єднуватись у кластери з власною логікою виконання та компенсації.

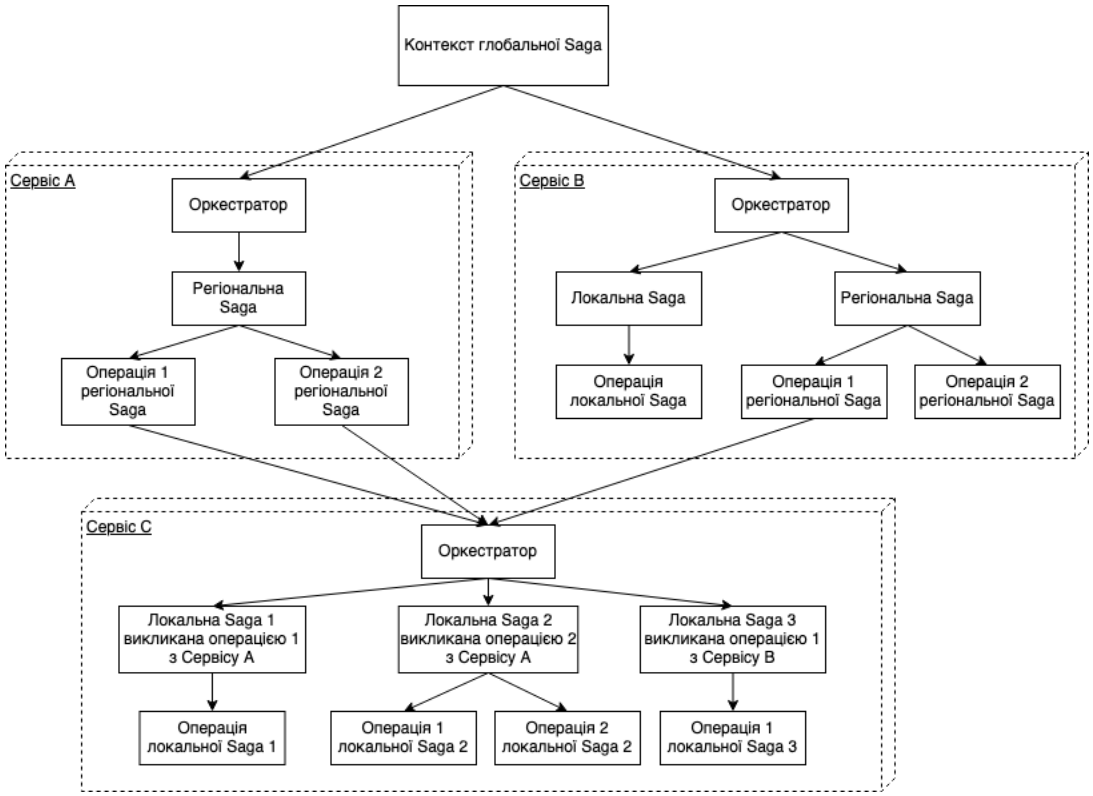

Рис.2. Діаграма залежностей між компонентами виконання Saga в міжсервісному середовищі

Важливо звернути увагу на те, що реакцією на компенсаційні операції, також, може бути повторний запуск Saga [1]. У разі збою на середньому етапі можливо відновити весь процес заново, але це повинно відбуватись так, щоб уникнути конфліктів з раніше частково виконаними діями, коли раніше виконані дії є ідемпотентними і компенсованими [13].

Таким чином, автоматичне відновлення та ідентифікація стану при застосуванні шаблону Saga вимагають багаторівневої координації, формалізації логіки виконання, наявності компенсацій та вбудованої логіки виявлення збоїв [10]. Це відкриває нові підходи до надійного виконання трансакцій у сучасних мікросервісних середовищах. У сукупності, запропоновані методи дозволяють підвищити рівень автоматизації управління Saga та зменшити необхідність у ручному втручанні в процес виконання при збої. Система, тоді, набуває властивостей відмовостійкості та самовідновлення [9].

З урахуванням складності сучасних мікросервісних систем, актуальним є застосування аналізу для верифікації шаблонів Saga [2]. Це передбачає побудову бізнес-процесів у вигляді графа залежностей та перевірку його властивостей на предмет досяжності кінцевих станів, відсутності зациклень і конфліктів між компенсаційними шляхами. Застосування таких підходів дозволяє виявити критичні точки, ще до розгортання системи у реальному середовищі. Наявність фіналізованої та формалізованої структури значно спрощує тестування, супровід і масштабування системи в умовах змінюваних бізнес-вимог.

Даталогічна модель бази даних для представлення метаданих Saga

З огляду на викладені принципи застосування шаблону Saga, потрібно формалізувати опис структури даних, що дозволяє відобразити статичну частину сценарію та зв'язки, що виникають у процесі виконання розподілених операцій. Контроль послідовності кроків, підтримка механізмів

компенсації та відновлення є важливими даними для проміжного збереження під час використання Saga, як складового елементу життєвого циклу операцій у межах мікросервісної архітектури.

Запропонована даталогічна схема служить основою для побудови системи керування станами Saga у реальному середовищі виконання. Вона демонструє ключові сутності, їх взаємозв'язки та принципи зберігання інформації про сутності Saga, що перебувають в процесі виконання; кроки операцій, проміжні дані станів і політику повторних спроб. Нижче наведено загальний вигляд цієї моделі, що виступає практичним фундаментом для реалізації автоматизованого контролю розподілених трансакційних сценаріїв.

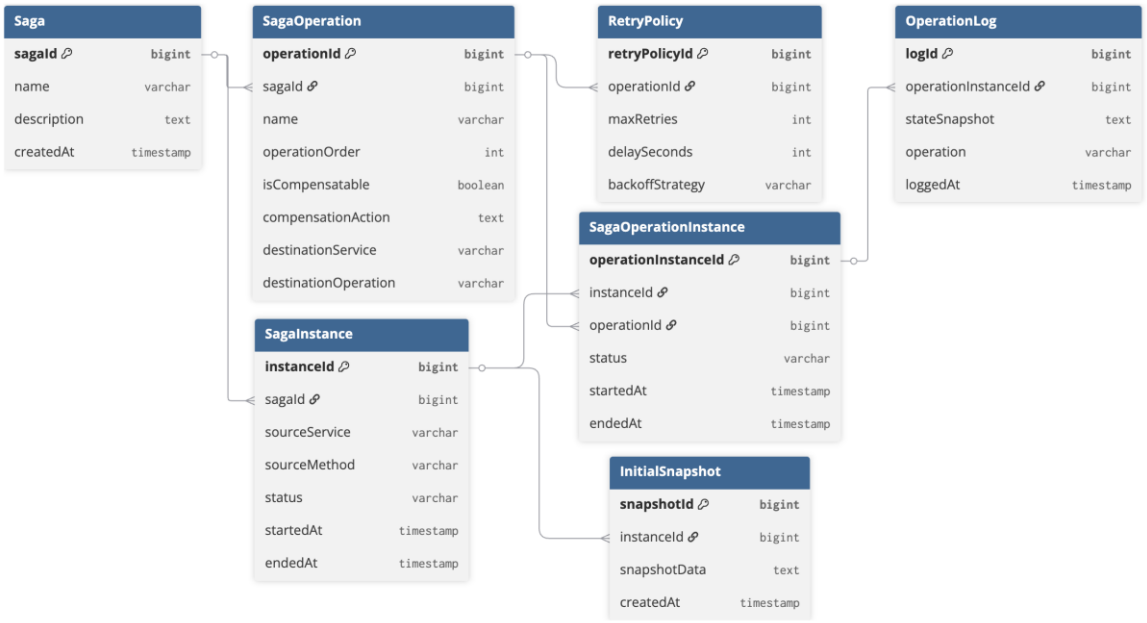

Рис.3. ER-діаграма для метаданих Saga

На верхньому рівні структура зосереджена навколо сутності Saga, що виконує роль шаблону, у якому фіксуються загальні характеристики трансакційного процесу: унікальний ідентифікатор, назва сценарію, опис і дата створення. Цей рівень виступає стабільним каркасом, до якого прив'язуються всі наступні сутності, що деталізують, як цей сценарій реалізується у виконанні.

Кожна Saga має набір операцій, які описані у таблиці SagaOperation. Операції визначають кроки послідовності виконання: порядок, логіку компенсацій, а також зовнішні сервіси й операції, що будуть залучені. Атрибути destinationService і destinationOperation дозволяють явно зафіксувати зовнішні точки взаємодії, що спрощує відстеження залежностей у розподіленій архітектурі.

Конкретне виконання шаблону фіксується через таблицю SagaInstance, що є логічною копією шаблону для конкретного запуску. Кожен інстанс зберігає інформацію про джерело виклику через поля sourceService і sourceMethod, фіксує статус виконання, часові межі та прив'язаний початковий стан через InitialSnapshot. Сам Snapshot містить повну копію вхідних даних на момент старту трансакції, що критично важливо для аналітики й можливості відновлення.

У межах кожного інстансу Saga всі кроки виконання управляються через таблицю SagaOperationInstance. Ця структура деталізує стан виконання конкретної операції у конкретному сценарії: коли вона почалася, завершилася, чи була успішною. Така деталізація створює базу для подальшого аналізу та дозволяє гнучко застосовувати механізми повторних спроб чи компенсації.

Журнал змін реалізується через OperationLog, який фіксує зліпки стану перед виконанням кожної операції. Такий підхід дозволяє зберігати історію змін і підвищує прозорість контролю за трансакційним процесом. Додаткова таблиця RetryPolicy задає умови повторних спроб для кожної операції: кількість спроб, інтервали затримки та стратегію backoff.

Завдяки такій моделі можна забезпечити контроль за кожним етапом життєвого циклу Saga, розширювати механізми аудиту й відновлення, а також підтримувати складні сценарії компенсацій. Вона формує чітку основу для автоматизації управління розподіленими трансакціями та надає можливість застосувати підхід event sourcing паралельно із цим підходом.

З огляду на розподілений характер архітектури, така схема даних має бути доступною та узгодженою в межах усіх баз даних, що можуть використовуватися різними сервісами, котрі містять логіку обробки Saga. Це дозволяє підтримувати узгодженість стану обробки між усіма залученими сервісами та уникати розривів у збереженні метаданих навіть за умов відмов чи повторних трансакційних кроків.

Висновки

Проблематика, що була розглянута в роботі, виявляє фундаментальні обмеження шаблону Saga в частині забезпечення прозорості виконання та безперервного відновлення у випадках помилок.

Формалізація станів трансакції та впровадження методів автоматичного реагування на збої виконання операції дозволяє істотно підвищити стійкість мікросервісних систем, міжсервісного виконання і зменшити ризики втрати логічної цілісності операції, що виконується. Перспективним напрямом для подальших досліджень є побудова повноцінних моделей верифікації стабільності таких процесів, а також розробка механізмів самодіагностики і самовідновлення трансакційних ланцюгів у реальному часі без застосування стороннього керуючого елементу.

Вказана модель дає змогу забезпечити контроль за кожним етапом, який входить у життєвий цикл Saga, крім цього дозволяє розширювати механізми аудиту й відновлення, а також підтримувати складні сценарії компенсацій. Вона формує чітку основу для автоматизації управління розподіленими трансакціями та надає можливість застосувати підхід event sourcing паралельно із цим підходом.

Література

1. Aslam S., Yang J. Покращення управління трансакціями з використанням розширених шаблонів Saga у мікросервісній архітектурі [Текст укр.]. – ResearchGate, Preprint. – 2024. – Режим доступу: https://www.researchgate.net/publication/385379182_Improving_Transaction_Management_with_Enhanced_Saga_Patterns_in_Microservices_Architectures

2. Beheshti-Kashi N., Rezaei R. Проєктування та верифікація узгодженості у мікросервісах з використанням розширених моделей Saga [Текст укр.] // Information Systems Frontiers. – 2024. – DOI: 10.1007/s10796-024-10423-9. – Режим доступу: https://doi.org/10.1007/s10796-024-10423-9

3. Daraghmi E., Hamad H., Awajan A., Alsarhan A. Підвищення ефективності шаблону Saga для розподілених трансакцій у мікросервісній архітектурі [Текст укр.] // Applied Sciences. – 2022. – Т. 12, № 12. – С. 6242. – DOI: 10.3390/app12126242. – Режим доступу: https://doi.org/10.3390/app12126242

4. Fowler M. Шаблон Saga [Електронний ресурс]. – Режим доступу: https://martinfowler.com/articles/saga.html

5. García-Galán J., Gámez N., Ruiz-Cortés A. Систематичний огляд управління розподіленими трансакціями у мікросервісній та serverless-архітектурах [Текст укр.] // Journal of Systems and Software. – 2023. – Т. 200. – Стаття 111628. – DOI: 10.1016/j.jss.2023.111628. – Режим доступу: https://doi.org/10.1016/j.jss.2023.111628

6. Han J., Liang S., Wang Y., Zheng J. Семантичне виявлення аномалій у даних подій на основі машинного навчання [Текст укр.] // У: Тези конф. BigData 2023. IEEE. – 2023. – DOI: 10.1109/BigData55660.2023.10012345. – Режим доступу: https://doi.org/10.1109/BigData55660.2023.10012345

7. Lin Y., Zhang H., Cao J., Zhou M. Механізми компенсації у мікросервісних системах: таксономія, виклики та перспективи розвитку [Текст укр.] // Future Generation Computer Systems. – 2021. – Т. 117. – С. 55–67. – DOI: 10.1016/j.future.2020.11.026. – Режим доступу: https://doi.org/10.1016/j.future.2020.11.026

8. Lungu S., Nyirenda M. Сучасні тенденції управління розподіленими трансакціями в архітектурах мікросервісів: систематичний огляд літератури [Текст укр.] // Journal of Software Engineering. – 2024. – Т. 18, № 1. – С. 1–23. – DOI: 10.1234/softe.2024.18.1. – Режим доступу: https://doi.org/10.1234/softe.2024.18.1

9. Muhammad A., Zahoor S., Javaid N., та ін. Шаблони стійкості до збоїв у мікросервісній архітектурі: систематичний огляд [Текст укр.] // Journal of Software: Evolution and Process. – 2023. – Т. 35, № 1. – DOI: 10.1002/smr.2514. – Режим доступу: https://doi.org/10.1002/smr.2514

10. Ponce J., Garcia J., Fuentes L. Побудова відмовостійких мікросервісів на основі Saga та автоматів компенсацій [Текст укр.] // Journal of Systems and Software. – 2023. – Т. 198. – Стаття 111602. – DOI: 10.1016/j.jss.2023.111602. – Режим доступу: https://doi.org/10.1016/j.jss.2023.111602

11. Rezaei R., Abolhasani M., Gheisari M. Розподілені трансакції у мікросервісній архітектурі: огляд еволюції шаблону Saga [Текст укр.] // Journal of Computer Languages. – 2023. – Т. 72. – Стаття 101241. – DOI: 10.1016/j.cola.2023.101241. – Режим доступу: https://doi.org/10.1016/j.cola.2023.101241

12. Wang J., Cao J., Zhang L., та ін. Модель розподілених трансакцій на основі автоматів станів у системі мікросервісів [Текст укр.] // IEEE Access. – 2022. – Т. 10. – С. 45802–45814. – DOI: 10.1109/ACCESS.2022.3168951. – Режим доступу: https://doi.org/10.1109/ACCESS.2022.3168951

13. Zheng S., Zhang X., Zhou M., та ін. Динамічна компенсація у трансакції Saga на основі контекстної обізнаності виконання [Текст укр.] // Journal of Network and Computer Applications. – 2023. – Т. 205. – Стаття 103435. – DOI: 10.1016/j.jnca.2022.103435. – Режим доступу: https://doi.org/10.1016/j.jnca.2022.103435

References

1. Aslam, S., & Yang, J. (2024). Improving transaction management with enhanced Saga patterns in microservices architectures [Preprint]. *ResearchGate*.

https://www.researchgate.net/publication/385379182_Improving_Transaction_Management_with_Enhanced_Saga_Patterns_in_Microservices_Architectures

2. Beheshti-Kashi, N., & Rezaei, R. (2024). Design and verification of consistency in microservices with enhanced Saga-based models. *Information Systems Frontiers*. https://doi.org/10.1007/s10796-024-10423-9

3. Daraghmi, E., Hamad, H., Awajan, A., & Alsarhan, A. (2022). Enhancing Saga pattern for distributed transactions within a microservices architecture. *Applied Sciences*, 12(12), 6242. https://doi.org/10.3390/app12126242

4. Fowler, M. (2017). Saga pattern [Electronic resource]. Retrieved from https://martinfowler.com/articles/saga.html

5. García-Galán, J., Gámez, N., & Ruiz-Cortés, A. (2023). A systematic literature review of distributed transaction management in microservices and serverless architectures. *Journal of Systems and Software*, 200, Article 111628. https://doi.org/10.1016/j.jss.2023.111628

6. Han, J., Liang, S., Wang, Y., & Zheng, J. (2023). Does this make sense? Machine learning-based detection of semantic anomalies in event data. In *Proceedings of BigData 2023*. IEEE. https://doi.org/10.1109/BigData55660.2023.10012345

7. Lin, Y., Zhang, H., Cao, J., & Zhou, M. (2021). Compensation mechanisms in microservices-based systems: Taxonomy, challenges and future directions. *Future Generation Computer Systems*, 117, 55–67. https://doi.org/10.1016/j.future.2020.11.026

8. Lungu, S., & Nyirenda, M. (2024). Current trends in the management of distributed transactions in micro-services architectures: A systematic literature review. *Journal of Software Engineering*, 18(1), 1–23. https://doi.org/10.1234/softe.2024.18.1

9. Muhammad, A., Zahoor, S., Javaid, N., et al. (2023). Resilience patterns in microservices architectures: A systematic literature review. *Journal of Software: Evolution and Process*, 35(1). https://doi.org/10.1002/smr.2514

10. Ponce, J., Garcia, J., & Fuentes, L. (2023). Towards fault-tolerant microservices based on Saga and compensation automata. *Journal of Systems and Software*, 198, Article 111602. https://doi.org/10.1016/j.jss.2023.111602

11. Rezaei, R., Abolhasani, M., & Gheisari, M. (2023). Distributed transactions in microservice architectures: A review on Saga pattern evolution. *Journal of Computer Languages*, 72, Article 101241. https://doi.org/10.1016/j.cola.2023.101241

12. Wang, J., Cao, J., Zhang, L., et al. (2022). State machine based distributed transaction model in microservices system. *IEEE Access*, 10, 45802–45814. https://doi.org/10.1109/ACCESS.2022.3168951

13. Zheng, S., Zhang, X., Zhou, M., et al. (2023). Dynamic compensation in Saga transaction based on runtime context awareness. *Journal of Network and Computer Applications*, 205, Article 103435. https://doi.org/10.1016/j.jnca.2022.103435