DOI: 10.20998/2079-0023.2024.02.16
УДК 004.9

В. О. КОЛБАСІН, кандидат технічних наук (PhD), доцент, доцент кафедри системного аналізу інформаційно-аналітичних технологій Національного технічного університету «Харківський політехнічний інститут», м. Харків, Україна; e-mail:viacheslav.kolbasin@khpi.edu.ua; ORCID: https://orcid.org/0009-0005-6367-7574

ОРКЕСТРУВАННЯ РОБОЧИХ ПРОЦЕСІВ ВЗАЄМОДІЇ З КЛІЄНТАМИ В КОРПОРАТИВНИХ ЗАСТОСУНКАХ

Автоматизація процесів є важливим чинником розвитку сучасних корпоративних систем, зокрема систем електронної комерції. Однією з задач, що треба вирішувати при побудові подібних систем є оркестрування процесів взаємодії з клієнтом, які за своєю природою є асинхронними. Сучасні популярні фреймворки для оркестрування, зокрема Airflow, не дуже пристосовані для одночасного очікування відповіді від клієнта великою кількість робочих процесів. Це призводить до зайвих витрат обчислювальних ресурсів та зниження економічної ефективності системи. В даній роботі розглядається задача побудови системи оркестрування робочих процесів взаємодії з клієнтом на основі подійно-орієнтованого підходу. Замість умовно послідовного централізованого виконання графової моделі робочого процесу пропонується представити робочий процес у вигляді послідовності подій та реакцій системи на них, що дозволяє позбутися потреби в явному очікуванні відповіді від клієнта. В такий моделі операції робочого процесу будуть виконуватися під час обробки подій, а перехід до наступної операції відбуватиметься за рахунок відправлення нового повідомлення з описом цієї операції. Розглянуто централізований та розподілений підходи для виконання переходів між операціями робочого процесу, показані переваги та недоліки кожного з них. Також розглянуто реалізацію механізму очікування відповіді від клієнта, для чого пропонується перед початком взаємодії з останнім зберігати повідомлення у спеціалізованому сховищі, прив'язане до ідентифікатору сеансу взаємодії з клієнтом. А після надходження даних від клієнта – додавати їх до повідомлення і відправляти його назад в чергу повідомлень для продовження виконання робочого процесу. Для опису моделі робочого процесу пропонується використовувати формат JSON, зважаючи на можливість використання обробників, написаних на різних мовах програмування. Описано один з підходів до побудови такого формату опису та продемонстровано його використання для реалізації прикладу робочого процесу. Результати дослідження можуть бути корисними при створенні систем оркестрування робочих процесів взаємодії з клієнтом.

Ключові слова: оркестрування робочого процесу, подійно-орієнтований підхід, взаємодія з клієнтом, асинхронні операції.

Вступ. Здатність зручно та економно організувати взаємодію з клієнтами є однією з конкурентних переваг сучасної компанії, орієнтованої на взаємодію з кінцевим споживачем. Для зниження витрат компанії треба збільшувати рівень автоматизації та переводити все більше і більше процесів в режим з найменшим залученням людини. З іншого боку клієнти не завжди радіють такий автоматизації, бо відчутна частина їх запитів не підтримується програмним забезпеченням системи і вони не можуть бути опрацьовані.

Використання чат-ботів з підтримкою штучного інтелекту покращує задоволеність клієнтів процесом спілкування, але слабка здатність системи вирішувати нетипові запити все ще є проблемою [1, 2]. Кожен нетиповий запит потребує певного аналізу людиною та можливо отримання дозволів на його виконання. А після того, як він стає одним з типових сценаріїв – він має бути доданий до системи.

Обробка запиту та інші бізнес-процеси компанії зазвичай описуються робочим процесом (workflow). Робочий процес зазвичай складається з певної послідовності типових операцій. Це можуть бути операції виконання певних запитів до інших систем, очікування додаткової інформації від клієнта, операції вибору подальшого шляху обробки запиту та інші подібні операції.

Завдяки тому, що базові операції повторюються в різних робочих процесах, їх можна реалізувати окремо та викликати в ході виконання робочого процесу. Таким чином, реалізація бізнес-процесів буде виглядати як послідовний виклик стандартних операцій під управлінням певного управляючого процесу – оркестратора. Його задачею є управління послідовністю виклика операцій робочого процесу та обробка особливих ситуацій під час його виконання.

Найпоширенішою моделлю робочого процесу є направлений ациклічний граф (directed acyclic graph, DAG), в якому вузлами графу є операції, а гілками графу – переходи між ними. Вузли можуть виконувати як бізнес-операції (виконати запит до платіжної системи або надіслати інформацію користувачеві), так і службові операції або перевірки (чи відповідає користувач, або його замовлення певним умовам – наприклад чи перший раз він використовує промокод). Гілки ж задають допустимі послідовності операцій.

Типовими реалізаціями оркестраторів графових моделей робочого процесу є платформи Apache Airflow [3] та Step Functions в хмарній платформі Amazon Web Services [4]. Вони дозволяють реалізувати довільний ациклічний граф робочого процесу з базових та допоміжних операцій. Кількість моделей робочих процесів та процесів, що одночасно виконуються є доволі великою, але вона обмежена ресурсами інфраструктури виконання.

З ростом кількості моделей робочих процесів та збільшенням відсотка операцій очікування, зручність використання наведених вище засобів оркестрування знижується. Особливо це відчутно при їх використанні для автоматизації процесів взаємодії з клієнтом, де система може доволі довго чекати на відповідь. Причиною таких обмежень є те, що оркестратор має відслідковувати стан виконання процесу та зберігати його контекст, тобто дані, що мають передаватися між

© Колбасін В.О., 2024

Дослідницька стаття: Цю статтю опубліковано видавництвом ***НТУ «ХПІ»*** у збірнику «Вісник Національного технічного університету "ХПІ" Серія: Системний аналіз, управління та інформаційні технології». Ця стаття поширюється за міжнародною ліцензією Creative Common Attribution (CC BY 4.0). **Конфлікт інтересів:** Автор/и заявив/или про відсутність конфлікту.

операціями. А для цього потрібні певні ресурси процесора та оперативної пам'яті.

Таким чином використання найбільш поширених платформ для оркестрування виконання робочих процесів має обмеження за рахунок наявності в них централізованого відстеження ходу виконання процесу та зберігання його контексту. Зменшення потреби в обчислювальних ресурсах та розробка більш гнучкого механізму оркестрування робочих процесів взаємодії з клієнтом в корпоративних застосунках є актуальною задачею, яка розглядається в даному дослідженні.

Виконання робочого процесу. Графова модель робочого процесу задається спеціальним описом графа у форматі JSON для платформи AWS Step Functions та програмою спеціального виду на мові програмування Python для платформи Airflow [3, 4]. Дана модель зберігається у централізованому сховищі платформи оркестрування та зчитується з нього при запуску процесу. Платформа відстежує результат виконання поточної операції процесу та залежно від нього виконує перехід на виконання наступної операції.

Процес взаємодії з клієнтом є за своєю природою асинхронним і очікує різної поведінки сторін. Клієнт може відповісти на питання системи або надати додаткову інформацію в будь який момент часу протягом певного періоду очікування. Якщо він не відповідає – система може йому нагадати про потребу надати дані. Якщо ж клієнт на взаємодіє з системою протягом певного періоду часу – сеанс взаємодії буде завершений та виконання бізнес-процесу перервано. Від системи навпаки очікується швидка відповідь та здатність спілкуватися з клієнтом в будь який час доби.

Приклад робочого процесу взаємодії з клієнтом для надання йому інформації про стан рахунку наведено на рис. 1, де жовтим позначені звичайні операції робочого процесу, синім – операції очікування та червоним – кінцеві операції процесу.

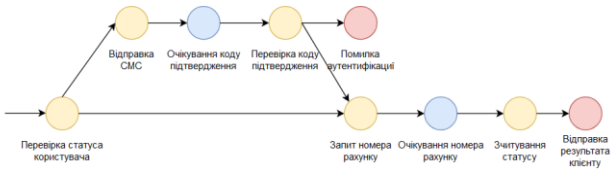

Рис. 1. Робочий процес отримання стану рахунку.

При такому сценарії взаємодії більшість часу виконання робочого процесу буде витрачатися на очікування відповіді від клієнта. Але пряма його реалізація базовими засобами того ж Airflow призведе до блокування одного зі слотів виконання на весь час очікування. Реалізація через відкладені оператори та тригери [5] є економнішою, але вона є схильною до помилок та також використовує ресурси відповідного обмеженого компоненту платформи.

Відмовитися від явної операції очікування можна зміною моделі виконання робочого процесу з умовно послідовного виконання операцій на графі на модель, засновану на обробці подій [6, 7]. Подія представляється повідомленням, що несе інформацію про подію – в даному випадку це може бути відповідь від клієнта та запитані дані, або результат обробки даних клієнта системою. Подія передається споживачу за допомогою брокера повідомлень, такого як RabbitMQ [8] або Apache Kafka [9]. Споживач подій виконує обробку повідомлень та передає результат у вигляді іншого повідомлення в певну чергу.

В цій моделі операції робочого процесу будуть виконуватися під час обробки подій, а перехід до виконання наступної операції буде відображатися надсиланням повідомлення з описом цієї операції.

Модель робочого процесу може виконуватися централізовано, і тоді після закінчення виконання кожної операції робочого процесу повідомлення з його результатами має надсилатися диспетчеру (споживач спеціального типу), який буде відповідати за аналіз результатів та вибір наступної операції. Іншим варіантом є покладення обов'язків робити вибір наступної операції робочого процесу на споживачів, що виконують операції бізнес-процесу.

Використання другого підходу є компромісним і разом з очевидними перевагами у вигляді зменшення числа елементів системи містить і певні ризики, бо вимагає строго однакової обробки моделі процесу всіма обробниками.

Приймаючи до уваги, що процес взаємодії з користувачем може бути доволі тривалим і має потребу в постійному покращенні та адаптації під потреби підприємства та клієнтів, ймовірно, що одномоментно будуть існувати різні версії одного й того ж самого робочого процесу. Для забезпечення послідовності виконання існуючих процесів різних версій при централізованому підході до зберігання моделі треба буде зберігати різні версії моделі робочого процесу і у повідомленнях передавати ідентифікатор версії. З іншого боку, враховуючи не дуже великий розмір типових робочих процесів взаємодії з клієнтом, в даній роботі пропонується зберігати всю модель робочого процесу як частину повідомлення.

Окремі операції можуть виконуватися як за допомогою загальних обробників повідомлень, так і з використанням спеціалізованих обробників, що можуть відповідати за розпізнавання мови та зображень (вони потребують специфічного апаратного забезпечення) або знаходитися всередині внутрішньої захищеної мережі компанії. У випадку використання спеціалізованих обробників треба забезпечити можливість диспетчеризації повідомлень залежно від типу операції.

Очікування відповіді від клієнта є частиною робочого процесу та має описуватися окремим повідомленням. На час очікування це повідомлення має бути збережено в певному сховищі, а потім при появі відповіді клієнта, має бути надіслано обробнику.

Зазвичай інформацію від клієнта система отримує через механізм зворотного виклику (callback), який не є специфічним для клієнта – тобто повідомлення від всіх клієнтів приходять на одну точку входу. Більш того, повідомлення від клієнта може бути початковим для старту взаємодії з системою. Такі повідомлення треба розрізняти від тих, що є відповіддю клієнта. А для останніх треба отримати робочий процес, що чекає відповіді, та збережене повідомлення. Це можна

зробити за рахунок використання ідентифікатора взаємодії з клієнтом, який може бути як випадковим ідентифікатором сесії у випадках веб-чату, так і постійним ідентифікатором користувача у месенджері.

З урахуванням всього наведеного вище структурна схема виконання робочого процесу буде виглядати як наведено на рис. 2.

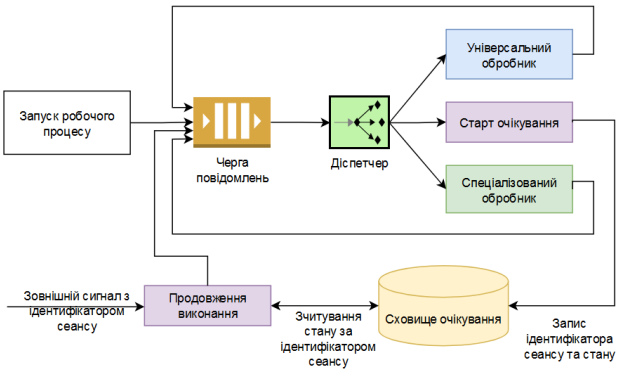

Рис. 2. Структурна схема виконання робочого процесу.

Виконання робочого процесу запускається ззовні надсиланням відповідного повідомлення з даними та моделлю процесу у чергу фреймворку. Далі відбувається виконання операцій процесу загальними та спеціалізованими обробниками, кожен з яких після закінчення виконання операції обчислює наступний стан та надсилає повідомлення зі зміненим станом процесу назад до черги. Так повторюється до тих пір, поки виконання процесу не закінчиться.

Для програмної реалізації даного підходу треба створити відповідні обробники та модулі взаємодії з зовнішніми системами взаємодії з клієнтом. Механізм диспетчеризації повідомлень також має бути реалізований програмно.

При використанні мови програмування Python для створення системи оркестрування є сенс розглянути фреймворк Celery [10], який реалізує розподілену чергу задач, яка управляється повідомленнями. Він дозволяє робити ациклічні моделі робочих процесів на рівні Python коду, але не підтримує напряму зупинки обробки задачі та її подовження. Тобто його можна використати для виконання лінійних частин графу без очікування. Також одним з напрямків подальших досліджень є вивчення можливостей адаптації даного фреймворку для реалізації очікування зовнішнього сигналу, що управляється повідомленнями [11].

Структура повідомлень. Повідомлення має містити всю необхідну інформацію для виконання поточної та всіх наступних операцій робочого процесу. Це означає, що воно має зберігати модель робочого процесу, ідентифікатор поточної операції та контекст виконання. Всі ці дані мають бути розташовані у повідомленні, для серіалізації якого в даній роботі пропонується використати формат JSON [12].

Контекст виконання робочого процесу потрібен для забезпечення зв'язку по даним між операціями. Оскільки оркестратор на може знати всі можливі поля даних та їх комбінації – контекст має підтримувати довільний формат даних, що зручно JSON об'єктом.

Кожна операція повинна мати свій власний унікальний ідентифікатор в моделі, також вона має містити всю необхідну інформацію для виклика операції обробником. Пропонується використати наступний опис операції:

- **handler** – опис обробника, визначає який обробник має виконати операцію;
- **function** – функція, яка має бути виконана обробником, містить посилання на код функції;
- **parameters** – параметри виконання функції;
- **next** – операція, що має бути виконана після виконання даної.

Будь-яка функція отримує два параметри – контекст виконання та параметри самої функції. Обидва параметрі є словниками, отриманими через десеріалізацію відповідних JSON об'єктів.

Службові операції на кшталт вибору гілки графу або запуску паралельних операцій помічаються спеціальним значенням типу обробника **system** і вони мають бути опрацьовані тим самим обробником, що і операція перед ними.

З використанням даного підходу до опису робочого процесу, приклад процесу отримання стану рахунку, наведений на рис. 1, буде виглядати так:

```
{
    "checkAuth": {
        "function": "com.company.CheckAuth",
        "parameters": {
            "result": "authStatus"
        },
        "next": "authChoice",
    },
    "authChoice": {
        "handler": "system",
        "function": "choice",
        "parameters": {
            "var": "context.authStatus",
            "options": {
                "1": "askAccount",
                "0": "startAuth"
            }
        }
    },
    "startAuth": {
        "function": "com.company.Request2FA",
        "next": "check2FA"
    }
    "check2FA": {
        "function": "com.company.Check2FA",
        "parameters": {
            "result": "authResult"
        },
        "next": "2FAChoice"
    },
```

Рис. 3. Фрагмент опису процесу отримання стану рахунку.

Як можна побачити на цьому описі, інформація про те, чи потрібна аутентифікація, отримується окремою функцією та її результат записується в змінну **authStatus** контексту. Далі цей самий обробник перевіряє значення змінній контексту та залежно від нього надсилає у чергу повідомлення з активним станом **askAccount** або **startAuth**.

Операція **startAuth** надсилає СМС клієнту та очікує його відповіді. Вона має спочатку зберегти стан моделі в сховищі і тільки після цього надіслати повідомлення клієнту для запобігання виникнення стану гонки (race condition).

Після отримання повідомлення від клієнта, обробник зворотного виклику зчитує повідомлення за ідентифікатором сеансу, додає до контексту отримане від клієнта значення та надсилає повідомлення в чергу. Обробник перевіряє наявність отриманого від клієнту значення і надсилає в чергу повідомлення з активним станом **check2FA**.

Висновки. У ході дослідження було розглянуто оркестрування робочих процесів взаємодії з клієнтом та було виявлено, що популярні платформи не дуже економно виконують очікування відповіді від клієнта. Було запропоновано підхід до оркестрування таких процесів на основі архітектури обробки подій та визначені головні структурні елементи оркестратора та сформульовано базовий формат опису моделі процесу. Результати дослідження можуть знайти використання при розробці корпоративних інформаційних систем, зокрема систем електронної комерції.

Список використаної літератури

1. Uzoka A., Cadet E. Ojukwu P.U. Leveraging AI-Powered chatbots to enhance customer service efficiency and future opportunities in automated support. *Computer Science & IT Research Journal*. 2024. Vol. 5(10). P. 2485–2510. DOI: 10.51594/csitrj.v5i10.1676.
2. Hsu C.-L., Lin J.C.-C. Understanding the user satisfaction and loyalty of customer service chatbots. *Journal of Retailing and Consumer Services*, Vol. 71, 103211. DOI: 10.1016/j.jretconser.2022.103211.
3. Ruiter J., Harenslak B. *Data Pipelines with Apache Airflow*. Manning, 2021. 480 p.
4. Sbarski P., Cui Y., Nair A. *Serverless Architectures on AWS, Second Edition*. Manning, 2022. 256 p.
5. *Deferrable Operators & Triggers.* URL: https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/deferring.html (дата звернення: 10.12.2024).
6. Hohpe G., Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2003. 736 p.
7. Stopford B. *Designing Event-Driven Systems*. O`Reilly Media Inc., 2018. 171 p.
8. Roy G. *RabbitMQ in Depth*. Manning, 2017. 264 p.
9. Shapira G, Palino T., Sivaram R., Petty K. *Kafka: The Definitive Guide, 2nd Edition*. O`Reilly Media Inc., 2021. 485 p.
10. Zhang C. *Celery. A detailed guide on how Celery asynchronous task queue works*. URL: https://python.plainenglish.io/asynchronous-task-queuing-with-celery-d9709364e686. (дата звернення: 10.12.2024)
11. Custom Message Consumers. *Celery documentation*. URL: https://docs.celeryq.dev/en/latest/userguide/extending.html#custom-message-consumers. (дата звернення: 10.12.2024)
12. Bray T. *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF, 2017. DOI: 10.17487/RFC8259.

References (transliterated)

1. Uzoka A., Cadet E. Ojukwu P.U. Leveraging AI-Powered chatbots to enhance customer service efficiency and future opportunities in automated support. *Computer Science & IT Research Journal*. 2024. Vol. 5(10). P. 2485–2510. DOI: 10.51594/csitrj.v5i10.1676.
2. Hsu C.-L., Lin J.C.-C. Understanding the user satisfaction and loyalty of customer service chatbots. *Journal of Retailing and Consumer Services*, Vol. 71, 103211. DOI: 10.1016/j.jretconser.2022.103211.
3. Ruiter J., Harenslak B. *Data Pipelines with Apache Airflow*. Manning, 2021. 480 p.
4. Sbarski P., Cui Y., Nair A. *Serverless Architectures on AWS, Second Edition*. Manning, 2022. 256 p.
5. *Deferrable Operators & Triggers.* URL: https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/deferring.html (access date: 10.12.2024).
6. Hohpe G., Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2003. 736 p.
7. Stopford B. *Designing Event-Driven Systems*. O`Reilly Media Inc., 2018. 171 p.
8. Roy G. *RabbitMQ in Depth*. Manning, 2017. 264 p.
9. Shapira G, Palino T., Sivaram R., Petty K. *Kafka: The Definitive Guide, 2nd Edition*. O`Reilly Media Inc., 2021. 485 p.
10. Zhang C. *Celery. A detailed guide on how Celery asynchronous task queue works*. URL: https://python.plainenglish.io/asynchronous-task-queuing-with-celery-d9709364e686. (access date: 10.12.2024).
11. Custom Message Consumers. *Celery documentation*. URL: https://docs.celeryq.dev/en/latest/userguide/extending.html#custom-message-consumers. (access date: 10.12.2024)
12. Bray T. *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF, 2017. DOI: doi:10.17487/RFC8259.

Надійшла (received) 06.12.2024

UDC 004.9

V. O. KOLBASIN, Candidate of Technical Sciences (PhD), Docent, Associate Professor at the Department of System Analysis and Information-Analytical Technologies, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine; e-mail: viacheslav.kolbasin@khpi.edu.ua; ORCID: https://orcid.org/0009-0005-6367-7574

ORCHESTRATION OF CUSTOMER INTERACTION WORKFLOWS IN ENTERPRISE APPLICATIONS

Process automation is an important factor in the development of modern corporate systems, in particular e-commerce systems. One of the tasks that must be solved during development of such systems is the orchestration of customer interaction processes, which are asynchronous in nature. Modern popular orchestration frameworks (Airflow for example), are not very well adapted to simultaneously waiting for a response from the client for a large number of workflows. This leads to unnecessary costs of computing resources and a decrease in the economic efficiency of the system. This paper considers the task of building a system for orchestrating customer interaction workflows based on an event-driven approach. Instead of a sequential-like centralized execution of the graph model of the workflow, it is proposed to present the workflow as a sequence of events and the system's reactions to them, which eliminates the need to explicitly wait for a response from the client. In such model workflow operations will be performed during event processing, and the transition to the next operation will occur by sending a new message with a description of the operation. Centralized and distributed approaches for performing transitions between workflow operations are considered, the advantages and disadvantages of each of them are shown. The implementation of waiting for client response is also considered, for which it is proposed to store the message tied to the client session identifier in a specialized storage before starting interaction with client. And after receiving data from the client, add them to the message and send it back to the message queue to continue the workflow. Taking into account the possibility of using handlers written in different programming languages, JSON format based description of workflow model is proposed to be used. One of the approaches to building model description format is described and its use for demonstrating example of a workflow. The results of the study can be useful when creating systems for orchestrating workflows of interaction with the client.

Keywords: workflow orchestration, event-driven approach, client interaction, asynchronous operations.

Повне ім'я автора / Author's full name

Колбасін Вячеслав Олександрович / Kolbasin Viacheslav Oleksandrovych