УДК 004.42 http://doi.org/10.15407/pp2022.02.037

Ю. Дивак, Т. Мамедов

ІНТЕГРАЦІЯ І КОМПОЗИЦІЯ ВЕБ-СЕРВІСІВ НА ОСНОВІ МОДЕЛІ

Анотація. В цій роботі я хотів би представити новий підхід для розробки застосунку і архітектури для роботи із семантичними веб-сервісами який інтегрує декілька корпоративних застосунків та поєднує результати з різних сервісів через застосування технік, методологій та анотацій, забезпечених технологіями розробки програмних систем і моделюванням бізнес-процесів. Я пропоную застосовувати існуючі техніки моделювання бізнес-процесів (BPMN, BPLWS, OWL-S, WSDL, WebML) для моделювання процесів, які проходять через декілька застосунків, для інтеграції програмних систем та композиції веб-сервісів. Головна мета - знайти методологію для розробки семантично багатих веб-застосунків з напівавтоматичним визначенням семантичного опису сервісу для моделі бізнес процесів. Це підвищить якість проєктування та зменшить обсяг додаткової роботи при розробці застосунків з обробкою семантично анотованої інформації, що проходить через корпоративні застосунки.

Ключові слова: Інтеграція програмних систем, Композиція веб-сервісів, Моделювання бізнес-процесів, Семантичні веб-сервіси, Проєктування, засноване на моделюванні, Методології

Вступ

Працюючи інтеграційним інженером і інтегруючи застосунки з усього світу, я виділив декілька проблем, з якими мав справу щодня. Серед них інтеграція двох незалежних застосунків між собою, інтеграція веб-сервісів із застосунками, і композиція веб-сервісів для отримання більш відповідного результату за менший проміжок часу. Звісно, ці проблеми можна розглядати і вирішувати окремо одна від одної, але, якщо розглядати їх разом, можна знайти нові шляхи вирішення. Дана стаття – це спроба знайти новий підхід для вирішення поставлених задач.

Усі аспекти корпоративного світу розроблені на основі бізнес-процесів, і часто ці бізнес-процеси проходять через декілька корпоративних застосунків, а іноді ці застосунки належать різним вендорам. Робота через декілька застосунків потребує асинхронного зв′язку між ними і здійснюється за часто змінюваними сценаріями. Такі інтеграційні рішення, як описані в [1] : **Файл трансфер**, **Спільна база данних**, **Віддалений виклик процедур** та **Надсилання повідомлень** – це опис підходів у загальному плані, конкретні реалізації інтеграцій. Такі, зокрема, як **мости**, **інтеграція через дані**, **надсилання повідомлень**, **інтеграція на рівні компонентів**, **інтеграція через інтерфейси**, **брокери**, **(ESB) enterprise service buses**, **інтеграційні сервери** та **інтеграційні архітектури**. Всі вони мають обмеження та застосовуються в конкретних зважених випадках. Якщо ми хочему обійти деякі обмеження ми можемо застосувати **семантичні веб сервіси, саме семантика дозволить обійти деякі обмеження**. Семантичні веб сервіси - це парадигма програмування яка базується на анотації процесів та само-описуючої реалізації, яка може бути застосована для побудови крос-корпоративних за стосунків, що вирізняються **гнучкістю**, **автоматичним виявленням ресурсів**, та **динамічним розвитком застосунку**. Одна з найважливіших задач це адаптувати семантичні технології та семантичні анотації до вже розроблених коммпонентів та застосунків. Напівавтоматичний підхід може бути застосований для виділення семантичного опису веб-сервісів.

Запропонований підхід базується на принципах розробки за стосунків, які працюють на основі моделей. Це означає, що спочатку ми маємо тільки модель, а згодом на основі цієї мо-

делі застосунок будує схему виконання процесів, в якій будує композицію веб-сервісів, шукаються описи веб-сервісів у спеціальних репозиторіях. Далі згідно зі схемою виконуються запити до веб-сервісів, збираються і компонуються прийнятні відповідно до критеріїв результати. Результат отримує користувач програмної системи. Інший підхід, побудований на основі роботи з моделлю процесів, описаний в [2] і [4], пропонується для додаткового вивчення.

Під час розробки нового підходу треба реалізувати декілька аспектів проєктування:

- Розробка продукту, який працює на вищому рівні абстракції глобальної хореографії веб-сервісів для взаємодії між застосунками і веб-сервісами.
- Робота згідно із запропонованою бізнес-моделлю. Модель не прив'язана до конкретних сервісів чи компонентів. Опис композиції веб-сервісів, забезпечений за допомогою анотацій, дає додатковий рівень абстракції моделі.
- Архітектура пропонує рішення для роботи декількох корпоративних за стосунків, працюючих одночасно.
- Архітектура пропонує рішення для роботи з декількома веб-сервісами чи акторами.

Приклади застосування

У цій статті ми спробуємо застосувати такий підхід до реального випадку, де ми маємо за стосунок, який працює із сервісами. Цей застосунок має комбінувати результати виконання декількох сервісів в одну задовільну відповідь кінцевому користувачеві згідно з наданої бізнес-моделі. В загальному плані композиція виглядає, як на рисунку 1.

Схема запитів на рисунку 2 показує взаємодію між сервісами та місця, де запит може бути перервано. Під час дослідження ми маємо брати до уваги, що наша система не повинна обмежуватися тільки двома взаємодіючими сервісами. Вона має працювати із сотнями і тисячами різних сервісів, налаштовувати взаємодію правильно та підбирати найбільш відповідні сервіси для композиції та об'єднувати результати їх виконання.

Ми можемо застосувати однаковий підхід для будь-якого випадку, де нам потрібна композиція декількох сервісів. Другий приклад, зображений на рисунку 3 та 4, описує композицію сервісу бронювання квитків на літак і бронювання номера в готелі.

Загальні принципи побудови системи

У загальному прикладі SOAP архітектури ми маємо декілька рівнів:

Рівень користувачів сервісів – в нашому конкретному випадку користувачі сервісів - це за стосунки, розроблені різними способами (B2B, Portals, .NET, JS, Java, Salesforce, Microsoft Dynamics CRM, etc.), які користуються сервісами через медіатор і отримують скомпонований результат виконання запиту з декількох сервісів. Це можуть бути як портали, готові рішення на основі CRM систем, або написані програми, які можуть бути взагалі без серверної частини і забезпечувати до-

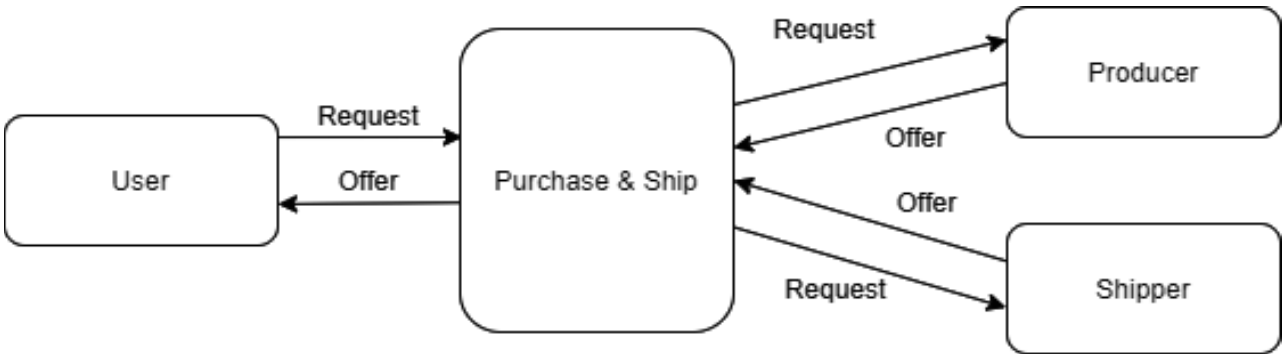

Рисунок 1. Схема композиції сервісів (приклад 1)

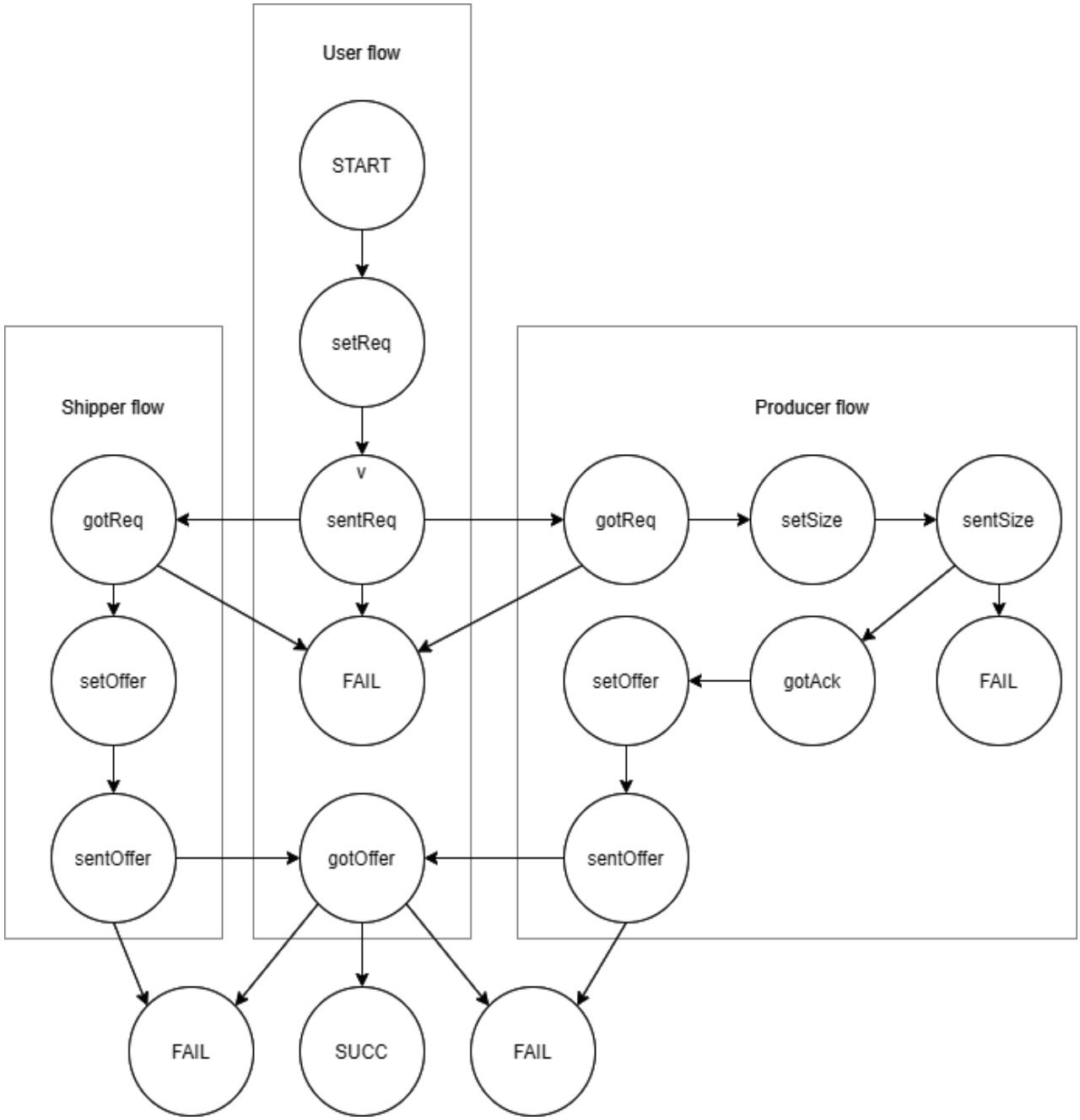

Рисунок 2. Схема виконання (приклад 1)

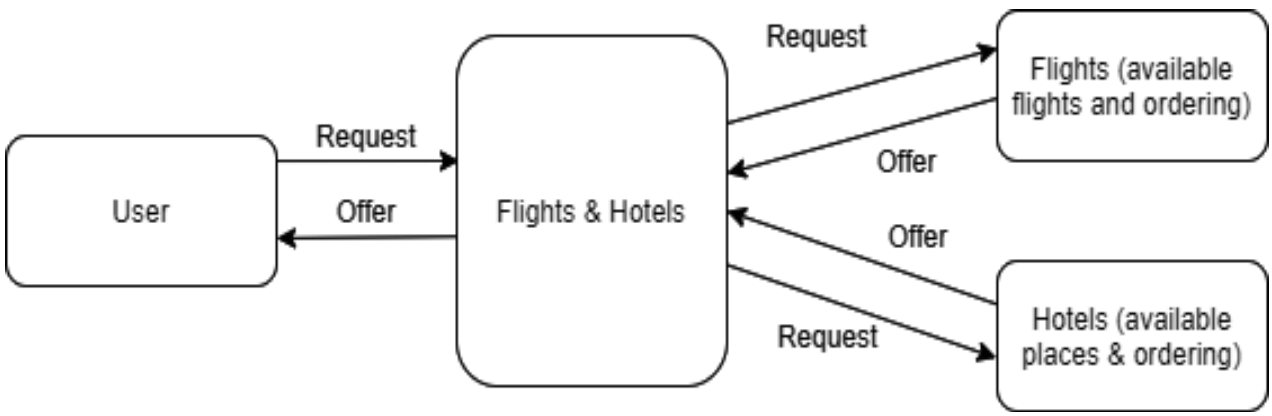

Рисунок 3. Схема композиції сервісів (приклад 2)

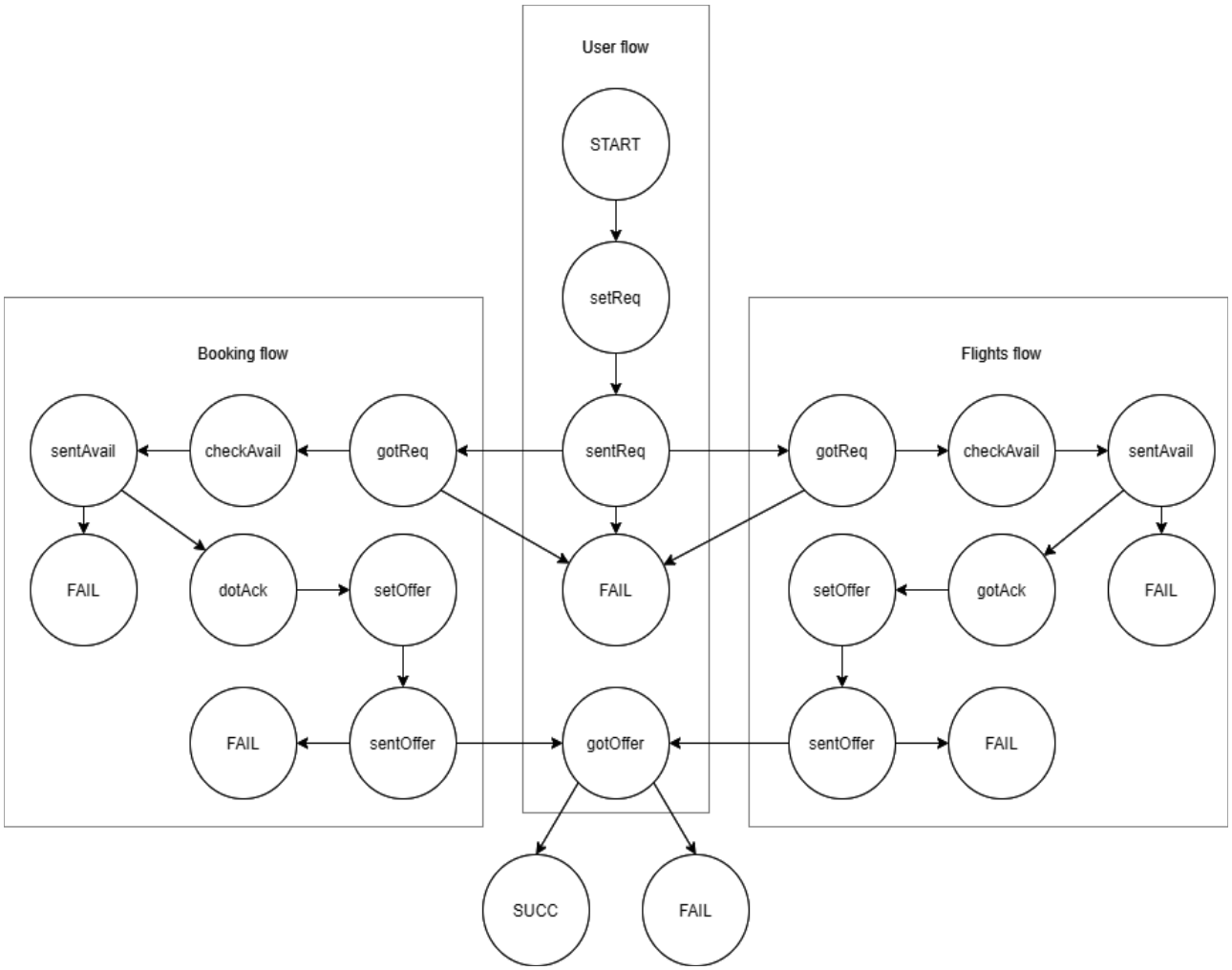

Рисунок 4. Схема виконання композиції (приклад 2)

ступ через медіатор до сервісів через розроблений UI, або системи, в яких тільки частина серверного застосунку замінена на роботу через медіатор із зовнішніми сервісами.

Рівень бізнес-процесів – В нашому конкретному випадку це документ (скрипт, модель бізнес-процесів, BPEL процеси), який описує бізнес-процеси в семантичній манері. Ми отримуємо цей документ від клієнта і згідно з ним забезпечуємо композицію сервісів і повертаємо задовільний результат. Клієнт може не турбуватися про пошук і композицію свервісів, про все піклується медіатор.

Рівень сервісів (композиція) – в нашому випадку ми маємо сховище, або декілька окремих сховищ, де зберігаються описи сервісів в форматах: WSDL, XML, SCHEMA, або WS-Policy. Сховища для описів сервісів можуть бути різні. Наприклад, файлове сховище, SQL, або NoSQL база даних, UDDI репозиторії, де зберігається необмежена або обмежена кількість описів для сервісів, які ми можемо використовувати. Медіатор підключений до сховищ і здійснює пошук у сховищах сервісів, які найбільш відповідають описові сервісу в документі [4].

Рівень компонентів та операційних систем – цей рівень виходить за межі дослідження в цій статті і реалізація покладається на розробників сервісів.

Агрегатори – це за стосунки, які компонують результати виконання запитів у декілька сервісів, зазвичай служать спільній меті - комбінувати відповіді з декількох сервісів та трансформувати в загальну відповідь, яка підходить для кінцевого користувача. Зокрема, це може бути пошук певного продукту та додаткової інформації про нього. Водночас ми повинні агрегувати результати з різних сервісів (онлайн-магазинів) та комбіну-

вати їх в якусь структуровану відповідь. Інший агрегатор може знаходити рейси літаків від різних компаній, номери різних готелів та сортувати за прийнятною ціною або за датами, чи пересадками та поєднувати рейси з доступними номерами в готелях. Це все збирається та компонується у відповідний формат, зрозумілий кінцевому користувачеві (кінцевим користувачем в нашому прикладі є так само застосунки).

Агрегатори в свою чергу не можуть забезпечити дослідження простору інтернету для виявлення нових сервісів та використання їх у системі. Для цього існують спеціальні кравлери, такі ж, що використовуються, наприклад Гуглом для індексації веб-сторінок. У нашій системі може використовуватися кровлер, що збирає описи сервісів, а також ми можемо використовувати сховище, налаштоване вручну.

У цій статті ми поговоремо про декілька компонентів і розробок щодо побудови посередника, який може використовуватися замість частини бекенду.

Медіатор під'єднаний до сервісів або апішок та отримує дані, що відповідають переданим параметрам. Однак тут маємо проблему: якщо один сервіс стає недоступним, ми не можемо отримати з нього відповідь. Тому медіатор може отримати дані з інших сервісів, система динамічна.

Витрати на розробку сервісів або інтеграцій з уже існуючими сервісами, можуть бути знижені завдяки такій системі, адже вона може взяти на себе частину роботи з динамічної композиції вебсервісів.

За основу для нової архітектури системи для інтеграції із сервісами і композиції веб-сервісів взято стандартну архітектуру SOAP за стосунку, яка зображена на рисунку 5, і перероблено згідно з нашими вимогами до системи.

Вже існує схоже рішення і розробка схожого за стосунку, який забезпечує композицію сервісів. Цей застосунок розроблявся 2013 року групою дослідників, і принципи його роботи описані в [5]. Архітектуру системи iServe зображено на рисунку 6.

Проблеми

Маємо зважати на проблеми, що виникають під час розробки подібних систем, які стосуються виконання, проєктування та розробки системи.

Під час виконання ми можемо стикнутися з такими проблемами як: **Доступність сервісу; надійність системи,**

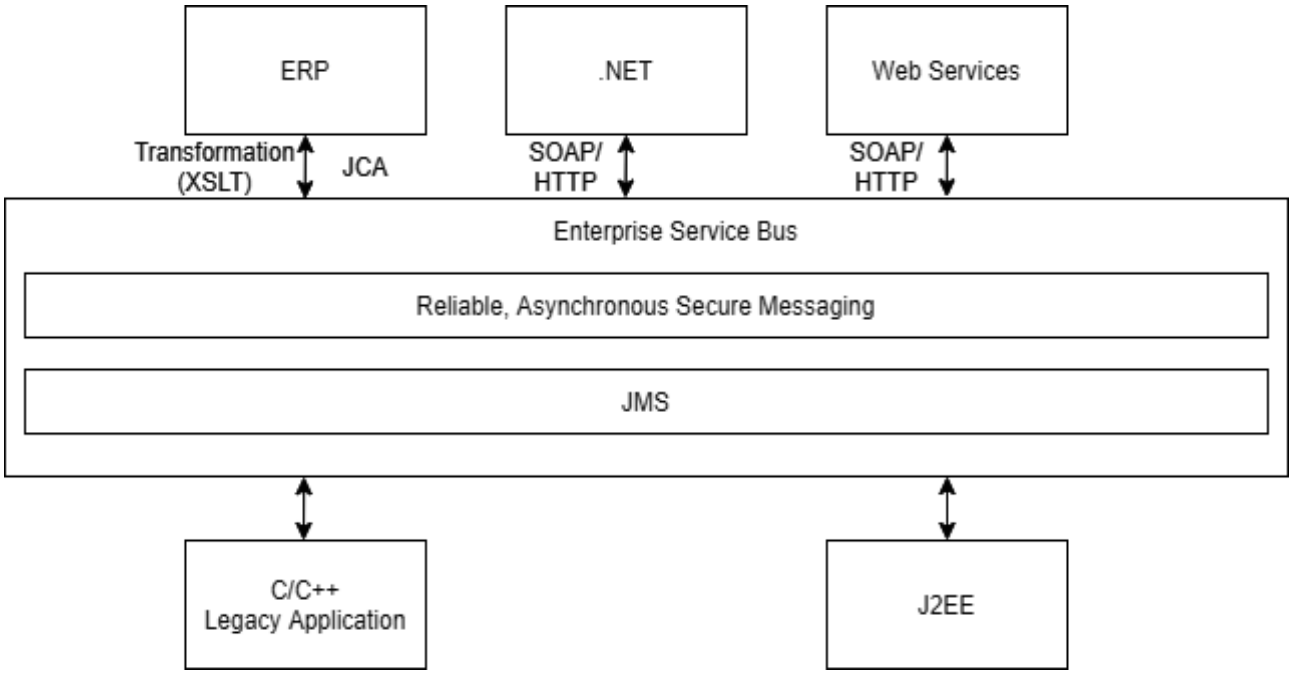

Рисунок 5. SOAP Архітектура

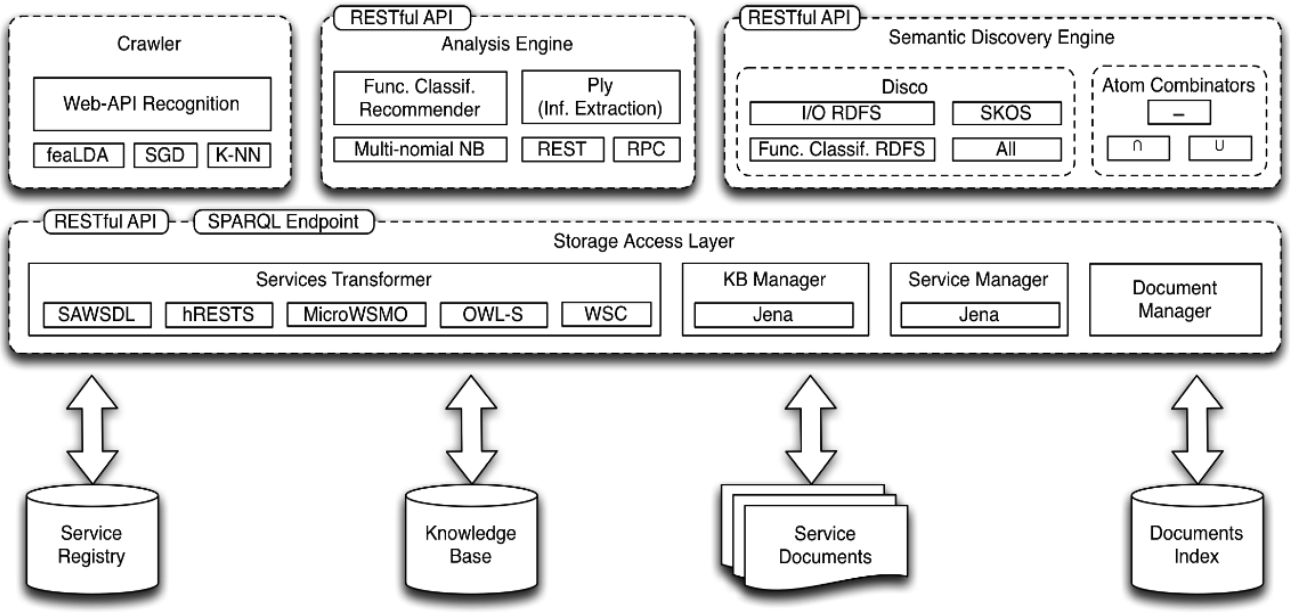

Рисунок 6. iServe архітектура

довговічність (як довго ми можемо її використовувати), масштабованість, зручність використання, безпека, можливість налаштування, продуктивність, обмеження.

Під час проєктування і розробки маємо інші проблеми: **можливість повторного використання, розширюваність, портативність, сумісність, модульність, тестованість, локалізація та інтернаціоналізація**.

Затримки

У подібних системах, де існує декілька компонентів, які взаємодіють через протокол HTTP, цей протокол є найслабшим місцем. Навантаження на систему може перевищити допустимі ліміти пропускання даних, що призведе до затримки відповіді на запити. Вирішенням такої проблеми може бути робота декількох таких систем одночасно на кількох серверах і балансування запитів між застосунками за допомогою наприклад NGINX або на рівні застосунків (балансування між застосунками відбувається на рівні компонентів). Можна також мати систему яка буде автоматично збільшувати кількість застосунків і серверів залежно від навантаження системи. Позаяк наша система складається з декількох компонентів, то їх можна запускати на окремих серверах, тим самим розподіляти навантаження між серверами.

Також можна розбивати нашу композицію на загальні частини і тоді результати виконання деяких частин, які використовуються найчастіше, зберігати в кеші застосунку для прискорення доступу до результатів виконання (Таким чином ми можемо отримати композицію композицій веб-сервісів).

Надійність

Інша проблема може виникнути, якщо один з сервісів, опис якого зберігається в сховищі, стає недоступним. Можливий випадок, коли ми безмежно довго чекаємо на відповідь сервісу. Але в нашому випадку медіатор автоматично перевіряє доступність сервісів для композиції. Якщо ж сервіс недоступний – шукає заміну з інших описів сервісів, які зберігаються в сховищі.

Відповідність результату

Проблема відповідності полягає в тому, чи відповідає результат виконання композиції тим параметрам і вимогам, які ставить користувач системи. Відповідність повністю лягає на бік медіатора.

Для забезпечення більшої відповідності результату виконання композиції вхідним параметрам, в нашій системі не потрібно кешувати окремі запити

до сервісів, бо нам потрібна тільки актуальна інформація, і ми легко можемо замінити один сервіс іншим у побудові маршруту, один результат виконання заміниться іншим.

Кешування може бути корисним, якщо ми хочемо прискорити виконання композиції. До прикладу, ми можемо зберігати на рівні кеша застосунку певну вже готову композицію і повертати результат виконання з кеша, водночас оминаючи повторну композмцію 2 - 3 - 4 сервісів (Композиція композицій).

Доступність

Інша проблема полягає в тому, що всі сервіси різні, і ми мусимо по-різному з ними взаємодіяти. Вирішенням може бути – взаємодія через єдиний інтерфейс, що забезпечить взаємозаміну сервісів іншими, зручність взаємодії із сервісами, у разі, якщо їх велика кількість і є швидкий перехід від одного сервісу до іншого.

В єдиному інтерфейсі також має бути передбачатися можливість перевірки сервісу на доступність і маркування його як недоступний або доступний, і періодичну перевірку доступності сервісів.

Компоненти

1. Користувачі сервісів

Найпростішим користувачем системи може бути звичайна HTML сторінка, що автоматично надсилає стандартний документ, який описує бізнес-процеси і композицію сервісів та написаний, наприклад, за допомогою BPLWS або OWL-S (модель бізнес-процесів) до нашої системи. Цей документ отримує відповідь і рендерить її на сторінці. Основна задача документу, який ми надсилаємо до медіатора - це опис станів, умов та параметрів для сервісів. За допомогою цього підходу ми зберігаємо час на розробку взаємодії із сервісами і доручаємо всю роботу медіатору. Той робить композицію в автоматичному режимі, повертаючи клієнту результат, який відповідає переданим параметрам. Обов'язковою є фронт-енд частина, коли клієнт може обійтися без серверної частини застосунку.

2. Модель бізнес-процесів

Документ (Execution script), який передається від клієнта до медіатора. Вочевидь цей документ - звичайний текстовий файл, складений за допомогою певних правил і принципів, в якому є параметри такі, як передумови та умови, перевірки, сутності та моделі даних.

У цьому документі ми можемо описати дані (Модель даних), очікуючи, що вони будуть повернуті з медіатора до клієнта, а також які між ними зв′язки.

3. Медіатор

Медіатор приймає запити від клієнтів, контролює вхідні параметри, підбирає потрібні сервіси за параметрами, складає маршрут виконання композиції та виконує запити, компонує результат і відповідає клієнту. Це центральний і найважливіший компонент нашої системи, який отримує багато запитів на композицію, працює зі сховищами описів сервісів, перевіряє сервіси на доступність та працює із сервісами, збираючи відповіді. Коли медіатор парсить документ розбираючи всі вхідні і визідні параметри сервісів, він підбирає сервіси з бази і знаходить найкоротші маршрути. Медіатор у постійному контакті із сервісами.

Ми можемо маніпулювати параметрами також на рівні побудови маршруту для отримання найкращого результату (преобробка), та обробляти дані, отримані із сервісів вже згодом (постобробка) таким чином досягяючи найкращого результату виконання композиції. Важливо чітко розуміти, який результат ми хочемо отримати, який результат композиції для досягнення більш точного результату. Це дозволить підібрати відповідні сервіси з більш надійними результатами, скласти більш вдалий маршрут виконання композиції сервісів, а на етапі пост обробки - узагальнити результати так, щоб задовольнити вимоги клієнтів застосунку.

4.Сховище описів сервісів

Опис веб сервісу може зберігатися по-різному. Наприклад, це може бути

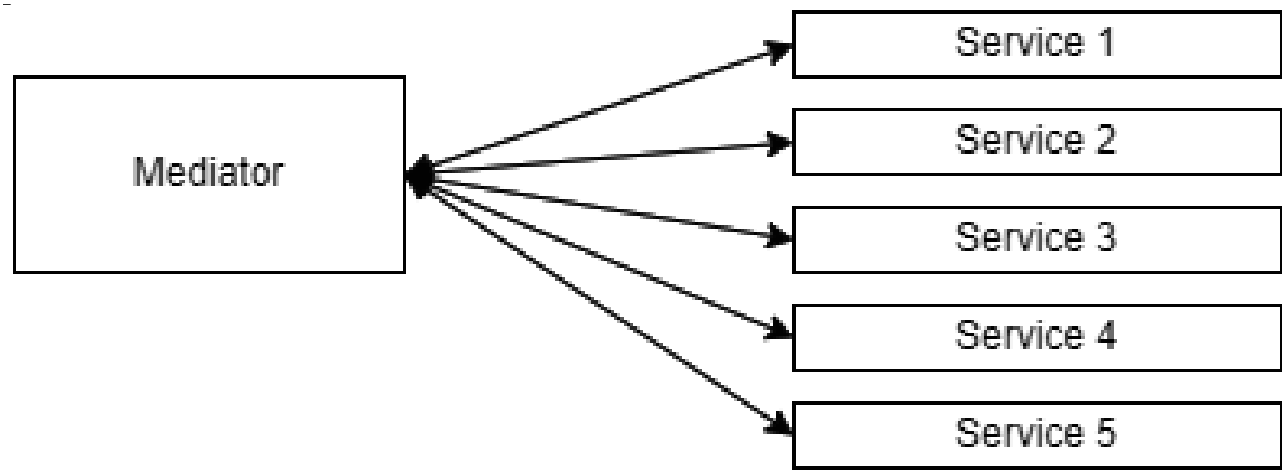

Рисунок 7. Взаємодія медіатора з сервісами

файлове сховище, де зберігаються текстові документи у форматі WSDL (як зроблено частково в застосунку iServe [5]) з описами сервісів. Це можуть бути описи, які зберігаються в реляційній базі даних, з якої ми отримуємо опис в зібраному вигляді з декількох таблиць. Це може бути також документоорієнтована база, яка використовує JSON як документ (NoSql). Описи зберігають інформацію про ендпоінти, URLs, вхідні параметри сервісу, вихідні параметри сервісу, кондішени та прекондішени, моделі тощо. Також ми можемо використовувати UDDI репозиторії як сховища для описів сервісів (як зроблено частково в застосунку iServe[5]), публічні або приватні репозиторії. В цьому випадку наш медіатор стає користувачем репозиторію [3].

5. Веб-кравлер

Що ж до веб-кравлера, ми маємо на увазі окремий за стосунок, який в автоматичному режими здійснює пошук мережею інтернет доступних веб-сервісів. Прикладом застосування такого типу систем є індексація гуглом сторінок в інтернеті. Таким чином наша база даних описів веб-сервісів для подальшої взаємодії може весь час доповнюватися, а це забезпечить кращий результат виконання композиції і взаємодії із сервісами.

Архітектура

На рисунку нижче ми можемо бачити, як виглядає вся система в загальному плані. Всі компоненти системи працюють разом у тісній взаємодії для досягнення єдиного результату. А саме:

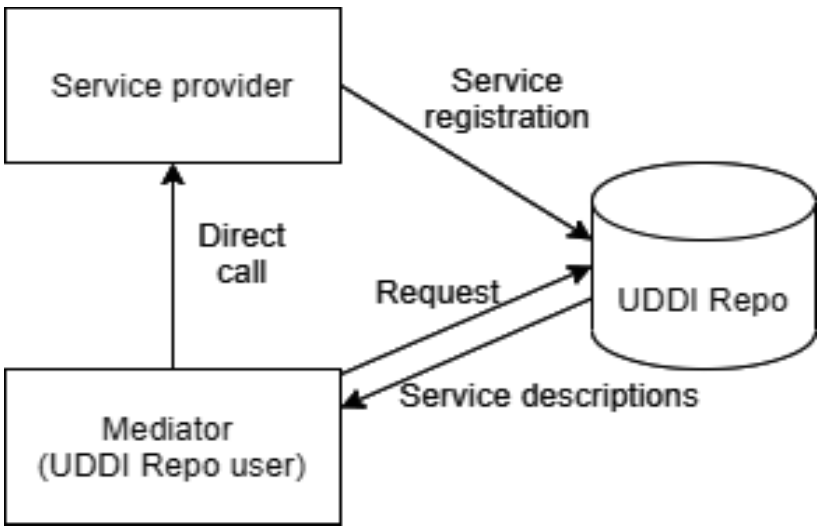

Рисунок 8. Взаємодія медіатора з репозиторіями і конкретним сервісом

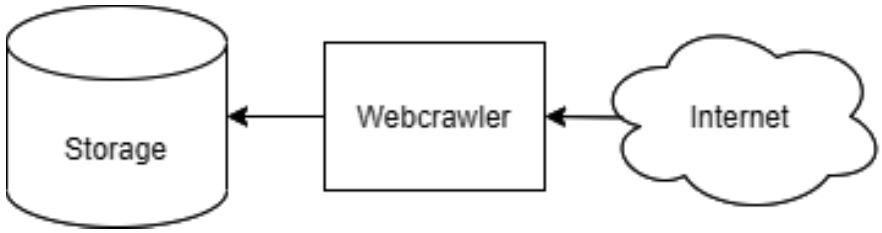

Рисунок 9. Додатковий веб-кровлер для автоматичного пошуку описів сервісів та додавання в базу

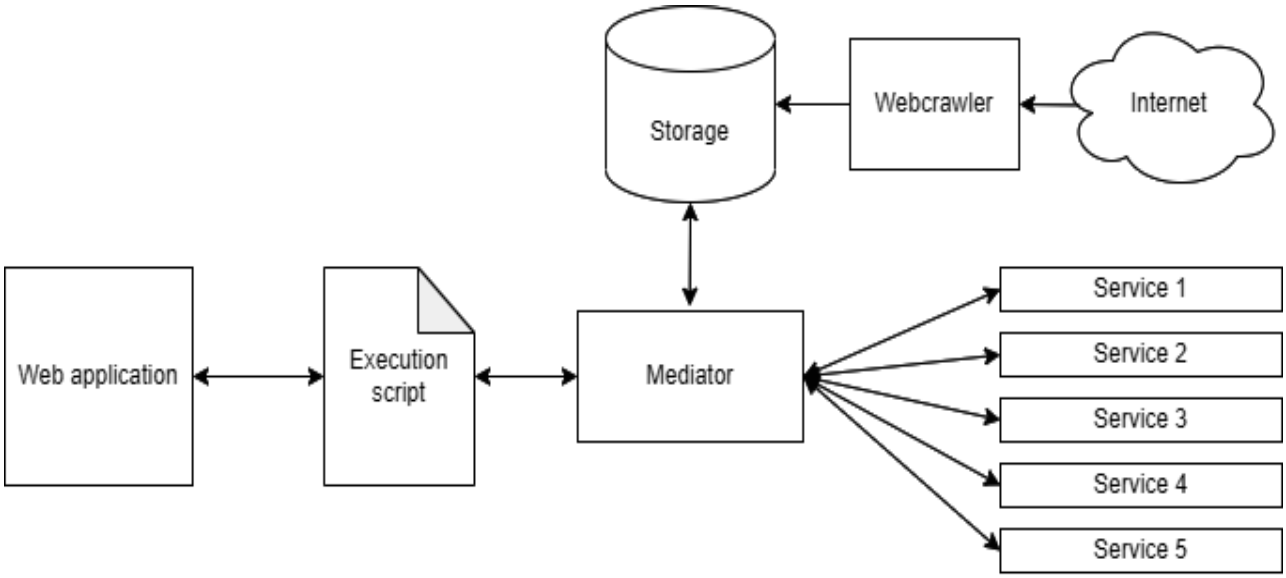

Рисунок 10. Архітектура для композиції сервісів

збір даних із зовнішніх джерел, композиція результатів виконання сервісів та конструювання загального результату виконання композиції, який відповідає вхідним параметрам і вимогам для результату. Одна з основних переваг такої системи - це (як ми бачимо на рисунку 10) гнучкість. Усі компоненти працюють незалежно і можуть бути покращені откремо один від одного без втручання у роботу інших компонентів. Клієнти застосунку можуть бути розроблені за допомогою будь-якої з відомих технологій розробки і проєктування програмних систем.

Висновок і майбутня праця

Нова архітектура може допомогти кінцевим користувачам отримати кращий результат за коротший проміжок часу і водночас значно скоротити витрати на розробку окремих компонентів, інтеграції та композиції сервісів. Цей підхід покращує вже існуючі підходи до композиції веб-сервісів й інтеграції із зовнішніми сервісами, а також дає можливість застосувати до цієї архітектури принципи антології (тому що інтеграція і композиція заснована на моделі). Наступні роботи будуть пов′язані із конкретною реалізацію медіатора і застосування алгоритмів для отримання кращого результату композиції та інтеграції, що забезпечить підвищення продуктивності та покращить інші характеристики системи. Ми спробуємо виміряти продуктивність системи із застосуванням різних підходів до композиції і оберемо найкращий варіант реалізації медіатора.

Алгоритми, які можуть бути застосовані при композиції веб-сервісів в рамках цієї архітектури.

- Євристичні алгоритми пошуку,
- A* Алгоритм,
- Мурашиний алгоритм,
- Генетичний алгоритм,

• Алгоритми динамічного програмування,

• Метод гілок і меж.

Залежно від вимог ми можемо обрати відповідніший алгоритм.

Література

1. Dyvak Y. A. (2021) Analytical review of existing approaches to integration of program systems.
2 Brambilla M., Celino I., Ceri S., Cerizza D., Della Valle E., Michele Facca F., (2006) A Software Engineering Approach to Design and Development of Semantic Web Service Applications.
3. Andon P. I. (2014) The Problems of programming in the semantic web environment. UkrPROG`2014
4. Arpinar B., Aleman-Meza B., Zhang R., Maduko A., (2012) Ontology-driven Web services composition platform.
5. C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert (2014) iServe: a Linked Services Publishing Platform

Отримано: 04.07.2022

Про авторів

Дивак Юрій, аспірант
Мамедов Турал, аспірант
ORCID: 0000-0001-6016-999X

Місце роботи:

Інститут програмних систем
НАН України, 03187, м. Київ-187,
проспект Академіка Глушкова, 40.
yurii.dyvak@gmail.com