НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ННК «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Системного проектування
(повна назва кафедри)

«На правах рукопису»

УДК 004:004.453

«До захисту допущено»

Завідувач кафедри

_____ А.І. Петренко
(підпис) (ініціали, прізвище)

«___» _____ 2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 122 – комп'ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва спеціальності)

на тему Семантична хореографія REST-сервісів

Виконав: студент 6 курсу, групи ДА-62м
(шифр групи)

Піпіч Артем Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доцент, к.т.н. Булах Б.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розробка стартап-проекту доцент, к.т.н. Булах Б.В. _____
(назва розділу) (науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ННК «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Системного проектування
(повна назва кафедри)

Рівень вищої освіти другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 122 – комп'ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва спеціальності)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ А.І. Петренко
(підпис) (ініціали, прізвище)

«___» _____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Піпічу Артему Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Семантична хореографія REST-сервісів

науковий керівник дисертації Булах Богдан Вікторович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. № _____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження: REST-сервіси

4. Предмет дослідження: ефективні способи організації хореографії REST-сервісів за використання семантичних технологій.

5. Перелік завдань, які потрібно розробити

- Проведення огляду концепцій та основних понять семантичної павутини
- Проведення порівняльного аналізу інструментів для використання семантичних технологій у веб-сервісах
- Дослідження підходів організації хореографій веб-сервісів

- Формулювання вимог до системи «Семантична хореографія REST-сервісів»
- Розробка та тестування додатку для вирішення поставленої задачі
- Аналіз результатів, отриманих в результаті роботи додатку
- Розроблення стартап-проекту «Семантична хореографія REST-сервісів»

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація на тему «Семантична хореографія REST-сервісів»

7. Орієнтовний перелік публікацій

Pipich A. The use of semantic web technologies for web services describing / Artem Pipich. // International Scietific Journal «Internauka». – 2018. – №8.

Pipich A. The use of choreography in saga design pattern / Artem Pipich. // International Scietific Journal «Internauka». – 2018. – №8.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Реалізація стартап-проекту	Булах Б.В., доцент		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1.	Отримання завдання	02.10.2017	
2.	Огляд літературних джерел	22.01.2018	
3.	Огляд концепцій семантичного вебу	12.02.2018	
4.	Дослідження підходів до побудови семантичних веб-сервісів та їх хореографії	05.03.2018	
5.	Розробка програмного додатку	25.03.2018	
6.	Хмарне розгортання додатку	09.04.2018	
7.	Тестування додатку	04.05.2018	
8.	Отримання допуску до захисту та подача роботи в ДЕК	14.05.2018	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ

виконану на тему: Семантична хореографія REST-сервісів

студентом: Піпічем Артемом Андрійовичем

Робота виконана на 82 сторінках, містить 5 ілюстрацій, 24 таблиці. При підготовці використовувалась література з 37 джерел.

Актуальність теми

На сьогоднішній день з’являється все більше систем, в яких використовується велика кількість веб-сервісів. Для організації їх ефективної взаємодії використовуються різні підходи, проте більшість з них мають свої переваги та недоліки, які часто стають критичними для певної ситуації.

Саме тому дослідження семантичної хореографії REST-сервісів як одного з можливих підходів до такої організації є актуальним. Використання даного підходу може дати суттєві результати при застосуванні в системах, в складі яких значну роль відіграють веб-сервіси.

Мета та задачі дослідження

Метою даної роботи є дослідження семантичної хореографії REST-сервісів а також способів використання даного підходу в системах, в складі яких значну роль відіграють веб-сервіси.

Рішення поставлених завдань та досягнуті результати

В роботі розглянуто засоби, за допомогою яких семантична хореографія REST-сервісів може бути ефективно реалізована. Запропоновано реалізацію такого підходу на основі обміну сервісами метаданими про запит через брокер повідомлень.

Було реалізовано описаний підхід, в реалізації застосовано патерн проектування Сага для ефективної обробки помилок, пов’язаних в тому числі і з комунікацією між сервісами.

Реалізацію було протестовано на багатьох тестових сценаріях; зроблено висновки щодо особливостей даного підходу, його переваг та можливостей покращення запропонованої реалізації.

Об’єкт досліджень

Системи з REST-сервісами.

Предмет досліджень

Взаємодія REST-сервісів із застосування хореографії, що реалізована за допомогою семантичних засобів.

Методи досліджень

Для розв’язання зазначеної проблеми в роботі застосовано методи синтезу та аналізу, системного порівняння та аналізу, композиції логічних структур даних та логічного узагальнення отриманих результатів.

Наукова новизна

Наукова новизна роботи полягає у реалізації нового підходу до семантичної хореографії REST-сервісів, який засновано на використанні брокеру повідомлень та патерні проектування Saga.

Практичне значення одержаних результатів

Отримані результати реалізації підходу можуть використовуватись в системах, в складі яких значну роль відіграють веб-сервіси. Представлений приклад реалізації демонструє, що отримані результати можуть бути використані для реалізації системи медичного обслуговування.

Апробації результатів дисертації

Результати попередніх досліджень опубліковано у міжнародному науковому журналі «Інтернаука», та можуть бути знайдені у випуску №8 2018 року.

Публікації

Піпіч А. А. Застосування семантичних веб-технологій для опису веб-сервісів // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.

Піпіч А. А. Застосування хореографії в шаблоні проектування Сага // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.

Ключові слова

Веб-сервіс, семантичний веб, онтологія, семантична мова, семантичний веб-сервіс, хореографія, патерн поведінки.

РЕФЕРАТ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

выполненную на тему: Семантическая хореография REST-сервисов
студентом: Пипичем Артемом Андреевичем

Работа выполнена на 82 страницах, содержит 5 иллюстраций, 24 таблицы. При подготовке использовалась литература из 37 источников.

Актуальность темы

На сегодняшний день появляется все больше систем, в которых используется большое количество веб-сервисов. Для организации их эффективного взаимодействия используются различные подходы, однако большинство из них имеют свои преимущества и недостатки, которые часто становятся критическими для определенной ситуации.

Именно поэтому исследования семантической хореографии REST-сервисов как одного из возможных подходов к такой организации является актуальным. Использование данного подхода может дать существенные результаты при применении в системах, в составе которых значительную роль играют веб-сервисы.

Цель и задачи исследования

Целью данной работы является исследование семантической хореографии REST-сервисов, а также способов применения данного подхода в системах, в составе которых значительную роль играют веб-сервисы.

Решение поставленных задач и достигнутых результатах

В работе рассмотрены средства, с помощью которых семантическая хореография REST-сервисов может быть эффективно реализована. Предложена реализация такого подхода на основе обмена сервисами метаданными о запросе через брокер сообщений.

Было реализовано описанный подход, в реализации применен паттерн проектирования Сага для эффективной обработки ошибок, связанных в том числе и с коммуникацией между сервисами.

Реализацию было протестировано на многих тестовых сценариях; сделаны выводы относительно особенностей данного подхода, его преимуществ и возможностей улучшения предложенной реализации.

Объект исследований

Системы с REST-сервисами.

Предмет исследований

Взаимодействие REST-сервисов с применением хореографии, реализованной с помощью семантических средств.

Методы исследований

Для решения указанной проблемы в работе применены методы синтеза и анализа, системного сравнения и анализа, композиции логических структур данных и логического обобщения полученных результатов.

Научная новизна

Научная новизна работы заключается в реализации нового подхода к семантической хореографии REST-сервисов, основанной на использовании брокера сообщений и паттерне проектирования Saga.

Практическое значение полученных результатов

Полученные результаты реализации подхода могут использоваться в системах, в составе которых значительную роль играют веб-сервисы. Представленный пример реализации показывает, что полученные результаты могут быть использованы для реализации системы медицинского обслуживания.

Апробации результатов диссертации

Результаты предварительных исследований опубликованы в международном научном журнале «Интернаука», и могут быть найдены в выпуске №8 2018 года.

Публикации

Пипич А. А. Применение семантических веб-технологий для описания веб-сервисов // Международный научный журнал "Интернаука". - 2018. - №8.

Пипич А. А. Применение хореографии в шаблоне проектирования Сага // Международный научный журнал "Интернаука". - 2018. - №8.

Ключевые слова

Веб-сервис, семантический веб, онтология, семантический язык, семантический веб-сервис, хореография, паттерн поведения.

ABSTRACT
ON MASTER'S THESIS

on topic: Semantic choreography of REST-services

student: Artem A. Pipich

Work carried out on 82 pages containing 5 figures, 24 tables. The paper was written with references to 37 different sources.

Topicality

To date, there are more and more systems that use a large number of web services. Various approaches are used to organize their effective interaction, however, most of them have advantages and disadvantages, which often become critical for a particular situation.

That is why the research of semantic choreography of REST-services as one of the possible approaches to such an organization is topical. The use of this approach can yield significant results when applied in systems in which a significant role is played by web services.

Purpose

The aim of this work is to investigate the semantic choreography of REST-services, as well as how to apply this approach in systems in which a significant role is played by web services.

Solution

In this paper, we examined the means by which the semantic choreography of REST-services can be effectively implemented. The implementation of this approach based on the exchange of services by metadata about the request through the message broker is suggested.

The described approach was implemented, the Saga design pattern was applied in the implementation for efficient error handling, including, among other things, communication between services.

The implementation was tested on many test scenarios; conclusions were drawn regarding the specifics of this approach, its advantages and possibilities for improving the proposed implementation.

The object of research

Systems with REST-services.

The subject of research

Interaction REST-services with the use of choreography, realized with the help of semantic means.

Research methods

To solve the described problem in this work methods of synthesis and analysis, system comparison and analysis, composition of logical data structures and logical generalization of the obtained results are applied.

Scientific novelty

The scientific novelty of the work is to implement a new approach to the semantic choreography of REST-services, based on the use of the message broker and Saga design pattern.

The practical value of research

The obtained results of the implementation of the approach can be used in systems in which a significant role is played by web services. The presented example of implementation shows that the results obtained can be used to implement the health care system.

Research results approbation

The results of pre-researches were published in the international scientific journal "Internauka", and can be found in issue No. 8 of 2018.

Publications

Pipich A. The use of semantic web technologies for web services describing / Artem Pipich. // International Scientific Journal «Internauka». – 2018. – №8.

Pipich A. The use of choreography in saga design pattern / Artem Pipich. // International Scientific Journal «Internauka». – 2018. – №8.

Keywords

Web Service, Semantic Web, Ontology, Semantic Language, Semantic Web Service, Choreography, Behavior Pattern.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

2PC - Two-Phase Commit Protocol

AC - API Consistency

API - Application Programming Interface

ACID - Atomicity, Consistency, Isolation, Durability

AD - API Difference;

AI - Alternative Implementations

AS - Additional Services

CO - Completeness

DSL - Domain Specific language

IP - Input Parameters

JMX - Java Management Extensions

OWL - Web Ontology Language

POM - Project Object Model

RDBMS - Relational Database Management System

RDF - Resource Description Framework

REST - Representational State Transfer

SBT - Scala Build Tool

SPARQL - SPARQL Protocol and RDF Query Language

STOMP - Streaming Text Oriented Messaging Protocol

SWOT - Strengths, Weaknesses, Opportunities, Threats

SWS - Semantic Web Services

UDDI - Universal Description Discovery & Integration

WSDL - Web Services Description Language

WSMF - Web Services Modeling Framework

WSML - Web Service Modeling Language

WSMO - Web Service Modeling Ontology

БЗ - База(и) знань

ВСТУП

При реалізації високорівневих процесів бізнес-логіки, в яких можуть брати участь різні підприємства, організація взаємодії Web-сервісів потребує стандартизації шаблонів їх взаємодії. Ряд стандартів галузі є близькими до імплементації в конкретних програмних продуктах. Два підходи до проектування процесів бізнес-логіки, що базується на організації взаємодії Web-сервісів, носять назви хореографії та оркестровки.

Стандарти хореографії і оркестровки мають відповідати набору технічних вимог, що описують проектування процесів бізнес-логіки за використання Web-сервісів. Ці вимоги застосовні як до мови для опису основного алгоритму у бізнес-процесі, так і до інфраструктури для його реалізації.

Можливість використовувати певний сервіс асинхронно є вкрай важливою для того, щоб гарантувати відмовостійкість, точність та універсальність, які на сьогоднішній день є обов’язковими атрибутами обчислювальних систем. Засобом для покращення ефективності процесу є можливість паралельного виклику одразу багатьох сервісів. Певний підхід до кореляції запитів є необхідним у випадку розробки асинхронних Web-сервісів.

Можливість керування винятковими ситуаціями і цілісністю транзакцій є вкрай важливою; така можливість має забезпечуватися структурою процесів бізнес-логіки. Крім обробки помилок і тайм-аутів Web-сервіси, до яких було застосовано підхід хореографії, мають забезпечувати доступність ресурсів під час проведення об’ємних розподілених транзакцій

Звичайні транзакції як правило не надто застосовні для застосуванні в об’ємних розподілених процесах бізнес-логіки, так як не надають можливості достатньо довго виконувати блокування використовуваних ресурсів. При застосуванні компенсуючих транзакцій кожен процес натає також і зворотню операцію, яку механізм хореографії може застосовувати за необхідності (під компенсуючою транзакцією мається на увазі включення в процес відкату операції при її скасуванні самим процесом або користувачем).

Хореографія Web-сервісів має бути не статичною, гнучкою і адаптивною, для того щоб задовільняти сучасним вимогам бізнесу. На впровадження гнучкості впливає чіткий поділ логіки процесу і Web-сервісів, що використовуються. Це як правило реалізується шляхом застосування підходу хореографії. Він скеровує потік задач процесів бізнес-логіки, надаючи можливість використання актуальних Web-сервісів та визначаючи, які наступні задачі слід виконати.

Таким чином, підхід до організації взаємодії Web-сервісів із застосуванням хореографії, значно покращує характеристики системи. Впровадження в системі хореографії Web-сервісів дозволяє робити цю систему адаптивно, гнучкою та динамічною навіть за необхідності реалізації транзакційності процесів бізес-логіки. Тобто дослідження способів впровадження хореографії в системі Web-сервісів є вкрай актуальною проблемою сьогодення. В даній роботі розглядається один з таких способів, заснований на використанні семантичних технологій, що має забезпечити високу ефективність у організації взаємодії Web-сервісів між собою.

Метою даної роботи є дослідження семантичної хореографії REST-сервісів а також способів використання даного підходу в системах, в складі яких значну роль відіграють веб-сервіси.

В роботі буде розглянуто засоби, за допомогою яких семантична хореографія REST-сервісів може бути ефективно реалізована, та досліджено можливість реалізації такого підходу на основі обміну сервісами метаданими про запит через брокер повідомлень.

1 СЕМАНТИЧНИЙ ВЕБ

1.1 Веб-сервіс

В даний час, як в сфері комерційної діяльності, так і в державному управлінні розвивається інфраструктура для здійснення різних електронних операцій на основі веб-сервісів.

Веб-сервіс - це технологія побудови розподілених систем. Вона заснована на відкритих стандартах - XML, SOAP, WSDL і HTTP. Консорціум W3C визначає веб-сервіси, як програмні системи, розроблені для забезпечення стандартних засобів інтероперабельної взаємодії між різними програмними додатками. Веб-служби характеризуються функціональною сумісністю, розширюваністю і можливістю міжкомп'ютерної (machine-to-machine) взаємодії через мережу [16].

Технологія інтеграції веб-сервісів дозволяє легко об'єднувати прості веб-сервіси, впроваджувати і включати їх в необхідні додатки через Інтренет. Можливе об'єднання веб-сервісів здійснюється незалежно від платформи і мови розробки додатків. При виклику веб-сервісів користувач отримує у відповіді ті дані, які має можливість використати у своєму додатку. Веб-сервіс однозначно описується через URI (Uniform Resource Identifier) та надає стандартний інтерфейс. В логіці веб-сервісу взаємодіють три сутності: замовник (service requestor), виконавець (service provider) і реєстр сервісів (service broker). В якості service broker може бути застосований універсальний інтерфейс визначення UDDI (Universal Discovery, Description and Integration), який використовується для публікації і пошуку вже існуючих веб-служб. Веб-сервіси є реалізацією точно визначених інтерфейсів обміну даними між різними інтернет-додатками, які можуть перебувати в різних вузлах мережі і працювати на різних апаратних і програмних платформах, розроблятися на різних мовах.

1.2 База знань

База знань, БЗ — це специфічного виду база даних, що розроблена для керування знаннями (метаданими), а саме збором, сховищем, пошуком і наданням знань. Напрям штучного інтелекту, який досліджує бази даних і методи роботи зі знаннями, носить назву інженерії знань [29].

Відповідно до іншого визначення: база знань — це об’єднання фактів (про певні процеси, явища, події або об'єкти), що мають відношення до конкретної теми або задачі, та побудована так, щоб надавати зручне представлення описаного об’єднання фактів як всього, так і будь-якого його фрагменту [32]. Відповідно до цього система керування базою знань має надавати обробку і представлення моделі, порівняною за рівнем складності з моделлю, відповідно до якої функціонує свідомість людини [6].

Найбільш важлива характеристика БЗ — це якість знань, що зберігаються в ній. Найякісніші БЗ містять релевантну і актуальну інформацію, оснащені прогресивними системами пошуку і детально пропрацьовані формат і структуру збережених знань [37, 34].

1.2.1 Застосування бази знань

Прості бази знань можуть застосовуватися для збереження даних про організації. Основна мета використання таких баз — надати можливість менш досвідченим користувачам знайти вже наявний опис способу розв’язання проблеми певної предметної області [6].

Онтологія може використовуватися для відображення в базі знань ієрархії понять і структурних відношень між цими поняттями. Онтологія, яка додатково зберігає екземпляри об'єктів, які описує в загалом, являється в такому випаду ще і базою знань [32].

База знань — життєво важливий елемент інтелектуальної системи. Найпоширеніший клас подібних додатків — експертні системи. Такі програми використовуються для пошуку способу розв’язання специфічних задач, базуючись на даних БЗ і на описі ситуації з боку користувача [32].

Створення і застосування систем штучного інтелекту вимагає значних за обсягом баз знань.

1.3 Ідея семантичного вебу

Дві основні проблеми сучасного Інтернету - це переповненість інформацією і спосіб подання цієї інформації, який орієнтований в основному на людей, а не на сервіси[29, 33].

Подальший розвиток Інтернету часто пов’язується із такою концепцією, як семантичний веб. Таку концепцію запропонував Тім Бернерс-Лі, автор і засновник принципу www [19].

За задумом Бернерс-Лі головна ідея полягає в реалізації такого надання даних в мережі, щоб допустимою була не тільки їх візуалізація, а такої і ефективна автоматична обробка додатками різних розробників. Семантичний веб - це модифікація сучасної мережі Інтернет, в якій інформація надається в чіткому змістовому значенні, що надає можливість сервісам та користувачам працювати зі значно вищим рівнем розуміння [37]. Передбачається об'єднання різних типів інформації в одну структуру, де кожному змістовому елементу даних має відповідати синтаксичний блок (тег). Теги мають утворювати єдину ієрархічну структуру [37].

1.4 Структура семантичного вебу

Семантичний веб може бути описаний як симбіоз двох основних напрямів.

Перший з них охоплює мови відображення даних. На поточний момент основними з таких мов є XML (eXtensible Markup Language) та RDF (Resource Description Framework). Наявний також і ряд інших, менш популярних мов та форматів представлення даних [19, 29].

Другий концептуальний напрямок включає в себе теоретичне уявлення про доступні моделі предметних областей. Онтологіями такі моделі називаються саме у термінології семантичного вебу [19].

Коли мова йде про логічні рівні семантичного вебу, то той рівень, що є найнижчим - це URI, уніфікований ідентифікатор, що специфікує спосіб запису адреси довільного ресурсу, в тому числі URL, що виступає такою адресою ресурсу в Інтернеті [19].

Наступний рівень представлений XML та засобами опису XML-даних (XML Shema, DTD). В даному випадку є можливість виділити конкретний рівень, що застосовується при роботі з цифровими підписами [19, 24].

На базі XML розгорнуто засоби опису ресурсів у вигляді RDF та RDF-схем, що пояснюють, як поєднуються XML-дані в мережі і надають можливість утворювати словники і каталоги понять. RDF надає можливість виконувати пошук запитуваних понять в семантичному вебі [19].

Більш повну та ретельну автоматичну обробку контенту забезпечує мова мережевих онтологій, що надає додаткову семантичну підтримку поряд із формальною семантикою [32].

1.4.1 Засоби описання ресурсів RDF

RDF зазвичай використовується для описання відношень у мережі між мережевими ресурсами та її інформацією. RDF є елементом між XML-документами і додатками, що виконуються пошук і навігацію на базі логічних стверджень [36]. RDF є засобом представлення змісту понять і термінів у форматі доступному для автоматичного опрацювання. Принцип структуризації відношень між мережевими ресурсами полягає у наявності трьох основних компонентів, а саме - об’єкта, атрибута і значення, що є повним аналогом звичайної схеми, такої як підмет - присудок - означення / доповнення [36, 32].

Кожен учасник такого триплету має наданий йому ідентифікатор URI за використання якого при виборі одного з елементів триплету є можливість отримати ланцюжок повністю. З триплетів RDF можуть складатися мережі на базі зв’язків між об’єктами. Застосування URI ідентифікаторів гарантує унікальність кожного поняття для мережі Інтернет повністю [32].

Основний блок RDF відображають по-різному: A(O,V), де O – об’єкт, A - характеристика із значенням V. RDF надає можливість міняти місцями значення і об’єкт. Початково в RDF застосовувався синтаксис мови XML, проте наявні також і інші мови описання, наприклад, у форматі ряду трійок [32, 29].

Окрім того є допустимою форма відображення у якій довільний вираз RDF у триплеті може бути значенням або ж об’єктом, тобто, надає можливість зазначити, що даний елемент має кокретний тип [32].

<rdf:Description about= www.about...>

<rdf: type rdf: resource=http://description/schema/#book>

</rdf:Description>

RDF специфікація гарантує підтримку тегів, які надають можливість описувати практично довільні поняття. Так наприклад, <order>, <item>, яким відповідають поняття замовлення та продукту відповідно. Інформаційні ресурси описуються за використання сукупності термінів та понять. Семантика термінів та їхніх словників фіксується за використання глобальних імен URI. Окрім того наявна RDF-схема, яка надає можливість визначати, які саме терміни можуть бути використані в RDF твердженнях про характеристики описуваних ресурсів, а також надає ієрархію сутностей предметної області та описує характеристики кожного з використаних термінів [36, 29].

1.4.2 Онтологія

Онтології є більш універсальними і ефективними засобами.

Зазвичай онтологія використовується як ієрархія сутностей об’єднаних відношенням певних особливих видів. Подібні онтології є аналогами семантичних мереж, для яких є можливість задавати структуру у форматі орієнтованих графів, в яких вершини відображають сутності або їх характеристики, а ребра – відношення певних типів. Прикладом таких відношень часто можуть бути «є наслідком» або «належить» [32].

Складніші онтології формалізуються засобами мов логіки і надають можливість логічного виведення. У найбільш простому випадку онтології застосовуються для підвищення ефективності пошуку в мережі Інтернеті [19, 29]. Окрім того, у випадку якщо одні і ті самі поняття інтерпретуються різними термінами, механізм онтологій надає можливість формувати створювати ієрархічні зв'язки між елементами, узагальнювати різні дані, надавати засоби для нечіткого пошуку [19].

Формальна семантика мови OWL, яка рекомендована W3C, описує як отримати логічні виводи на основі онтологій, тобто, отримати факти, які не представлені буквально, а випливають із семантики онтологій. При чому ці виводи можуть будуватися на аналізі як одного документа, так і множити документів, розподілених у мережі [32, 29].

Побудова онтологій на практиці розпочинається з формування ієрархії типів понять, які утворюють предметну область. Вираз типу "rdfs: subClassOf" є фундаментальним конструктором для типів та класів. Він поєднує конкретний тип із загальним типом.

Характеристики можуть бути обмеженими діапазоном та доменами. Так наприклад, характеристика "виготовлено-з-молока" може мати домен "сметана" і діапазон "молоко". Множинні домени означають, що доменом характеристики є область перетину вказаних типів [19, 32].

1.5 Семантичний веб-сервіс

Семантичні веб-сервіси (Semantic Web Services, SWS) застосовують семантичні веб-технології для опису веб-сервісів і є подальшим розвитком веб-сервісів і мови WSDL [30].

Якщо опис ресурсів мережі на базі онтологій, який було визначено в якості ключового технічного будівельного блоку семантичного вебу, можна вважати його статичної частиною, то SWS є динамічною частиною семантичного вебу [19, 32]. Відмінністю семантичних веб-сервісів є їх збагачення семантичної анотацією, що

дає можливість отримувати семантичні веб-служби з машинно-оброблюваної семантикою [30].

Під семантичною анотацією розуміється постачання веб-сервісами додаткового опису їх властивостей, яке визначається як метадані [4]. Метадані - це дані, які містять інформацію про інші дані. Консорціум W3C визначає метадані, як машиночитанну інформацію про веб-ресурси та інші об'єкти [1]. Для анотування семантичних веб-сервісів застосовуються структурні дані, семантика яких заснована на загальних укладеннях, наприклад, стандартний словник Dublin Core [34]. Семантика семантичних метаданих точно і формально задається за допомогою онтологій і тому є автоматично доступною і коректно інтерпретується для комп'ютерів [30].

Під терміном «анотація» мається на увазі прикріплення даних до інших даних і встановлення певної категорії зв'язку між анотованими та анотаційними даними [30].

Під формальною моделлю анотації A розуміють кортеж (a_s, a_p, a_o, a_c), де:

a) a_s - суб'єкт анотації (анотовані дані),

b) a_o - об'єкт анотації (анотаційні дані),

c) a_p - предикат (тип відношення між a_s і a_o ,

d) a_c - контекст, в якому створюється анотація.

Для створення онтологій веб-служб можуть використовуватися [30] такі технології:

- Web Services Description Language (WSDL) [27] - мова опису веб-служб, яка використовується для опису форматів повідомлень і способів обміну простими повідомленнями (операції), що надаються веб-службами;
- Web Service Modeling Ontology (WSMO) [28] - мова моделювання онтологій, яка забезпечує концептуальну основу і формальну мову для семантичного опису всіх відповідних аспектів веб-служб з метою автоматизації пошуку, об'єднання і виклику електронних послуг через глобальну мережу;

- OWL-S: Semantic Markup for Web Services [15] - мова семантичної розмітки веб-сервісів, яка побудована на базі Ontology Web Language (OWL). OWL-S є онтологією в рамках OWL - основи семантичного вебу для опису семантичних веб-сервісів [33]. Цей опис дозволяє користувачам і програмним агентам здійснювати автоматичний пошук, виклик, і моніторинг послуг, пропонованих веб-ресурсами, з урахуванням певних обмежень;

Web Service Modeling Language (WSML) [26] - мова моделювання веб-сервісів, яка забезпечує офіційні синтаксис і семантику для онтологій моделювання веб-служби WSMO.

Опис веб-сервісу за допомогою мови WSDL має наступні три шари [30]:

- абстрактний інтерфейс, який містить одну або кілька операцій - найпростіших одиниць спілкування з веб-службами [24]. Операції включають одне або більше повідомлень в залежності від шаблонів обміну повідомленнями. Формат повідомлень, як правило, задається за допомогою XML-оголошення елементів, найчастіше в XMLSchema [30].
- протокол зв'язування, який визначає, як повідомлення, що містяться в інтерфейсі, можна серіалізувати для обміну протягом певного мережевого протоколу. WSDL надає два зумовлених варіанти прив'язок - SOAP і HTTP [30].
- точка доступу до сервісу - набір адрес, разом з відповідними прив'язками, де веб-служби роблять доступним єдиний інтерфейс [30].

Підхід WSMO, заснований на концепції Web Services Modeling Framework (WSMF) - схемі моделювання веб-сервісів з семантичним описом даних і послуг, при цьому він уточнює і розширює цю основу і розвиває формальну мову онтології [30].

Модель WSMF виділяє чотири основні елемента, необхідні для опису семантичних веб-сервісів [30]:

- онтології, які надають термінологію, яка використовується всіма елементами [27];

- сховища цілей, які визначають завдання, які повинні бути вирішені веб-сервісами [26];
- описи веб-сервісів, які визначають різні аспекти веб-служби: пропоновані послуги, вхідні / вихідні дані, можливі збої, якість і ціну даних, походження даних, достовірність інформації і т. д.
- mediators (посередники), які вирішують проблему сумісності різних словників і взаємодії платформ.

WSML це мова, яка використовується для опису всіх цих елементів. Вона дозволяє описувати бізнес-процеси, об'єкти, цілі та посередників [30].

Мова OWL-S являє собою розширення UDDI можливістю явного опису семантики веб-сервісів, як в статичній їх частині (інформаційні ресурси), так і динамічної (процеси) [34]. Іншими словами, OWL-S - це набір онтологій, що дозволяють описувати об'єкти предметної області та бізнес-процеси [30].

Для створення семантичних анотацій веб-сервісів на OWL-S можна використовувати OWL-S Editor.

У Таблиці 1.1 представлені мови розробки семантичних веб-сервісів, запропонованих консорціумом W3C.

Таблиця 1.1. Мови розробки SWS [32]

Технологія	Опис	Семантична мова	Формальний апарат
OWL-S	OWL онтології високого рівня для веб-сервісів	OWL	дескриптивна логіка
WSDL-S	Використання розширення елементів WSDL, для анотації	Доповнення до мов онтологій (може працювати з OWL, WSMO, UML, XML)	Можливість вибору формального апарату
WSMO	Онтології моделювання Вебсервіс виражається за допомогою WSML.	WSML	Дескриптивні логіки, логіки першого порядку і логіки програмування (F-Logic)

Семантичний веб не замінює можливості веб-сервісів, а доповнює їх: в той час, як завданням більшості сучасних веб-сервісів є забезпечення зв'язку між додатками, семантичний веб вирішує складнішу проблему - підвищити релевантність пошуку інформації в глобальній мережі [34]. Для вирішення цього завдання необхідно побудувати мережу, що складається не з різнорідних, несумісних між собою документів, а з семантично структурованих об'єктів, з описом зв'язків і взаємин між ними [30].

Передбачувана перевага використання семантичних веб-сервісів полягає в можливості їх автоматичного пошуку (а також вибору, заміни, композиції) інтелектуальними програмними агентами відповідних сервісів для вирішення поставлених завдань [30].

1.6 Висновки

Даний розділ роботи було підготовлено в ході виконання переддипломної практики.

Були більш точно сформульовані цілі та задачі дипломної роботи, визначено основний обсяг роботи та послідовність кроків для її виконання.

Проведено огляд предметної області роботи, розглянуто основні поняття та засоби, що будуть використані при виконанні роботи, такі як веб-сервіс та семантичний веб-сервіс, бази знань та їх застосування.

Ознайомлення з означеними поняттями та принципами більш точно окреслює предмет дослідження даної роботи, дозволяє ефективно використовувати ті інструменти, які знадобляться для реалізації підходу хореографії REST-сервісів, що базується на семантичних процесах.

2 ХОРЕОГРАФІЯ ВЕБ-СЕРВІСІВ

2.1 Проблема пошуку веб-сервіса

Створення власного веб-сервісу - достатньо працемісткий процес, який потребує витрат значної кількості ресурсів на проектування, реалізацію, тестування, розміщення та просування власного продукту. Над одним повноцінним веб-сервісом, який має якісний API, ефективні системи логування та виявлення помилок, підтримку тощо команда розробників працюватиме протягом значного часу. Розширення ж такого сервісу потребує ще більших ресурсів.

Зовсім не часто веб-сервіс є поодиноким продуктом, що на вхід отримує лише дані користувача та віддає йому результат, отриманий власними засобами веб-сервісу. Прикладами таких служб можуть бути лише найпростіші продукти, такі як інженерний калькулятор або міні-гра.

Значно частіше веб-сервіси інтегруються з іншими сервісами, утворюють складні системи, що обмінюються даними та надають користувачу результат своєї спільної роботи, що є значно більш повноцінними за той результат роботи, що здатен надати один веб-сервіс окремо.

Побудова такої системи від початку і до кінця є задачею іншого порядку, порівняно із повним циклом розробки окремого сервісу, і для розв’язання цієї задачі має бути залучена значно більша кількість розробників.

Альтернативою повній розробці системи є використання вже існуючих веб-служб як допоміжних для власного веб-сервісу - для реалізації такого рішення достатньо узгодити API сервісів. Задачею що потребує вирішення є пошук відповідного сервісу, що надаватиме необхідні дані.

Вирішення цієї задачі на більш загальному рівні дозволяє створити механізм пошуку веб-сервісу, необхідного даному, характеристики якого можуть бути записані у вигляді набору параметрів даних, з якими працює сервіс; або ж параметрів процесів, що сервіс над даними виконує.

Такий механізм дозволяв би для будь якого сумісного веб-сервісу знаходити необхідний йому для роботи інший сумісний з механізмом веб-сервіс, якщо такий існує.

2.2 Брокер як вирішення проблеми пошуку

Реалізація такого механізму, що дозволяє веб-сервісам обмінюватися даними між собою, носить назву брокера. Брокер отримує запити від одного веб-сервісу, адаптує (за необхідності) та перенаправляє їх на сервіси, що можуть надати необхідні дані; після отримання відповідей агрегує дані та повертає вихідному веб-сервісу у зрозумілому для нього форматі.

Можливою є реалізація брокера, який не забов’язує сервіс надсилати разом із запитом список інших сервісів, до яких запит має бути перенаправлено; замість цього брокер сам виконує пошук веб-сервісів, до яких запит можна перенаправити.

Опис відповідних сервісів може бути отримано брокером із запит. Або ж запит може бути відправлено веб-сервісом одразу у узгодженому з брокером форматі, який описує і характеристики сервісів, до яких запит може бути перенаправлений.

2.2.1 Реалізація брокеру

Описані вимоги до механізму пересилки запитів та агрегування можуть бути реалізовані в окремому сервісі, який і забезпечуватиме транзитивне спілкування інших веб-сервісів системи.

Такий сервіс має бути здатним до одночасного прийому та передачі багатьох запитів, якими обмінюються сервіси. Для реалізації такого брокера доцільним було в використання брокеру повідомлень.

Брокер повідомлень є посередником програмного модуля, який перетворює повідомлення з офіційного протоколу передачі повідомлень від відправника до формального протоколу обміну повідомлень приймача. Брокери повідомлень - це елементи в телекомунікаційних або комп'ютерних мережах, де зв'язуються програмні додатки, обмінюючись офіційно визначеними повідомленнями. Брокери

повідомлень є будівельним блоком інформаційного проміжного програмного забезпечення, є архітектурним шаблоном для підтвердження повідомлень, перетворення та маршрутизації. Він опосередковує комунікацію між додатками, мінімізуючи ймовірність того, що програми повинні мати безпосередній канал зв’язку один з одним, щоб мати можливість обмінюватися повідомленнями.

Мета брокера - приймати вхідні повідомлення з додатків та виконувати певні дії на них. Нижче наведено приклади процесів, які б міг здійснити брокер:

- Маршрутні повідомлення до одного або декількох напрямків
- Перетворення повідомлень в альтернативне представлення
- Виконання агрегації повідомлень, розкладання повідомлень у декілька повідомлень та відправлення їх у пункт призначення
- Взаємодія з зовнішнім сховищем, щоб зберегти повідомлення
- Виклик веб-сервісів для отримання даних
- Відповідь на події чи помилки
- Надання змісту і маршрутизація повідомлень на основі теми за допомогою шаблону "Публікатор-Підписник"

Отже, брокер повідомлень і є основним компонентом брокеру веб-сервісів. Вибір брокера повідомлень при створенні брокера веб-сервісів системи цілком залежить від характеристик цієї системи.

Нижче представлена порівняльна характеристика кількох основних брокерів повідомлень та обґрунтування вибору брокера для даної роботи.

2.2.2 Порівняння брокерів повідомлень

Переглядаючи різні брокери повідомлень, є деякі важливі проблеми:

- Їх поведінка, коли є відставання повідомлень,
- Їх здатність створювати кластер, і
- Їх здатність захищати дані, не блокую видавців, якщо виникає збій вузла в кластері.

2.2.3 Брокер повідомлень: RabbitMQ

RabbitMQ був розроблений і підтримується Pivotal. Офіційний веб-сайт RabbitMQ містить багато корисної інформації та технічної літератури. Erlang, на якій брокер написано, не є для нього обов'язковою мовою програмування, але добре працює з ним.

Має простий процес встановлення. У rabbitmq.config доступні багато регульованих параметрів, сконфігурованих значеннями за замовчуванням. Клієнтський API RabbitMQ підтримує багато мов та стандартні протоколи, такі як STOMP (доступний через плагін). Можна використовувати API-клієнт або веб-інтерфейс для створення тем та черг. Додаткові вузли можна кластеризувати, а черги та теми можна використовувати для інших сервісів. Швидкість публікації близько 20000 / с. Споживає значну кількість додаткової пам'яті, процесорного часу.

2.2.4 Брокер повідомлень: Kafka

Написана на Java, була розроблена компанією LinkedIn, зараз належить Apache. Архітектура брокера визнається дуже високофункціональною. Повідомлення зберігаються у звичайних файлах, і споживачі можуть запитувати повідомлення наборами зі зсувом. Схожий на сервер MySQL (в якості продюсера), що зберігає повідомлення, і споживачі можуть запитувати повідомлення наборами зі зсувом. Сервер досить простий та дуже швидкий. Є можливість зберігати старі повідомлення не більше певного часу.

На сервері використовується Zookeeper, що забезпечує належність в кластері та маршрутизацію; ту саму програму можуть використовувати для синхронізації і споживачі.

Вбудований веб фронт-енд відсутній, деякі з них доступні через екосистему. Маршрутизація відсутня, правила не доступні, а статистика - звичайний JMX. З іншого боку, продуктивність досягла швидкості публікації 165000 повідомлень / с. в одному потоці. Продуктивність порядку 3000000 повідомлень / с. Невисокий рівень використання процесорного часу та пам'яті.

Потребує кількох секунд для операції resync репліки після її відключення та додавання великої кількості повідомлень.

2.2.5 Брокер повідомлень: ActiveMQ

Ще один популярний брокер повідомлень. Має деякі потужні можливості. Хоча і написаний на Java (як Kafka), проте більш порівняна зі стандартом, встановленим RabbitMQ. Серверний накопичувач забезпечує рівень High Availability, реплікація підтримується на рівні бази даних.

Під використанням мережі брокерів розуміється підключення до одного з них, де повідомлення публікується або споживається. Невідомо, де знаходиться черга, а саме який вузол відомий брокеру, який ви використовуєте, з яким він з'єднується і якому надсилає ваш запит. Швидкість вставки порядку 5000 повідомлень / с.

2.2.6 Брокер повідомлень: Kestrel

Більш схожий на Kafka. Написано на мові Scala; спілкується протоколом memcached. Дозволяє вказати обмеження на зберігання в черзі, термін дії та поведінку при досягненні лімітів. Так наприклад для виконання вимоги ніколи не блокувати продюсерів, може бути використано параметр "discardOldWhenFull = true".

Kestrel дещо обмежений його здатністю до кластеризації. Проте, він може публікувати доступність для Zookeeper, а видавці та споживачі будуть повідомлятися про відсутність сервера. Процес стає складнішим, якщо існує багато серверів Kestrel, і всі вони мають одну і ту ж саму чергу. Публікація порядку 10000 повідомлень / с.

2.3 Використання хореографії сервісів у патерні проектування Сага

Транзакції є важливою частиною програм. Без них неможливо зберегти узгодженість даних. Один з найбільш потужних типів транзакцій називається двоетапним комітом, суть якого в тому, що коміт першої транзакції залежить від

завершення другої. Це корисно, особливо якщо доводиться одночасно оновлювати декілька об'єктів, наприклад підтвердження замовлення та оновлення доступного ліміту одночасно [17, 18].

Однак, при роботі, наприклад, з мікросервісами, стан речей значно ускладнюється. Кожна служба є окремою системою, що має свою власну базу даних, і вже відсутня можливість використовувати простоту місцевих двофазних комітів для забезпечення узгодженості всієї системи.

За втрати цієї властивості, RDBMS стає досить поганим вибором для збереження, оскільки можна виконати ту саму «одиночну атомну транзакцію», але в десятки разів швидше, просто використовуючи базу даних NoSQL, таку як Couchbase. Саме тому більшість компаній, що працюють з мікросервісами, також використовують NoSQL.

Щоб продемонструвати цю проблему, розглянемо наступну архітектуру мікросервісів на високому рівні системи електронної комерції (Рисунок 2.1).

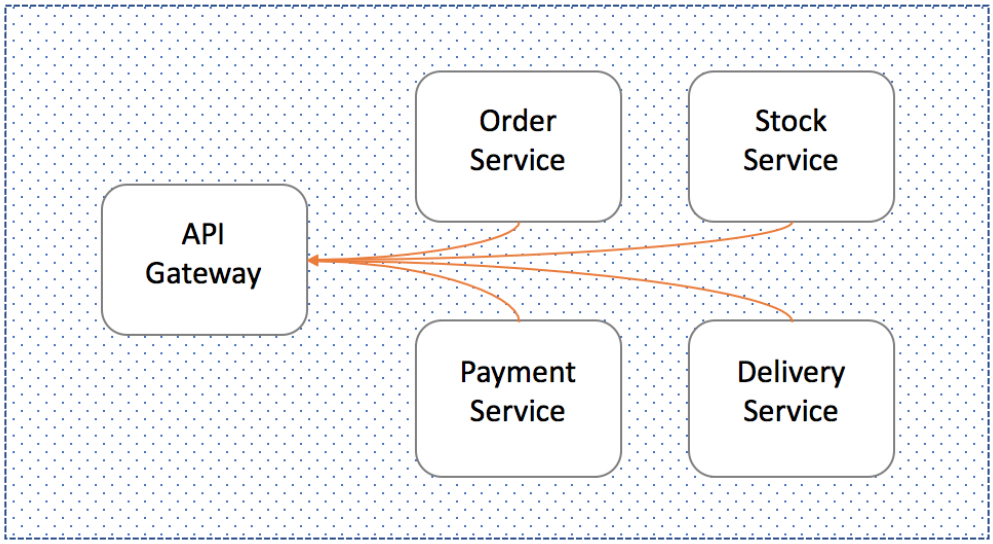

Рисунок 2.1 - Архітектура мікросервісів електронної комерції [18]

У наведеному вище прикладі не можна просто розмістити замовлення, здійснити його оплату замовником, оновлювати запас і відправити замовлення у службу доставки - все в єдиній транзакції ACID. Щоб послідовно виконати цю всю послідовність, потрібно буде створити розподілену транзакцію.

Досить розповсюдженим є досвід того, як важко реалізувати щось розподілене, і, на жаль, транзакції не є винятком. Робота з тимчасовими станами, не

гарантована консистентність між службами, ізоляція та відкати - це сценарії, які всі слід розглянути на етапі проектування.

На щастя, було розроблено кілька ефективних моделей, на основі 20-річного досвіду здійснення розподілених транзакцій їхніми авторами. Те, про що піде мова в даній статті, називається шаблоном проектування Сага.

2.3.1 Шаблон SAGA

Одна з найбільш відомих моделей для розподілених транзакцій називається Сага. Перший документ про неї був опублікований ще в 1987 році і це зробило патерн досить популярним рішенням на той час.

Сага - це послідовність локальних транзакцій, де кожна з них оновлює дані в межах однієї служби. Перша транзакція ініціюється зовнішнім запитом, який відповідає певній події в системі, після чого кожен наступний етап ініціюється завершенням попереднього.

Існує кілька різних способів реалізації транзакції із сагами, але два найбільш популярні:

Події / Хореографія: коли немає центральної координації, кожна служба публікує та слухає події іншої служби та вирішує, чи потрібно застосовувати власну відповідну дію чи ні.

Команда / Оркестровка: коли сервіс-координатор відповідає за централізацію прийняття рішень саги, а також за послідовність бізнес-логіки.

Слід трохи заглибитися у кожну з реалізацій, щоб зрозуміти, як саме вони працюють.

2.3.2 Події / Хореографія

У підході "Події / Хореографія" перший сервіс виконує транзакцію, після чого публікує свою подію. Ця подія слухається одним чи кількома сервісами, які виконують власні локальні транзакції та публікують (або ж ні) власні нові події.

Розподілена транзакція закінчується, коли останній сервіс виконує свою локальну транзакцію і не публікує жодних подій, або ж подія, що була опублікована, не буде слухатися жодним із учасників саги.

Order Service зберігає нове замовлення, встановлює його стан в очікування та публікує подію під назвою ORDER_CREATED_EVENT.

Payment Service слухає ORDER_CREATED_EVENT, сплачує замовлення (локальна транзакція) та публікує власну подію BILLED_ORDER_EVENT.

Stock Service слухає BILLED_ORDER_EVENT, оновлює доступний ліміт, готує продукти, куплені в замовленні, та публікує ORDER_PREPARED_EVENT.

Delivery Service прослуховує ORDER_PREPARED_EVENT, після чого забирає та доставляє товар, так як знає, що він був підготовлений у Stock Service. Зрештою, він публікує ORDER_DELIVERED_EVENT

Нарешті, Order Service прослуховує ORDER_DELIVERED_EVENT і встановлює стан замовлення як завершене.

У вищезгаданому випадку, якщо стан замовлення треба відстежувати, Order Service може просто слухати всі події та оновлювати стан.

2.3.3 Відкат у розподілених транзакціях

Відкат розподіленої транзакції не є аж занадто простим. Зазвичай необхідно ввести ще одну операцію / транзакцію, щоб компенсувати те, що було зроблено раніше.

Припустимо, що Stock Service не виконав свою локальну транзакцію. В такому разі відкат виглядатиме так:

- Stock Service публікує PRODUCT_OUT_OF_STOCK_EVENT;
- Order Service та Payment Service слухають дану подію;
- Payment Service повертає кошти клієнта;
- Order Service встановлює стан замовлення як невдалий

Слід звернути увагу на те, що важливо визначити зовнішній ідентифікатор для кожної транзакції, тому щоразу, коли публікується подія, всі слухачі можуть одразу дізнатися, якої транзакції це стосується.

2.3.4 Переваги та недоліки дизайну Подій / Хореографії Saga

Події / Хореографія є природним способом реалізації моделі Саги; вона проста для зрозуміння, не вимагає багато зусиль для побудови, а всі учасники є слабо пов'язаними (loosely coupled), оскільки не мають безпосереднього знання один одного. Якщо певна транзакція включає в себе 2-4 кроки, застосування цього патерну може бути доволі доцільним.

Однак цей підхід може швидко стати досить заплутаним, якщо продовжувати включати додаткові кроки до транзакції, оскільки в такому випадку стає важко відстежити, які саме сервіси прослуховують певні події. Окрім того, це також може додати і циклічну залежність між сервісами, оскільки вони в такому разі мають підписатися на події один одного.

І, врешті, тестування було б доволі складним, так як передбачало б імітування поведінки транзакції, в якій повинні працювати всі сервіси.

2.4 Висновки

В даному розділі було більш детально розглянуто конкретні засоби, які мають стати основою реалізації взаємодії між набором сервісів.

Одним з таких засобів є брокер повідомлень. Зважаючи на особливості практичної частини роботи, Kafka найкраще відповідає її вимогам. З її переваг - існує гарантія того, що сервіс доступний завдяки неблокуючим операціям. Повідомлення легко реплікуються для забезпечення більш високої доступності даних. Продуктивність Kafka висока, а використання ресурсів досить економне.

Було розглянуто шаблон проектування Saga, який дозволяє за використання хореографії ефективно організувати взаємодію між сервісами системи та забезпечити її консистентний стан, не зважаючи на помилки при обробці запитів.

Ця особливість є дуже важливою для системи, в якій мають бути відсутні чіткі залежності між сервісами. Такі залежності мають утворюватися динамічно відповідно до результату обміну метаданими через брокер. Обробка помилок в патерні Saga і дозволяє використовувати брокер без ризику встановлення в системі неконсистентного стану через помилку в комунікації сервісів.

3 РЕАЛІЗАЦІЯ СИСТЕМИ СЕРВІСІВ З ВИКОРИСТАННЯМ СЕМАНТИЧНОЇ ХОРЕОГРАФІЇ

3.1 Формулювання вимог до продукту

Метою створення даного продукту є дослідження ефективності використання хореографії REST-сервісів за використання засобів семантичного аналізу.

Розроблений програмний засіб повинен моделювати розподілену систему, що складається з незалежних сервісів. В системі наявні сервіси, подібні за призначенням, проте різні за наборами вхідних параметрів або різними за форматом відповіді за запит. Ці особливості API сервісів мають бути описані у вигляді онтологій.

Використовуючи можливість обирати сервіс для надсилання запиту за допомогою цих онтологій, сервіси мають взаємодіяти таким чином, щоб надавати користувачеві повноцінну відповідь на його запит.

У тому випадку, коли набір сервісів не є повним для того, щоб успішно обробити запит користувача, користувач має бути коректно повідомлений, що його запит було виконано частково, або ж взагалі не виконано.

3.2 Засоби реалізації принципів семантичної хореографії сервісів

3.2.1 Мова програмування: Scala

Scala - мультипарадигмова мова програмування, що була спроектована короткою і типобезпечною для простого і швидкого створення компонентного програмного забезпечення, що поєднує можливості функціонального та об'єктно-орієнтованого програмування.

Перші версії мови були створені в 2003 році колективом лабораторій програмного забезпечення Федеральної політехнічної школи Лозанни під керівництвом Мартіна Одерски; мову було реалізовано для платформ Java та .Net.

Ключові аспекти мови

Scala-програми багато в чому схожі на Java-програми, і можуть вільно взаємодіяти з Java-кодом. Мова включає єдинообразну об'єктну модель - в тому сенсі, що будь-яке значення є об'єктом, а будь-яка операція - викликом методу. При цьому це також функціональний мова в тому сенсі, що функції - це повноправні значення.

У Scala включені потужні та єдині концепції абстракцій як для типів, так і для значень. Зокрема, мова містить гнучкі симетричні конструкції додатків для композиції класів і типів. Дозволяє здійснювати декомпозицію об'єктів шляхом порівняння зі зразком; зразки та вирази при цьому були успільнені для підтримки природної обробки XML-документів. В цілому ці конструкції дозволяють легко виражати самостійні компоненти, що використовують бібліотеки Scala, не використовуючи спеціальні мовні конструкції.

Мова допускає зовнішні розширення компонентів з використанням представлень (views). Можливості загального програмного забезпечення реалізуються за рахунок підтримки загальних функцій (generics), у тому числі вищого типу (generics of a higher kind). Крім різних класичних структурних типів даних, до мови включено підтримка екзистенціальних типів.

Об'єктно-орієнтована мова

В мові використовується чиста об'єктно-орієнтована модель, подібна до застосованої в Smalltalk: кожне значення - це об'єкт, і кожна операція - це відправлення повідомлень. Наприклад, додавання x + y інтерпретується як x. + (y), тобто як виклик методу + з аргументом y і x у якості об'єкта-приємника.

Роздивимось інший приклад: 1 + 2. Цей вираз інтерпретується як (1).+ (2). Зверніть увагу, що дужки навколо чисел обов'язкові, тому що лексичний аналізатор Scala розбиває вираз на лексеми за принципом найдовшого можливого зіставлення. Таким чином, вираз 1. + (2) розбивається на лексеми 1, + і 2, тому що лексема 1. довша за лексему 1 і перший аргумент додавання буде інтерпретовано, як тип Double замість Int.

Функціональна мова

Кожна функція - це значення. Мова надає легкий синтаксис для визначення анонімних і карійованих функцій. Кожна конструкція повертає значення. Співставлення зі зразком природно застосовуване до обробки XML за допомогою регулярних виразів.

Повторне використання та адаптація

Кожна компонентна система з потужними конструкціями абстракції і композиції стикається з проблемою, коли справа доходить до інтеграції підсистем, розроблених різними командами в різний час. Проблема полягає в тому, що інтерфейс компонентів, розроблений тією чи іншою групою, часто не підходить клієнтам, що мають намір використати цей компонент.

Scala представляє нову концепцію вирішення проблеми зовнішньої розширюваності - представлення (views). Вони дозволяють розширювати клас новими членами та типами. Представления в Scala в певному сенсі відповідають класам типів, що використовуються в Haskell, але на відміну від класів типів, область видимості представлень можна контролювати, причому в різних частинах програми можуть співіснувати паралельні представлення.

Визначні характеристики / переваги мови:

- Доступ до значної кількості існуючих Java-бібліотек;
- Велика кількість бібліотек, написаних на Scala і для Scala;
- Порівняно висока швидкість (в 2-3 менша за C [26, 25]);
- Велика розвинена екосистема;
- Легкість створення DSL (Domain Specific language);
- Розвинена система типів;
- Використовується в ряді потужних інтернет-компаній (Twitter, LinkedIn, Foursquare);
- Найбільш поширена в промисловості функціональна мова програмування;

- Навний ряд розвинених web-фреймворків, наприклад, Play, Scalatra і ряд інших;
- Наявний ряд розвинених фреймворків для використання з Hadoop, наприклад, Scalding;
- Наявна реалізація системи акторів за принципом Ерланга (Akka);
- Розвинена підтримка графічного інтерфейсу (Swing, JavaFX);
- Потужна система збирання - SBT;
- Наявна бібліотека Scalaz, яка дозволяє писати на Scala як на повністю функціональній мові програмування.

Мінусами мови можна назвати:

- Складність, порівняно високий поріг входження;
- Перевантаженість мови.

3.2.2 Контейнер класів: SpringBoot

Spring Framework (або коротко Spring) - універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Також існує форк для платформи .NET Framework, названий Spring.NET.

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги «Expert One-on-One Java EE Design and Development» (Wrox Press, жовтень 2002 року).

Фреймворк був вперше випущений під ліцензією Apache 2.0 license в червні 2003 року. Перший стабільний реліз 1.0 був випущений в березні 2004. Spring 2.0 був випущений в жовтні 2006, Spring 2.5 - в листопаді 2007, Spring 3.0 в грудні 2009, і Spring 3.1 в грудні 2011. Поточна версія - 5.0.1.

Не зважаючи на те, що Spring не забезпечує певну конкретну модель розробки, він став доволі поширеним в Java-співтоваристві в основному як аналог і заміна моделі Enterprise JavaBeans. Його перевага полягає в тому, що Spring надає достатню свободу Java-розробникам в розробці архітектури; крім того, він постачає добре документовані і прості у використанні засоби розв’язання проблем, що зазвичай виникають при розробці програм корпоративного масштабу.

В той самий час, особливості Spring Core можуть бути використані в будь-якому Java-додатку, а для розробки веб-додатків на платформі Java Enterprise наявні безліч удосконалень і розширень. З описаних причин Spring набув високої популярності і розглядається розробниками як стратегічно важливий фреймворк.

Spring забезпечує вирішення численних завдань, з якими зустрічаються Java-розробники і організації, що намагаються створити інформаційну систему, що базується на Java платформі. Завдяки доволі широкій функціональності достатньо важко зазначити найбільш значущі структурні частини, з яких фреймворк складається. Spring не цілкм зав'язаний а Java Enterprise платформу, не зважаючи на його глибоку інтеграцію з нею, що є не останньою причиною популярності фреймворку.

Spring багатьом відомий саме як джерело розширень (features), необхідних для ефективної розробки великих бізнес-систем не використовуючи великовагові шаблонні моделі, що історично були переважними в промисловості. Його ще одна перевага полягає в тому, що Spring ввів перед цим невикористовувані функціональні засоби в сьогоднішні головні методи розробки, в тому числі поза Java платформи.

Даний фреймворк надає послідовну модель і забезпечує її придатність для переважної більшості видів додатків, що були вже розроблені на базі Java платформи. Spring пропонує модель розробки, що базується на найвищих стандартах індустрії, і постачає її доступною у багатьох областях використання Java.

Основним елементом Spring фреймворку є контейнер Inversion of Control, який постачає засоби конфігурації і керування об'єктами Java за використання рефлексії. Контейнер відповідає за керування життєвим циклом об'єкта: конструювання об'єктів, виклик функцій їх ініціалізації і конфігурації шляхом слабкого зв'язування їх з іншими об’єктами.

Об'єкти, що конструюються контейнером, також носять назву керованих об'єктів або бінів (beans). Як правило конфігурація такого контейнера виконуєтсья шляхом затягування XML-файлів, що описують визначення бінів та надають необхідну для конструювання бінів інформацію.

Такі об'єкти можуть бути отримані одним з двох основних способів:

- Пошук залежності - шаблон проектування, в якому викликаний об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям або певного типу.
- Впровадження залежності - патерн проектування, в якому контейнер надає екземпляри певних об'єктів по їх імені або ідентифікатору іншим керованим об'єктам за використання їх конструктора, поля або фабричного методу.

Spring Boot - це рішення Spring-Con-over-configuration для створення автономних, продуктивних Spring-додатків, які можна "просто запустити". Він попередньо налаштований з "поважним поглядом" Spring на найкращу конфігурацію та використання платформи Spring та сторонніх бібліотек, щоб надавати можливість почати з мінімальної невпорядкованості проекту. Більшість додатків Spring Boot потребують дуже малої конфігурації Spring. Особливості:

- Створення окремих програм Spring
- Підключення Tomcat або Jetty безпосередньо (не потрібно розгортати файли WAR)
- Надання орієнтованих "початкових" Project Object Models (POMs) для спрощення конфігурації Maven
- Автоматичне налаштування Spring, коли це можливо
- Надання таких готових функцій, як метрики, health-тести та зовнішня конфігурація
- Абсолютно відсутні генерація коду і вимоги до обов’язкового конфігурування через XML.

3.2.3 Брокер повідомлень: Akka Actors

Модель акторів

У комп'ютерних науках модель акторів є математичною моделлю паралельних обчислень, яка трактує поняття «актор» як універсальний примітив паралельного чисельного розрахунку: у відповідь на одержувані повідомлення актор може приймати локальні рішення, створювати нові актори, посилати свої повідомлення, а

також встановлювати, як слід реагувати на наступні повідомлення. Модель акторів виникла в 1973 році, використовувалася як основа для розуміння обчислення процесів і як теоретична база для ряду практичних реалізацій паралельних систем.

Akka

Akka - це безкоштовний інструментарій із відкритим вихідним кодом, який спрощує побудову одночасних та розподілених програм на JVM. Акка підтримує кілька моделей програмування для паралелізму, але з акцентом на паралелізм, заснований на системі акторів, і в якому наявний вплив мови Erlang.

Прив'язки до мови існують як для Java, так і для Scala. Акка написана на Scala, і, як відповідно до Scala 2.10, реалізація Akka Actors включена як частина стандартної бібліотеки Scala.

Особливості

Ключовими відмінностями програм, що використовують Akka Actors, є:

- Паралельність на основі повідомлень і асинхронності: зазвичай ніякі мутабельні дані не відриваються для доступу і ніякі примітиви синхронізації не використовуються; Akka реалізує модель актора.
- Актори взаємодіють однаково, не залежно від того, чи вони на тому ж хості або окремі ж на окремому хості, спілкуються безпосередньо або через об'єкти маршрутизації, працюють в кількох або багатьох потоках тощо. Ці дані можуть бути змінені під час розгортання через механізм конфігурації, що дозволяє маштабувати (для використання більш потужних серверів) та випускати (для використання інших серверів) програму без модифікації.
- Актори впорядковуються ієрархічно відносно до падінь програми, які розглядаються як події, що підлягають обробці супервізором актора (незалежно від того, який актор відправив повідомлення, що викликало падіння). На відміну від Erlang, Akka застосовує батьківський супервізор, а це означає, що кожен актор створюється та контролюється його батьківським актором.

Akka має модульну структуру, з основним модулем, що забезпечує модель акторів. Інші модулі доступні для додавання таких функцій, як мережевий розподіл акторів, підтримка кластерів, Command та Event Sourcing, інтеграція з різними сторонними системами (наприклад, Apache Camel, ZeroMQ) та навіть підтримка інших моделей паралелізму, таких як Futures та Agents.

3.2.4 Сховище даних: Apache Jena

Apache Jena - це open source Semantic Web фреймворк для Java. Вона надає API для вибору даних та написання графів RDF. Графи представлені у вигляді абстрактної "моделі". Модель може бути отримана з файлів даних, баз даних, URL-адрес або їх комбінацій. Модель також може запитуватися через SPARQL 1.1.

Jena схожа на Sesame; однак, на відміну від Sesame, Jena надає підтримку OWL (Web Ontology Language). Всередині має ряд внутрішніх різонерів, і для роботи в Jena може також бути налаштований Pellet різонер (з відкритим вихідним кодом Java OWL-DL).

Jena підтримує серіалізацію графів RDF у:

- реляційнау базу даних
- RDF/XML
- Turtle
- Notation 3

Fuseki це HTTP-інтерфейс для даних RDF. Він підтримує SPARQL для запитів та запису. Проект є підпроектом Jena і розроблений як сервлет. Fuseki також може працювати на автономному сервері, оскільки він постачається з налаштованим веб-сервером Jetty.

3.3 Архітектурні рішення при реалізації принципів семантичної хореографії сервісів

3.3.1 Використані при реалізації патерни

Фасад

Структурний шаблон проектування, що надає можливість приховати високу складність системи методом зведення всіх можливих викликів ззовні до єдиного об'єкту, делегує їх відповідними об'єктами системи.

Використовується щоб забезпечити уніфікований інтерфейс з набором розрізнених імплементацій або інтерфейсів, або з підсистемою у тому випадку якщо слід уникнути небажано високого рівня зв'язування з цією підсистемою або ж реалізація такої підсистеми в майбутньому може змінитися.

В якості рішення можливо визначити єдину точку взаємодії з такою підсистемою - а саме фасадний об'єкт, що як раз забезпечує загальний інтерфейс інтеграції з такою підсистемою, і покласти на цю точку обов'язок по взаємодії з компонентами всередині. Фасад - це зовнішній об'єкт, що надає єдину точку входу для різних служб підсистеми. Імплементація інших компонентів такої підсистеми закрита та її не видно іншим, зовнішнім компонентам. Такий фасадний об'єкт гарантує реалізацію GRASP шаблону Protected Variations з точки зору захисту від можливих майбутніх змін в реалізації такої підсистеми.

Command

Ціль використання даного патерну - створення структури, в якій клас-відправник і клас-отримувач не залежать один від одного. Це організація зворотного виклику до класу, який включає в себе клас-відправник.

В ООП шаблон проектування Команда є поведінковим шаблоном, в якому об'єкт використовується для інкапсуляції всієї інформації, необхідної для виконання дії або виклику події в більш пізній час. Ця інформація включає в себе ім'я методу, об'єкт, який є власником методу і значення параметрів методу.

Чотири терміна завжди пов'язані з шаблоном Команда: команди (command), приймач команд (receiver), що викликає команди (invoker) і клієнт (client). Об'єкт Command знає про приймачі та викликає метод приймача. Значення параметрів приймача зберігаються в команді. Довільний об'єкт (invoker) знає, як виконати команду і, можливо, робить облік і запис виконаних команд. Довільний об'єкт (invoker) нічого не знає про конкретну команді, він знає тільки про інтерфейс. Обидва об'єкти (що викликає об'єкт і кілька об'єктів команд) належать об'єкту клієнта (client). Клієнт вирішує, які команди виконати і коли. Щоб виконати команду він передає об'єкт команди викликає об'єкту (invoker).

Використання командних об'єктів спрощує побудову загальних компонентів, які необхідно делегувати або виконувати виклики методів в будь-який час без необхідності знати методи класу або параметри методу. Використання виклику об'єкта (invoker) дозволяє ввести облік виконаних команд без необхідності знати клієнтові про цю модель обліку (такий облік може стати в нагоді, наприклад, для реалізації відміни і затримки команд).

Медіатор

Використовується у випадку, коли необхідно забезпечити взаємодію багатьох об'єктів, сформувавши при цьому слабку зв'язність і позбавивши самі об'єкти від необхідності явно посилатися на друге.

Медіатор визначає інтерфейс для обміну інформацією з об'єктами, Конкретний Медіатор координує дії об'єктів. Кожен клас знає про свій об'єкт Медіатор, всі об’єкти обмінюються інформацією лише з медіатором, при його відсутності їм слід обмінюватися інформацією безпосередньо. Об’єкти відправляють запити медіатору і отримують запити від нього. Медіатор реалізує кооперативну поведінку, пересилаючи кожен запит одному або декільком об’єктам.

Сага

При застосуванні патерну проектуваня Database per service кожнен сервіс має власну базу даних. Деякі бізнес-транзакції, однак, охоплюють кілька сервісів, тому і потрібний механізм забезпечення узгодженості даних у різних сервісах. Прикладом

може бути магазин електронної комерції, де клієнти мають ліміт кредиту. Додаток має гарантувати, що нове замовлення не перевищить кредитний ліміт замовника. Оскільки замовлення та клієнти знаходяться в різних базах даних, програма не може просто використовувати локальну транзакцію ACID.

Для того, щоб зберегти узгодженість даних у різних сервісах, є можливість реалізувати кожну бізнес-транзакцію, яка охоплює кілька сервісів як сагу. Сага - це послідовність місцевих транзакцій. Кожна локальна транзакція оновлює базу даних та публікує повідомлення або подія, щоб викликати наступну локальну транзакцію в сазі. Якщо локальна транзакція не вдається, оскільки вона порушує ділове правило, сага виконує ряд компенсаційних транзакцій, які скасовують зміни, внесені попередніми локальними транзакціями.

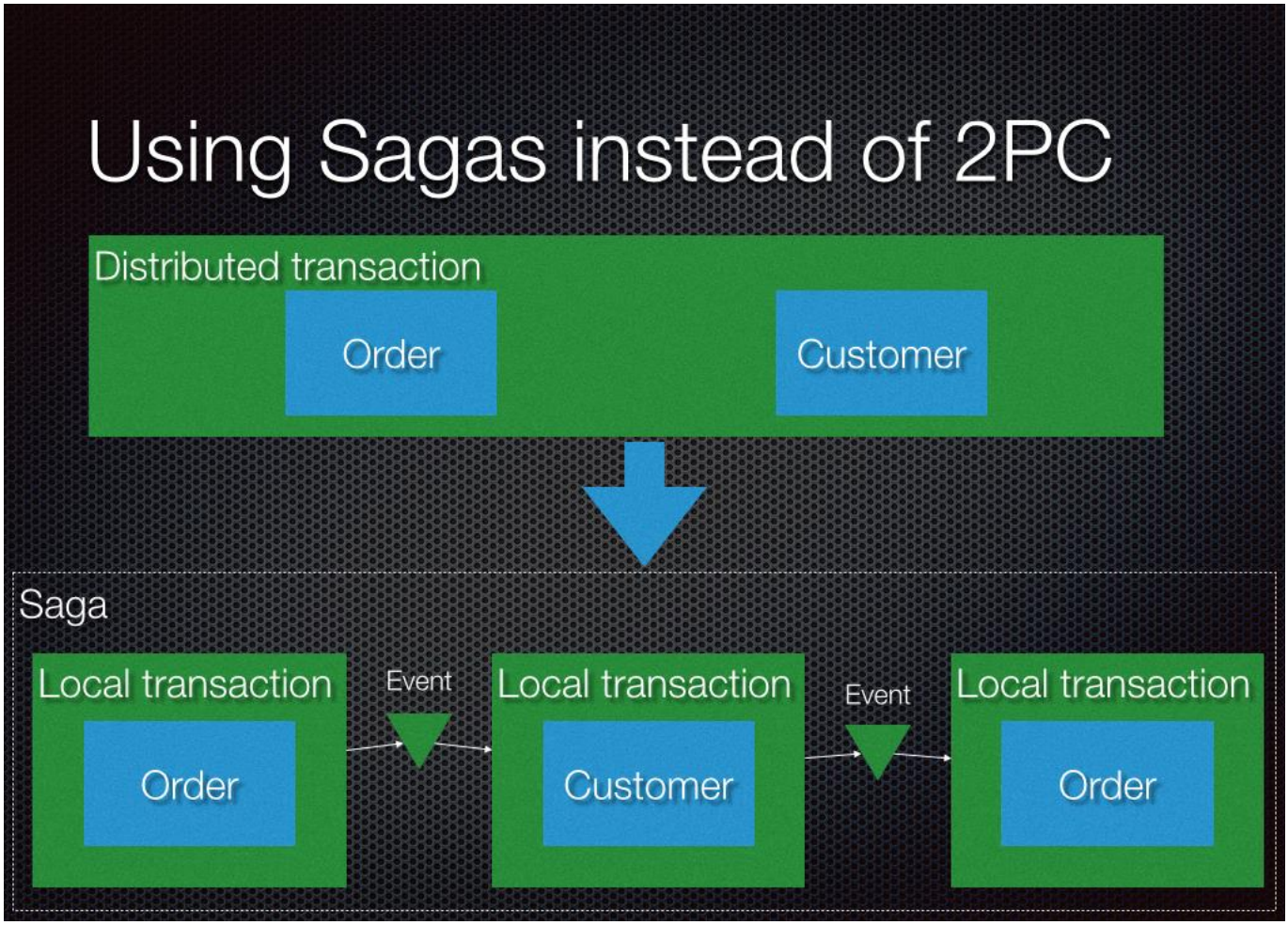

Рисунок 3.1 - Використання патерну Сага замість 2PC [17]

3.3.2 Обмін повідомленнями між сервісами

Обмін даними між сервісами відбувається через HTTP-протокол. Кожен сервіс має власний ендпоінт, або кілька ендпоінтів. Адреси ендпоінтів не є відомою всім іншим сервісам - вони їх отримують в залежності від змісту їхніх запитів.

Сервіс безпосередньо не зберігає ніяких даних про попередні запити до нього, що є однією з вимог, яким має задовольняти REST сервіс. Також сервіс не зберігає і ніяких даних про те, в яких саме сервісів можна отримати інформацію, що необхідна для того, щоб відповісти на поточний запит.

Якщо при отриманні запиту через REST ендпоінт сервіс потребує тих даних, якими не володіє, він має зробити запит до іншого сервісу, що такі наді може надавати. Проте поточний сервіс не знає жодного іншого сервісу. Їх взагалі може не бути в певний момент часу, а згодом до системи вже може бути підключено кілька таких сервісів, що підійшли б поточному.

Для отримання необхідних даних сервіс перш за все має визначити, в кого саме такі дані можуть бути отримані. Для цього він відправляє запит до брокеру повідомлень надати йому ендпоінти тих сервісів, що необхідні дані йому можуть надати.

В системі відсутнє централізоване сховище ендпоінтів та опису до них - такий підхід означав би побудову системи, заснованої на оркеструванні сервісів. На відміну від нього, підхід що використовується тут більш характерний для хореографії.

Цей запит розсилається брокером до кожного з сервісів системи. Вже на цьому етапі можливі дві основні реалізації даного процесу.

Поточний сервіс може надати брокеру назву тематики, до якої належать необхідні сервісу дані. В цьому випадку брокер зробить розсилку запиту ендпоінтів використовуючи окремий канал, за яким закріплено зазначену тематику. Лише сервіси, що прослуховують даний канал аналізуватимуть запит ендпоінтів та нададуть через брокер відповідь, чи здатні вони надати шукані дані.

Поточний сервіс може не надавати брокеру назву тематики, до якої належать необхідні сервісу дані. В такому випадку брокер робитиме розсилку запиту

ендпоінтів не використовуючи канали. Усі сервіси системи отримають запит ендпоінтів і надаватимуть відповідь, чи здатні вони надати шукані дані.

Після отримання ендпоінту сервісу, що має можливість надати поточному сервісу необхідні дані, він на цей ендпоінт надсилає запит, надаючи необхідні для отримання даних параметри.

Сервіси системи повинні мати можливість обробити запит, що може не містити всіх необхідних для надання необхідних даних параметрів. В такому випадку вони або відповідають тими даними, які вдалося отримати, використовуючи лише надані параметри, або ж одразу відповісти сервісу, що надіслав їм запит, що дані, якими сервіс оперує, не можуть бути надані в якості відповіді на такий запит.

Сервіси також повинні мати властивість ігнорувати параметри, які не можуть використати для отримання даних. Такі параметри не мають описувати обов’язкову вимогу до необхідних даних, так як в такому випадку сервіс не мав би надавати через брокер свою адресу (так як цей обов’язковий параметр він обробляти не може і на запит відповісти не має можливості).

Для прийняття рішення, чи відповідає даний сервіс вимогам, отриманим через брокер, використовуються онтології. Вимоги передаються у вигляді триплетів. Використовуючи їх, а також власні метадані, що описують сервіс та дані, які він надає, сервіс робить висновок, чи здатен він буде відповісти на наступний запит через ендпоінт. У тому випадку, якщо сервіс здатен це зробити, він надає адресу ендпоінту, на який такий запит може бути направлено.

3.4 Реалізація семантично-керованої системи сервісів

3.4.1 Модульна структура системи сервісів

Окремі сервіси додатку було винесено в окремі модулі, які спілкуються між собою за принципом, описаним в розділі Хореографія сервісів - REST HTTP та брокер повідомлень.

Основним модулем є semantic-choreography, який містить в собі усі інші модулі. Всі вони наслідують semantic-choreography-parent, в якому підключені глобальні залежності додатку, такі як Spring Boot та Akka.

Також до глобального parent-модуля підключено модуль semantic-choreography-common, який не містить логіки, а лише допомагає більш ефективно працювати з об'єктами. Типовим представником цього модуля є об'єкт FutureTools:

```
package utils

import scala.concurrent.duration._
import scala.concurrent.{Await, Future}
import scala.language.postfixOps
import scala.util.Try

object FutureTools {

  implicit class FutureTools[T](future: Future[T]) {

    def awaitFor(duration: Int): Option[T] = Try {
      Await.result(future, duration seconds)
    }.toOption

    def await: Option[T] = awaitFor(2)
  }
}
```

Модулем, що приймає запит від користувача є semantic-choreography-web.

Модулі сервісів, для яких було застосовано патерн Saga та хореографія зібрано в окремому модулі semantic-choreography-modules. Окрім модулів сервісів він також містить semantic-choreography-modules-api - API для сервісів, що будуть підключені до процесу хореографії за допомогою брокера повідомлень; semantic-choreography-modules-parent, від якого модулі сервісів наслідуються.

Модуль semantic-choreography-modules містить наступний список модулів сервісів:

- semantic-choreography-module-bank
- semantic-choreography-module-delivery
- semantic-choreography-module-diagnostics-A
- semantic-choreography-module-diagnostics-B
- semantic-choreography-module-insurance
- semantic-choreography-module-pharmacy-A
- semantic-choreography-module-pharmacy-B

Кожен з них представляє собою незалежний сервіс, що має власний HTTP REST API. В даному списку наявні модулі, що мають подібний функціонал, проте різний API.

Так наприклад semantic-choreography-module-pharmacy-A та semantic-choreography-module-pharmacy-B використовуються сервісом діагностики для підбору лікарського засобу в залежності від діагнозу, проте вони мають різний набір лікарських засобів та вхідні параметри їх ендпоінтів відрізняються.

Така структура модулів дозволяє певному сервісу не залежати від конкретного іншого модуля, а динамічно визначати найбільш доречний сервіс чи сервіси з поміж доступних.

Структура та зв’язок описаних модулів додатку представлені на Рисунку 3.2 та Рисунку 3.3.

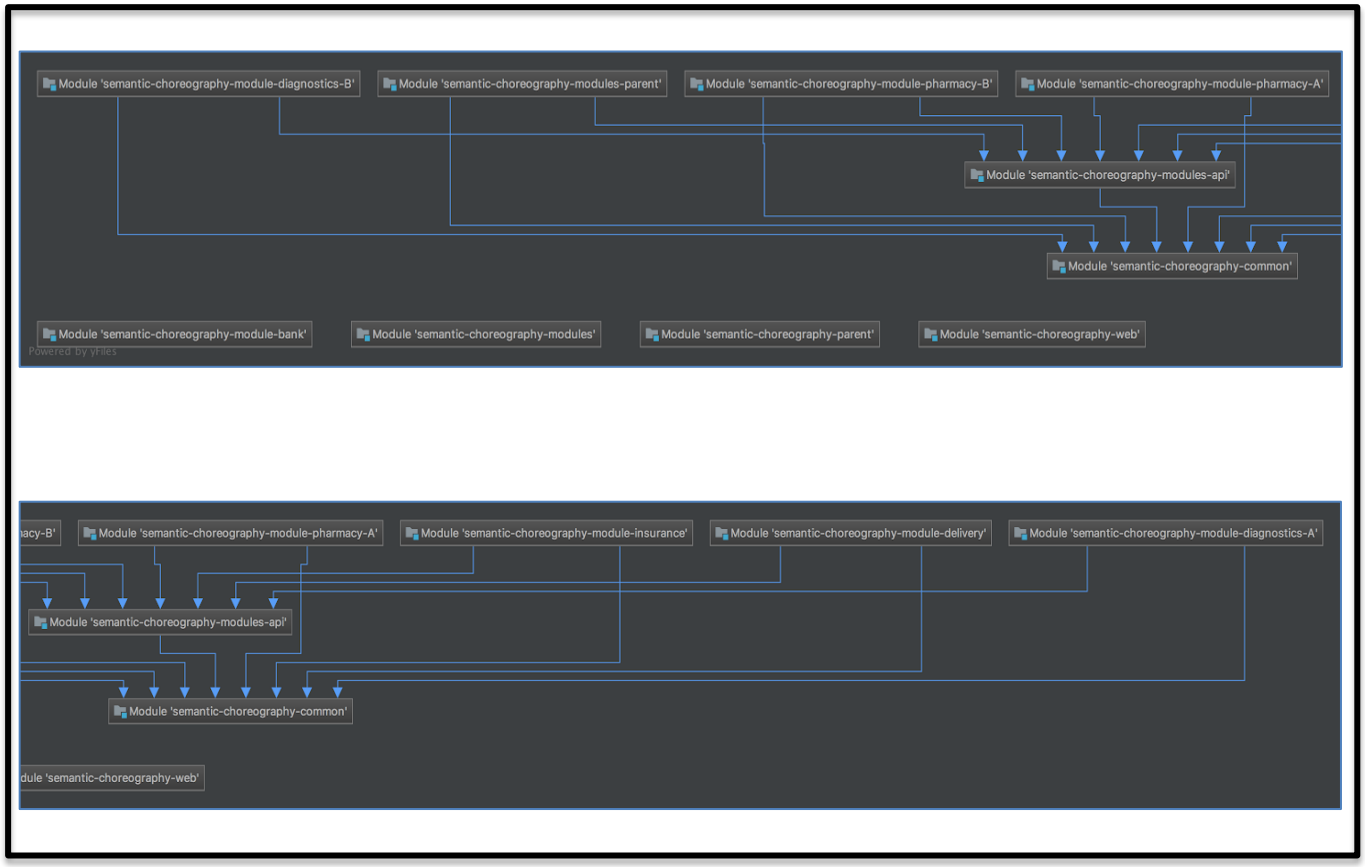

Рисунок 3.2 - Діаграма модулів додатку [Розробка автора]

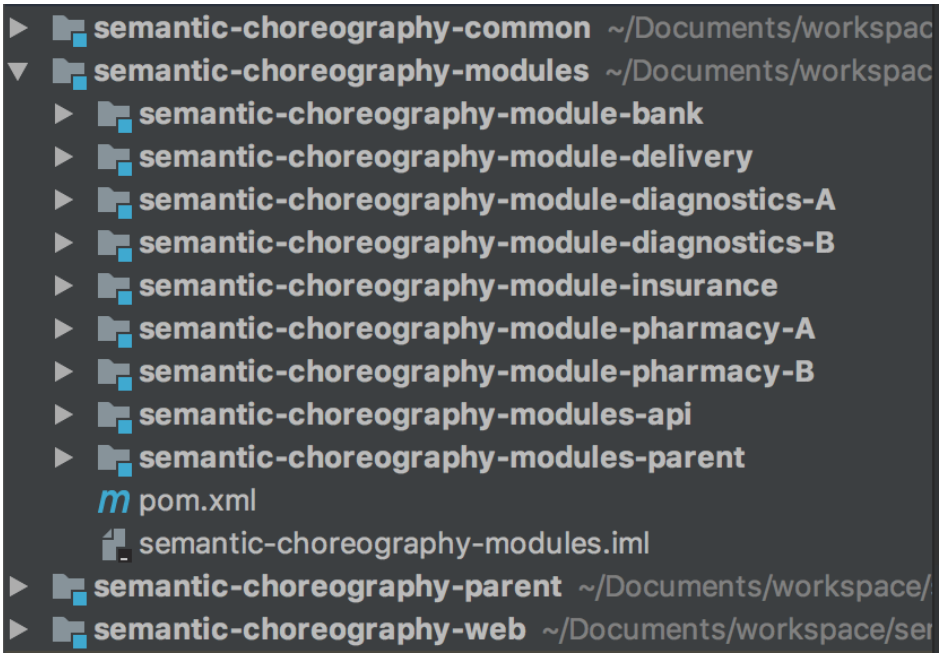

Рисунок 3.3 – Ієрархічна структура модулів додатку [Розробка автора]

Далі на прикладі Модуля діагностики А представлена структура окремого незалежного модуля сервісу.

3.4.2 Опис модулю semantic-choreography-module-diagnostics-A

HTTP REST інтерфейс, через який із сервісом взаємодіють інші сервіси, надається класом DiagnosticsController, який зареєстрований як REST-контроллер SpringBoot.

```
package web

@RestController
@RequestMapping(path = Array("/api"))
class DiagnosticsController(diagnosticsService: DiagnosticsService) {

  @PostMapping(path = Array("/diagnose"))
  def diagnose(@RequestBody symptoms: Seq[String]): Conclusion = {
    val diagnoses = diagnosticsService.diagnosis(symptoms)
    diagnosticsService.conclusion(diagnoses)
  }
}
```

Контролер віддає у response результат аналізу вхідних параметрів, що виконується у основному класі модуля DiagnosticsService:

```
package service

import domain.{Conclusion, Diagnosis}

class DiagnosticsService(pharmacyClient: PharmacyClient)
  extends SourceRequestResolver(source = "ontology/diagnostics-service") {
  val validProbability = 0.8d

  def diagnosis(symptoms: Seq[String]): Seq[Diagnosis]

  def conclusion(diagnoses: Seq[Diagnosis]): Conclusion = {
    val diagnosis = diagnoses.filter(_.probability >
validProbability).maxBy(_.probability)

    Conclusion(diagnosis.name, diagnosis.probability, diagnosis.treatment)
  }

  implicit class `Diagnosis → Treatment`(diagnosis: Diagnosis) {

    def treatment: Option[String] = {
      val pharmaceuticals = treatments(diagnosis.name)

      pharmacyClient.pharmaceutical(pharmaceuticals)
    }

    private def treatments(diagnosis: String): Seq[String]

  }

}
```

Сервіс реалізує інтерфейс SourceRequestResolver, описаний в наступному пункті semantic-choreography-modules-api. Інтерфейс дозволяє іншим сервісам отримувати дані про DiagnosticsService, передаючи йому метадані свого запиту.

Таким чином, DiagnosticsController і DiagnosticsService отримуватимуть лише ті запити, що відповідають функціоналу, реалізованому в модулі. Інші ж модулі не

мають залежностей на даний, через що його може бути у будь-який час відключено або підключено. Даний підхід є реалізацією патерну проектування Фасад.

Параметр source = "ontology/diagnostics-service" позначає де знаходиться онтологія з метаданими про DiagnosticsService (semantic-choreography-module-diagnostics-A).

Методи diagnosis() та treatments() в залежності від їх реалізації надають конкретні медицинські дані про можливі діагноз та лікування пацієнта.

```
case class Diagnosis(name: String, probability: Double, details: Option[String])
```

Метод conclusion() використовується для отримання висновку Conclusion, що буде передано в DiagnosticsController і відправлено в якості відповіді на запит.

```
case class Conclusion(name: String, probability: Double, treatment: Option[String] =
None, note: Option[String] = None)
```

Об’єкт Conclusion містить в тому числі і конкретне (можливе) лікування, що було визначено з поміж можливих варіантів в залежності від логіки DiagnosticsService (діагностичного модуля) та PharmacyService (фармацевтичного модуля).

Звернення до фармацевтичного сервісу відбувається за допомогою впровадженої в DiagnosticsService залежності PharmacyClient.

```
package service

import domain.MetaRequest

class PharmacyClient extends HttpOntologyClient[Seq[String], String] {

  protected def metaRequest: MetaRequest

  protected def endpoint: Option[String]

  def pharmaceutical(pharmaceuticals: Seq[String]): Option[String] =
    send(pharmaceuticals)
}
```

PharmacyClient не є реалізацією фармацевтичного сервісу та не залежить на жодну з конкретних реалазацій. Даний клас реалізує інтерфейс

HttpOntologyClient[Request, Response], що знаходиться в semantic-choreography-modules-api, та параметризує його конкретними типами запиту та відповіді на запит, що відповідають форматам даних, які є характерими саме для цього діагностичного сервісу.

Зазначені формати також описані в метаданих semantic-choreography-module-diagnostics-A та використовуються в PharmacyClient через метод metaRequest.

3.4.3 Опис модулю semantic-choreography-modules-api

Є модулем додатку, що реалізує взаємодію між іншими незалежними модулями за принципом їх хореографії. В модулі використовуються залежності брокеру повідомлень, в даному випадку Akka Actors. Це дозволяє іншим модулям наслідувати інтерфейси даного модулю і не зав’язуватися за реалізацію взаємодії з іншими модулями (не тільки на їх конкретну реалізацію). Тобто при створенні нового модуля сервісу немає необхідності реалізувати взаємодію і для нього - достатньо наслідуватися від інтерфейсів semantic-choreography-modules-api.

Брокер повідомлень або принцип роботи з онтологіями можуть бути змінені в модулі semantic-choreography-modules-api, при цьому всі інші модулі сервісів не відчують даних змін, а продовжуватимуть працювати.

Інтерфейс SourceRequestResolver, який наслідують сервіси з модулів сервісів додатку, використовується для визначення, чи є реалізація сервісу актуальною для запиту, метадані якого передаються в SourceRequestResolver.

```
package service

import akka.actor.Actor
import akka.cluster.pubsub.DistributedPubSub
import akka.cluster.pubsub.DistributedPubSubMediator.Subscribe
import domain.{MetaRequest, Triple}

class SourceRequestResolver(source: String) extends RequestResolver with Actor {
  private val mediator = DistributedPubSub(context.system).mediator
  mediator ! Subscribe("content", self)
```

```
  override def receive: Receive = {
    case request: MetaRequest ⇒ resolveEndpoint(request.requestData)
    case _ ⇒
  }

  protected def resolveEndpoint(requestData: Seq[Triple]): Option[String]
}
```

Параметр source: String конкретизує розташування онтології з метаданими конкретного сервісу. Вона буде використана тоді, коли будуть отримані метадані запиту-кандидата до конкретного сервісу.

Метадані про запит передаються у вигляді триплетів.

```
package domain

case class Triple(tSubject: String, tPredicate: String, tObject: String)

object Triple {
  type MetaRequest = Seq[Triple]
}
```

Метадані запиту отримуються з брокера повідомлень, що розсилає їх всім сервісам-реалізаціям SourceRequestResolver. Для цього використовується патерн медіатор, описаний в підрозділі Використані патерни. Даний патерн реалізується засобами Akka Actors та Akka Cluster. Метод receive(), що є Partial Function, в SourceRequestResolver (який він має, так як реалізує інтерфейс Actor) реагує тільки на ті повідомлення брокера, які є метаданими запитів MetaRequest. Інші повідомлення ігноруються. В даному інтерфейсі реалізовано патерн проектування Command.

До такого конкретного сервісу, реалізації SourceRequestResolver, інші сервіси отримують доступ за допомогою інтерфейсу HttpOntologyClient[Request, Response].

```
package service

trait HttpOntologyClient[Request, Response] extends HttpClient[Request, Response]
with Requester {
  protected def send(request: Request): Option[Response] =
    endpoint.map(e ⇒ send(request, e))

  protected def endpoint: Option[String]
}
```

Так, наприклад, діагностичний сервіс звертається до фармацевтичного сервісу через фармацевтичний клієнт, що є частиною діагностичного сервісу. Клієнт є реалізацією HttpOntologyClient, параметризованою типами, що належать діагностичному сервісу і що відповідають форматам даних, що діагностичний сервіс може надавати, та які йому треба отримати. Таким чином, клієнт ніяк не зав'язаний на реалізацію конкретного фармацевтичного сервісу, а обирає актуальні варіанти з поміж сервісів.

HttpOntologyClient є реалізацією інтерфейсу HttpClient, що параметризується тими самими типами запиту та відповіді на запит. Інтерфейс HttpClient використовується лише для відправки запиту та отримання відповіді у відповідних форматах.

HttpOntologyClient також реалізує інтерфейс Requester, який визначає параметр endpoint, необхідний для HttpClient.

```
package service

import scala.concurrent.Future
import scala.concurrent.duration.FiniteDuration

trait Requester {
  val requestActor = new RequestActor

  protected def metaRequest: MetaRequest

  def endpoints: Seq[String] =
    requestActor.resolveEndpoint(metaRequest).await.toSeq.flatten
}
```

```
object Requester {

  class RequestActor extends Actor {
    private implicit val timeout = Timeout(FiniteDuration(50, TimeUnit.MILLISECONDS))
    private val mediator = DistributedPubSub(context.system).mediator

    def resolveEndpoint(request: MetaRequest): Future[Option[String]] = {
      mediator ? request mapTo manifest[Seq[String]]
    }

    override def receive = {  case _ ⇒
    }
  }
}
```

Requester використовує патерн mediator, реалізаваний засобами Akka Actors та Akka Cluster, для того щоб розіслати метадані запиту всім реалізаціям SourceRequestResolver. Ті з них, що відповідають формату запиту та відповіді на нього, надають свої ендпоінти, які і використовує Requester та його реалізація HttpOntologyClient.

Спрощена UML діаграма класів модуля semantic-choreography-module-diagnostics-A та інтерфейсів модуля semantic-choreography-modules-api, які вони реалізують, представлена на Рисунку 3.4.

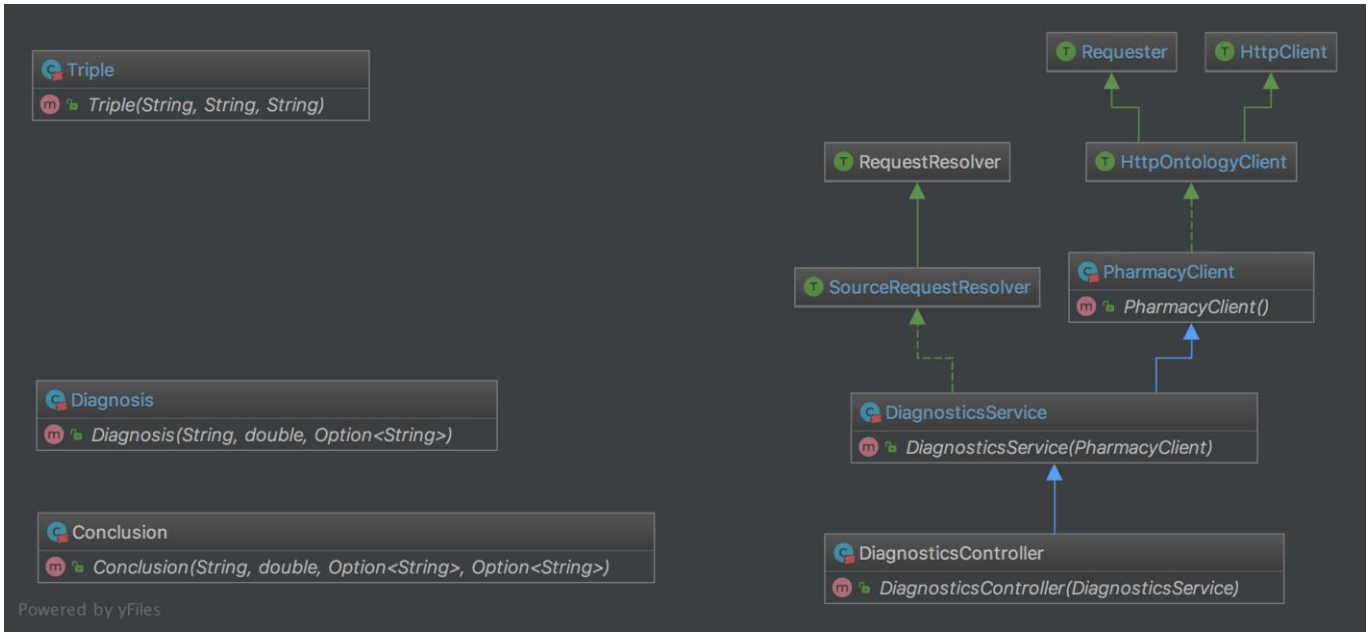

Рисунок 3.4 - Спрощена UML діаграма класів одного сервісного модуля [Розробка автора]

3.5 Результати обробки запиту семантично-керованою системою сервісів

3.5.1 Тестування розробленого продукту

Для тестування додатку було розроблено кілька тестових сценаріїв, що покривають різні можливі випадки поведінки сервісів в системі.

Для сервісів діагностування було надано значно спрощену реалізацію постановки діагнозу та визначення набору можливого лікування.

Тестові сценарії передбачають перевірку комбінацій обставин випадків, що характеризуються такими параметрами системи, як:

- CO (completeness) - повнота набору сервісів, необхідного для повної обробки запиту;
- AC (API consistency) - відповідність опису API сервісів за допомогою онтологій реальному API сервісів;
- IP (Input parameters) - кількість вхідних параметрів сервісів та деталізація їх опису в онтології;
- AI (Alternative implementations) кількість альтернативних реалізацій сервіса;
- AD (API difference) - різниця в API альтернативних сервісів;
- AS (additional services) - наявність сервісів, що не є необхідними для обробки запиту;

Для кожної з розглянутих комбінацій наведених характеристик було складено кілька варіантів онтологій, що задовольняють описаній системі.

Результати виконання описаних тестових сценаріїв наведено у Таблиці 3.1.

З отриманих результатів можливим є висновок щодо впливу окремих параметрів на результат обробки запиту.

Таблиця 3.1 - Результати виконання тестових сценаріїв [Розробка автора]

Сценарій	CO	AC	IP	AI	AD	AS	Результат
1	+	+	середня	2	середня	відсутні	0.8
2	+	-	середня	2	середня	відсутні	0.2
3	-	+	середня	2	середня	відсутні	0.0
4	-	-	середня	2	середня	відсутні	0.0
5	+	+	низька	2	середня	відсутні	0.7
6	+	+	висока	2	середня	відсутні	0.9
7	+	+	середня	відсутні	середня	відсутні	0.8
8	+	+	середня	3	середня	відсутні	0.8
9	+	+	середня	4	середня	відсутні	0.7
10	+	+	середня	2	низька	відсутні	0.6
11	+	+	середня	2	висока	відсутні	1.0
12	+	+	середня	2	середня	1	0.8
13	+	+	середня	2	середня	2	0.8
14	+	+	середня	2 (AS)	середня	2	0.7
15	+	+	низька	2	низька	відсутні	0.4
16	+	+	висока	2	висока	відсутні	1.0
17	+	+	низька	2	висока	відсутні	0.8
18	+	+	висока	2	низька	відсутні	0.7
19	+	+	середня	3	низька	відсутні	0.6

За відсутності одного з видів сервісів запит не може бути виконаним.

Консистентність між API сервісу та його описом в онтології має високий вплив на те, якому з сервісів буде відправлено запит. Її відсутність спричиняє множинні помилки в обробці сервісом запиту, формат якого не відповідає тому, що сервіс здатен обробити.

Більша кількість вхідних параметрів покращує здатність сервісів правильно розпізнавати запити, що мають бути оброблені саме цими сервісами. Аналогічна залежність спостерігається і для того, наскільки відрізняються API різних сервісів. Причиною цього має бути більша кількість характерних особливостей запиту, що заважає йому бути ідентифікованим псевдопозитивно сервісом, що не має його обробляти.

Кількість альтернативних реалізацій сервісів зі схожим функціоналом та кількість сервісів, що не мають брати учать в обробці запиту користувача не чинять суттєвого впливу на хід обробки цього запиту. При зростанні кількості будь-яких сервісів зростає ймовірність того, що будь-який з сервісів зробить псевдопозитивний висновок щодо запиту.

3.5.2 Переваги розробленого рішення

Представлене рішення дозволяє створювати систему сервісів, які не мають прямих залежностей один від одного та не мають зберігати список альтернативних сервісів на той випадок, якщо основний сервіс, з яким до цього працював поточний, раптово виходить з ладу. Бродкастинг метаданих запиту на всі сервіси гарантує те, що всі альтернативи сервісу, що вийшов з ладу, зможуть взяти участь в обробці саме цього запиту.

Інша перевага такого підходу полягає в тому, що новий сервіс може бути підключено без будь-яких перешкод зі сторони робочої версії додатку, помилок в ній або ж додаткових модифікацій в самому новому сервісі - в ньому достатньо лише реалізувати API сервісного модуля і описати формат вхідних та вихідних даних сервісу в онтології. Інші сервіси зможуть одразу розпочати використовувати новий.

Так само один з сервісів може бути вилучено ,якщо для нього існують аналоги - відповідні запити одразу почнуть оброблятися ними. Таке рішення може знадобитися в тому випадку, якщо, наприклад, в одному з сервісів було знайдено помилку, що має негативні side-ефекти. Такий сервіс може бути просто призупинено.

3.5.3 Засоби покращення розробленого рішення

Деталізація метаданих в онтології

Однією з проблем даного рішення, виявлених під час тестування, було псевдопозитивне розпізнавання запиту сервісом, що насправді не має його

обробляти. Причина цього полягає в тому, що відповідно до правил, записаних в онтологіях, тип запиту не завжди може бути однозначно ідентифікований.

Це є проблемою лише даної реалізації. Структура онтології повністю залежатиме від того, які саме параметри сервіси приймають на вхід. Зазвичай в реальних системах структури даних значно більші і різноманітні, що спричинить виконання більш деталізованого опису сервісу в онтології і зменшить кількість псевдопозитивних результатів.

Динамічне конструювання онтологій

Недоліком даного підходу є те, що API сервісу доводиться описувати принаймні двічі - при його розробці та при наданні опису сервісу в онтології. Окрім того, що це збільшує час та ресурси, необхідні для побудови окремого сервісу, є ймовірність того, що буде порушено консистентність між реальним API та його описом в онтології, а це, як було виявлено під час тестування, вкрай негативно вливає на результат роботи всіх сервісів.

Рішенням даної проблеми може стати використання автоматичної генерації опису API в онтології. На кшалт того, як документація до API генерується за допомогою Swagger, є можливою і автоматична генерація опису у вигляді онтологій структури форматів, з якими цей API працює. Можливим є і зворотній процес, але зазвичай автоматична генерація коду дає не найкращі результати, а сам підхід може перешкоджати технікам розробки програмного забезпечення (таким як TDD).

Заміна брокера повідомлень

В даній реалізації в якості брокера повідомлень були використані такі бібліотеки як Akka Actors та Akka Cluster. Вони є дуже простми у використанні та налаштуванні, але при зростанні самої системи та її розподіленості, більш доречним стає використання такого брокеру повідомлень як Kafka, переваги якого були вже описані в розділі Хореографія Веб-сервісів.

Використання тематик повідомлень

При зростанні кількості сервісів зросте також і кількість одночасних перевірок кожним з сервісів, чи є запит, що може бути сервісу надісланий, тим запитом, який сервіс може обробити.

Для того, щоб не перевіряти кожен з запитів, є можливість впровадити тематичну розсилку метаданих запиту. Підхід було описано в пункті Хореографія сервісів розділу Реалізація, проте не було застосовано в поточні версії додатку.

Тематика запиту може автоматично визначатися сервісом-відправником за метаданими запиту, або ж може бути просто задано константно саме для цього запиту (проте, хардкод тематики змушує розробника кожен раз перевіряти її при зміні формату даних, якими оперує сервіс). Такий підхід робить сервіси менш гнучкими, так як певним чином обмежує список доступних сервісів, проте він в той самий час дає більші гарантії того, що запит не буде оброблено псевдопозитивно, що також вирішує одну з проблем даного підходу.

3.6 Висновки

В ході роботи над даним розділом було створено реалізацію системи сервісів, що взаємодіють між собою по HTTP-протоколу згідно REST. Між сервісами відсутні прямі залежності - не встановлено, до яких сервісів поточний має звертатися для отримання даних, необхідних для подальшої роботи. Для визначення цього використовується розсилання метаданих наступного запиту всім сервісам через брокер повідомлень; відповідні сервіси аналізують метадані запиту і порівнюють із власними метаданими, а в разі відповідності надають ендпоінт для надсилання цього запиту.

Відсутність прямих з’вязків між сервісами дозволяє додавати та видаляти окремі модулі системи без надмірних зусиль, змін у вже наявних модулях чи певних підготовчих етапів, що впливають на поведінку поточної версії продукту.

В реалізації використано патерн проектування Сага, що забезпечує транзакціність обробки запиту та дозволяє обробляти помилки, пов’язані в тому числі і з встановленням правильного порядку взаємодії сервісів.

4 СТАРТАП-ПРОЕКТ

4.1 Ідея проекту

Розділ містить економічне обґрунтування стартап-проекту “Розподілена хореографічна система медичної допомоги”. Розділ має на меті ознайомлення з економічними та функціональними характеристиками майбутнього проекту, економічними аспектами його реалізації та впровадження у використання.

Опис основної ідеї майбутнього стартап-проекту наведено в Таблиці 4.1.

Таблиця 4.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Адаптивна система, що аналізує симптоми пацієнта та рекомендує і допомагає впроваджувати лікування	Підвищення швидкості та якості обслуговування пацієнтів лікарень	Відсутність черг, необхідності взаємодії зі страховою та фармацевтичною службами; Підвищення точності діагнозу, його віддалена постановка
	Полегшення роботи лікарів	Полегшення взаємодії з пацієнтом

Таблиця містить опис ідеї стартап-проекту “Розподілена хореографічна система медичної допомоги”. Таблиця має на меті ознайомлення з економічними та функціональними вигодами майбутнього проекту.

Для оцінювання конкурентноспроможності та можливості і складності виходу стартапу на ринок було виконано порівняння із рядом потенційних конкурентів, до яких можуть бути застосовані обрана для порівняння характеристики (Наявність адміністратора для налаштування, Наявність Інтернету, Кросплатформність тощо). Результати порівняння наведені в Таблиці 4.2.

В таблиці 4.2 наведено порівняння із рядом потенційних конкурентів, що надає можливість оцінки конкурентоспроможності та можливості і складності виходу стартапу на ринок.

Таблиця 4.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характе-риситики ідеї	(Потенційні) товари/концепції конкурентів					W слабка сторона	N нейтра-льна сторона	S сильна сторона
		Мій проект	К-т 1	К-т 2	К-т 3	К-т 4			
1.	Форма виконання	Система сервісів	Додаток	Веб-сервіс	Веб-сервіс	Веб-сервіс		+	
2.	Собівар-тість	Низька	Висока	Середня	Низька	Середня			+
3.	Наявність адміністратора для налаштування	Не треба	Не треба	Треба	Треба	Треба		+	
4.	Наявність Інтернету	Треба	Не треба	Треба	Треба	Не треба	+		
5.	Кросплатформність	Так	Так	Ні	Так	Ні			+
6.	Масштабованість	Так	Ні	Ні	Ні	Ні			+

Проект має сильні сторони, що відсутні в існуючих аналогах та здатний із ними конкурувати. Сильними сторонами є відсутність необхідності в адміністраторі для налаштування, низька собівартість реалізації та кросплатфоменість, що дозволяє мати швидкий старт. Така перевага як масштабованість є відсутньою в потенційних конкурентів, тому є основною перевагою проекту і дозволяє розширяти його новими сервісами без надмірних зусиль.

Слабкою стороною є необхідність використання Інтернету, проте умова відсутності доступу до Інтернету в потенційному середовищі застосування проекту є малоймовірною.

4.2 Технологічний аудит ідеї проекту

Проводиться аудит технологій, за допомогою якої може бути реалізована ідея проекту (технології створення товару).

Таблиця 4.3 - Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технології
1.	Створення системи сервісів	Java, SpringBoot, Kafka	Наявна	Безкоштовна, доступна
		Scala, SpringBoot, Akka Actors	Наявна	Безкоштовна, доступна
		Python, AWS	Наявна	Частково платна, доступна
Для створення сервісу обрані технології (Scala, SpringBoot, Akka Actors), які є безкоштовними, доступними та добре дослідженими потенційними розробниками.				

У таблиці Таблиця 4.3 надано результати огляду основних стеків технологій, що можуть бути використані з метою реалізації системи стартап-проекту, описаного вище. Було обрано стек технологій, що не потребує допрацювань та додаткових витрат.

4.3 Аналіз ринкових можливостей

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Результати аналізу представлені у Таблиці 4.4.

У Таблиці 4.4 надано результати аналізу характеристик попиту на ринку. На ринку присутні чотири основних представника, а загальний обсяг продаж є значним. Проте зростання динаміки ринку та відсутність суттєвих обмежень для входу на ринок дозволяють це виконати стартап-проекту з наведеними у попередніх підрозділах характеристикими.

Таблиця 4.4 - Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од	4
2.	Загальний обсяг продаж, грн/ум.од	13500 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Конфіденційність оброблюваних даних
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Дії, необхідні для виходу на такий ринок, залежать в тому числі і від потенційних споживачів. Аналіз цільових аудиторій споживачів даного продукту наведено у Таблиці 4.5.

Таблиця 4.5 - Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Система сервісів, що спрощує процес отримання медичних послуг	Клініки, які обслуговують населення, що має медичне страхування	Очікується низька зацікавленість в стартапі державних клінік	Надійність та конфіденційність процесу

Відповідно до результатів аналізу цільових аудиторій споживачів описаного продукту, наведеного у Таблиці 4.5 та необхідного для виходу на такий ринок, який було описано в Таблиці 4.4, слід спрямовувати зусилля на активне просування проекту в приватних клініках.

Важливим процесом для виходу на ринок та орієнтації на певну цільову аудиторію споживачів є аналіз можливих загроз стартап-проекту, що можуть спричинити значні проблеми для його розвитку. Результати відповідного аналізу факторів загроз продукту наведено в Таблиці 4.6.

Таблиця 4.6 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Динаміка ринку	Уповільнення росту ринку	Співпраця з іншими компаніями для поліпшення ситуації на ринку
			Розширення на суміжні ринки
2.	Конкуренція	Вихід на ринок великої компанії	Вихід з ринку
			Запропонувати великій компанії поглинути себе
			Надати додаткові переваги власного продукту лише за появи сильного конкурента
3.	Конкуренція	Поява іноземних конкурентів з товарами низької вартості	Акцентування уваги на якості результату, що надає продукт (або ж на будь якій іншій явній перевазі відповідно до Таблиці 4.2)
4.	Потреби користувачів	Користувачам необхідний сервіс з іншим функціоналом	Надання нового функціоналу вже існуючій системі (завдяки її адаптивності)
5.	Держава	Зростання податкового тягаря	Перегляд виконання умов, що зменшують податки
			Поступове підвищення тарифів

Відповідно до результатів аналізу можливих факторів загроз стартап-проекту описаного продукту, наведеного у Таблиці 4.6 та необхідного для виходу на ринок, описаний в Таблиці 4.4, існує ряд ризиків які слід враховувати при планування виходу продукту на ринок та мати орієнтовні сценарії їх мінімізації та компенсування їх впливу, наведені в таблиці вище.

Аналогічно до загроз стартап-проекту, що можуть спричинити значні проблеми для його розвитку, важливою частиною є огляд можливих сприятливих умов, використання яких може значно покращити становище спартап-проекту та надати перевагу порівняно із конкурентами. Такі сприятливі умови та відповідні можливості розглянуто в Таблиці 4.7.

Таблиця 4.7 - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Можливості користувачів	Зростання можливостей потенційних покупців	Таргетингова реклама
2.	Конкуренція	Зниження довіри до конкурента 1 внаслідок помилок в функціонуванні	Акцентувати увагу на надійності системи
3.	Конкуренція	Зменшення числа конкурентів за рахунок появи бар'єрів входу на ринок	Заохочення співробітників конкурентів до зміни компанії
4.	Технології	Поява нових технологій	Аналіз технології, включення нових модулів (що технологію використовують) до адаптивної системи
5.	Держава	Послаблення обмежень в законодавстві	Оптимізація діяльності для скорочення витрат

Відповідно до результатів аналізу можливих сприятливих умов для описаного продукту, наведеного у Таблиці 4.7 та вкрай бажаного для виходу на ринок, описаний в Таблиці 4.4, існує ряд можливих реакцій компанії, правильне застосування яких може надати значну перевагу порівняно із конкурентами

Конкуренція на ринку може стати як причиною занепаду компанії, так і стимулом завдяки якому стартап-проект значно покращить якість послуг та отримає корисний для майбутнього розвитку досвід. У зв’язку з цим в Таблиці 4.8 проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку стартап-проекту.

Відповідно до результатів аналізу пропозиції описаного продукту, наведеного у Таблиці 4.8 та необхідного для виходу на ринок, описаний в Таблиці 4.4, визначено загальні риси конкуренції на ринку стартап-проекту та зазначено можливий вплив на діяльність підприємства - можливі дії та заходи компанії, спрямовані на підвищення її конкурентоспроможності.

Таблиця 4.8 - Ступеневий аналіз конкуренції на ринку

Характеристика конкурентного середовища	Особливість конкурентного середовища	В чому проявляється характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип	Досконале	Існує 4 фірми-конкуренти на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
Рівень конкурентної боротьби	Міжнародне	Один зарубіжній конкурент	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок
Галузева ознака	Внутрішньогалузеве	Конкуренти мають ПЗ, який використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко його переробити для використання у інших галузях
Вид товарів	Товарно-видове	Однаковий вид товарів (ПЗ) і послуг (сфера медицини)	Створити ПЗ, враховуючи недоліки конкурентів
Характер конкурентних переваг	Нецінове	Вдосконалення технології створення ПЗ, для зменшення його собівартості	Використання менш дорогих технологій, більш ефективних методологій
Інтенсивність	Немарочне	Бренди відсутні	-

Як було зазначено вище, конкуренція відіграє надзвичайно валиву роль у розвитку компанії. У зв’язку з цим в Таблиці 4.9 наведено аналіз конкуренції в галузі за кількома її складовими, для кожної з якої наведено висновки.

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти (бар'єри входження в ринок)	Фактори сили постачальників	Фактори сили споживачів	Фактори загроз з боку товарів-замінників
Висновки	Існує 4 конкуренти на ринку Найбільш схожим за виконанням є конкурент 3	Можливості для входу на ринок наявні Надане рішення спрощує та пришвидшує роботу спеціаліста.	Постачальники відсутні	Важливим для користувача є зручність у користуванні	Використання дешевних технологій створення ПЗ Менша собівартість товару

Відповідно до результатів аналізу пропозиції описаного продукту, наведеного у Таблиці 4.9 та необхідного для виходу на ринок, описаний в Таблиці 4.4, визначено основні особливості конкуренції на ринку стартап-проекту та зазначено висновки щодо кожної зі складових проведеного аналізу для успішної конкуренції на ринку.

На основі аналізу конкуренції із урахуванням характеристик ідеї проекту, вимог споживачів до товару та факторів маркетингового середовища (Таблиці 4.8 та 4.9) визначається та обґрунтовується перелік факторів конкурентоспроможності що наведено у Таблиці 4.10.

Таблиця 4.10 - Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентноспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Кросплатформність	Відсутність жорстких вимог до платформи для встановлення
2.	Масштабованість та адаптивність	Без додаткових зусиль розширюється як функціональність проекту, так і апаратне забезпечення, необхідне для роботи

Відповідно до результатів аналізу факторів конкурентоспроможності продукту, наведеного у Таблиці 4.10 та необхідного для виходу на ринок, описаний в Таблиці 4.4, обґрунтувано головні фактори конкурентоспроможності, такі як

кросплатформність, масштабованість та адаптивність, переваги яких також було розкрито в Таблицях 4.6 та 4.7.

На основі результатів аналізу зазначених вище факторів конкурентоспроможності даного стартап-проекту проведено аналіз сильних та слабких його сторін, результат якого надано в Таблиці 4.11 за використання оцінки конкурентноспроможності за 20-бальною шкалою.

Таблиця 4.11 - Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентноспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з нашим підприємством						
			-3	-2	-1	0	+1	+2	+3
1.	Кросплатформність	17		+					
2.	Масштабованість та адаптивність	20	+						
3.	Необхідність в доступі до Інтернету	3						+	

Відповідно до результатів аналізу сильних та слабких сторін проекту, наведеного у Таблиці 4.10 та необхідного для виходу на ринок, описаний в Таблиці 4.4, було визначено, що найбільшою перевагою стартап-проекту є його масштабованість та адаптивність, що робить можливим надання нового функціоналу вже існуючій системі; а також кросплатформність, що дозволяє не встановлювати жорстких вимог до платформи для розміщення системи.

Результати аналізу, що надані в Таблиці 4.11, використано для проведення SWOT-аналізу, що є фінальним етапом ринкового аналізу можливостей впровадження проекту. Результат SWOT-аналізу представлено у вигляді матриці аналізу сильних (Strength) та слабких (Weak) сторін стартап-проекту, зазначених в Таблиці 4.2, загроз (Troubles) з Таблиці 4.6 та можливостей (Opportunities) з Таблиці 4.7.

Таблиця 4.12 – SWOT-аналіз стартап-проекту

S	Можливість надання нового функціоналу вже існуючій системі, відсутність жорстких вимог до платформи	Необхідність в доступі до Інтернету	W
O	Таргетингова реклама, акцентуваня уваги на надійності системи, заохочення співробітників конкурентів до зміни компанії	Надання додаткових переваг власного продукту лише за появи сильного конкурента, перегляд виконання умов, що зменшують податки, поступове підвищення тарифів	T

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Для визначених альтернатив виконано аналіз з точки зору строків та ймовірності отримання ресурсів, результати якого наведенов Таблиці 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення ПЗ використовуючи патерн Saga	70%	8 місяців
2.	Створення ПЗ використовуючи патерн 2PC	55%	10 місяців

Відповідно до результатів аналізу з точки зору строків та ймовірності отримання ресурсів, наведеного у Таблиці 4.10, для реалізації проекту більш доцільним є вибір альтернативи 1 - Створення ПЗ використовуючи патерн Saga, так як вона передбачає вищу ймовірність отримання ресурсів та більш стислі 8 строки реалізації, тобто є переважною по обом параметрам.

4.4 Розробка ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.14 - Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Державні клініки	Спрощення процесу діагностування та організації лікування	Середній	4 конкурента з продуктами відповідної функціональності, проте гіршої гнучкості	Адаптивність та кросплатформність
2.	Приватні клініки	Спрощення процесу діагностування та організації лікування	Великий		Адаптивність та кросплатформність
Обрано цільові групи: Державні клініки, Приватні клініки					

Результати розгляду цільових груп потенційних споживачів стартап-проекту, наведені в Таблиці 4.14, дозволяють визначити які переваги продукту є можливість використати для виходу на цей сегмент ринку, а також чи є доцільною витрата ресурсів для впливу на певну групу.

За результатами аналізу потенційних груп споживачів (сегментів) було обрано дві основні цільові групи, яким проект буде запропоновано для використання. Для ефективного впровадження продукту в обраних групах має бути розроблена стратегія охоплення ринку.

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1.	Створення транзакційної системи сервісів на основі патерну Сага	Ринкове позиціонування	Адаптивність, кросплатформність, простота в налаштуванні	Диференціація

За результатами визначення базової стратегії розвитку, представленими в Таблиці 4.15, було обрану оптимальну стратегію розвитку за альтернативою, обраною в Таблиці 4.13, що має задовольняти основним потенційним групам споживачів стартап-проекту, розглянутим в Таблиці 4.14.

Відповідно до результатів отриманих в ході вибору базової стратегії розвитку стартап-проекту, що в включає в себе такі компоненти як альтернатива розвитку, стратегія охоплення ринку, конкурентноспроможні позиції, було обрано диференціальну базову стратегію. В той самий час необхідним є також і вибір стратегії конкурентної поведінки, наданий в Таблиці 4.16.

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	«Першопроходець» на ринку	Агресивний пошук нових споживачів	Копіювання основних характеристик в конкурент	Стратегія конкурентної поведінки
1.	Ні	Так	Так: - кросплатформність (к-ти 1 та 3) - простота налаштування (к-т 1)	Зайняття конкурентної ніші

За результатами визначення базової конкурентної поведінки, представленими в Таблиці 4.16, було обрану оптимальну стратегію конкурентної поведінки за використання основних характеристик-переваг конкурентів, описаних в Таблиці 4.2, для якої були зазначені наявність агресивного пошуку споживачі та зайняття конкурентної ніші як стратегія безпосередньо, що має надати можливість ефективної боротьби з вищезгаданими конкурентами.

На основі вимог споживачів з двох основних сегментів, описаних в Таблиці 4.14, до стартап-компанії та до її системи сервісів, а також зважаючи на диференціальну базову стратегіію розвитку та стратегії зайняття конкурентної ніші - стратегії конкурентної поведінки, - розроблено стратегію позиціонування, а саме сформовано ринкову позицію як комплекс асоціацій, для ідентифікації споживача торгівельної марки та проекту.

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Адаптивність, простота в налаштуванні, конфіденційність	Диференціація	Адаптивність, кросплатформність, простота в налаштуванні	Адаптивність Конфіденційність Простота

За результатами визначення стратегії позиціонування, представленими в Таблиці 4.17, було обрану оптимальну стратегію позиціонування стартап-проекту за орієнтації на основні вимоги до товару в цільової аудиторії. З основних ключових конкурентоспроможних позиції стартап-проекту, наданих в Таблиці 4.15, в якості асоціацій було обрано дві, що задовольняють поставленим вимогам: адаптивність та простота. Третьою позицією для асоціацій було обрано не конкурентноспроможну позицію, а основну вимогу до будь-яких проектів в даній сфері - конфіденційність.

4.5 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у Таблиці 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Економність в ресурсах ОС	Використання баз знань, що ефективно зберігають дані	Бази знань
2.	Адаптивність	Включення нових модулів в систему без зайвих зусиль	Слабка зв’язаність компонентів

За результатами визначення ключових переваг концепції потенційного товару, представленими в Таблиці 4.18, було розглянуто основні потреби на ринку та проведено оцінку вигод проекту. Було визначено ключові переваги перед

конкурентами, що задовольняють наведеним основним потребам. Такими ключовими перевагами стали використання баз знань та слабка зв’язаність компонентів (має бути впроваджено відповідно до патерну GRASP).

В Таблиці 4.19 наведено трирівневу маркетингову модель товару.

1-й рівень. Задум товару - засобом вирішення якої потреби / проблеми буде даний товар, яка його основна вигода (основа технічного завдання).

2-й рівень. Рішення реальної реалізації товару: якість, властивості, упаковка.

3-й рівень. Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін).

Таблиця 4.19 - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Адаптивна система, що аналізує симптоми пацієнта та рекомендує і допомагає впроваджувати лікування		
II. Товар у реальному виконанні	Властивості / характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Адаптивність	Нм	Технологічна
	2. Кросплатформність	Нм	Технологічна
	3. Конфіденційність	Нм	Технологічна
	4. Простота	Нм	Технологічна
	Якість: згідно до стандарту ISO 4444 буде проведено тестування		
	Маркування відсутнє		
	Моя компанія: “Binary Health”		
III. Товар із підкріплюнням	1-місячна пробна безкоштовна версія		
	Постійна підтримка для користувачів		
За рахунок чого потенційний товар буде захищено від копіювання: патент, електронні ключі			

За результатами виконання трирівневевої маркетингової моделі товару, представленими в Таблиці 4.19, було визначено задум товару його властивості та характеристики, стандарт якості, маркування та назва. Було визначено підкріплення товару - пробна версія та безкоштовна регулярна підтримка.

Було визначено засоби захисту від копіювання: патент на законодавчому рівні та електронні ключі на фізичному рівні.

Надалі було визначено цінові межі, якими слід керуватись при встановленні ціни на потенційний товар, в ході чого було експертним методом проведено аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів, результати якого представлені в Таблиці 4.20.

А також визначено оптимальну систему збуту, в межах якого приймається рішення, результати чого представлені в таблиці 4.21.

Таблиця 4.20 - Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники, грн	Рівень цін на товари-аналоги, грн	Рівень доходів цільової групи споживачів, грн	Верхня та нижня межі встановлення ціни на товар/послугу, грн
1.	33000	45000	260000	27000-40000

Таблиця 4.21 - Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Придбання місячної або річної підписки	Продаж	0 (напряму) 1 (через одного посередника)	Власна та через посередників

За результатами визначення цінових меж та оптимальної системи збуту, представленими в Таблиці 4.20 та Таблиці 4.21, із врахуванням цін на товари-аналоги та товари-замінники було встановлено верхню та нижню границі ціни. До того ж було сформовано систему збуту, що передбачає продовження ліцензії шляхом придбання періодичної підписки: місячної чи річної (дешевшої за місячну сумарно) та описано канали збуту, в результаті чого система збуту була визначена як власна та через посередників.

В якості останньої складової маркетингової програми було розроблено концепцію маркетингових комунікацій, для чого було використано результат попереднього вибору основи для позиціонування та визначену специфіку поведінки клієнтів, результати розробки якої наведено в Таблиці 4.22.

Таблиця 4.22 - Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1.	Придбання ліцензії шляхом отримання унікального ключа	Інтернет	Адаптивність Конфіденційність Простота	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик застосування Таргетингова реклама

За результатами розробки концепції маркетингових комунікацій, представленими в Таблиці 4.22, із використанням результатів формування системи збуту, наведених в Таблиці 4.21, було визначено специфіку поведінки цільових клієнтів; Інтернет, як головний канал комунікації цільових клієнтів; Адаптивність, Конфіденційність, Простота як ключові позиції для позиціонування; а також сформовані завдання рекламного повідомлення, концепція рекламного звернення.

4.6 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами.

В ході складання документації було сформовано ідею стартап-проекту “Розподілена хореографічна система медичної допомоги”, наведено порівняння із рядом потенційних конкурентів, що надало можливість оцінки конкурентноспроможності та можливості і складності виходу стартапу на ринок; надано результати огляду основних стеків технологій, що можуть бути використані

з метою реалізації системи стартап-проекту та обрано оптимальний стек; надано результати аналізу характеристик попиту на ринку; виконано аналіз цільових аудиторій споживачів описаного продукту; аналіз можливих факторів загроз та аналіз можливих сприятливих умов для стартап-проекту.

Було виконано обґрунтування факторів конкурентоспроможності та порівняльний аналіз сильних та слабких сторін проекту, результати яких пізніше були використані для SWOT-аналізу стартап-проекту.

Було обрано цільові групи потенційних споживачів, визначено базову стратегію розвитку та стратегії конкурентної поведінки та позиціонування; розроблено маркетингову програму.

В результаті було сформовано висновок, що для успішного виконання проекту необхідно реалізувати систему сервісів транзакційність подій в якій буде забезпечуватися патерном проектуванням Сага. В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків.

Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

ВИСНОВКИ

В даній роботі було досліджено такий підхід до проектування системи веб-сервісів, як їх семантична хореографія.

Було проведено огляд предметної області роботи, розглянуто основні поняття та засоби, що були пізніше використані при виконанні роботи, такі як веб-сервіс та семантичний веб-сервіс, бази знань та їх застосування.

На етапі постановки завдання було визначено мету та цілі роботи, окреслено її головні подальші етапи та визначено засоби її виконання. Було вказано головні характеристики майбутнього додатку, що в процесі виконання роботи має бути розробленим.

Ознайомлення з означеними поняттями та принципами більш точно окреслило предмет дослідження даної роботи, дозволило ефективно використовувати інструменти для реалізації підходу хореографії REST-сервісів, що базується на семантичних процесах.

Відповідно до прийнятих рішень, було встановлено головні кроки розробки програмного продукту з використанням відповідних інструментів. Спроектований програмний продукт було імплементовано на мові програмування Scala, що надає можливість використовувати його на різних платформах.

В ході роботи було створено реалізацію системи сервісів, що взаємодіють між собою по HTTP-протоколу згідно REST. Між сервісами відсутні прямі залежності - не встановлено, до яких сервісів поточний має звертатися для отримання даних, необхідних для подальшої роботи. Для визначення цього використовується розсилання метаданих наступного запиту всім сервісам через брокер повідомлень; відповідні сервіси аналізують метадані запиту і порівнюють із власними метаданими, а в разі відповідності надають ендпоінт для надсилання запиту.

Відсутність прямих з’вязків між сервісами дозволяє додавати та видаляти окремі модулі системи без надмірних зусиль, змін у вже наявних модулях чи певних підготовчих етапів, що впливають на поведінку поточної версії продукту.

В реалізації використано патерн проектування Сага, що забезпечує транзакціність обробки запиту та дозволяє обробляти помилки, пов’язані в тому числі і з встановленням правильного порядку взаємодії сервісів.

Надані також і рекомендації щодо покращення запропонованого рішення, такі як деталізація метаданих в онтології, динамічне конструювання онтологій, заміна брокера повідомлень та використання тематик повідомлень в брокері.

Програмний продукт було розроблено за використанням TDD. Було підтверджено його придатність до використання для досягнення поставлених в роботі цілей.

Для дослідження підходу використання семантичної хореографії у системі REST-сервісів продукт було протестовано за набором тестових сценаріїв. Відповідно до результатів тестування було окреслено основні чинники, що впливають на ефективність використання зазначеного підходу, описані його переваги та засоби покращення створеного рішення.

Було проведено дослідження щодо можливості ринкової комерціалізації проекту. В ході складання документації до проекту було сформовано ідею стартап-проекту “Розподілена хореографічна система медичної допомоги”, наведено порівняння із рядом потенційних конкурентів, що надало можливість оцінки конкурентноспроможності та можливості і складності виходу стартапу на ринок. Було виконано обґрунтування факторів конкурентоспроможності та порівняльний аналіз сильних та слабких сторін проекту, результати яких пізніше були використані для SWOT-аналізу стартап-проекту. В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків.

Отримані результати свідчать про те, що розглянутий підхід може бути успішно використаний для підвищення ефективності взаємодії між сервісами. Також, було складено рекомендації щодо того, як уникнути основних проблем, пов’язаних з використанням даного підходу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amazon.com: Web Services V.2.0: http://associates.amazon.com/exec/panama/associates/ntg/browse/- /1067662/ref=gw_hp_ls_1_3/
2. Apache Software Foundation: WSIF: http://ws.apache.org/wsif/
3. DAML Joint Committee.: Daml+oil (march 2001) language.: http://www.daml.org/2001/03/daml+oil-index.html 2001
4. DAML-S Coalition.: Daml-s: Web services for the semantic web: In ISWC2002.
5. Dan King: The Dublin Core Element Set Ontology: http://www.daml.org/ontologies/201
6. Davenport T., PrusakL. WorkingKnowledge: how organizations manage what they know. - Boston:Harvard Business School Press, 1998. - 200 p.
7. E. Christensen, F. Curbera, G. Meredith, and S.Weerawarana.: Web Services Description Language (WSDL): http://www.w3.org/TR/2001/NOTE-wsdl-20010315 2001.
8. European Guide to good Practice in Knowledge Management -Part 1: Knowledge Management Framework // [Электронный ресурс]. - 2004. - Режим доступу: ftp://cenftp1.cenorm.be/PU-BLIC/e-Europe/KM/CWA14924-01-2004-Mar.pdf
9. F. Curbera, Y. Goland, J. Klein, Microsoft, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana: Business Process Execution Language for Web Services, Version 1.0: http://www106.ibm.com/developerworks/webservices/library/ws-bpel/
10. Gruber T.A. Translation Approachto Portable Ontology Specifications // Knowledge Acquisition.- 1993. - V. 5. - № 2. - P. 199-220.
11. Hohpe, Gregor and Woolf, Bobby. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. s.l. : Addison-Wesley, 2003.
12. Hub and Spoke [or] Zen and the Art of Message Broker Maintenance. Enterprise Integration Patterns. 2003-11-12. Retrieved 2010-10-14.
13. M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and xml schemas. In Electronic Trans. on Artificial Intelligence, 2001.

14. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara.: Semantic matching of web services capabilities. In ISWC2002, 2002
15. Metadata and Resource Description // W3C Technology & Society Domain. - 2001. - [Електронний ресурс] - Режим доступу: http://www.w3.org/Metadata - 20.01.2016.
16. OWL-S: Semantic Markup for Web Services // W3C Member Submission. - 2004. - [Електронний ресурс] - Режим доступу: http://www.w3.org/Submission/OWL-S - 20.01.2016.
17. Pattern: Saga // Microservice Architecture. - 2017. - [Електронний ресурс] - Режим доступу: http://microservices.io/patterns/data/saga.html - 10.04.2018.
18. Saga Pattern | How to implement business transactions using Microservices - 2018. - [Електронний ресурс] - Режим доступу: https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part/ - 12.04.2018.
19. T. Berners-Lee, J. Hendler, and O. Lassila.: The semantic web.: Scientific American, 284(5):34--43, 2001.
20. UDDI: The UDDI Technical White Paper.: http://www.uddi.org/ 2000.
21. W3C Web Services Architecture Working Group: Web services Glossary: http://www.w3.org/TR/2002/WD-ws-gloss-20021114/
22. W3C Web services Choreography Working Group: Charter: http://www.w3.org/2003/01/wscwg-charter
23. W3C, «RDF/XML Syntax Specification (Revised)» // [Электронный ресурс]. - 2003.-Режимдоступа: http://www.w3.org/TR/rdf-syntax-grammar/
24. Web Ontology Language. Overview // [Электронный ресурс]. -2003. - Режим доступа: http://www.w3.org/TR/owl-features/
25. Web Service Modeling Language (WSML) // W3C Member Submission. 2005. - [Електронний ресурс] - Режим доступу: https://www.w3.org/Submission/WSML - 20.01.2016.
26. Web Service Modeling Ontology (WSMO) // W3C Member Submission. - 2005. - [Електронний ресурс] - Режим доступу: https://www.w3.org/Submission/WSMO 20.01.2016.

27. Web Services Description Language (WSDL) // W3C. - 2001. - [Електронний ресурс] - Режим доступу: http://www.w3.org/TR/wsdl - 20.01.2016.
28. Web Services Statement // W3C Process Document. - 2013. - [Електронний ресурс] - Режим доступу: https: // www.w3.org/2002/ws/Activity - 20.01.2016.
29. Елена Г.У. ТЕХНОЛОГИИ ОПИСАНИЯ СЕМАНТИЧЕСКИХ ВЕБ-СЕРВИСОВ // Естественные и математические науки в современном мире: сб. ст. по матер. XLVI междунар. науч.-практ. конф. № 9(44). – Новосибирск: СибАК, 2016. – С. 29-34.
30. Піпіч А. А. Застосування семантичних веб-технологій для опису веб-сервісів // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.
31. Піпіч А. А. Застосування хореографії в шаблоні проектування Сага // Міжнародний науковий журнал "Інтернаука". — 2018. — №8.
32. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: Навчальний посібник. — Запоріжжя: ЗНТУ, 2008. — 341 с.
33. Тузовский А.Ф. Метод объединения онтологий предметных областей // Известия Томского политехнического института. - 2006. - № 7. - С. 138-141.
34. Тузовский А.Ф. Работа с онтологической моделью организации на основе дескриптивной логики // Известия Томского политехнического университета. - 2006. - Т. 309. - № 7. -С. 134-137.
35. Тузовский А.Ф., Васильев И.А., Усов М.В. Программная реализация основных компонент информационно-программного обеспечения системы управления знаниями // Известия Томского политехнического университета. - 2004. - Т. 307. - № 7. - С. 116-122.
36. Тузовский А.Ф., Козлов С.В., Чириков С.В., Ямпольский В.З. Использование онтологий в системах управления знаниями организаций // Известия Томского политехнического университета. - 2006. - Т. 309. - № 3. - С. 180-184.
37. Тузовский А.Ф., Чириков С.В., Ямпольский В.З. Системы управления знаниями (методы и технологии). - Томск: Изд-во НТЛ, 2005. - 260 с.