Презентація Магістерської дисертації
на тему

# Семантична хореографія REST-сервісів

Піпіч Артем

# Актуальність та мета роботи

Актуальність теми полягає в тому, що динамічно налаштована взаємодія сервісів дозволяє синтезувати розширену функціональність програмної системи на основі існуючих функцій, оформлених як сервіси.

Мета роботи - дослідження ефективності способів реалізації хореографії сервісів та створення власної реалізації, заснованої на семантичних технологіях, її тестування.

# Постановка завдання

Аналіз існуючих підходів до організації взаємодії веб-сервісів в системі

Приділення особливої уваги підходам, в основі яких лежить хореографія
(виділення їхніх характерних переваг та недоліків)

Розробка семантичної хореографії як підходу до автоматизації децентралізованої обробки запиту сервісами, який використовує онтології

Розробка програмного прототипу, що реалізує запропонований підхід

# Постановка завдання

Розроблений програмний засіб повинен моделювати систему, що складається з незалежних сервісів.

В системі мають бути 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. Механізм хореографії в системі має вміти на сонові онтологій визначати, до якого сервісу слід звернутися поточному сервісу.

# Існуючі рішення

Передача WS-CDL опису запиту разом із самим запитом та його обробка у фільтрі сервісу

Попередній запит даних про актуальні сервіси за використання BPEL та мережі Петрі

Business Process Execution Language for Web Services

# Використані інструменти та фреймфорки

- Мова програмування:
  Scala
- Контейнер класів:
  SpringBoot
- Брокер повідомлень:
  Akka Actors
- Сховище даних:
  Apache Jena

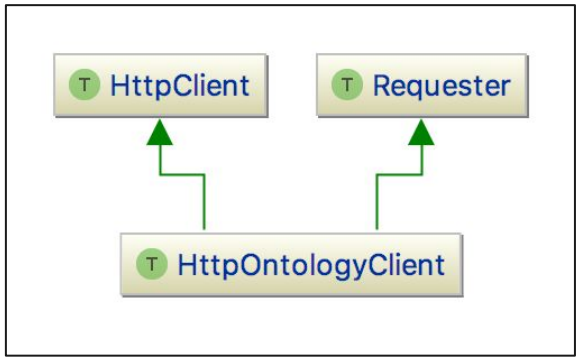
Maven™



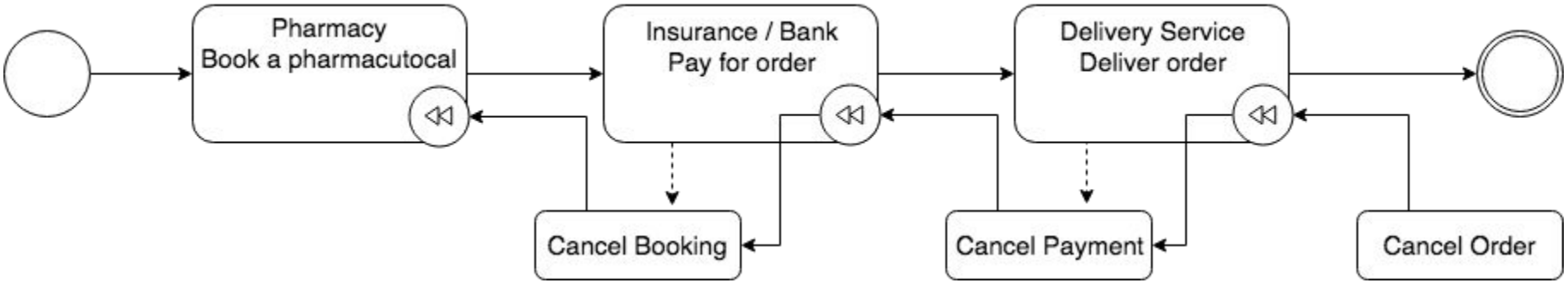
# Використані патерни проектування

```
package service

import ...

trait SourceRequestResolver extends RequestResolver with Actor {
 private val mediator = DistributedPubSub(context.system).mediator
 mediator ! Subscribe("content", self)

 def source: String

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

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

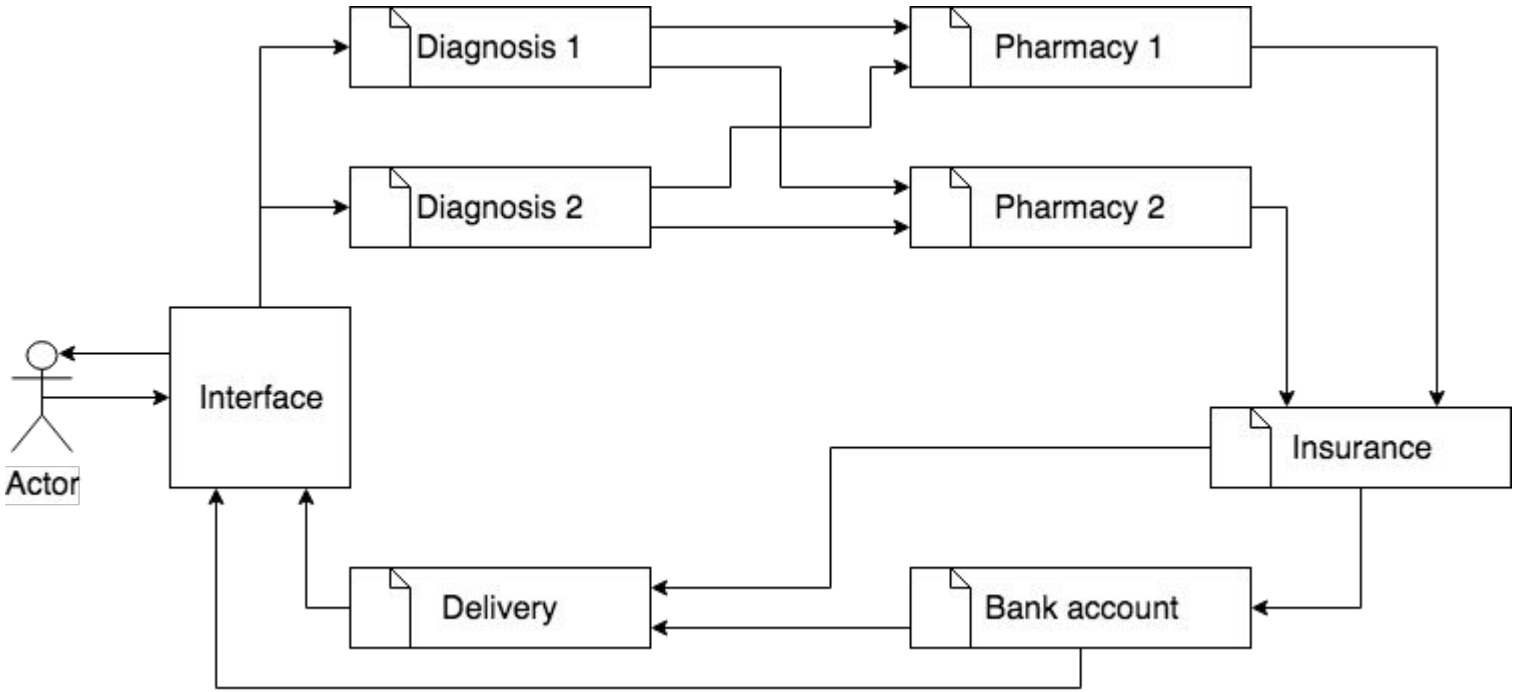
- Фасад
- Command
- Медіатор
- Сага



# Патерн проектування Сага



# Схема взаємодії сервісів



# Структура модулів додатку

**Основні модулі додатку**

semantic-choreography-web

semantic-choreography-modules
semantic-choreography-parent

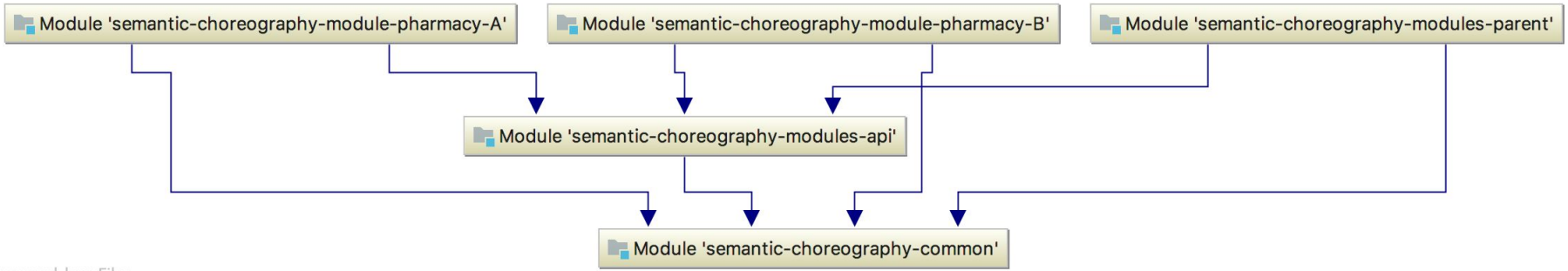
semantic-choreography-modules-api
semantic-choreography-common

**semantic-choreography-modules:**

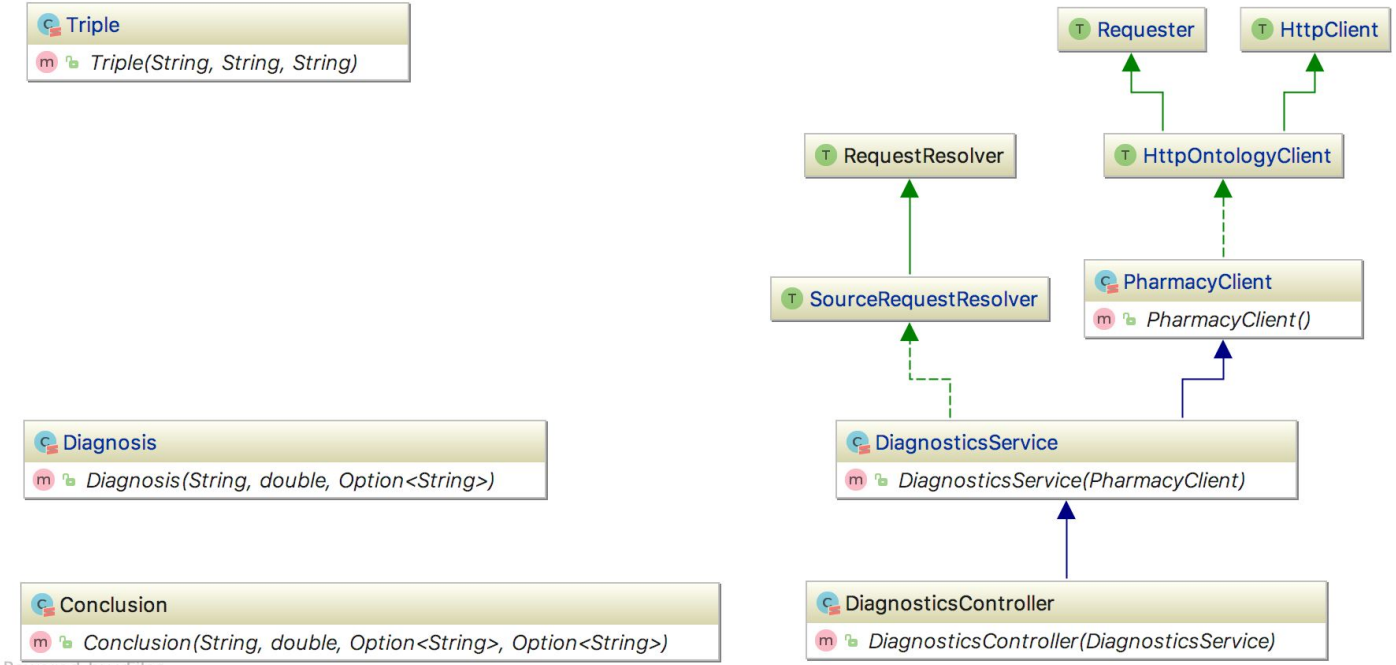
- semantic-choreography-module-bank
- semantic-choreography-module-delivery
- semantic-choreography-module-diagnostics-A
- semantic-choreography-module-diagnostics-B
- semantic-choreography-module-insurance
- semantic-choreography-module-pharmacy-A
- semantic-choreography-module-pharmacy-B

semantic-choreography-modules-parent
semantic-choreography-modules-api
semantic-choreography-common

# Фрагмент структури модулів додатку



# Спрощена діаграма класів модулю semantic-choreography-module-diagnostics-A



# Приклади фрагментів опису сервіса

```
scapi:Service a rdfs:Class

scapi:Endpoint a rdfs:Class

scapi:Parameter a rdfs:Class
```

```
scdiaga:Name a rdfs:Class
             rdfs:subClassOf scapi:Name
             rdfs:domain scapi:Person

scdiaga:Age a rdfs:Class
            rdfs:subClassOf scapi:Parameter
            rdfs:range rdfs:Literal
            rdfs:domain scapi:Person

scdiaga:Diet a rdfs:Class
             rdfs:subClassOf scapi:Parameter
             rdfs:domain scapi:Behaviour
```

```
scdiagae:getDiagnosis rdf:type scapi:Endpoint;
                      scapi:requires scdiaga:Name
                      scapi:requires scapi:Age
                      scapi:accepts scdiaga:Diet

scdiagae:serviceDiagA rdf:type scapi:Service
                      rdfs:label "Diagnostics A"
                      scapi: hasProperty scapi:getDiagnosis
```

# Отримані результати

| 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 |

# Отримані результати

- На виконанні звичайного тестового сценарію успішно оброблено 8 / 10 запитів
  - API сервісів відповідає опису в онтології
  - 2 сервіси мають аналоги зі схожим API
- Сценарій неконсистентного API - успішно оброблено 2 / 10 запитів
  - API сервісів має суттєві відмінності від свого опису в онтології
- Сценарії модифікованих API
  - API сервісів відповідає опису в онтології
  - ендпоінти сервісів мають менше або більше вхідних параметрів - 7 та 9 запитів з 10 успішно оброблено
  - API сервісів відрізняються за набором параметрів більше або менше - 10 / 10 та 6 / 10 відповідно
- Кількість сервісів в системі
  - сервіси мають більше або менше аналогів - 8 / 10
  - наявні сервіси, що не мають брати участі в обробці запиту 8 / 10
  - зростання кількості сервісів іноді призводить до результату гіршого за стандартний (7, рідко 6 з 10), причина чого полягає у зростанні ймовірності випадкової відповідності запиту певному API

# Переваги розробленого рішення

- відсутність прямих залежностей між сервісами, динамічне отримання цих залежностей
  - спрощений процес підключення / видалення сервісу
    - не треба змінювати існуючі сервіси
    - відсутність підготовчих етапів (до підключення сервісу), що впливають на роботу системи
  - взаємозамінність сервісів
    - відмовостійкість системи
- використання патерну проектування Сага
  - транзакційність обробки запиту
  - можливість якісної обробки помилок

# Засоби покращення розробленого рішення

## Деталізація метаданих в онтології

Однією з проблем даного рішення, виявлених під час тестування, було псевдопозитивне розпізнавання

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

Чим менше надано правил, тим складніше визначити, який саме запит буде надіслано.

Більш деталізований опис  API сервісу зменшує кількість псевдопозитивних результатів, що також відповідає результатам, отриманим під час тестування

# Засоби покращення розробленого рішення

## Динамічне конструювання онтологій

API сервісу доводиться описувати принаймні двічі - при його розробці та при наданні опису сервісу в онтології

- витрачаються час та ресурси
- зростає ймовірність того, що буде порушено консистентність між реальним API та його описом в онтології

Пропонується автоматично генерувати опис API сервісу в онтології
(на кшалт того, як звичайна документація до API генерується за допомогою Swagger)

# Засоби покращення розробленого рішення

## Заміна брокера повідомлень

Для реалізації обміну сервісів повідомленнями були використані такі бібліотеки як Akka Actor та Akka Cluster.

При зростанні системи та у випадку реалізації розподіленої системи сервісів, більш доречним стає використання Kafka у поєднанні з Akka Stream.

- з самого спроектований як розподілена система, яку легко масштабувати,
- підтримує високу пропускну здатність як з боку джерел, так і систем-підписників
- підтримує об'єднання підписників в групи
- забезпечує можливість тимчасового зберігання даних для подальшої пакетної обробки

# Засоби покращення розробленого рішення

## Використання тематик повідомлень

При зростанні кількості сервісів зростає і кількість семантичних перевірок запиту сервісами.

Щоб не перевіряти кожен з запитів, є можливість впровадити тематичну розсилку метаданих

Тематика запиту може автоматично визначатися сервісом-відправником за метаданими

- робить сервіси менш гнучкими, так як певним чином обмежує список доступних сервісів
- зменшує кількість семантичних перевірок
- дає більші гарантії того, що запит не буде оброблено псевдопозитивно

# 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.

# Висновки

В ході роботи було проаналізовано підходи до організації взаємодії веб-сервісів що спілкуються між собою по HTTP протоколу.

Було виділено децентразілований семантичний підхід, в якому для визначення динамічних залежностей між сервісами використовується розсилання метаданих майбутнього запиту сервісам через брокер повідомлень

Сервіси аналізують такі метадані запиту на основі онтологій, що описують їх API, і в разі відповідності надають ендпоінт для надсилання запиту.

# Висновки

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 конкретній реалізації даного підходу було 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.

# Висновки

Для дослідження підходу використання семантичної хореографії у системі  веб-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.

# Висновки

Отримані результати свідчать про те, що розглянутий підхід може бути успішно використаний для підвищення ефективності взаємодії між сервісами.

Було також зазначено рекомендації щодо того, як уникнути основних проблем, пов’язаних з використанням даного підходу.

# Дякую за увагу!