МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 01.00.00.000 ПЗ

Група ІПм-23-1

Гавучак Назар

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Гавучак Назар Вікторович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методи та методологія побудови репозиторіїв моделей як сервісів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Гавучак Н.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Вовк Роман Богданович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Гавучаку Назару Вікторовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методи та методологія побудови репозиторіїв моделей як сервісів”

керівник проекту (роботи) Вовк Роман Богданович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні методи побудови та функціонування інформаційних та програмних технологій репозиторіїв моделей

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз предметної області дослідження побудови моделей для аналітики даних
2. Дослідження та опис технології побудови репозиторіїв моделей
3. Методи трансформації моделей бізнес-процесів
4. Імплементація системи автоматизації побудови та аналізу репозиторіїв моделей

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Еталонна модель аналізу даних CRISP-DM (рис. 1.1)
2. Огляд процесу збирання та аналізу UML (рис. 1.2)
3. Основні елементи BPMN (рис. 1.3)
4. Основні елементи EPC (рис. 1.4)
5. Обмін послугами між постачальником і споживачем в SOA (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Аналіз предметної області дослідження побудови моделей для аналітики даних	15.10.2024	виконано
4	Дослідження та опис технології побудови репозиторіїв моделей	08.11.2024	виконано
5	Методи трансформації моделей бізнес-процесів	20.11.2024	виконано
6	Імплементація системи автоматизації побудови та аналізу репозиторіїв моделей	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 80 с., 30 рис., 1 табл., 50 джерел.

Тема: Методи та методологія побудови репозиторіїв моделей як сервісів

Об'єкт дослідження: процеси автоматизації побудови та управління репозиторіями моделей як сервісів у контексті інформаційних технологій.

Мета роботи: розробка та обґрунтування ефективних методів, методології та системи автоматизації побудови репозиторіїв моделей як сервісів для забезпечення швидкого доступу, управління та інтеграції моделей у різні інформаційні системи, що дозволить підвищити продуктивність, масштабованість та гнучкість процесів моделювання.

Предмет дослідження: методи, методології та системи автоматизації побудови репозиторіїв моделей, їх інтеграція у сервіси та забезпечення ефективної роботи з ними.

Результати дослідження

В роботі виконано розробку нових методів автоматизації побудови репозиторіїв моделей як сервісів, які забезпечують інтеграцію різних моделей і платформ у єдине середовище.

Висновок

Представлено підхід до модульної архітектури системи автоматизації репозиторіїв, що дозволяє масштабувати та інтегрувати різні компоненти без втрати продуктивності та розроблено механізми адаптивної інтеграції моделей, що дозволяє використовувати репозиторії в умовах різнорідних даних та систем.

РЕПОЗИТОРІЙ МОДЕЛЕЙ, СЕРВІС-ОРІЄНТОВАНА АРХІТЕКТУРА, ІНТЕГРАЦІЯ МОДЕЛЕЙ, ЖИТТЄВИЙ ЦИКЛ МОДЕЛЕЙ, МОДЕЛІ ДАНИХ, ІНТЕРОПЕРАБЕЛЬНІСТЬ, МЕТОДИ МОДЕЛЮВАННЯ, МІКРОСЕРВІСНА АРХІТЕКТУРА

ABSTRACT

Master Thesis: 80 pp., 30 fig., 1 tab., 50 sources.

Thesis Subject: Methods and methodology of building repositories of models as services

Object of research: processes of automation of construction and management of repositories of models as services in the context of information technologies.

The purpose of the work: development and substantiation of effective methods, methodology and systems for automating the construction of model repositories as services to ensure quick access, management and integration of models into various information systems, which ensure increased productivity, scalability and relevance of modeling processes.

The subject of research: methods, methodology and automation systems for building model repositories, their integration into services and ensuring effective work with them.

Research results

The work involves the development of new methods of automating the construction of model repositories as services, which ensure the integration of various models and platforms into a single environment.

Conclusion

An approach to the modular architecture of the repository automation system is presented, which allows scaling and integration of various components without loss of productivity, and the mechanisms of adaptive integration of models are developed, which allows the use of repositories in conditions of heterogeneous data and systems.

MODEL REPOSITORY, SERVICE-ORIENTED ARCHITECTURE, MODEL INTEGRATION, MODEL LIFE CYCLE, DATA MODELS, INTEROPERABILITY, MODELING METHODS, MICROSERVICE ARCHITECTURE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 9
ВСТУП 10

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ ПОБУДОВИ МОДЕЛЕЙ ДЛЯ АНАЛІТИКИ ДАНИХ 15
1.1. Опис сутності процесів аналітики даних 15
1.2. Особливості аналітики даних та моделей 18
1.3. Особливості мов моделювання бізнес-процесів 20
1.3.1. BPMN: модель бізнес-процесу 21
1.3.2. EPC: ланцюг процесів, керований подіями 22
1.4. Дослідження особливостей Arrowhead Framework 23
1.4.1. Сервісно-орієнтована архітектура 24
1.4.2. Системи та пристрої 25
1.4.3. Система систем (SoS) 25
1.4.4. Концепція локальної хмари 25
Висновки до розділу 27

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ОПИС ТЕХНОЛОГІЇ ПОБУДОВИ РЕПОЗИТОРІЇВ МОДЕЛЕЙ 28
2.1. Огляд архітектур репозиторіїв моделей 28
2.2. Дослідження функціоналу та архітектури платформи Apromore: Advanced Process Model Repository 35
2.3. Дослідження майнінгу репозиторію програмного забезпечення 39
2.4. Методи трансформації моделей бізнес-процесів 42
2.4.1. Пряме перетворення моделі 43
2.4.2. Непряме (опосередковане) перетворення моделі 44
Висновки до розділу 46

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ПОБУДОВИ ТА АНАЛІЗУ РЕПОЗИТОРІЇВ МОДЕЛЕЙ 47
3.1. Представлення основних вимог до функціональності системи побудови репозиторіїв моделей 47
3.2. Представлення архітектури системи автоматизації аналізу моделей ... 49
3.3. Використання Arrowhead Framework в системі аналізу моделей 51
3.4. Реалізація рівня керування репозиторієм моделей 52
3.4.1. Майнер моделі сховища 52
3.4.2. Методи сканування моделі 55
3.4.3. Моделі фільтрації 59
3.5. Реалізація та опис рівня зберігання моделей 63
3.5.1. Система Elasticsearch 64
3.5.2. Моделі та метадані 65
3.6. Реалізація рівня представлення моделей 66
Висновки до розділу 72

ВИСНОВКИ 73
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 75

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

BPM - Business Process Management

BPMN - Business Process Model Notation

CPF - Canonical Process Format

EPC - Event-driven Process Chain

EPML - EPC Markup Language

MAAS - Model Analytics Automation System

MDE - Model-Driven Engineering

OMG - Object Management Group

SoS - System of Systems

SUS - System Usability Scale

SysML - The Systems Modeling Language

TAM - Technology Acceptance Model

mTAM - modified TAM

NPS - Net Promoter Score

PU - Perceived Usefulness

PEU - Perceived Ease-of-Use

YAWL - Yet Another Workflow Language

XMI - XML Metadata Interchange

XSD - XML Schema Definition

ВСТУП

Актуальність теми.

У сучасних умовах стрімкого розвитку інформаційних технологій, моделі стають ключовим інструментом для аналізу, прогнозування та автоматизації процесів у різних галузях. Побудова та використання репозиторіїв моделей як сервісів є важливим напрямом розвитку систем автоматизації, що дозволяє спростити процес інтеграції та управління моделями в різних прикладних областях. Актуальність дослідження зумовлена необхідністю забезпечення швидкого доступу до моделей, їх актуалізації та інтеграції в різні системи для підвищення ефективності бізнес-процесів, наукових досліджень та інноваційних рішень.

Особливості розробки методів та системи автоматизації побудови репозиторіїв моделей як сервісів:

- Модульна архітектура системи. Система повинна мати гнучку модульну структуру, що дозволяє додавати або видаляти компоненти без шкоди для роботи всієї системи. Така архітектура полегшує масштабування та адаптацію до різних вимог користувачів.

Використання мікросервісної архітектури дозволяє розробляти кожну функцію (пошук, керування моделями, версійність, інтеграція) окремо, що спрощує підтримку та оновлення системи.

- Інтероперабельність та стандартизація. Репозиторій повинен підтримувати різні типи моделей (наприклад, моделі машинного навчання, симуляційні моделі, аналітичні моделі) та різноманітні формати даних. Використання загальновизнаних стандартів для обміну моделями та даними (наприклад, PMML для моделей машинного навчання) забезпечує легку інтеграцію з існуючими інструментами та платформами.

- Автоматизація життєвого циклу моделей. Система повинна автоматизувати всі етапи життєвого циклу моделей: створення, оновлення, перевірку, публікацію та виведення з експлуатації. Це включає в себе

механізми для автоматичного оновлення моделей та управління їхніми версіями. Автоматичний вибір і розподіл ресурсів для тестування та виконання моделей в залежності від їх вимог.

- Забезпечення безпеки та доступу. Під час розробки системи важливо впровадити механізми автентифікації та авторизації для контролю доступу до моделей. Це може включати рольову модель доступу, що дозволяє розмежувати права на використання та редагування моделей. Забезпечення шифрування комунікацій та даних для захисту від несанкціонованого доступу.

- Інтеграція з хмарними сервісами та оркестрація. Система повинна підтримувати роботу в хмарних середовищах для забезпечення гнучкого масштабування, що дозволить зберігати та обробляти моделі без прив’язки до фізичних серверів. Використання систем оркестрації контейнерів, таких як Kubernetes, для автоматизації розгортання моделей у різних середовищах.

- Оптимізація управління моделями. Важливо реалізувати інструменти для пошуку та фільтрації моделей на основі метаданих, таких як тип моделі, дата створення, версія, автор і продуктивність. Це допоможе користувачам швидко знаходити потрібні моделі. Інструменти для оцінки продуктивності та порівняння моделей, що допоможе в прийнятті рішень щодо їх використання.

- Підтримка колаборативної роботи. Система повинна дозволяти кільком користувачам одночасно працювати з одними й тими ж моделями, забезпечуючи зручні інструменти для спільної роботи, обговорення та документування результатів.

- Масштабованість і адаптивність. Система повинна бути здатною до швидкого масштабування в залежності від обсягів даних та кількості користувачів. Адаптивність стосується можливості інтеграції нових методів або моделей без кардинальних змін архітектури. Використання хмарних обчислень для динамічного розподілу ресурсів відповідно до навантаження.

- Моніторинг та підтримка експлуатаційної діяльності. Система має забезпечувати моніторинг продуктивності та ефективності моделей, щоб вчасно виявляти та виправляти можливі проблеми, такі як зниження точності моделі або відхилення від очікуваних результатів.

Ці особливості розробки допомагають створити ефективну, масштабовану та безпечну систему для автоматизації управління моделями, що відповідає вимогам сучасних бізнес- та наукових процесів.

З огляду на постійне зростання обсягів даних, необхідність оптимізації роботи з моделями та їх повторного використання, зростає потреба в ефективних методах, методологіях та системах автоматизації побудови репозиторіїв моделей. Це дозволяє зменшити часові та ресурсні витрати на пошук, адаптацію та впровадження моделей, що є критично важливим для компаній, що працюють з великими даними, машинним навчанням та інноваційними технологіями.

Мета дослідження – розробка та обґрунтування ефективних методів, методології та системи автоматизації побудови репозиторіїв моделей як сервісів для забезпечення швидкого доступу, управління та інтеграції моделей у різні інформаційні системи, що дозволить підвищити продуктивність, масштабованість та гнучкість процесів моделювання.

Об’єкт дослідження - процеси автоматизації побудови та управління репозиторіями моделей як сервісів у контексті інформаційних технологій.

Предмет дослідження - методи, методології та системи автоматизації побудови репозиторіїв моделей, їх інтеграція у сервіси та забезпечення ефективної роботи з ними.

Відповідно до мети роботи було сформовано наступні **задачі:**

- виконати аналіз предметної області дослідження побудови моделей для аналітики даних;

- розглянути особливості аналітики даних та моделей;
- виконати дослідження особливостей arrowhead framework;
- дослідити функціонал та архітектуру платформи advanced process model repository;
- описати особливості майнінгу репозиторію програмного забезпечення;
- дослідити методи трансформації моделей бізнес-процесів;
- виконати імплементацію системи автоматизації побудови та аналізу репозиторіїв моделей.

Методи дослідження.

В представленій магістерській роботі використано наступні методи дослідження:

- аналіз та синтез: використовується для вивчення існуючих підходів до автоматизації побудови репозиторіїв моделей, аналізу сучасних інструментів та платформ.

- моделювання: для розробки та перевірки ефективності запропонованих методів і систем автоматизації.

- експериментальний метод: полягає у тестуванні розробленої системи автоматизації на основі реальних даних і моделей.

- метод порівняльного аналізу: застосовується для оцінки переваг і недоліків різних підходів до організації репозиторіїв моделей.

- алгоритмічний підхід: для розробки та оптимізації алгоритмів автоматизації роботи з моделями.

Наукова новизна отриманих результатів полягає в розробці нових методів автоматизації побудови репозиторіїв моделей як сервісів, які забезпечують інтеграцію різних моделей і платформ у єдине середовище.

Практичне значення магістерської роботи полягає в розробці нового підходу до модульної архітектури системи автоматизації репозиторіїв, що

дозволяє масштабувати та інтегрувати різні компоненти без втрати продуктивності.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 80 сторінок, і містить 30 рисунків, 1 таблицю, перелік використаних джерел із 50 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ ПОБУДОВИ МОДЕЛЕЙ ДЛЯ АНАЛІТИКИ ДАНИХ

1.1. Опис сутності процесів аналітики даних

Аналітика даних стає все більш важливим інструментом для великих організацій. Аналітика даних стосується виявлення, інтерпретації та передачі значущих шаблонів у даних (наприклад, тексті, відео, аудіо тощо). Вона включає в себе процеси, інструменти та методи дослідження даних з метою отримання корисної інформації. Створення цінності для бізнесу на основі даних є рекордно високим, і сучасні організації впроваджують аналітику для перетворення інформації в розуміння з метою досягнення операційної ефективності підприємства.

За допомогою аналітики моделей моделі, що використовуються в різних сферах, таких як інженерна та бізнес-сфера, беруться як вхідні дані в аналітичний процес, щоб покращити процес прийняття рішень організаціями. Моделі являють собою спрощене представлення окремих частин розглянутої області. Вони пропускають деталі, які не мають відношення до заданого набору критеріїв, зберігаючи цікаві властивості щодо набору вимог. У інженерній сфері моделі можуть стосуватися проміжних артефактів, які використовуються для реалізації системи. У бізнес-сфері моделі (також їх називають моделями бізнес-процесів) описують набір дій для досягнення спільної мети. Моделі можуть бути визначені різними мовами моделювання. Мова моделювання – це штучна мова, яка використовується для представлення знань або інформації про домен. Вони визначаються в термінах своїх метамоделей, які вказують, як визначаються моделі, а також правила й обмеження, яким мають відповідати їхні моделі. Прикладами двох широко використовуваних мов моделювання є нотація моделі бізнес-процесу (BPMN) і уніфікована мова моделювання (UML).

Подібно до аналітики даних, загальний робочий процес аналітики моделі складається із збору, підготовки та остаточного аналізу моделей і пов’язаних артефактів моделювання. Збір і підготовка великих обсягів даних для аналізу типові для багатьох областей, таких як видобуток біологічних даних для біоінформатики і видобуток програмних репозиторіїв для вихідного коду. Зокрема, кількість сховищ для майнінгу артефактів програмного забезпечення постійно зростає тенденція останніх років. З одного боку, наявні на даний момент технології та інструменти інтелектуального аналізу даних зробили можливим отримувати великі обсяги даних із багатого та різнорідного спектру доменів. З іншого боку, такі платформи, як GitHub, Bit-Bucket і GitLab, які забезпечують велике джерело цих даних, зростають у популярності та використанні [21].

Аналіз великих обсягів артефактів моделювання був одним із головних напрямків у галузі досліджень. Було вжито кілька зусиль для видобутку загальнодоступних репозиторіїв для моделей UML і BPMN [21, 19]. Подальші кроки були розпочаті для зберігання та керування великими наборами моделей і пов’язаних артефактів [6, 31, 18]. Проте, як правило, недостатньо досліджень інструментів, які об’єднують методи збирання та підготовки моделей із сучасними методами зберігання та керування моделями. Крім того, існує обмежена кількість досліджень щодо автоматизації кроків у аналітиці моделей. Усі перераховані вище проблеми заважають дослідникам виконувати емпіричні дослідження у відповідних сферах. Ця робота спрямована на вирішення цієї проблеми шляхом представлення архітектури та впровадження системи автоматизації аналізу моделей (MAAS), що працює в Arrowhead Framework як набір систем для взаємодії, що забезпечує можливості зберігання та пошуку моделей, а також автоматичний підхід до збору та підготовка моделей до аналізу.

Робочий процес аналітики моделі включає різні кроки, починаючи від збору артефактів моделювання за допомогою таких методів, як видобуток репозиторію, до фактичного аналізу артефактів. Розробка системи, яка

повністю автоматизує робочий процес аналітики моделі, є складним і трудомістким завданням. Враховуючи обмеження магістерської роботи, повністю автоматизувати аналітичний процес дослідників було б неможливо. Тому необхідно вибирати частини робочого процесу з точки зору їх унікальності як специфічні послуги, що надаються MAAS. Крім того, послуги мають бути корисними для зацікавлених сторін (дослідників) MAAS. Це було обговорено з головною зацікавленою стороною, і на основі його пропозиції було обрано чотири основні послуги/компоненти:

- Майнінг репозиторію. Компонент майнера сховища, який використовується для пошуку репозиторіїв програмного забезпечення, наприклад GitHub, для потенційних моделей процесів (наприклад, BPMN і EPML).
- Фільтрація та перевірка. Компонент надає послуги для фільтрації моделей процесів за їхніми метаданими та можливість перевірки моделей процесів.
- Перетворення від моделі до моделі. Перетворення моделі процесу з вихідної мови на цільову мову. Зокрема, BPMN до EPML і навпаки.
- Зберігання моделі. Можливість зберігати та отримувати моделі процесів і пов’язані метадані.

Ця робота буде зосереджена на чотирьох вибраних службах і виведе архітектуру залучених компонентів. Крім того, архітектура має бути загальною та розширюваною, підтримуючи автоматизацію додаткових кроків у аналітичному робочому процесі моделі дослідників шляхом розширення системи додатковими сервісами.

Зовнішнє обмеження, накладене зацікавленими сторонами на архітектуру MAAS, полягає в тому, що вона повинна працювати в Arrowhead Framework як система або системи. Фреймворк забезпечує фундаментальну функціональність для ефективної підтримки розробки, розгортання та експлуатації взаємопов’язаних кооперативних систем на основі сервіс-орієнтованої архітектури (SOA).

1.2. Особливості аналітики даних та моделей

Аналіз даних — це процес виявлення, інтерпретації та передачі значущих шаблонів у даних [4]. Він містить широкий вибір процесів, інструментів і методів для вивчення даних з метою отримання корисної інформації для покращення процесу прийняття рішень бізнесом.

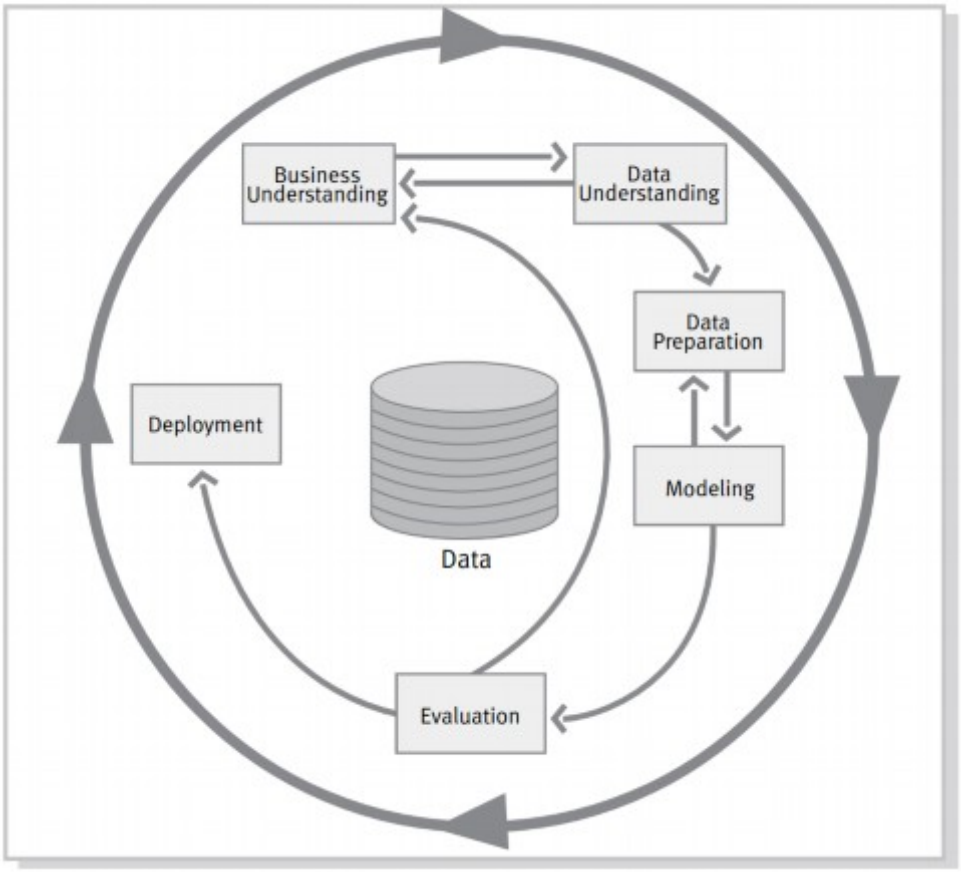

Рис. 1.1. Еталонна модель аналізу даних CRISP-DM

Рисунок 1.1 ілюструє модель аналізу даних CRISP-DM, яка є добре відомою методологією аналізу даних, яка використовується в широкому діапазоні дослідницьких областей. Збір даних можна визначити як: «процес збору та вимірювання інформації про цікаві змінні, який дає змогу відповісти на запитання дослідження, перевірити гіпотези та оцінити результати». Одним із початкових етапів процесу є розуміння даних. Протягом цього етапу дані збираються з різних джерел. Етап розуміння даних продовжується

діями, щоб отримати початкове розуміння даних. За цим етапом слідує підготовка даних, що включає кроки, необхідні для створення остаточного набору даних. Етапи включають очищення даних, перетворення та вибір атрибутів і записів, необхідних для аналізу. Отже, для проведення фактичного аналізу даних створюється модель або набір моделей з використанням кількох методів моделювання.

Прикладом процесу аналізу даних є команда дослідників, які хочуть провести аналіз зв'язків між різними сховищами на GitHub. Відповідно до моделі CRISP-DM, зображеної на рисунку 1.1, одним із початкових кроків є розуміння даних. Дослідники використовують методику збору даних, наприклад майнінг репозиторіїв, щоб зібрати необроблені дані зі сховищ програмного забезпечення. Після збору необроблених даних дослідники готують остаточний набір даних шляхом очищення, трансформації та вибору функцій, щоб зменшити кількість введених атрибутів. Нарешті, різні методи статистичного аналізу, такі як лінійна регресія, можуть бути використані для побудови моделі, яку дослідники використовують, щоб відповісти на низку запитань дослідження.

Як згадувалося раніше, аналітика даних використовує як вихідні вхідні дані, такі як текст, аудіо та відео. З іншого боку, модельна аналітика використовує моделі як вхідні дані з метою отримання корисної інформації для покращення процесу прийняття рішень бізнесом. Тому аналітику моделей можна розглядати як спеціалізацію аналітики даних, з особливим акцентом на моделюванні артефактів. Отже, загальні види діяльності, які можна знайти в аналітиці даних, такі як збір даних, підготовка, моделювання, можна застосовувати в аналітиці моделей.

На рисунку 1.2 зображено процес, використаний у [19] для збору та підготовки великих наборів моделей UML для аналізу. ЯК показано, процес складається з трьох загальних дій:

1. Збір даних. Майнер сховища використовується для видобутку GitHub для потенційних моделей UML.

2. Фільтрація та перевірка. Зібрані моделі перевіряються, якщо вони містять нотацію UML.

3. Вилучення та аналіз даних. Виконуються кроки для підготовки остаточного набору даних для проведення аналізу.

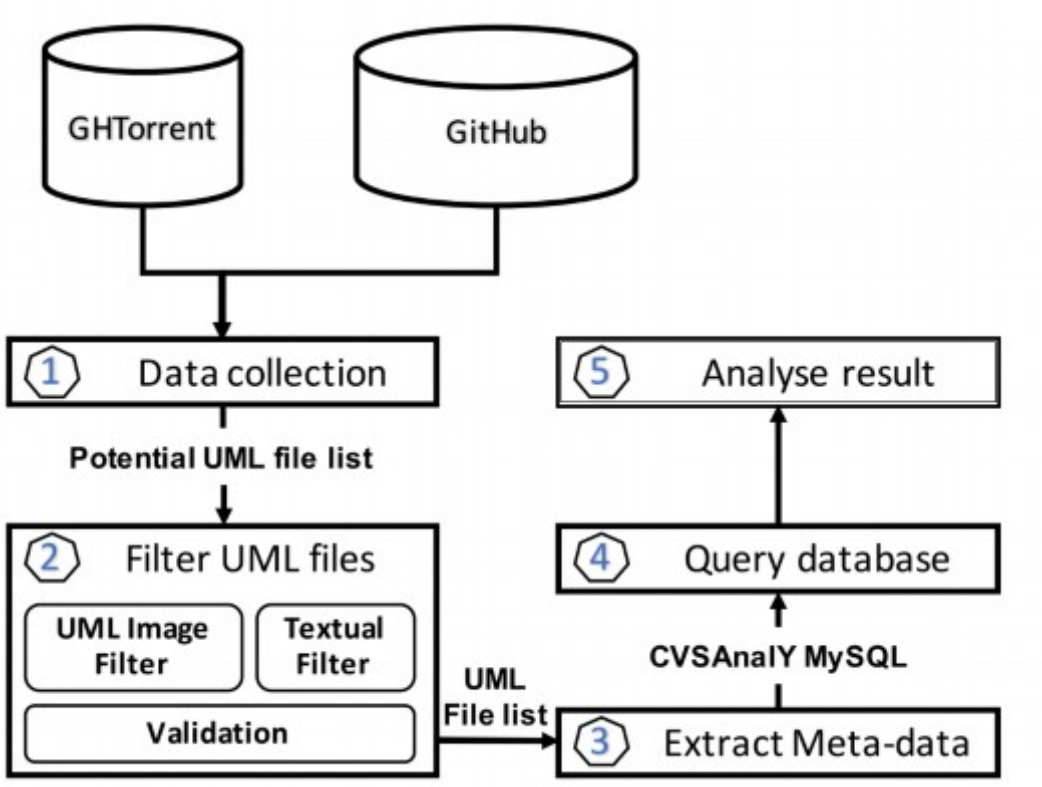

Рис. 1.2. Огляд процесу збирання та аналізу UML

Щоб проаналізувати моделі для отримання корисної інформації, ми бачимо, що загальні кроки в робочому процесі аналітики даних дійсно застосовуються в аналітиці моделей. Однак існуючі методи, які більше орієнтовані на текстові дані, такі як інтелектуальний аналіз сховищ для вихідного коду, можуть бути незастосовні безпосередньо в цьому контексті.

1.3. Особливості мов моделювання бізнес-процесів

Мова моделювання процесу може бути визначена як: «Мова, що забезпечує відповідний синтаксис і семантику для точного визначення вимог

бізнес-процесу, щоб підтримувати автоматизовану верифікацію процесу, валідацію, моделювання та автоматизацію процесу» [34]. Метою моделювання процесу є надання специфікацій, незалежних від реалізації таких специфікацій на загальному, абстрактному рівні.

Існує два переважаючих типи формалізмів моделювання процесів, а саме формалізми на основі графів і формалізми на основі правил. Мови моделювання на основі графів використовують такі концепції теорії графів, як вузли та ребра, щоб визначити моделі процесів. З іншого боку, мови моделювання на основі правил базуються на формальній логіці. У цій дипломній роботі ми зосередимо увагу на двох широко використовуваних мовах моделювання на основі графів, а саме на BPMN і EPC.

1.3.1. BPMN: модель бізнес-процесу

Нотація моделювання бізнес-процесів (BPMN) — це методологія моделювання бізнес-процесів, розроблена Ініціативою моделювання бізнес-процесів (BPMI). Перша версія мови (версія 1.0) була випущена в 2003 році. На момент написання останньою версією мови є BPMN 2.0, випущена в 2011 році. У цій версії внесено деякі помітні зміни, наприклад метамодель, яка визначає набір правил і конструкцій, необхідних для створення конкретної моделі процесу BPMN і формату обміну в XMI і XSD. Крім того, мова була стандартизована OMG [1], починаючи з версії 1.1, випущеної в 2008 році.

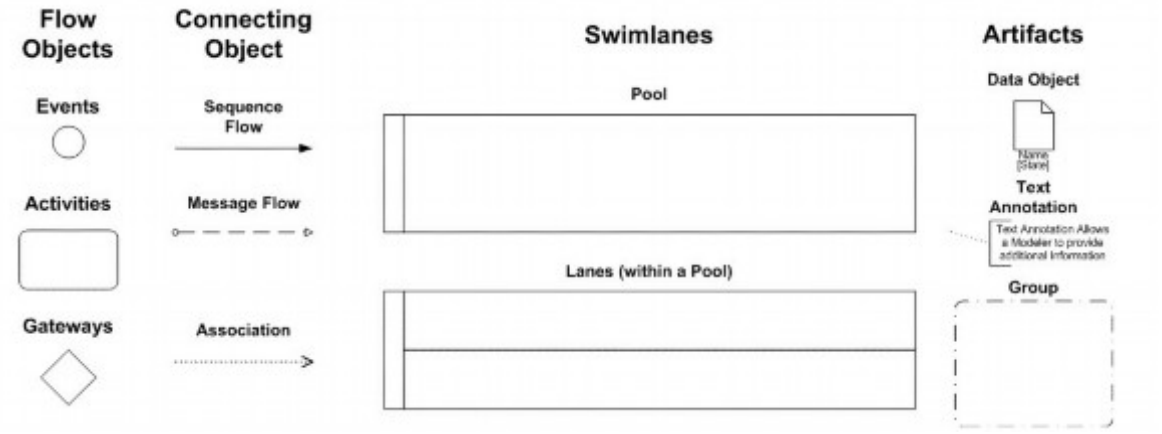

Рис. 1.3. Основні елементи BPMN

Рисунок 1.3 ілюструє основні елементи BPMN. BPMN надає п’ять основних конструкцій для моделювання, тобто об’єкти даних, об’єкти потоку, асоційовані об’єкти, доріжки для плавання та артефакти [26]. Крім того, користувачі можуть розширювати базові конструкції, проектуючи артефакти, що складаються з кількох основних елементів. Наприклад, підпроцес, який містить інший пул або смугу.

1.3.2. EPC: ланцюг процесів, керований подіями

Подійно-керований ланцюг процесів (EPC) — це мова моделювання бізнес-процесів. Основними мовними конструкціями EPC є функції, події та конектори [26]. Користувачі можуть використовувати основні конструкції для моделювання певного бізнес-процесу. Однак сумісні інструменти розширюють мову додатковими елементами, дозволяючи створювати більш виразні моделі бізнес-процесів. Мова підтримує різні формати обміну. Найпомітнішими є пропрієтарні формати VDX і AML, які використовуються Visio і ARIS, а також формат EPML з відкритим вихідним кодом і незалежний від інструментів. У цій роботі ми зосередимося на форматі EPML.

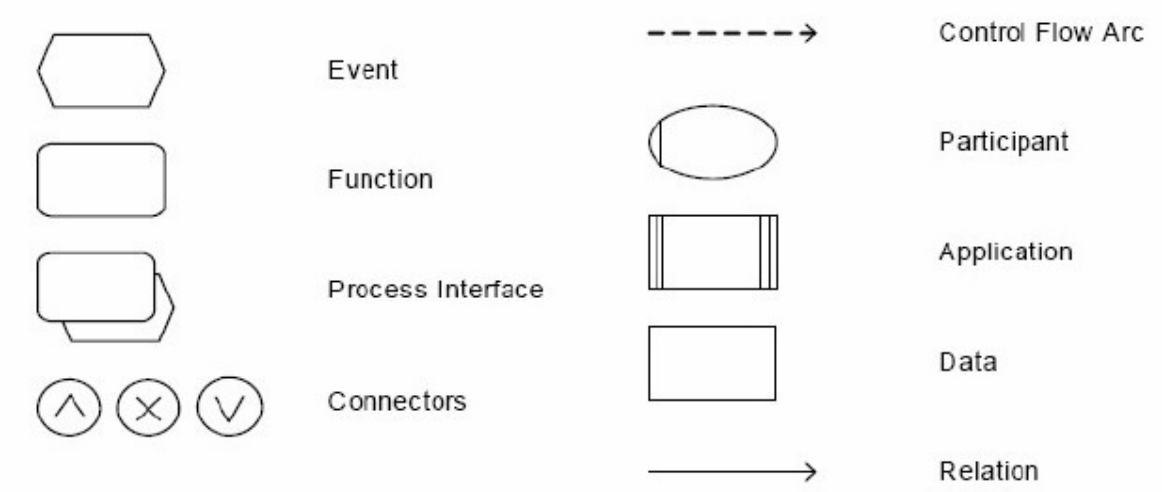

Рис. 1.4. Основні елементи EPC

1.4. Дослідження особливостей Arrowhead Framework

Arrowhead Framework — це набір програмних рішень, орієнтований на забезпечення інтероперабельності та інтеграції систем в контексті Індустрії 4.0, Інтернету речей (IoT) і кіберфізичних систем (CPS). Він розроблений у рамках європейського дослідницького проекту Arrowhead для полегшення взаємодії між різними сервісами і додатками, які використовуються на виробництві, в розумних містах, розумних мережах та інших складних екосистемах.

Основні особливості Arrowhead Framework:

1. Децентралізована архітектура: підтримує інтеграцію різних локальних систем без необхідності у централізованому керуванні.

2. Сервіс-орієнтована архітектура (SOA): кожен компонент функціонує як сервіс і може бути взаємодоповнений іншими сервісами через стандартизовані інтерфейси.

3. Інтероперабельність: дозволяє підключення і взаємодію різних систем, використовуючи стандарти та протоколи.

4. Безпека: включає механізми для забезпечення безпеки даних, автентифікації, авторизації та захисту комунікацій між компонентами.

Цей фреймворк особливо корисний для створення розумних систем у промислових середовищах, наприклад, для автоматизації виробничих процесів або управління енергетичними мережами.

Arrowhead Framework надає архітектуру, спрямовану на створення широкомасштабних систем на основі Інтернету речей (IOT) , побудованих на базових принципах Systems of Systems (SOS) з акцентом на [11]:

• Обробку даних у реальному часі для зв’язку з низькою затримкою в системах автоматизації.

• Безпеку систем автоматизації та даних.

• Розробку системи автоматизації для підвищення ефективності та гнучкості виробництва.

• Масштабованість систем автоматизації для підтримки великих систем автоматизації.

На момент написання останньою версією фреймворку проект пропонує реалізацію фреймворку на мовах програмування Java і C++. Реалізація Java базується на Spring-Boot Framework для побудови веб-служб R ESTful [3]. Крім того, проект Arrowhead Framework надає клієнтський скелетний код для багатьох мов програмування, зокрема: Java, C++, Python, NodeJS і Kalix.

1.4.1. Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (SOA) відноситься до стилю проектування програмного забезпечення, де розподілена програма або система в першу чергу будується як композиція кількох окремих служб. У Arrowhead Framework служба використовується для обміну даними між системою-постачальником і системою-споживачем. Крім того, Arrowhead Framework побудовано на основі принципів SOA [7]. Це дозволяє структурі вирішувати проблеми взаємозамінності даних між системами та пристроями. Далі ця концепція проілюстрована на рисунку 1.5. (1) Постачальник послуг реєструє свої послуги через реєстр послуг, (2) дозволяючи споживачеві послуг знаходити та використовувати надану послугу.

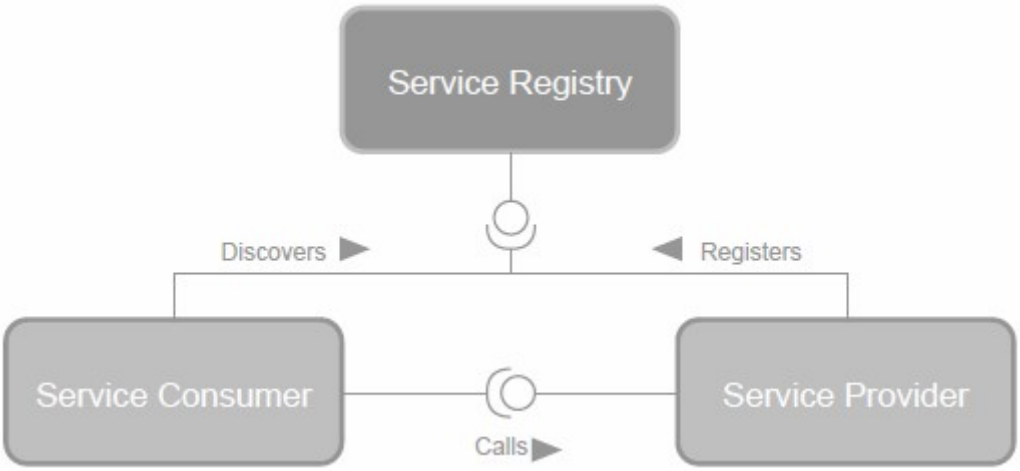

Рисунок 1.5 - Обмін послугами між постачальником і споживачем в SOA

1.4.2. Системи та пристрої

У контексті Arrowhead Framework система — це те, що має надавати або споживати послуги [7]. Крім того, система може бути як постачальником однієї або кількох послуг, так і одночасно споживачем однієї чи кількох послуг. У Arrowhead Framework існує суттєва різниця між програмними та апаратними системами. Уточнюючи, система реалізована як програмне забезпечення, що працює на апаратному забезпеченні, яке називається пристроєм. Інакше кажучи, програмна програма, відповідальна за надання або споживання послуг, називається системою, тоді як апаратне забезпечення, здатне розміщувати таку програмну програму, називається пристроєм. Крім того, пристрій може розміщувати декілька систем Arrowhead Framework.

1.4.3. Система систем (SoS)

Систему систем (SoS) можна визначити як [23] «великомасштабні інтегровані системи, які є різнорідними та незалежно діючими самостійно, але об’єднані разом для досягнення спільної мети».

У контексті Arrowhead Framework SOS складається з набору систем, які керуються основними локальними хмарними системами Arrowhead і співпрацюють для виконання певного завдання автоматизації [7]. Тому локальна хмара також є SOS у цьому контексті. Співпраця також може бути розширена до кількох локальних хмар, які обмінюються службами та інформацією між собою, ефективно надаючи рішення для більш складних завдань автоматизації.

1.4.4. Концепція локальної хмари

Локальну хмару можна розглядати як автономну мережу, яка може включати системи та пристрої для виконання певних значною мірою незалежних завдань автоматизації. Кожна локальна хмара матеріалізується шляхом включення трьох обов’язкових основних систем Arrowhead і

принаймні однієї прикладної системи. Ключовою характеристикою локальної хмари є захист від вхідного та вихідного мережевого трафіку з відкритого Інтернету. Локальна хмара забезпечує межу, спрямовану на «захист» внутрішньої частини локальної хмари, що включає системи та пристрої, від зовнішнього мережевого трафіку.

У локальній хмарі системи співпрацюють для виконання певного завдання автоматизації. Зв’язок між n пристроями в локальній хмарі також називається внутрішньохмарним зв’язком. Локальна хмара в основному характеризується як набір систем, які співпрацюють для формування однієї складної системи автоматизації. Таким чином, локальна хмара по суті стає SoS [7]. Крім того, через міжхмарний зв’язок, тобто коли дві системи знаходяться в двох різних локальних хмарах і обмінюються інформацією, це також стає SoS.

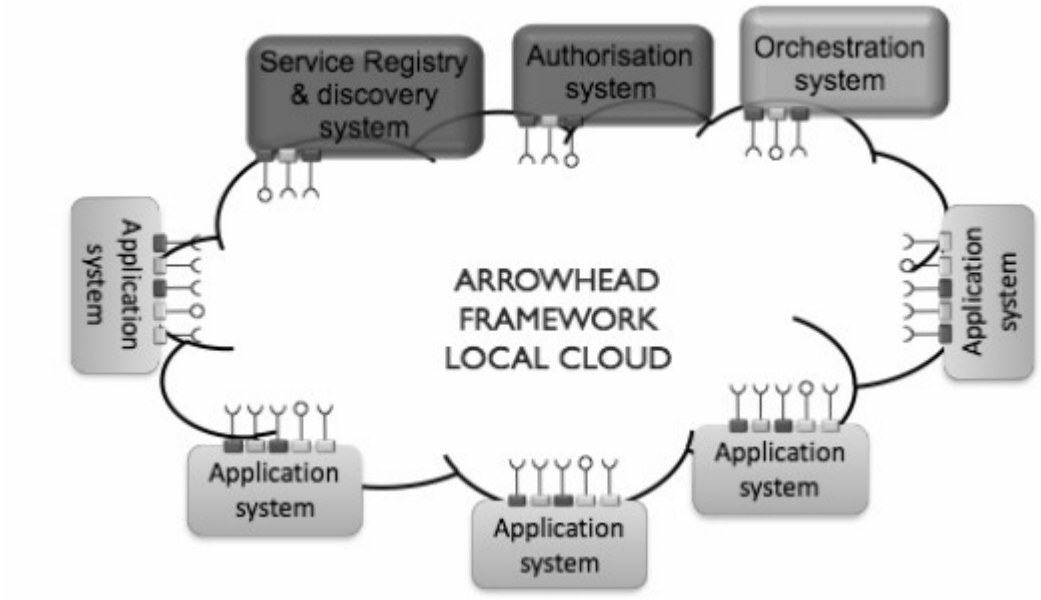

Рис. 1.6. Приклад локальної хмари

У локальній хмарі Arrowhead системи повинні використовувати кілька обов’язкових основних компонентів Arrowhead, перш ніж вони зможуть обмінюватися службами та даними [7] між собою. У випадку системи постачальника послуг вона повинна використовувати базові системи

ServiceRegistry та Authorization . У той час як система споживача послуг повинна використовувати всі три основні системи. Три основні системи локальної хмари:

• Авторизація. Компонент авторизації забезпечує автентифікацію та авторизацію споживачів послуг за набором правил. Постачальник послуг може визначити, чи може споживач послуг користуватися наданою послугою на основі цих правил.

• ServiceRegistry. Компонент ServiceRegistry має на меті забезпечити можливості зберігання для поточних активних служб, зареєстрованих у локальній хмарі. Це також робить послуги видимими для споживачів послуг.

• Оркестровка. Компонент Orchestration спрямований на можливість повторного використання існуючих служб для створення нових служб і функцій. Крім того, компонент також має на меті забезпечити розширене виявлення послуг шляхом виявлення та об’єднання постачальників і споживачів послуг.

Висновки до розділу

В даному розділі проведено аналіз досліджень щодо аналітики моделей і Arrowhead Framework, а також термінологія, пов’язана з цими областями. Наведено вступ до аналітики даних і моделей, основні компоненти мов моделювання BPMN і EPC, виконано короткий опис Arrowhead Framework, включаючи його архітектуру та основні концепції.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ОПИС ТЕХНОЛОГІЇ ПОБУДОВИ РЕПОЗИТОРІЇВ МОДЕЛЕЙ

2.1. Огляд архітектур репозиторіїв моделей

В останні роки було розроблено кілька перспективних підходів до технології репозиторію моделей. Дослідження в [15] представляють огляд сховищ моделей процесів, які різняться за методами управління моделлю та стійкістю. Дослідження в [12] показує репозиторії спільних моделей та пов'язані з ними перешкоди та проблеми. Дослідження [40] представляє огляд існуючих інструментів, які використовують репозиторії моделей як послугу. У цьому розділі розглядається існуюча технологія сховища моделей, пов'язана з MDE та BPM.

ModelBus [20]: це платформа на основі SOA, яка полегшує інтеграцію гетерогенних інструментів моделювання. Фреймворк спрямований на надання інформації про модель сервіс-орієнтованим архітектурам. Розроблено прототипну версію. Однак воно не відповідало на деякі технічні характеристики. Було розроблено нову та переглянуту версію ModelBus, яка розміщена як проект із відкритим кодом. Інструмент надає інтерфейси веб-служб, які можуть бути розширені розробником додатків, що дозволяє інтегрувати існуючі інструменти в структуру. Після завершення процесу інтеграції інструменту надані інструментом послуги стають доступними для використання іншими інструментами в рамках. Інструмент використовує шаблон взаємодії, що складається з компонентів служби моделювання, споживача моделювання та сховища. Центральним компонентом шаблону взаємодії є репозиторій моделей, що забезпечує спільний доступ до моделі між інтегрованими інструментами фреймворку.

Фреймворк постачається з вбудованим репозиторієм моделей, що забезпечує функціональні можливості для версії моделі, часткової перевірки та злиття версій моделі та фрагментів. Споживацький і сервісний компоненти

спілкуються з репозиторієм через скелет і заглушку. Інтерфейси каркаса та заглушки дозволяють передавати моделі за посиланням замість моделі. Розробники додатків також можуть реалізувати цей інтерфейс для інтеграції спеціального сховища моделей.

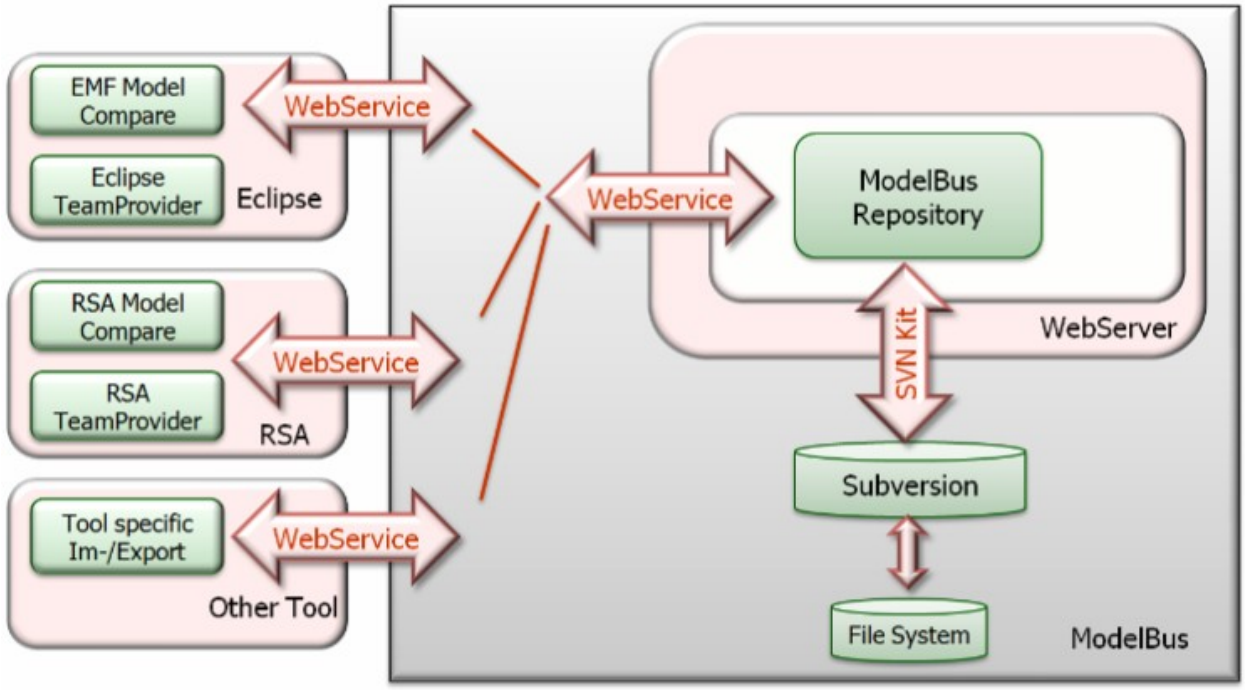

Рис. 2.1. Загальна архітектура розгортання ModelBus

CDO - Connected Data Objects [48] – це репозиторій чистої моделі Java. Він розроблений для підтримки як моделей EMF, так і метамоделей. CDO пропонує деякі цікаві функціональні можливості репозиторію, включаючи спільне моделювання, масштабованість моделі та стійкість моделі з настроюваними серверними частинами бази даних. CDO використовує архітектуру клієнт-сервер , підтримуючи клієнтські програми на основі EMF і надаючи репозиторії моделей на сервері. Компонент репозиторію складається з двох рівнів: загального рівня, який дозволяє клієнтам взаємодіяти з репозиторієм, і рівня бази даних, що дозволяє їм підключатися до різних баз даних, таких як реляційні або об’єктні бази даних. Однією з головних цілей репозиторію є забезпечення високої масштабованості. Це досягається шляхом використання відкладеного завантаження, що

супроводжується методами попередньої вибірки. Ці методи дозволяють завантажувати моделі за вимогою та кешувати моделі, що призводить до підвищення продуктивності під час доступу до великомасштабних і складних моделей.

GenMyModel [4] - це хмарне середовище моделювання, що підтримує моделі BPMN, UML, EMF, бази даних і блок-схеми. Інструмент надає веб-інтерфейс із можливостями керування моделлю. Він забезпечує функції запитів, дозволяючи користувачам шукати існуючі проекти спільноти за назвою. Також надається функція фільтрації, що дозволяє користувачам фільтрувати за типами проектів, наприклад UML або BPMN.

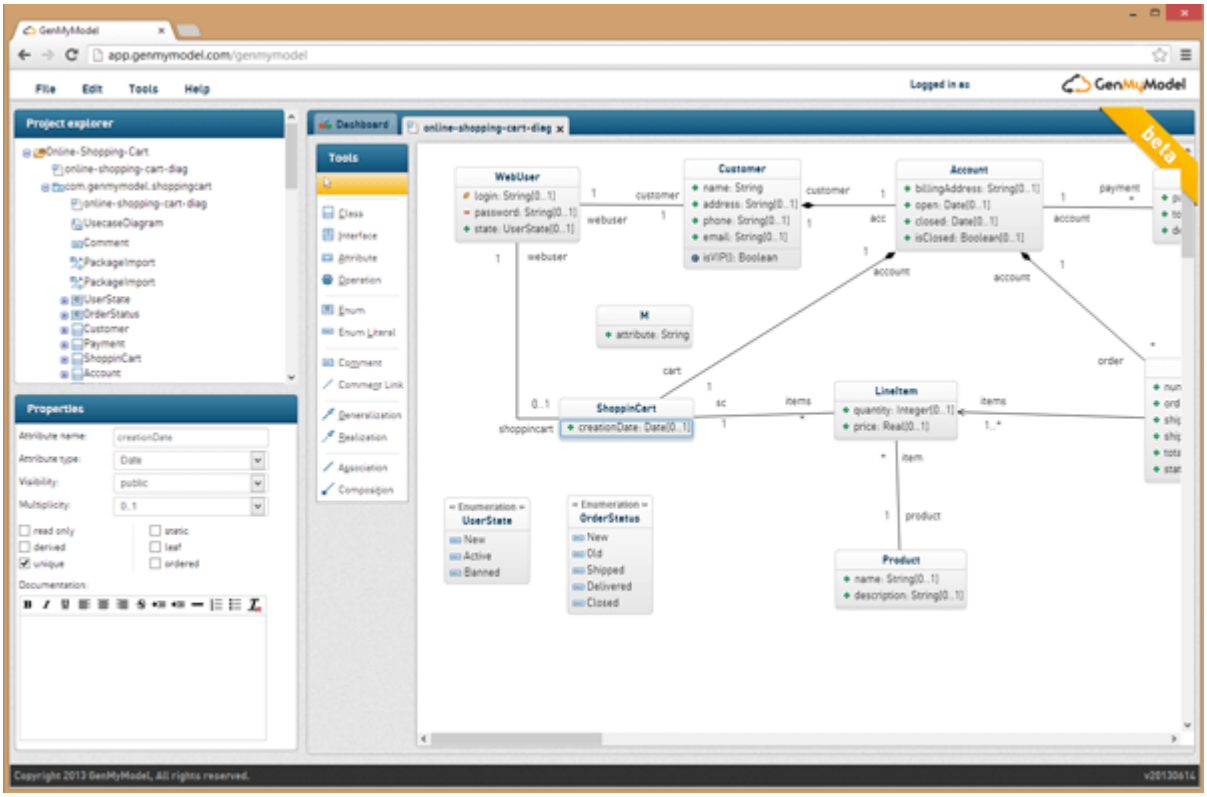

Рис. 2.2. GenMyModel - DBMS Tools

Перегляд результатів пошуку подібний до інших репозиторіїв моделей. Більш детальну інформацію про проект можна переглянути, вибравши його зі списку. Вибір проекту забезпечує візуалізацію моделі проекту. Подібно до репозиторіїв вихідного коду, також можна клонувати вибраний проект,

дозволяючи користувачам повторно використовувати моделі, створені спільнотою. Крім того, також підтримується спільне моделювання, що дозволяє користувачам працювати в реальному часі над тією самою моделлю. Крім того, за допомогою формату обміну XMI моделі, розроблені на інших платформах, можна інтегрувати в платформу. Отже, інструмент також можна розглядати як загальну платформу спільного моделювання.

MDEForge [6] – розширюване середовище моделювання. Хоча було можливо протестувати ранню версію інструменту, проект було відхилено та переведено в автономний режим. Проте все ще можна розгорнути локальний екземпляр сховища. MDEForge прагне підтримати спільноту MDE, надаючи функції для виявлення та повторного використання існуючих артефактів моделювання.

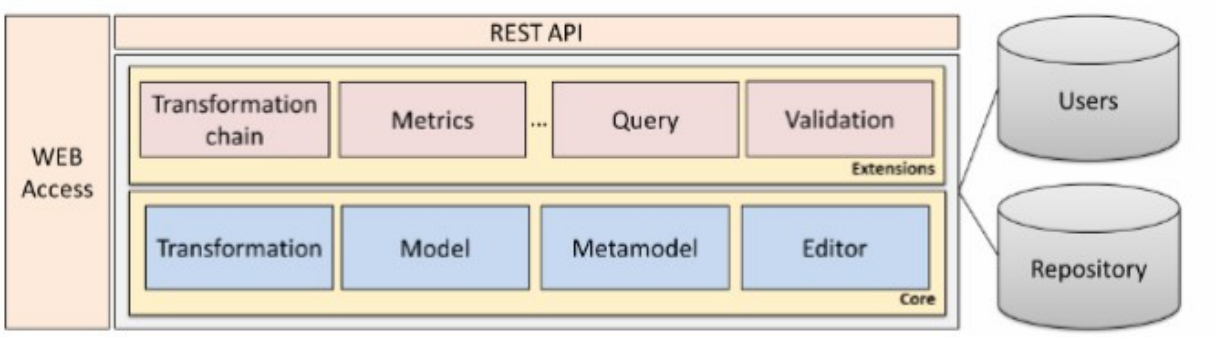

Рис. 2.3. Архітектура MDEForge

Особливістю, яка відрізняє це сховище від інших, є можливість керувати будь-яким артефактом моделювання, включаючи моделі, метамоделі, перетворення та редактори. Цей інструмент надає веб-графічний інтерфейс для взаємодії зі сховищем моделей. Він підтримує пошук і перегляд сховища вже розроблених артефактів моделювання. Репозиторій використовує трирівневу архітектуру, яка включає ядро, розширення та REST API, щоб розкрити надані служби. Основний рівень надає базові послуги CRUD для артефактів моделювання. Користувачі можуть розширити

репозиторій додатковими функціями за допомогою розширень, створених спільнотою, таких як показники моделі або розширені запити моделі.

AMOR - Adaptable Model Versioning [2] - це структура, яка намагається використовувати системи контролю версій у сфері MDE. Репозиторій моделей є частиною структури AMOR. Функції репозиторію включають виявлення конфліктів, інтелектуальне вирішення конфліктів і адаптивне керування версіями моделі. Точне виявлення конфліктів описується як уникнення раніше невиявлених конфліктів і неправильно вказаних конфліктів. Інтелектуальне вирішення конфліктів використовується для представлення суперечливих змін разом із пропозицією кроків вирішення. Існуючі методи керування версіями є або загальними, або пристосованими явно для мови моделювання. Таким чином, за допомогою адаптивного керування версіями моделі репозиторій має на меті надати користувачам можливість збалансувати загальність і конкретність.

EMFStore [27] – це репозиторій моделей, призначений для вирішення проблеми версії моделі. Репозиторій базується на Eclipse Modeling Framework і використовує архітектуру клієнт-сервер, де репозиторій розгортається на сервері.

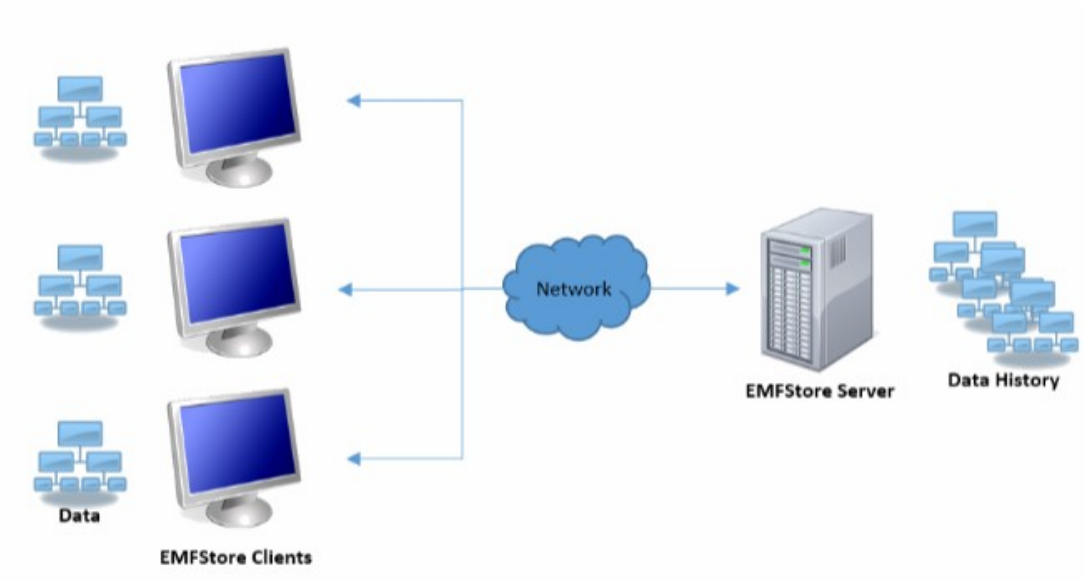

Рис. 2.4. Приклад архітектури EMFStore

Репозиторій підтримує відстеження змін на основі операцій, виявлення конфліктів і злиття. Відстеження змін на основі операцій є окремим випадком відстеження змін на основі змін. На відміну від відстеження змін на основі змін, де кожна зміна моделі записується, коли вона відбувається в репозиторії, відстеження змін на основі операцій записує операції перетворення щодо стану моделі. Виявлення конфлікту на основі операцій застосовується для виявлення конфліктних операцій на рівні атрибутів. Іншими словами, виявлення конфлікту позначає операції як конфліктні, якщо вони змінюють однакові значення того самого атрибута на інший результат. Злиття на основі операцій використовується для вирішення конфліктів.

WebGME [35] – це веб та хмарний інструмент метамоделювання, який спрямований на забезпечення масштабованості та спільного моделювання. Основним призначенням інструменту є підтримка проектування великомасштабних і складних моделей. Інструмент реалізовано за допомогою архітектури клієнт-сервер, де передній та внутрішній компоненти розгорнуті на клієнті та сервері відповідно. Внутрішня частина складається зі сховища моделей і набору веб-сервісів, що демонструють моделі через REST API, доступний клієнту. Крім того, серверну частину також можна розширити за допомогою плагінів, що дозволяє користувачам інтегрувати додаткові інструменти. Інструмент використовує методи контролю версій, а також об’єктно-орієнтовані концепції, такі як успадкування та композиція . Крім того, нові методи моделювання представлені для моделювання наскрізних проблем.

APROMORE є репозиторієм моделі бізнес-процесу. Репозиторій реалізовано як програмне забезпечення як послуга (Saa S) з відкритим кодом. Він надає користувачам широкий спектр послуг, таких як аналіз та керування моделями процесів. Архітектура має трирівневу архітектуру, що складається з рівня підприємства (презентація), базового (бізнес-логіка) і проміжного (проміжне програмне забезпечення). Репозиторій надає канонічний формат процесу, що дозволяє однаково обробляти різнорідні мови моделювання.

Таким чином, репозиторій моделей обмежений не лише моделями бізнес-процесів, що дозволяє використовувати його в інших областях.

ChronoSphere [18] – це репозиторій моделей EMF на основі графіків. Репозиторій моделей спрямований на вирішення проблеми керування великими та складними моделями, використовуючи концепції моделювання, керовані предметною областю, з техніками зберігання моделей на основі масштабованих графів і спеціальною мовою запитів до моделі . Крім того, розширені методи керування версіями моделі також надаються завдяки використанню версійного графа, що дозволяє користувачам використовувати розширені методи вирішення конфліктів і об’єднання.

Enterprise Architect [44] – це платформа моделювання для гетерогенних мов моделювання, включаючи BPMN, UML і SysML. Інструмент надає широкий спектр послуг, що підтримують процес розробки програмного забезпечення, від аналізу вимог до підтримки моделей.

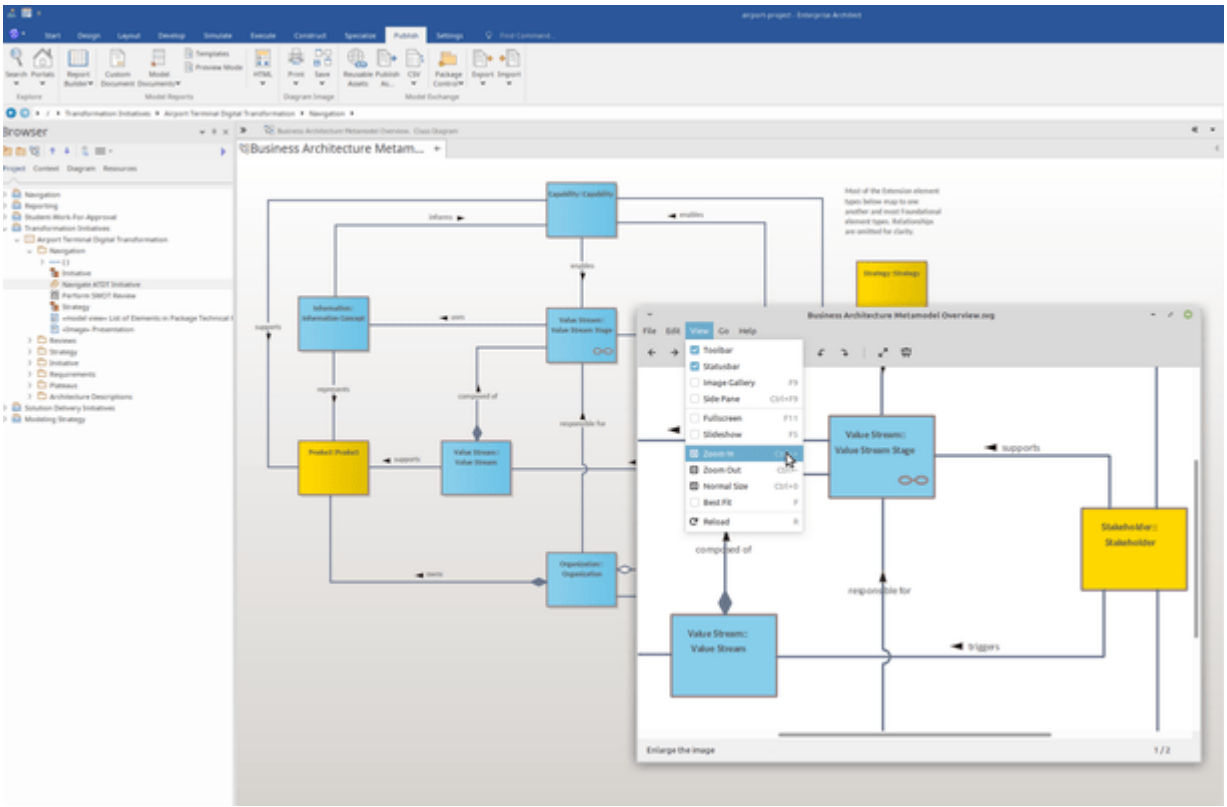

Рис. 2.4. Вигляд Enterprise Architect Professional Edition

Enterprise Architect надає настільну програму для взаємодії з репозиторієм моделей. Переглядати репозиторії та підключатися до них можна локально, на сервері або в хмарі. Інтерфейс робочого столу дозволяє користувачам запитувати моделі за атрибутами в кількох сховищах моделей. Результати запиту представлені у вигляді списку. Можна вибрати модель із результатів запиту, що дозволить отримати більш детальний перегляд вибраної моделі. Крім того, надаються функції співпраці у формі обговорень між n користувачами, а також перегляду моделей. Інструмент надає шаблони проектування для широкого діапазону діаграм UML, дозволяючи користувачам повторно використовувати існуючі моделі без необхідності починати з нуля.

Таблиця 2.1.

Порівняння технологій, що використовують репозиторії моделей

Technology	Managed Artifacts	Main Purpose	Model Persistence	MaaS	Open-source	Extensible
ModelBus [20]	Models Metamodels Transformation rules	Tool integration	RDBMS (built-in)	✓	✓	✓
CDO [48]	Models Metamodels	Scalability	RDBMS Documents Filesystem	✓	✓	✓
Enterprise Architect [44]	Models	Collaborative modeling	RDBMS	✓	✗	✓
GenMyModel [4]	Models	Collaborative modeling	RDBMS	✗	✗	✗
MDEForge [6]	Models Metamodels Transformation rules Editors	Storage	Documents	✓	✓	✓
AMOR [2]	Models Metamodels	Versioning	Filesystem	✗	✗	✓
EMFStore [27]	Models Metamodels	Versioning	Filesystem	✓	✓	✗
WebGME [35]	Models Metamodels	Scalability	RDBMS	✓	✓	✓
APROMORE [31]	Models Metamodels	Storage	RDBMS	✓	✓	✓
ChronoSphere [18]	Models Metamodels	Scalability	Graph	✓	✓	✓

2.2. Дослідження функціоналу та архітектури платформи Apromore: Advanced Process Model Repository

Apromore (Advanced Process Model Repository) - це платформа з відкритим кодом для SaaS (програмне забезпечення як послуга), що дозволяє зберігати, отримувати, перетворювати та аналізувати вміст моделей процесу.

Однією з його основних характеристик є здатність однаково обробляти моделі процесів, представлені різними мовами моделювання. Ідея цього полягає в тому, щоб перевести моделі процесів у загальний формат, який називається Canonical Process Format, який дає ряд переваг, наприклад, усуває необхідність мати різні алгоритми для кожного формату процесу.

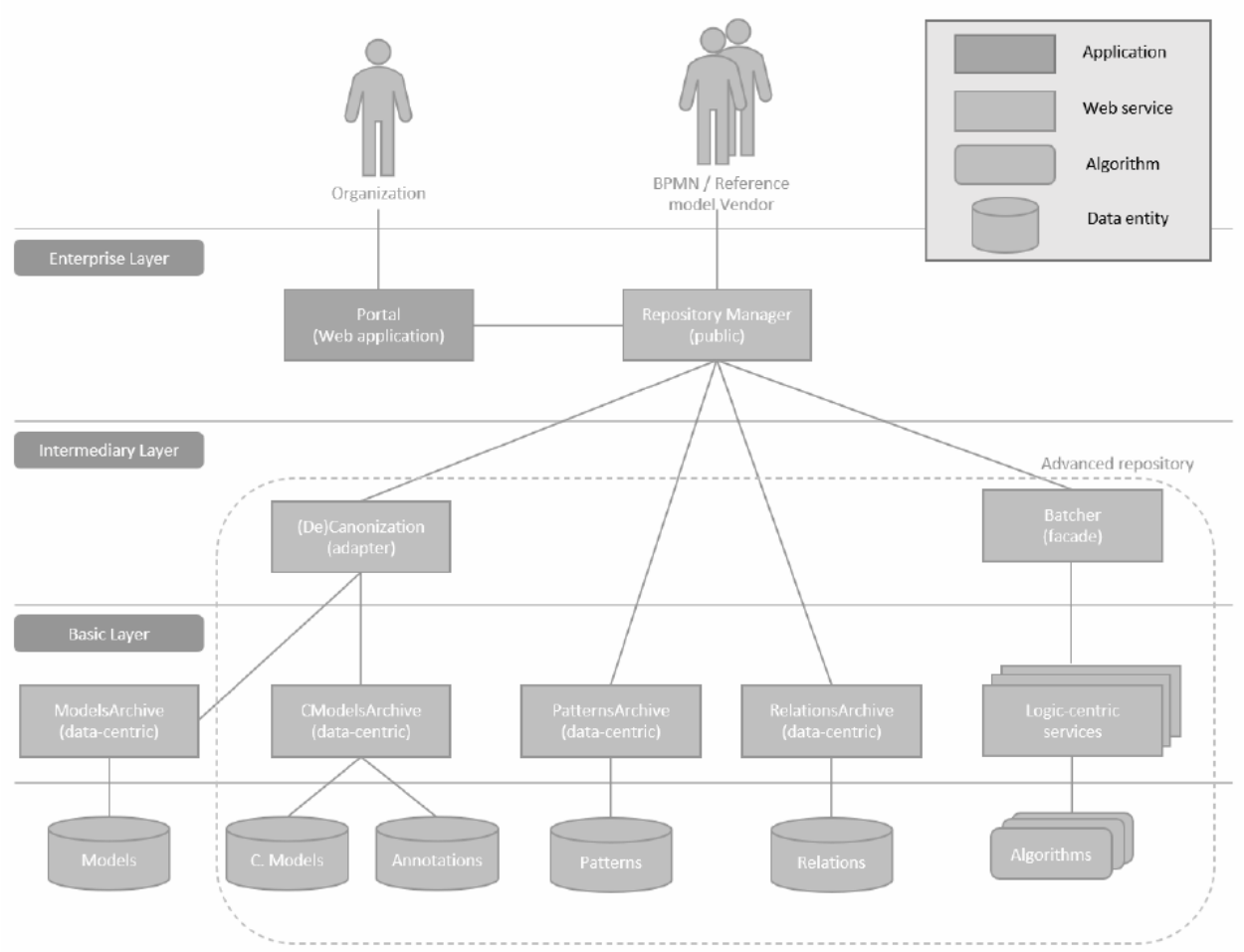

Рис. 2.5. Архітектура Apromore - сховища моделі процесу

Apromore базується на трирівневій архітектурі, що складається з таких рівнів:

• Корпоративний рівень. Корпоративний рівень діє як зовнішня частина сховища моделей. Він містить службу диспетчера репозиторію, яка забезпечує загальні функції репозиторію моделі: запити моделі, імпорт/експорт, версії моделі та безпека. Тому послугу можна розглядати як унікальну точку доступу до базових рівнів сховища моделей. Крім того,

через менеджер сховища можна отримати доступ до служб, які забезпечують аналітику моделей (подібність процесів, виявлення клонів, злиття процесів тощо) і зберігання моделей у канонічному форматі процесу.

• Проміжний шар. Проміжний рівень надає менеджеру сховища набір служб, включно з пакетом і адаптером канонізації. Служба пакетування дозволяє менеджеру сховища використовувати логіко-орієнтовані служби, надані на базовому рівні. На додаток до цього, користувачі можуть групувати набір алгоритмів за допомогою простого сценарію. Наприклад, отримати колекцію моделей, виконати виявлення клонів і візуалізувати результат. Адаптер канонізації дозволяє менеджеру сховища отримувати доступ до моделей як в їх оригінальному, так і в канонічному форматі. Крім того, передбачені можливості перетворення. Наприклад, перетворення з YAWL на BPMN.

• Базовий шар. Серцем архітектури є базовий рівень, який інкапсулює бізнес-логіку та логіку даних репозиторію моделі. Бізнес-логіка містить алгоритми, які виконують дії над великими наборами моделей процесів; типовими для цієї сутності є алгоритми зіставлення, алгоритми злиття та алгоритми індивідуалізації. Логіка даних складається з набору орієнтованих на дані служб, які забезпечують доступ до базових даних сховища. П’ять основних сутностей на рівні логіки даних:

1. Архів моделей: ця сутність містить моделі процесів у їх оригінальному форматі (тобто BPMN, EPML, YAWL тощо).

2. Архів канонічних моделей: ця сутність містить канонічний формат кожної відповідної моделі в архіві моделей.

3. Архів анотацій: для кожної моделі додаються метадані про її представлення (тобто товщина лінії, положення тощо).

4. Архів шаблонів: архів шаблонів містить набір визначень моделей або шаблонів, які користувачі можуть повторно використовувати для визначення інших моделей процесів.

5. Архів зв’язків: тут зберігаються зв’язки між моделями процесів у їх канонічному форматі або зв’язки між моделями процесів та їх розширеннями.

Канонічний формат процесу (CPF) забезпечує загальний однозначний стандарт моделей бізнес-процесів, представлених у різних нотаціях або на різних рівнях абстракцій, з наміром, щоб усі моделі процесів можна було розглядати як процеси, змодельовані тією самою мовою . Ідея полягає в тому, щоб включити лише ті структурні характеристики моделі процесу, які є загальними та повторюються в більшості мов моделювання. Оскільки CPF надає мову виключно для структурного типу, графічні аспекти моделей, такі як лінії, позиції та фігури, вважаються непотрібними та зберігаються окремо у формі анотацій. Інформація, що зберігається як анотації, використовується лише тоді, коли модель потрібно перетворити назад у вихідний формат.

Виділяють такі переваги використання CPF:

• Стандартизація: немає необхідності мати різні версії алгоритмів для різних типів моделей. Іншими словами, алгоритми, що працюють на CPF, можна використовувати для роботи з моделями, написаними різними мовами, враховуючи, що існує переклад з мови оригіналу на CPF.

• Ефективність: значно скорочується час перекладу моделі з однієї мови на іншу. Крім того, моделі можна індексувати за елементами їх канонічного формату, що покращує загальну ефективність системи запитів моделей.

• Взаємозамінність: неструктурну інформацію моделей, отриману як анотації, можна використовувати для перетворення CPF на мову оригіналу, а також на інші мови. Крім того, можна перемикатися між різними візуальними представленнями, зберігаючи ту саму структуру моделей процесу.

• Повторне використання: багаторазові шаблони бізнес-процесів також зберігаються в CPF, що дозволяє користувачам визначати нові процеси без необхідності проектувати процес з нуля.

• Гнучкість: елементи CPF визначаються механізмом успадкування, що дозволяє розглядати процеси з вищого рівня абстракції як спрямовані графи. Це дозволяє виконувати операції на різних рівнях деталізації.

Таким чином, завдяки використанню CPF усувається необхідність впровадження різних версій одного алгоритму для кожної мови моделювання. Рисунок 2.6 показує, як деякі з поширених конструкцій BPMN і EPC перекладаються в CPF.

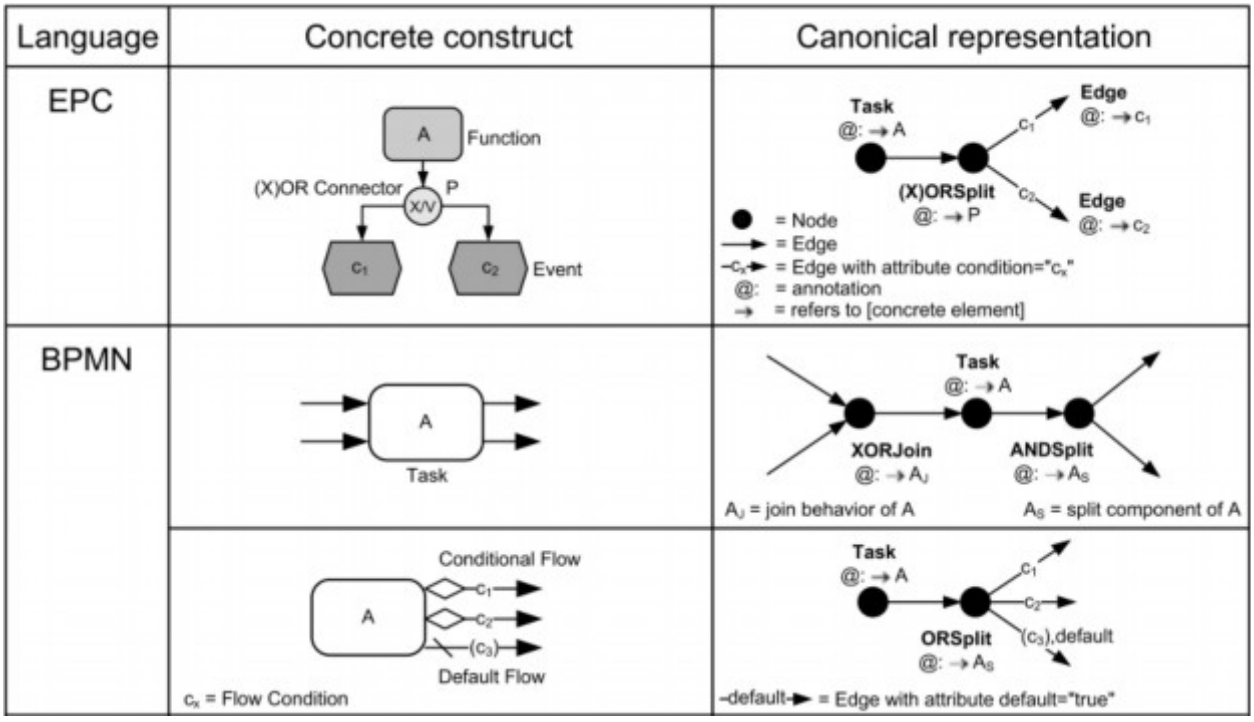

Рис. 2.6. Загальномовні конструкції в CPF

2.3. Дослідження майнінгу репозиторію програмного забезпечення

Майнінг репозиторіїв програмного забезпечення, який є систематичним способом збору, обробки та аналізу інформації про артефакти програмного забезпечення зі сховищ програмного забезпечення, набув популярності за останні роки. Видобування даних, наданих GitHub, наприклад, може виявити цікаву інформацію про проекти розробки програмного забезпечення. У той час як галузь дослідження в основному зосереджена навколо артефактів вихідного коду інтелектуального аналізу даних, інші артефакти, такі як

артефакти моделювання, також можуть скористатися тими ж методами інтелектуального аналізу даних.

Розглянемо напівавтоматичний підхід до отримання моделей UML з GitHub. Тут дотримуються системного підходу, який складається з наступних кроків: видобуток GitHub, ідентифікація моделей UML, перевірка моделей UML і вилучення метаданих. Результатом такого підходу став набір даних Lindholm, що складається з понад 93 000 загальнодоступних моделей UML. В якості основи дотримано подібного систематичного підходу до майнінгу GitHub для моделей BPMN. Рисунок 2.7 ілюструє процес майнінгу.

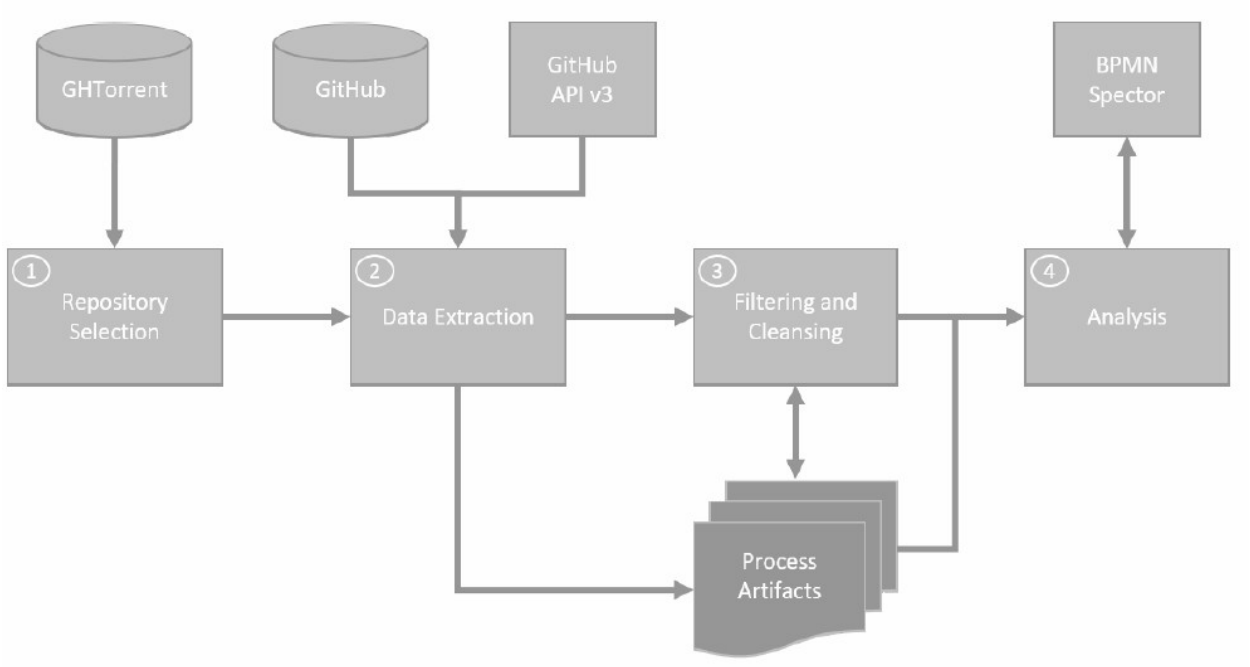

Рис. 2.7. Огляд процесу видобування BPMN

Як видно, процес майнінгу складається з наступних етапів:

1. Вибір сховища: першим кроком у процесі майнінгу є вибір сховищ для майнінгу. Для цього використовується база даних GHTorrent [17], щоб вибрати 10% репозиторіїв GitHub, які не розгалужені та не видалені.

2. Витягування даних: із пулу вибраних сховищ було надіслано запит до API GitHub для дерева файлів сховищ. Відзначається, що пропускна здатність була серйозно обмежена через використання GitHub API. Щоб

подолати цю проблему, вони використати облікові дані кількох користувачів. Використання цього підходу може скоротити час, необхідний для виконання цього кроку втричі.

3. Фільтрування та очищення: оскільки файли BPMN можуть надходити в різних форматах файлів, наприклад, як зображення або як файли XML. Вони зберегли моделі, які відповідали лише метамоделі BPMN 2.0, у результаті чого було отримано корпус із 8904 моделей BPMN.

4. Аналіз: останнім кроком у процесі видобутку є аналіз, який автори провели, щоб відповісти на низку запитань дослідження.

У [9] представляють набір даних GHS (GitHub Search), що містить дані 507 871 сховищ на GitHub, написаних 9 мовами програмування. За допомогою цього набору даних автори прагнуть отримати емпіричні дані, надаючи 25 характеристик кожного видобутого проекту. Крім того, дані кожного видобутого проекту постійно оновлюються. Автори використовують спеціальний інструмент видобутку, використовуючи GitHub Search API у поєднанні з сканером HTML-сторінок для видобутку додаткової інформації з проектів GitHub.

GitHub API Invoker використовується для збору ряду характеристик зі списку сховищ, написаних певною мовою програмування та створених або оновлених протягом певного періоду часу. Для цього використовують останню функцію, щоб сегментувати пошуковий запит GitHub на кілька часових інтервалів, щоб подолати обмеження в 1000 результатів пошуку на запит.

Веб-сканер GitHub використовується для збору додаткових характеристик сховища. Компонент аналізує сторінки HTML за допомогою селекторів CSS. Для цього завдання автори покладалися на бібліотеки jsoup і Selenium . Перший надав можливість аналізувати інформацію зі статичних HTML-сторінок. Однак через динамічний вміст кількох сторінок GitHub jsoup не вдалося отримати деяку цікаву інформацію. Тому часто використовують Selenium, щоб отримати інформацію з динамічних сторінок.

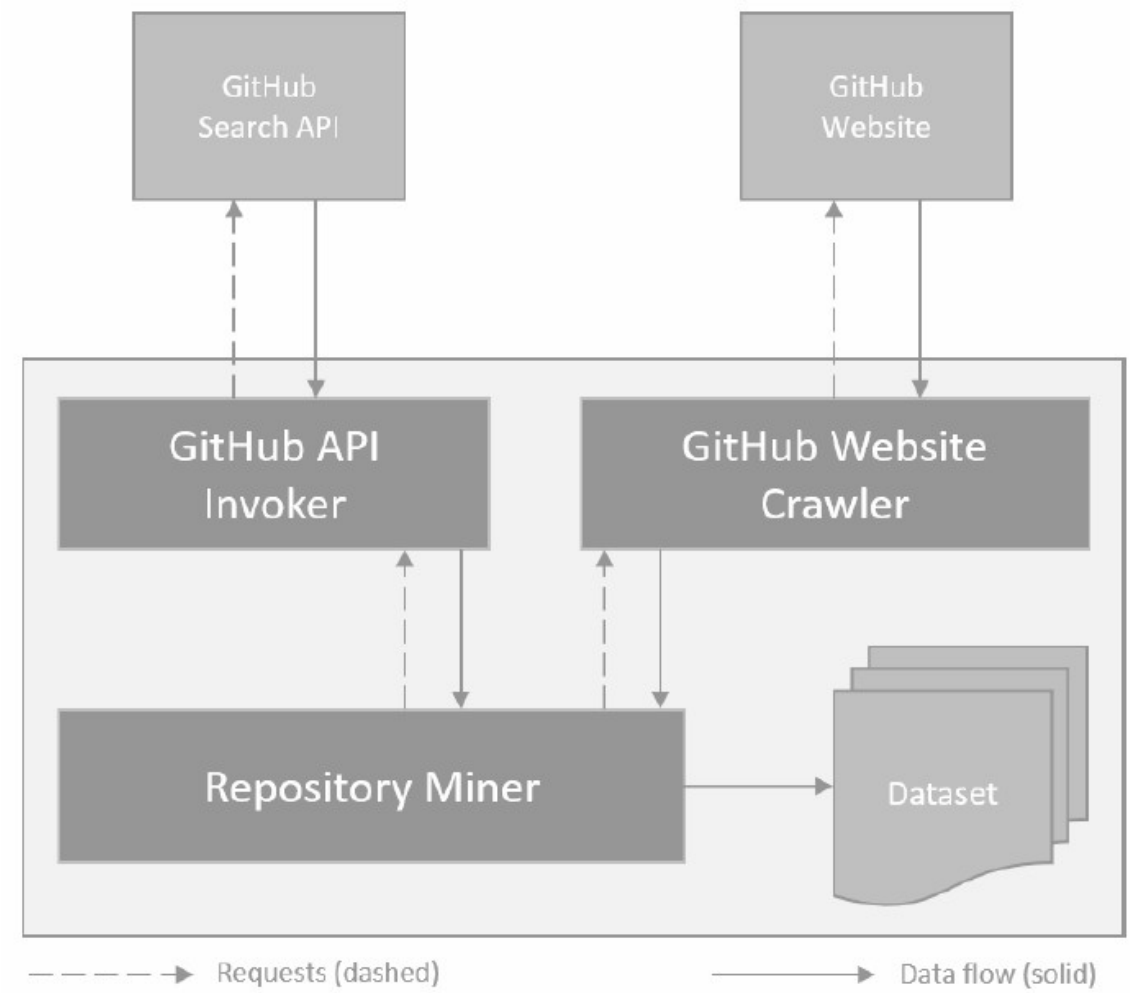

Рис. 2.8. Архітектура GHS

Repository Miner є основним компонентом інструменту GHS. Компонент можна розглядати як оркестратор, який наказує сканеру GitHub API і веб-сканеру GitHub збирати інформацію зі сховищ GitHub. Крім того, він використовує алгоритм періодичного майнінгу, який запускається кожні шість годин для оновлення зібраних даних у GHS.

2.4. Методи трансформації моделей бізнес-процесів

Моделювання бізнес-процесів стало значною частиною в галузі, головним чином для аналізу, документування та оптимізації робочих процесів. В даний час у промисловості широко використовуються моделі процесів, керованих подіями (EPC). З нинішнім зростанням нотації

моделювання бізнес-процесів (BPMN) галузь потребує значного перетворення EPC на BPMN [29].

У цьому розділі ми розглянемо останні роботи з перетворень моделі. Зокрема, перетворення EPC на BPMN і навпаки. Відомо, що EPC включає різні різнорідні формати, такі як VDX і AML, які є власними форматами, обмеженими інструментами Visio та ARIS відповідно. Тому ми обмежимо наш огляд незалежним від інструментів форматом, який називається мовою розмітки EPC (EPML).

Існує два загальних підходи до трансформації моделей процесу. Вони можуть бути трансформовані прямо або опосередковано [49]. При прямому перетворенні елементи моделі з однієї мови безпосередньо перекладаються на іншу мову. З іншого боку, непряме перетворення використовує проміжну мову для перекладу елементів моделі з однієї мови на іншу.

2.4.1. Пряме перетворення моделі

Пряме перетворення — це пряме або «один-до-одного» відображення елементів моделі, визначених однією мовою, на іншу. Наприклад, пряме відображення функціонального елемента EPC в елемент завдання BPMN. Перевага цього методу полягає в тому, що структурна або семантична інформація не втрачається, враховуючи, що існує пряме відображення для кожного елемента моделі на двох вихідних мовах. Дослідження [49] представляє прямий підхід до відображення EPC у BPMN. Однак через семантичні відмінності між двома мовами відображення один до одного було неможливим. Таким чином, лише деякі з елементів EPC, які автори називають основними елементами, безпосередньо перекладаються на BPMN.

Робота [29] розширює це питання та представляє інструмент для двостороннього перетворення між EPC та BPMN. Іншими словами, відображення від EPC до BPMN і навпаки. Автори мали намір забезпечити взаємозамінність моделі між різними інструментами моделювання , оскільки один набір інструментів моделювання для моделей EPC може мати більше

переваг, ніж набір інструментів моделювання для моделей BPMN. Подібно до [49], було неможливо відобразити кожен елемент з BPMN на EPC, і перетворення було обмежено лише основними елементами обох мов.

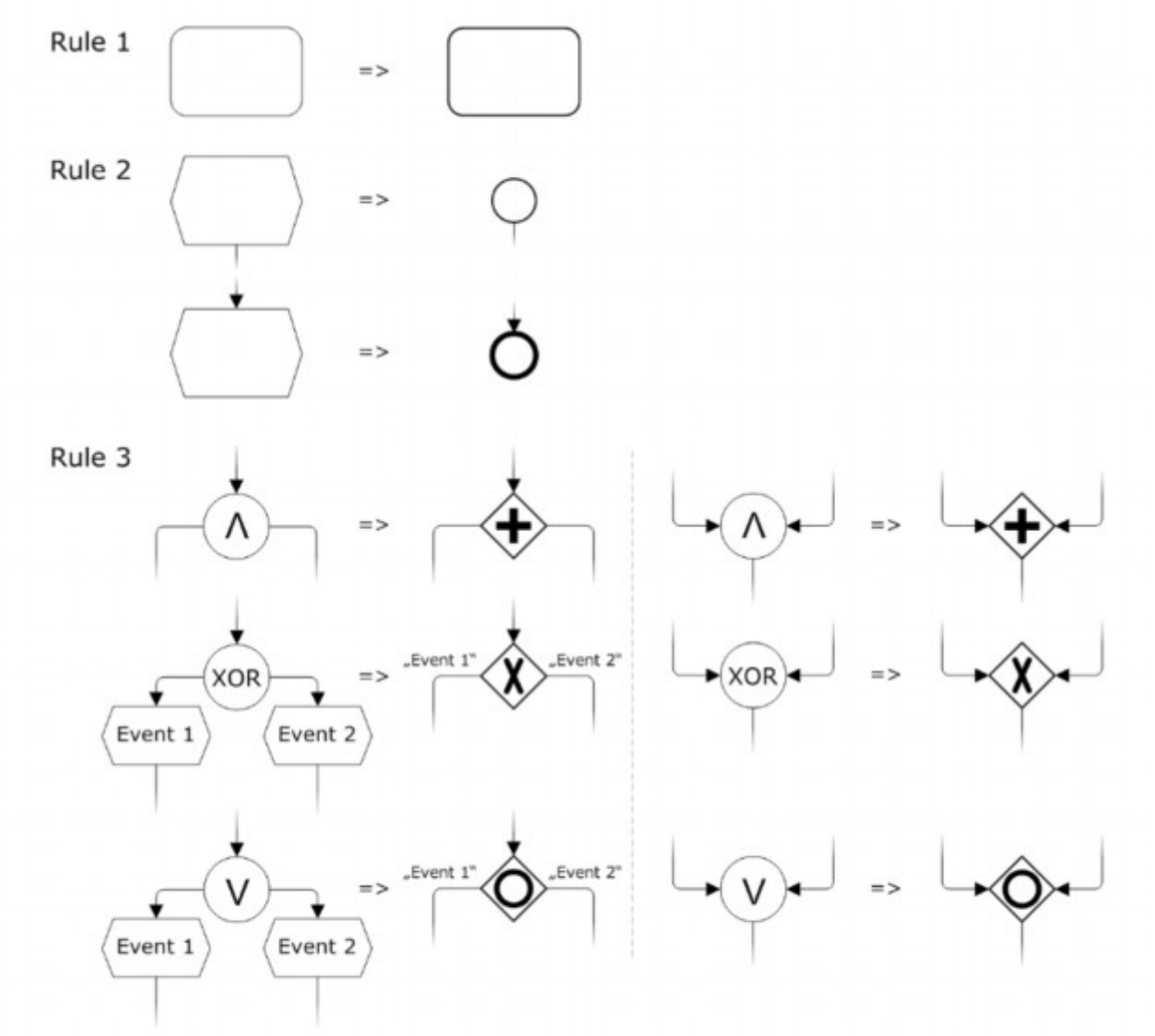

Рис. 2.9. Пряме перетворення від основного EPC до BPMN

2.4.2. Непряме (опосередковане) перетворення моделі

Непряме відображення використовує проміжну мову для перекладу EPC у BPMN. Наприклад, EPC відображається на мережі Петрі, яка, відповідно, перекладається на BPMN. Перевагою цього підходу є те, що вже існуючі картографи моделей можна використовувати для відображення двох нотацій. Однак цей метод має деякі недоліки. Зокрема, виразна сила

проміжної мови може бути нижчою порівняно з EPC та BPMN. Тому частина структурної та семантичної інформації може бути втрачена під час перетворення. У цьому контексті CPF забезпечує непряме перетворення з однієї мови на іншу. Модель, представлена на мові EPC, спочатку відображається в CPF, а потім перекладається в BPMN. Однак, як обговорювалося раніше, непрямий пінг відображення призводить до значної втрати структурної та семантичної інформації, якщо проміжна мова має нижчу виразну силу порівняно з цільовою та вихідною мовами. Щоб мінімізувати втрату інформації, виконують зберігання неструктурної інформації про моделі, таку як графічну інформацію, окремо як анотації. Інформація, що зберігається в анотаціях, пізніше поєднується зі структурною інформацією, що зберігається як CPF, для перетворення моделей на інші мови.

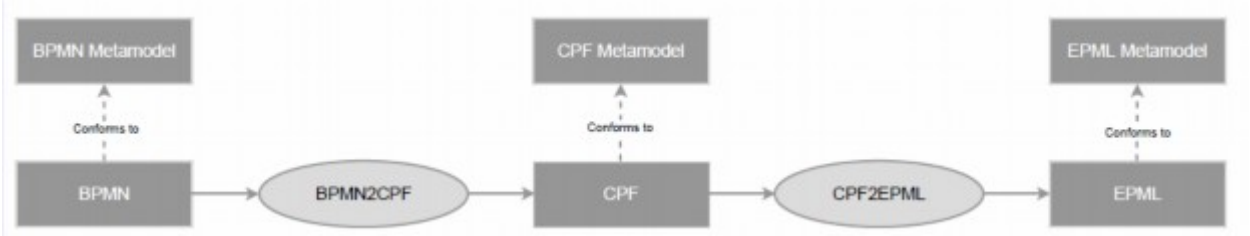

Рисунок 2.10. Приклад непрямого перетворення моделі

Рисунок 2.10 ілюструє пояснювальний ланцюг непрямих перетворень моделі. Зокрема, BPMN2CPF — це перетворення моделі, яке перетворює моделі, що відповідають метамоделі BPMN, на моделі, що підтверджують метамодель CPF. Крім того, CPF2EPML — це перетворення моделі, яке перетворює моделі, що відповідають метамоделі CPF, на моделі, що відповідають метамоделі EPML.

Висновки до розділу

В даному розділі проводиться аналіз поточних репозиторіїв моделей, щоб дізнатися, які функції вони надають. Визначено, що вони забезпечують певну форму автоматизації для покращення керування моделями. Зокрема, вони забезпечують версії моделі, реєстрацію/виписку, спільне моделювання, аналітику моделі та можливість розширення. Крім того, надається порівняння репозиторіїв моделей. Після огляду репозиторіїв моделей наведено більш детальний аналіз, на наш погляд, найбільш актуального репозиторію моделей: Apromore. Виконано аналіз великої кількості моделей і підготовка їх до аналізу, що є одним з основних кроків аналітики моделей, а також функцією, запропонованою головною зацікавленою стороною. Тому проведено огляд методів інтелектуального аналізу репозиторіїв, зокрема, сканування та сканування репозиторіїв програмного забезпечення для моделей процесу.

Наведено аналіз деяких методів перетворення моделі і визначено, що існує два методи перетворення моделі: пряме та непряме перетворення. Було зосереджено увагу на двох конкретних трансформаціях моделі: BPMN в EPML і навпаки.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ПОБУДОВИ ТА АНАЛІЗУ РЕПОЗИТОРІЇВ МОДЕЛЕЙ

3.1. Представлення основних вимог до функціональності системи побудови репозиторіїв моделей

На даний час, інструменти моделювання ще не на тому рівні, щоб надати користувачеві повну автоматизацію робочого процесу аналітики моделі. В попередньому розділі було розглянуто кілька технологій, які пропонують користувачеві деякі з цих функцій. Однак після огляду літератури ми дійшли висновку, що всі представлені функції розкидані по різних інструментах і технологіях.

Метою даної роботи є розробка та реалізація інструменту, який може інтегрувати служби, представлені в попередніх розділах. Крім того, інструмент має працювати в Arrowhead Framework як система, що складається з кількох підсистем. На рисунку 3.1 показано спрощену схему архітектури системи.

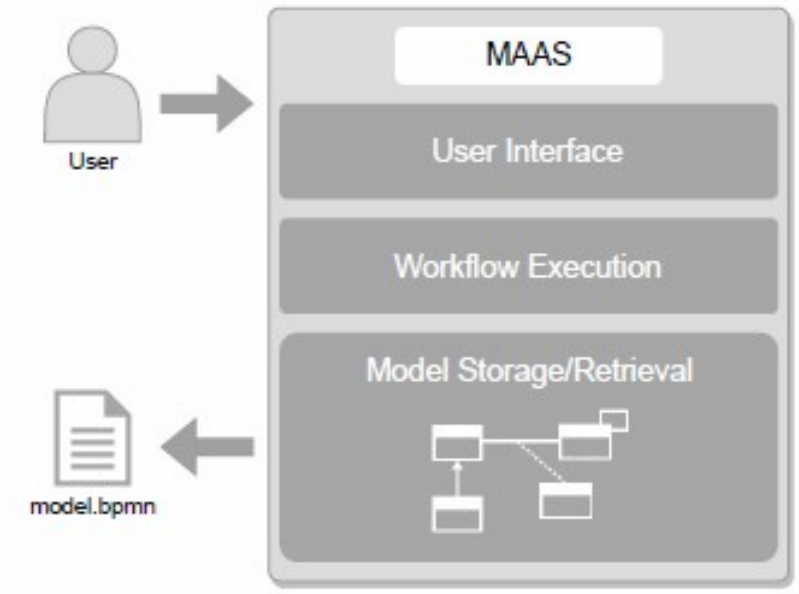

Рис. 3.1. Спрощена схема (архітектура) системи автоматизації аналізу моделей (MAAS)

Система має багаторівневу архітектуру, розподіляючи різні обов'язки між різними рівнями. Архітектура загалом складається з таких рівнів:

- Презентація (користувальницький інтерфейс) - основна відповідальність цього рівня полягає в тому, щоб перетворити результати на те, що користувачі можуть зрозуміти, і надати інтерфейс, який можна використовувати для легкої взаємодії з функціями, які надає система.

- Логіка (виконання робочого процесу) - цей рівень містить бізнес-логіку для роботи системи. Він діє як проміжний рівень, переміщуючи дані між презентаційним рівнем і рівнем даних. Цей рівень обробляє запити користувачів шляхом завантаження та збереження інформації на рівні даних.

- Дані (зберігання/вилучення моделі) - інформація зберігається та отримується з рівня даних, який підключено до певної серверної бази даних.

Ми вже розгляди і визначили, що робочий процес аналітики моделі включає різні етапи підготовки моделей до аналізу. Тому інструмент має бути загальним, щоб можна було інтегрувати нові функції шляхом розширення існуючих компонентів або додавання нових компонентів. Отже, інструмент також має дозволяти підключати інші компоненти, щоб розширити його можливості та таким чином автоматизувати більше завдань.

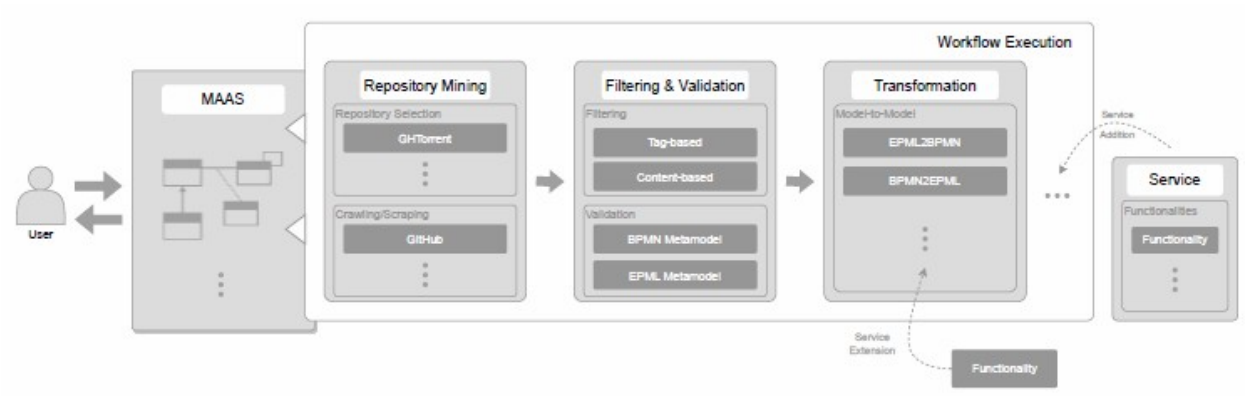

Рис. 3.2. Огляд MAAS із можливістю підключення та розширення послуг

На рисунку 3.2 представлено узагальнений огляд інструменту. Як показано, MAAS інтегрує набір служб, які можна підключити або видалити.

Крім того, існуючі послуги можна розширити новими функціями. Проілюстровано кілька конкретних основних послуг.

• Майнінг репозиторію: ця служба відповідає за сканування/збирання сховищ програмного забезпечення, таких як GitHub, для потенційних моделей процесів.

• Фільтрування та валідація : включає послуги перевірки та фільтрації для різних моделей процесу.

• Трансформація: містить функціональні можливості для виконання трансформацій моделі.

3.2. Представлення архітектури системи автоматизації аналізу моделей

Основною метою системи автоматизації аналізу моделей (MAAS), є вдосконалення робочого процесу аналізу моделей. Для того, щоб сприяти досягненню цієї мети, MAAS забезпечує автоматизацію деяких кроків у робочому процесі аналітики моделі. У цьому розділі представлено архітектуру для досягнення цієї мети. Архітектура містить різні необхідні шари для матеріалізації інструменту та взаємодії між різними компонентами на дуже абстрактному рівні.

На рисунку 3.3 показано трирівневу архітектуру MAASяка базується на наступних рівнях:

- Рівень презентації. Презентаційний рівень забезпечує інтерфейс до системи автоматизації аналітики моделі, що дозволяє користувачам взаємодіяти з базовими підсистемами через графічний інтерфейс користувача. Презентаційний рівень підключається до рівня керування репозиторієм через інтерфейс RESTful. Основні служби MAAS надаються рівнем керування репозиторієм, доступ до якого доступний через інтерфейс RESTful. Крім того, рівень керування репозиторієм діє як єдина точка входу до служб на нижніх рівнях.

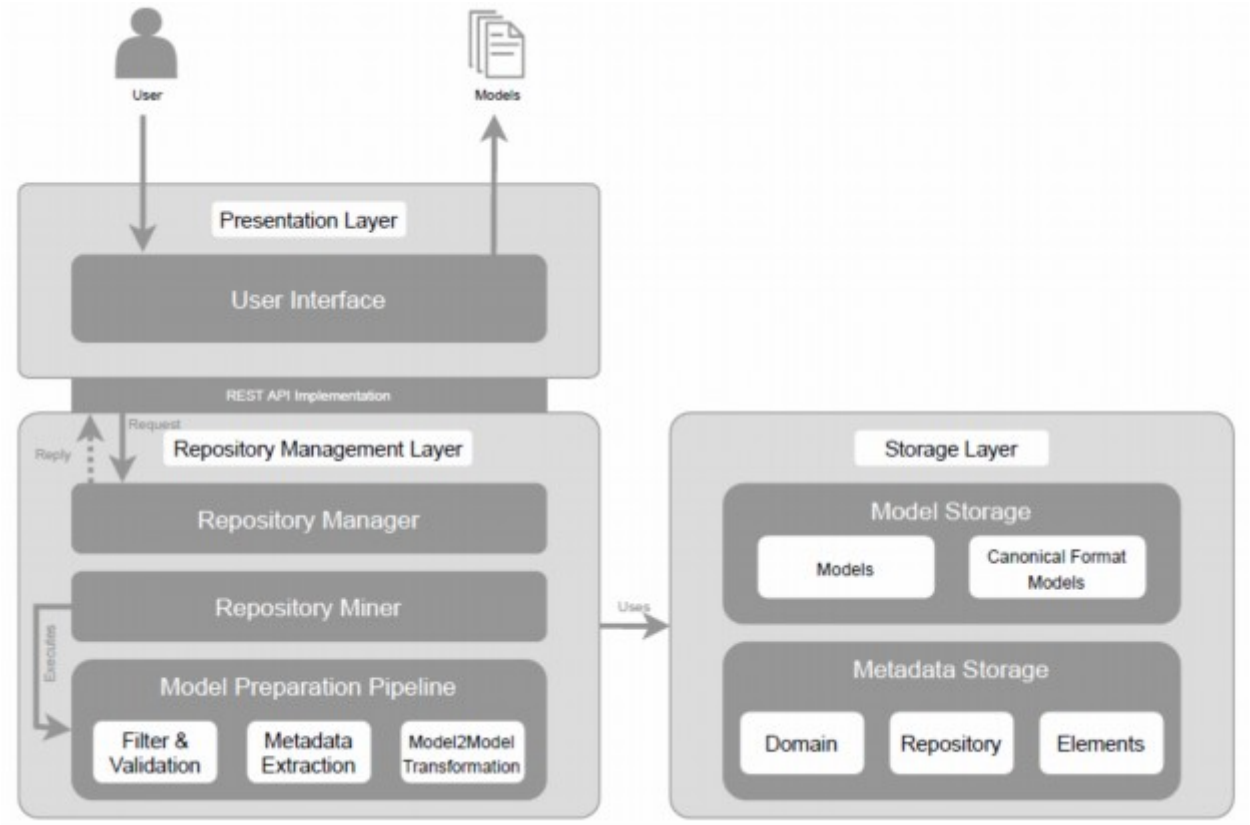

Рис. 3.3. Рівні архітектури MAAS

- Рівень керування репозиторієм. На додаток до менеджера сховища, який відповідає за функціональні можливості для зв'язку з репозиторієм моделей, рівень керування сховищем складається з двох додаткових суттєвих об'єктів, які називаються майнером сховища та конвеєром підготовки моделі. Майнер сховища відповідає за сканування та сканування сховищ програмного забезпечення для потенційних моделей процесу. Цей об'єкт виконує конвеєр підготовки моделі до того, як будь-які моделі процесів будуть збережені в репозиторії. Першим компонентом конвеєра підготовки моделі є компонент фільтрації та перевірки , який надає послуги очищення та перевірки моделі. Другим компонентом є екстрактор метаданих, який збирає додаткові метадані про моделі процесу. Останній компонент — це трансформатор model2model, що забезпечує перетворення моделі в модель. Три компоненти обмінюються інформацією з рівнем зберігання для зберігання проміжних артефактів, створених у конвеєрі підготовки моделі.

- Рівень зберігання. Цей рівень містить дані архітектури програмного забезпечення, де набір служб співпрацює, щоб працювати з даними в системі. Він зберігає моделі процесів і відповідні метадані. Моделі процесів зберігаються як в оригінальному, так і в канонічному форматі. У цьому випадку ми можемо значно підвищити ефективність перетворення, зменшивши кількість проміжних перетворень. Крім того, у більшості випадків перехід з однієї мови на іншу призводить до втрати інформації. Таким чином, наявність посилання на оригінальну модель усуває проблеми, спричинені перетворенням з одного формату на вихідний.

3.3. Використання Arrowhead Framework в системі аналізу моделей

Обов'язковою вимогою, якій має відповідати MAAS, полягає в тому, що він має працювати на основі Arrowhead Framework як система, що дозволяє іншим системам у локальній хмарі використовувати надані служби для автоматизації кроків аналітики моделі. MAAS тісно пов'язаний із концепціями SOA та SOS Arrowhead Framework.

Щоб уточнити, у нас є інструмент, що містить набір служб, які співпрацюють для автоматизації деяких завдань аналітики моделей. Крім того, служби можна розширити або додати, щоб розширити функціональні можливості інструменту. У перекладі на Arrowhead Framework цей інструмент можна розглядати як систему, що працює в локальній хмарі, що складається з різних підсистем, які співпрацюють як SOS для виконання певного завдання автоматизації.

Тому цілком розумно вважати, що архітектура локальної хмари Arrowhead розроблена для адаптації до модульної та сервіс-орієнтованої природи системи, як показано на рисунку 3.4 . Архітектура розподіляє різні обов'язки MAAS між різними підсистемами, які співпрацюють, щоб надавати послуги автоматизації.

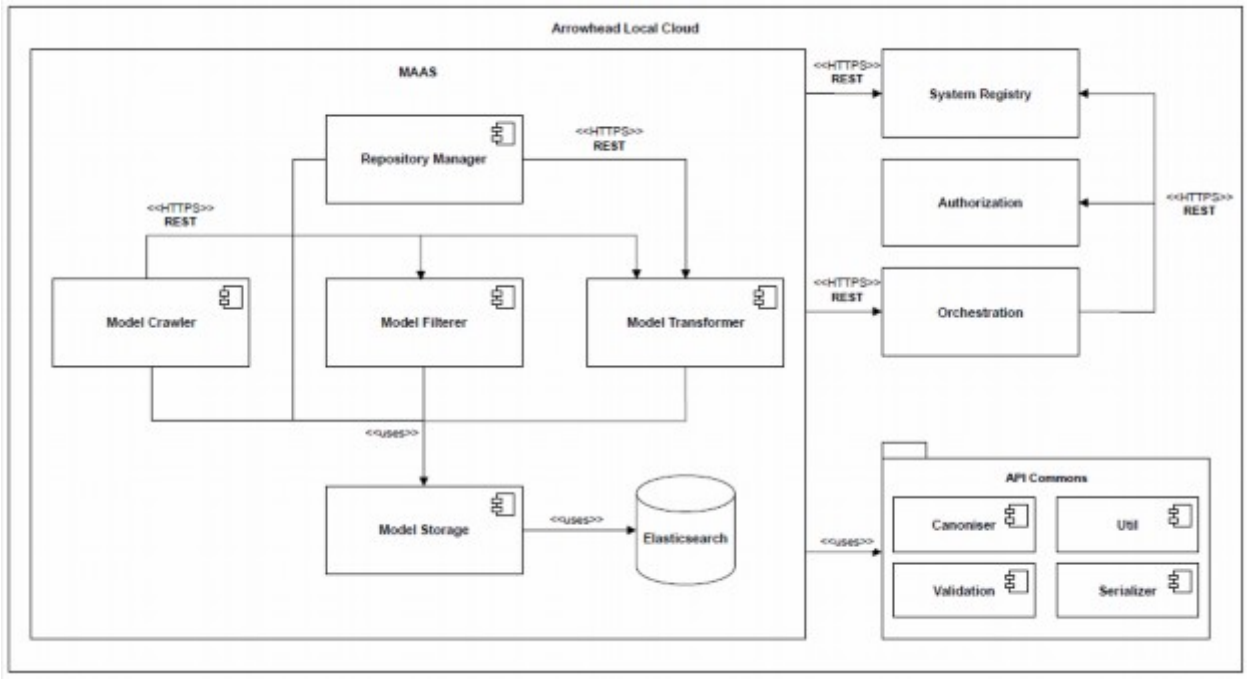

Рис. 3.4. Огляд систем і пакетів MAAS, що працюють у локальній хмарі Arrowhead.

На рисунку 3.4 стрілки позначають зв’язок використання. Крім того, пакет API Commons представляє компоненти, які є частиною MAAS і спільно використовуються різними підсистемами в MAAS.

3.4. Реалізація рівня керування репозиторієм моделей

У цьому розділі буде детально описано, як реалізуються основні служби MAAS і як рівень керування репозиторієм діє як проміжний рівень між рівнями презентації та зберігання моделі.

3.4.1. Майнер моделі сховища

Попередньо ми зазначили, що одним із основних компонентів рівнів керування сховищем є майнер сховища. Цей компонент збирає моделі зі сховищ програмного забезпечення в GitHub певної мови моделювання, указаної користувачем. У цьому розділі ми представляємо два методи сканування моделей з GitHub. Перший метод використовує GHTorrent для

вибору репозиторіїв-кандидатів. Другий метод використовує пошук GitHub для запиту моделей процесів у GitHub. У решті цього розділу ми докладно розглянемо реалізацію компонентів сканера та проектні рішення.

GitHub надає загальнодоступний REST API для отримання списку доступних репозиторіїв із їхніми метаданими. Однак API накладає певні обмеження на кількість запитів для певного періоду часу. Для неавтентифікованих користувачів це обмеження становить 60 запитів на годину, а для автентифікованих користувачів кількість запитів на годину становить 5000. API може серйозно заважати сканерам, які використовують API GitHub, і зазвичай вони реалізуються за допомогою великого набору автентифікованих користувачів облікові дані [21].

Щоб вирішити цю проблему, пропонована модель сканера використовує не GitHub API для отримання інформації про репозиторії, а підхід аналізу сторінки HTML за допомогою селекторів CSS. Для цього завдання сканер використовує бібліотеку jsoup [24]. Однак через динамічну природу сторінок GitHub інформація про деякі з необхідних елементів, необхідних сканеру моделі, не завжди доступна, оскільки для завантаження решти елементів сторінки GitHub потрібен деякий час, а jsoup не може цього зробити, оскільки це в основному статичний парсер HTML-сторінок. Тому ми було використано бібліотеку HtmlUnit [22], яка дозволяє аналізувати динамічні сторінки. Однак використання HtmlUnit має значний недолік продуктивності. Щоб зменшити недолік продуктивності, ми використовуємо HtmlUnit лише тоді, коли jsoup не працює.

Підхід аналізу сторінки HTML за допомогою селекторів CSS може створити деякі проблеми в найближчому майбутньому, коли GitHub вирішить суттєво змінити свій інтерфейс користувача. Якщо це так, наш компонент майнінгу репозиторію може потребувати майбутнього оновлення. ЩОБ зменшити витрати на обслуговування нашого сканера, ми дотримувалися тієї ж процедури, що й у [9], зробивши селектори CSS максимально загальними. Крім того, ми узагальнили нашу реалізацію

сканера та зробили деякі ключові компоненти модульними, які будуть детальніше описані пізніше.

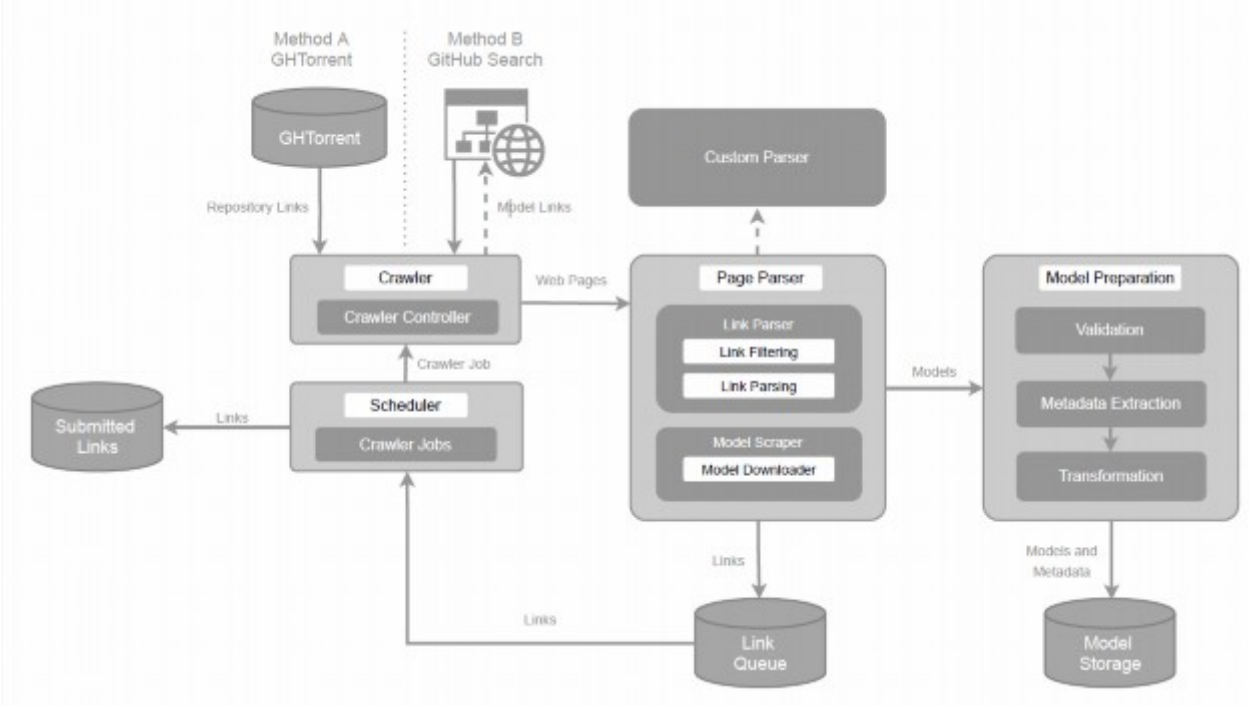

Рис. 3.5. Архітектура майнера сховища із зображенням двох методів сканування

Щоб підвищити ефективність збору моделі та метаданих, ми використовуємо багатопотоковість, що дозволяє сканеру завантажувати та аналізувати кілька сторінок HTML паралельно. Архітектура багатопотокового сканера зображена на рисунку 3.5. Як показано, загальна архітектура складається з таких ключових сутностей:

• Crawler є центральним блоком майнера сховища. Він відповідає за ініціювання кількох завдань сканування, відкритих API. Веб-сканер підтримує два методи сканування, які обговорюються далі в цьому розділі.

• Парсер сторінок відповідає за розбір HTML-сторінок і вилучення вихідних посилань і потенційних моделей процесів. Посилання фільтруються на основі набору критеріїв (наприклад, посилання, що закінчуються на «.bpmn/.epml») і надсилаються до черги посилань для подальшої обробки сканером. Скрапер моделі використовується для вилучення вмісту

потенційної моделі процесу. Нарешті, вміст потенційних моделей процесів передається в конвеєр підготовки моделей. Компонент є загальним, що дозволяє конкретні спеціалізації синтаксичного аналізу (наприклад, парсер для сховища програмного забезпечення BitBucket).

• Планувальник розподіляє робоче навантаження між кількома ядрами/потоками ЦП, отримуючи першу URL-адресу в черзі та плануючи її для завдання сканування, надісланого сканеру . Щоб подолати проблему циклічного сканування, компонент відстежує подані посилання під час процесу сканування.

• Компонент Model Preparation Pipeline використовується для «підготовки» моделей до аналізу. Він включає в себе етапи перевірки, вилучення метаданих і перетворення. Під час етапу перевірки моделі процесів перевіряються на їх метамоделі (тобто BPMN 2.0 і EPML 2.0). Далі витягуються метадані моделей процесу, наприклад елементи формату канонічного процесу. Нарешті, Transformation використовується для перекладу моделей у формат канонічного процесу. Отже, в результаті утворюється два формати моделей процесу: вихідний формат і канонічний формат процесу.

3.4.2. Методи сканування моделі

Перший метод заснований на підході видобутку корисних копалин, розглянутому в [21]. Однак HtmlUnit спричинює серйозне зниження продуктивності. Тому ми реалізували евристичний метод, який призвів до того, що сканер збирав моделі за розумний проміжок часу.

Метод GHTorrent

Даний метод видобутку репозиторію побудований на основі підходу, розглянутого у [21]. Підхід складається з двох кроків, для першого кроку яких потрібна ручна робота, тоді як система повністю автоматизує останній крок:

1) вибір зразка доступних сховищ GitHub,

2) пошук і вилучення моделей BPMN/EPML, а також метаданих їх сховища:

1. Вибір сховища. Першим кроком у нашому підході є збір списку доступних репозиторіїв із GitHub. Подібно до [21], використовується база даних GHTorrent [17], щоб завантажити найновіший дамп бази даних. Потім ми отримали таблицю проектів, щоб вибрати випадкову вибірку з 10% репозиторіїв, які не розгалужені та не видалені, що призвело до набору з 7 435 722 сховищ.

2. Вилучення моделі та метаданих. Після етапу вибору сховища ми використовували сканер для перевірки сховищ на наявність будь-яких файлів BPMN/EPML, «надаючи» йому список із 7 435 722 URL-АДРЕС до сховищ. Для кожної URL-адреси сканер використовує HtmlUnit для аналізу динамічного вмісту HTML-сторінок, а потім jsoup для отримання URL-адрес папок у «дереві файлів» сховища за допомогою селекторів CSS. Потім витягнуті URL-адреси буде поставлено в чергу для подальшої обробки сканером. Сканер використовує пошук у глибину, щоб пройти по дереву файлів сховища та знайти будь-які потенційні моделі BPMN/EPML. Як тільки він знаходить потенційну модель процесу (тобто URL-адресу, яка закінчується на .bpmn), сканер витягує модель і перевіряє її на відповідність метамоделі. Нарешті, якщо модель відповідає своїй метамоделі, вона зберігається в базі даних разом із метаданими та канонічним форматом процесу.

Як описано вище, HtmlUnit накладає серйозне вузьке місце на цей метод. Виходячи з наших спостережень щодо менших вибірок репозиторіїв (тобто 100, 1000), лише цей метод займе біля 1500 днів, а тому було обрано евристичний метод, який є на порядки швидшим.

Метод пошуку GitHub

Щоб збільшити пропускну здатність нашого модельного сканера, ми виконали модель за допомогою евристичного методу. Цей метод використовує функцію пошуку GitHub для запиту до сховищ файлів із

закінченням «.bpmn» або «.epml». Наприклад, якщо нам потрібен список файлів BPMN, запускається такий запит:

```
https://github.com/search?q=extension:bpmn
```

Результатом цього запиту є HTML-сторінка, що містить щонайбільше 10 «звернень» пошукового запиту з можливістю «перейти на сторінку» до наступної сторінки для отримання додаткових результатів. Для кожного елемента списку ми вставляємо URL-адресу в чергу. Крім того, URL-адреса наступної сторінки також додається до черги. Подібно до [9], щоб подолати 1000 результатів пошуку на запит, які нав’язує GitHub, ми сегментуємо наш запит на кілька підзапитів. Зокрема, ми сегментуємо наші запити за кількома інтервалами розміру файлу. Іншими словами, якщо нам потрібен список файлів BPMN із заданим інтервалом розміру (тобто від 0 до 250 кілобайт), ми використовуємо такий пошуковий запит:

```
https://github.com/search?q=extension:bpmn+size:0..250
```

Якщо запит повертає понад 1000 файлів для заданого інтервалу розміру, ми ділимо інтервал навпіл, і нові інтервали розміру вставляються в чергу, яка пізніше обробляється сканером. В іншому випадку, якщо інтервал призводить до 1000 або менше результатів, сканер повторює отриманий список, як зазвичай. Пошуковий робот витягує модель і метадані сховища для кожної потенційної моделі процесу (тобто URL-адресу, що закінчується на «.bpmn/.epml»).

Впровадження моделі Crawler

Діаграма класів служби сканування моделі продемонстрована на рисунку 3.6. Точка входу в службу здійснюється через ModelCmvjlService, яка надає набір функцій сканування, наданих сканером.

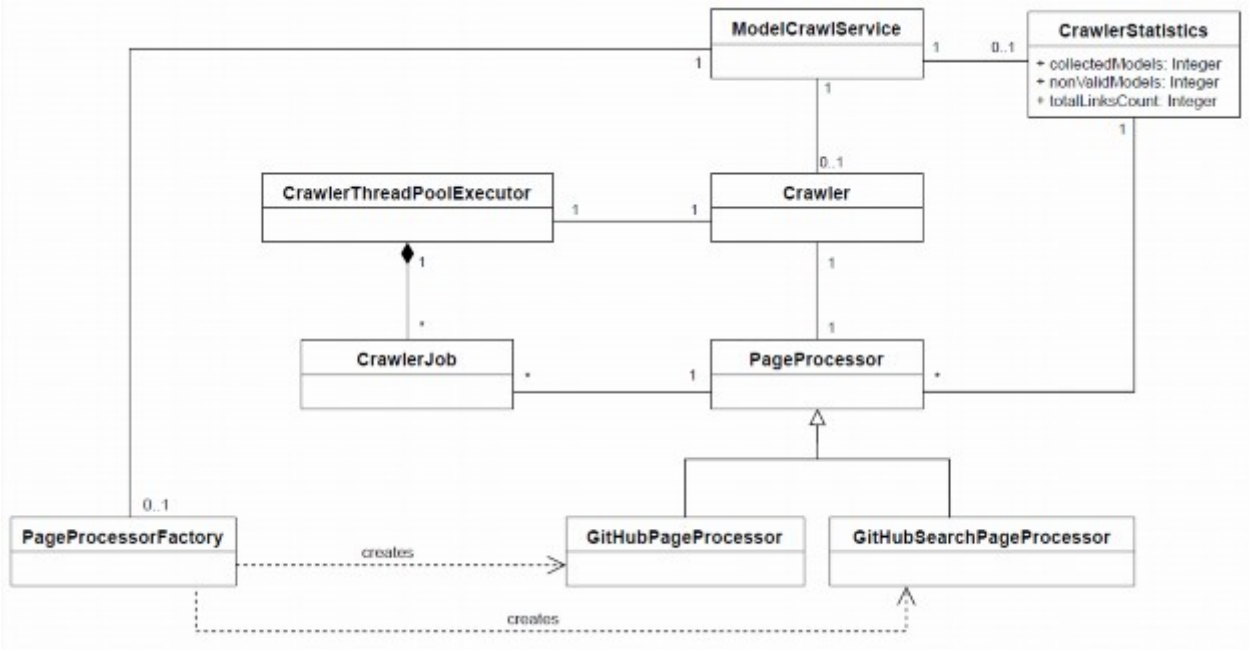

Рис. 3.6. Діаграма класів служби сканування моделі

Клас ModeACrawlService надає функціональні можливості для запуску та зупинки сканера. Крім того, він обробляє створення класу Crawler . Крім того, клас відповідає за введення PageProcessor у клас Crawler .

Клас Crawler відповідає за сканування заданого списку URL-АДРЕС. Клас використовує введений екземпляр PageProcessor для вилучення вихідних посилань зі сторінки HTML. Крім того, він відповідає за створення CrawlerThreadPoolExecutor.

CrawlerThreadPoolExecutor є підкласом ThreadPoolExecutor, який надається бібліотекою Java. Клас діє як планувальник служби сканування, який відповідає за планування завдань у класі CrawlerJob .

PageProcessor є базовим класом для GitHubPageProcessor і GitHubSearch - PageProcessor , які відповідають за методи GHTorrent і GitHub Search відповідно. Клас PageProcessor можна розглядати як компонент Parser служби сканування.

Клас CrawlerStatistics містить базову статистику сканера (тобто кількість зібраних моделей, кількість недійсних моделей і кількість зібраних посилань на сторінки).

3.4.3. Моделі фільтрації

В попередніх розділах ми представили підсистеми, які охоплюють систему автоматизації аналітики моделі. Однією з цих підсистем є система фільтрації моделей, яка надає набір послуг для фільтрації моделей на основі наданих користувачем критеріїв пошуку. У цьому розділі описано, як слід налаштувати систему фільтрації моделі та як її слід реалізувати в Arrowhead Framework.

Система фільтрації моделей надає користувачам такі типи фільтрації моделей:

- Загальна фільтрація.
- Фільтрація елементів моделі.
- Фільтрація сховища.

Загальну фільтрацію можна застосувати для вибору моделей, які містять певний рядок у своїй назві (наприклад, «доставка» поверне всі моделі, що містять цей рядок), або мають певне розширення (наприклад, «.bpmn» або «.epml»).

Моделі також можна відфільтрувати за елементами моделі канонічного формату. Користувач може вказати предикат, за яким слідує ціле число, щоб відфільтрувати елементи моделі (наприклад, «моделі, що мають ¬більше п’яти елементів типу події»). Користувач може вибирати з наступних типів предикатів: «більше», «менше», «не дорівнює» та «дорівнює». Крім того, передбачена можливість «ланцюжкових» умов. При цьому користувач може ініціювати умови фільтрації наступного вигляду:

“numberOfEdges > 5 AND numberOf Tasks < 3 AND numberOfORJoins = 1”

За допомогою фільтрації репозиторію моделі також можна фільтрувати за їхніми метаданими репозиторію (наприклад, кількість зірочок або

кількість розгалужень). Подібно до фільтрації за елементами моделі, фільтрацію репозиторію можна використовувати за допомогою предикатів.

Щоб визначити умови фільтрації, нам потрібні два компоненти в системі фільтрації моделей, щоб запустити фільтрацію моделей:

• Коли система отримує запит на фільтрування, компонент аналізу повинен проаналізувати та витягти умови із запиту.

• Після завершення синтаксичного аналізу умов компонент перекладу запиту повинен зіставити умови в запит до бази даних.

Для того, щоб отримати моделі на основі набору умов, вищевказані компоненти повинні бути реалізовані. Проблема в тому, що запити на фільтрацію є динамічними. Користувачі можуть ініціювати запит на фільтрацію на основі довільного набору предикатів і атрибутів моделі. Крім того, умови можуть бути об’єднані в ланцюжок, і вони можуть мати будь-який довільний розмір. Отже, кодування вручну всіх можливих комбінацій умов фільтрації є неможливим для цієї роботи та не є правильним дизайнерським рішенням.

Щоб подолати цю проблему, ми застосували шаблон проектування Builder [16]. Шаблон конструктора класифікується як шаблон створення, оскільки він описує, як класи мають бути створені. Патерн дозволяє покроково будувати об’єкти, які вважаються придатними для динамічної побудови умов фільтрації.

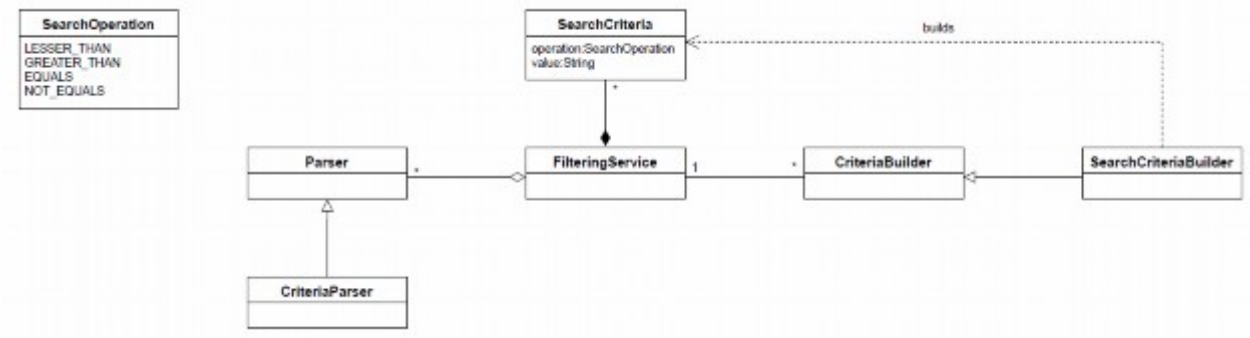

Рис. 3.7. Діаграма класів служби фільтрації

Оскільки алгоритми аналізу мають тенденцію бути складними загалом, бажано тримати їх закритими або повністю незалежними від загальної системи. В [16] пропонується ізолювати складні алгоритми, інкапсулюючи їх в об’єкти, що призводить до легшої взаємозамінності між різними алгоритмами під час виконання. Таким чином, щоб зробити алгоритми взаємозамінними під час виконання, визначаються загальний і конкретний клас аналізу.

3.4.4. Перевірка та валідація моделей

Очікується, що буде отримано широкий спектр моделей процесів під час процесу видобутку репозиторію, включаючи моделі з однаковими форматами файлів, змодельовані в різних версіях мови моделювання (тобто BPMN 1.0 і BPMN 2.0). Але більшість моделей процесів визначають стандартний формат обміну на основі XML, надаючи інструменти для імпорту та експорту моделей за допомогою цього формату [21]. Ця робота зосереджена на форматах BPMN і EPML, а моделі процесів цих форматів перевіряються відповідно на метамоделі BPMN 2.0 і EPML 2.0.

Щоб перевірити моделі процесів на відповідність їхнім метамоделям, потрібно вирішити наступну проблему: враховуючи, що запити на перевірку виникають, як ми створюємо правильний тип засобу перевірки моделі після його запуску.

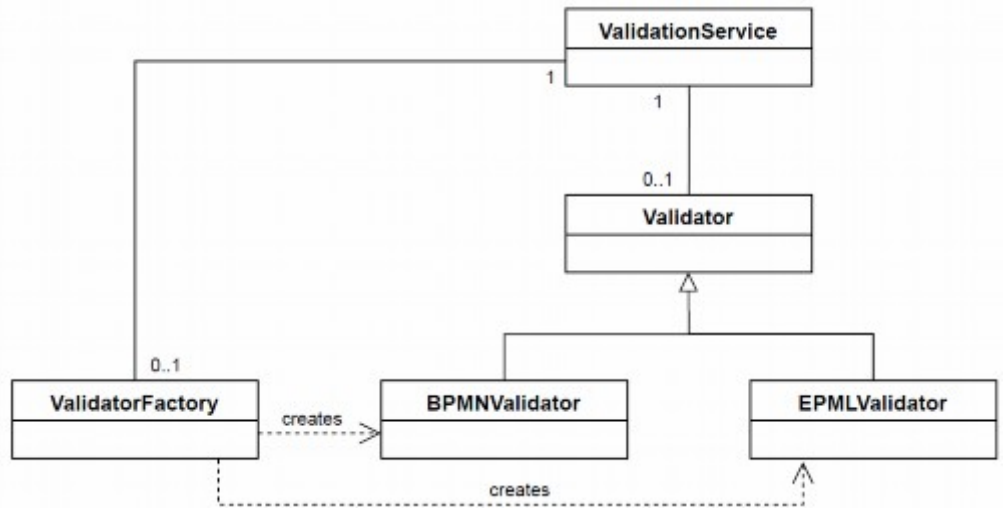

Рис. 3.8. Діаграма класів служб перевірки

Щоб створити правильний засіб перевірки моделі, заданий типом моделі, було використано шаблон проектування Factory Method [16] який забезпечує ідеальну структуру для цієї проблеми проектування. Цей шаблон абстрагує процес створення, так що конкретний тип об’єкта може бути визначений під час виконання, враховуючи параметри, передані фабричному методу. На рисунку 3.8 показано класи компонента валідатора.

Метою служби трансформації є перетворення моделі процесу з її вихідної метамоделі на цільову метамодель. Розглядаючи, як реалізувати необхідні служби перетворення, ми перевірили кодову базу Apromore і виявили, що існує набір плагінів, які надають функціональні можливості, необхідні для перетворення моделей процесів на цільову мову і ці плагіни називають плагінами канонізатора.

Зважаючи на це, було б доцільніше повторно використовувати функціональні можливості існуючих плагінів canoniser, а не розробляти власну службу трансформації моделі процесу, яка виходить за межі цієї тези. Однак, хоча плагіни надали деякі корисні функціональні можливості для трансформації моделей, вони не являли собою повністю готове рішення. Плагіни canoniser надали низку методів через свої інтерфейси. Щоб перетворити модель з вихідної мови на цільову, потрібні численні виклики методів у певній послідовності.

Щоб вирішити цю проблему, ми зрозуміли, що шаблон фасаду [16] добре підходить для цієї конкретної ситуації. Метою цього шаблону є створення спрощеного представлення більш складної підсистеми. Це представлення також можна описати як «обгортку» для функціональних можливостей складної підсистеми під фасадом.

Щоб перетворити BPMN на модель EPML, потрібно виконати кілька викликів методів у плагінах canoniser. Відомо, що інкапсуляція викликів методів разом в одному місці та надання єдиної точки доступу спростить процес трансформації, а не виконання серії викликів методів безпосередньо на самих плагінах канонізатора.

Щоб реалізувати шаблон фасаду, ми створили новий клас під назвою TransformationFacade, який обгорнув функціональні можливості плагінів canoniser для виконання трансформацій.

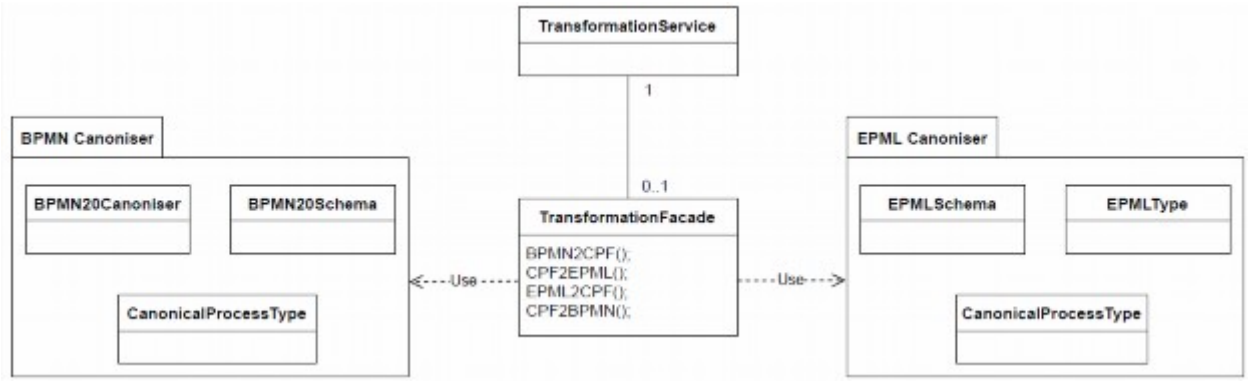

Рис. 3.9. Діаграма класів служби перетворення.

Як показано на рисунку 3.9, TransformationFacade надає кілька методів для ініціювання конкретного перетворення. Коли один із методів викликається, вони виконують численні виклики методів базових плагінів canoniser, щоб виконати дії, необхідні для трансформації моделей. Отже, складна послідовність викликів методів повністю прихована від клієнтів, які використовують фасад, яким у даному випадку є TransformationService.

3.5. Реалізація та опис рівня зберігання моделей

Рівень зберігання забезпечує можливості збереження для моделей процесів і пов’язаних метаданих. Моделі процесів зберігаються як у вихідному, так і в канонічному форматі процесу з причин, викладених у попередньому розділі. Ми зацікавлені в зберіганні великих обсягів моделей і додаткових метаданих. Хоча існують добре відомі традиційні рішення RDBMS, такі як MySQL і PostgresSQL, які легко налаштувати та запитувати. Вони погано підходять для аналітики великих даних, вимагаючи підтримки великих обсягів даних і адаптації до характеристик великих даних у реальному часі [28]. За наявності великих обсягів даних RDBMS вимагає

створення численних індексів, що накладає деякі значні недоліки в продуктивності під час процесу оновлення даних.

Беручи до уваги вищезазначені проблеми, ми пропонуємо підхід до зберігання на основі документів із використанням Elasticsearch як серверної бази даних, пошукової системи з відкритим вихідним кодом, яка розроблена таким чином, щоб бути масштабованою, розподільною та здатною працювати в реальному часі [14].

3.5.1. Система Elasticsearch

Elasticsearch — це пошукова система з відкритим кодом, яка забезпечує повнотекстовий пошук. Він розроблений таким чином, щоб бути масштабованим, розповсюджуваним і працювати в режимі реального часу. Запущений екземпляр Elasticsearch називається вузлом , і два або більше вузлів можуть утворювати кластер Elasticsearch [28]. Хоча RDBMS і ElasticSearch можуть відрізнятися різними способами, багато основних Концепції традиційної RDBMS аналогічні концепціям у світі Elasticsearch, як показано на рисунку 3.10.

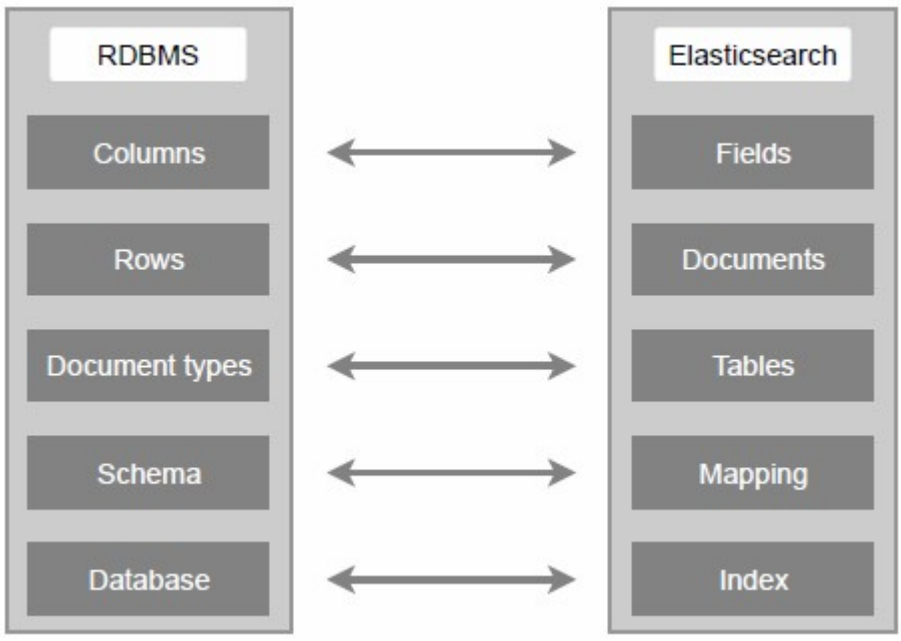

Рис. 3.10. Реалізація концепцій RDBMS засобами Elasticsearch

Поле схоже на стовпець у RDBMS: воно зберігає значення певного типу даних, однак поле може зберігати декілька значень, по суті стаючи списком, тоді як стовпець обмежений лише одним значенням. Аналогічним рядку таблиці RDBMS є документ в Elasticsearch. Документ, по суті, є об’єктом JSON в Elasticsearch. Він може містити кілька полів, подібних до рядків , які можуть містити кілька стовпців. Тип документа аналогічний таблиці RDBMS , оскільки він визначає поля, які можна вказати для певного документа. Відображення схожі на визначення схем у базах даних SQL. Вони визначають усі типи документів в індексі. Нарешті, індекс схожий на базу даних у RDBMS: забезпечує зберігання, пошук і оновлення для різних типів документів.

3.5.2. Моделі та метадані

Подібно до схеми в RDBMS, відображення визначає поля та типи даних, які містяться в індексі. Відображення складається з двох визначень індексу: індекс для моделей та індекс для доменів. Індекс моделі складається з таких полів:

• id: первинний ключ (PK) таблиці моделі, призначений базою даних,

• name: назва моделі,

• file_name: ім'я файлу моделі,

• description: довгий або короткий опис моделі,

• version: номер версії,

• modeling_language: назва файлу моделі,

• path: вихідний шлях до моделі,

• model: вміст моделі,

• c_model: канонічний формат моделі.

• elements: метадані про канонічні елементи.

• repository: метадані про оригінальний репозиторій.

• domains: метадані домену.

Індекс домену складається з таких полів:

• id: первинний ключ (PK) домену, призначений базою даних.

• name: назва домену.

• tags: список тегів домену.

На рисунку 3.11 зображено концептуальну схему рівня зберігання. Репозиторій може містити кілька моделей і доменів. Модель складається з оригінального вмісту файлу та формату канонічного типу процесу. Він складається з різних версій і може містити кілька доменів. Версія — це спеціалізація моделі. Домен складається з кількох тегів і може посилатися на нуль або кілька моделей.

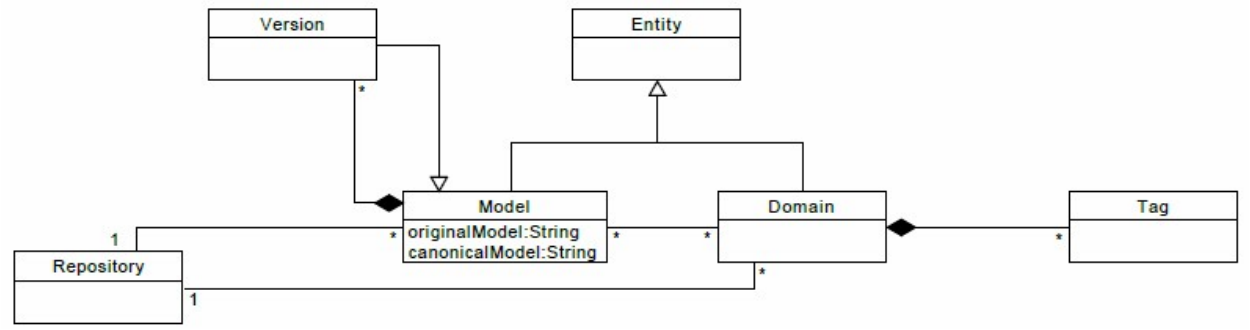

Рис. 3.11. Концептуальна схема рівня зберігання

3.6. Реалізація рівня представлення моделей

Презентаційний рівень надає користувачеві можливість взаємодії з MAAS. Він представляє графічний інтерфейс користувача (GUI), що дозволяє користувачам виконувати команди та запити в базовій системі. Графічний інтерфейс зв’язується з рівнем керування репозиторієм через REST API. Отже, два способи взаємодії з MAAS:

1) через GUI,

2) безпосередньо через REST API.

Реалізація рівня презентації досить проста. Нам потрібно лише розкрити функціональні можливості, надані репозиторієм моделей. Іншими словами, нам потрібно лише підключити рівень до REST API. Реалізація презентаційного рівня використовує простий веб-інтерфейс, оскільки

реалізація графічного інтерфейсу користувача, знайденого в реальних репозиторіях моделей, не є метою цієї роботи, і для її реалізації потрібно багато часу. Отже, графічний інтерфейс системи репозиторію моделей є базовим, який відображає лише мінімальні компоненти інтерфейсу користувача, необхідні для виконання дій CRUD та інших команд у системі. Взаємодія з системою здійснюється за допомогою GUI або безпосередньо через REST API. Користувач може взаємодіяти з такими аспектами системи:

API надає служби для ініціювання або завершення вже запущеного завдання сканування. ЩОБ розпочати роботу сканування, користувачеві пропонується набір параметрів. По-перше, користувач може вибрати метод сканування, детальніше описаний у попередньому розділі. Далі користувачеві пропонується вибрати формат моделі для сканування (наприклад, BPMN або EPML). Після встановлення необхідних параметрів користувач може почати сканування. Нижче наведено список функцій, доступних за допомогою API сканування:

• StartCrawler: ця функція повідомляє системі сканера розпочати видобуток GitHub для моделей процесів у вказаному форматі. Він підтримує методи GHTorrent і GitHub Search. Він також наказує системі підготувати моделі для аналізу. Для прикладу інтелектуального аналізу BPMN функція StartCrawler дає вказівки зібраним моделям пройти конвеєр підготовки моделі, звертаючись до систем фільтрації та трансформації в локальній хмарі.

• StopCrawler: ця функція дає команду системі завершити всі поточні завдання Crawler .

• Status: використовується для отримання останніх оновлень щодо кількості зібраних моделей, посилань і того, чи неактивний сканер.

Фільтрування моделі

Інша служба, доступна через API, — це служба фільтрації моделей. Веб-інтерфейс сервісу фільтрації моделі представлено на рисунку 3.12. Як показано, користувачі можуть ініціювати запит на фільтрацію на основі різних типів критеріїв.

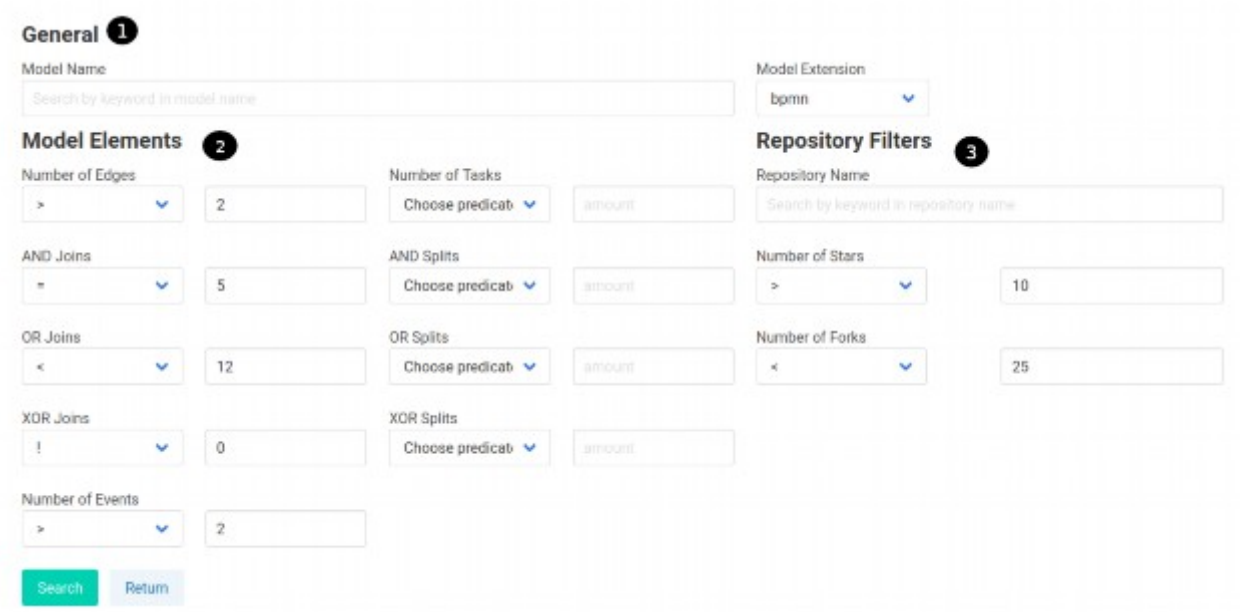

Рис. 3.12. Приклад графічного інтерфейсу для фільтрування моделі

Загальні фільтри (1), можна використовувати для фільтрації моделей за їх назвою. Наприклад, назва «медичний» поверне всі моделі, що містять цей рядок. Також можна фільтрувати моделі за їхніми розширеннями (наприклад, файли, що закінчуються на «.bpmn/.epml»).

Моделі також можна фільтрувати на основі елементів формату канонічного процесу. Користувачі можуть вказати певний предикат (наприклад, «<», «>», «=», «!»), а потім число, що дає можливість виконати такий пошуковий запит: «numberOfEdges > 0 AND numberOfEvents = 3».

Також, користувачі також можуть фільтрувати моделі за метаданими свого сховища (наприклад, кількістю зірочок, розгалужень) (3) і назвою сховища.

Натиснувши кнопку «Пошук», система фільтрації отримає вказівку зібрати та повернути моделі, які задовольняють критеріям пошуку, надаючи користувачам можливість завантажувати моделі як пакетний архів.

У запропонованій системі керування моделлю стосується виконання дій CRUD над набором моделей. Нижче наведено список функцій, доступних API:

• Створити: функція create використовується для завантаження моделі разом із додатковими метаданими до репозиторію моделей.

• Отримати: Ця функція використовується для пошуку останньої версії моделі з унікальним ідентифікатором (тобто «id»). Крім того, також надається історія версій моделі.

• Список: використовується для отримання колекції найновіших версій доступних моделей у сховищі.

• Пошук: Пошук використовується для пошуку моделі за її атрибутом «ім'я». Вказувати повну назву моделі не потрібно. Таким чином, можна отримати модель за її частковою назвою.

• Оновлення: ця функція оновлює вже доступний екземпляр моделі з його унікальним ідентифікатором.

• Видалити: функція видалення використовується для видалення існуючої моделі зі сховища за її унікальним ідентифікатором.

Служба керування моделлю генеруватиме відповідь JSON щоразу, коли буде зроблено виклик однієї з цих функцій. Приклад згенерованого результату функції Get

```
"id": 38,
"name": "Dispatch-of-goods",
"file_name" : "file1.bpmn"
"description": "A bpmn diagram",
"modeling_language": "bpmn",
"model": ...
```

```
"versions": [
    {
        "id": 86,
        "name": "Dispatch-of-goods",
        "file_name" : "file1.bpmn"
        "description": "A bpmn diagram",
        "modeling_language": "bpmn",
        "model": ...,
        "version_number": 1.0,
        ...
    },
...
...
]
```

Рисунок 3.13 ілюструє розділ веб-інтерфейсу для керування моделями. Інтерфейс дозволяє користувачам запитувати моделі за атрибутом «name». Результати запиту представлені у вигляді списку. Натиснувши кнопку «Редагувати», можна переглянути деталі моделі (тобто метадані) і застосувати деякі зміни. Натиснувши кнопку «Додати модель», користувачі можуть завантажити власні моделі процесу та пов’язані метадані.

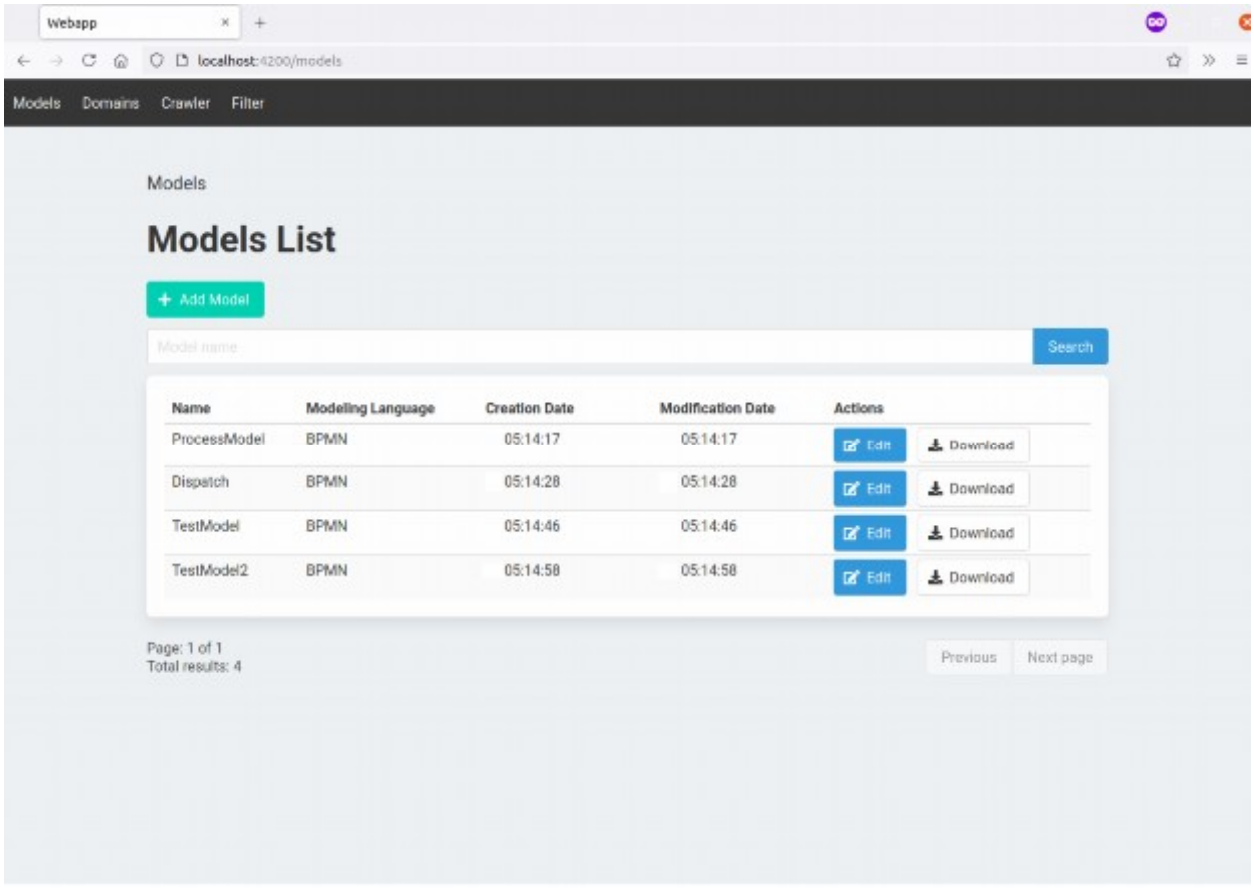

Рис. 3.13. Знімок екрана сторінки списку моделей

Подібно до керування моделлю, керування доменом стосується виконання дій CRUD над набором доменів. Користувачі можуть вказати ряд доменів додатків (наприклад, охорона здоров’я, банківська справа та роздрібна торгівля) через інтерфейс доменів разом із відповідними тегами/мітками. Система використовує мітки для обчислення перевірки подібності між доменами та зібраними моделями процесів. На рисунку 3.14

показано приклад визначення нового домену програми, включаючи пов’язані теги.

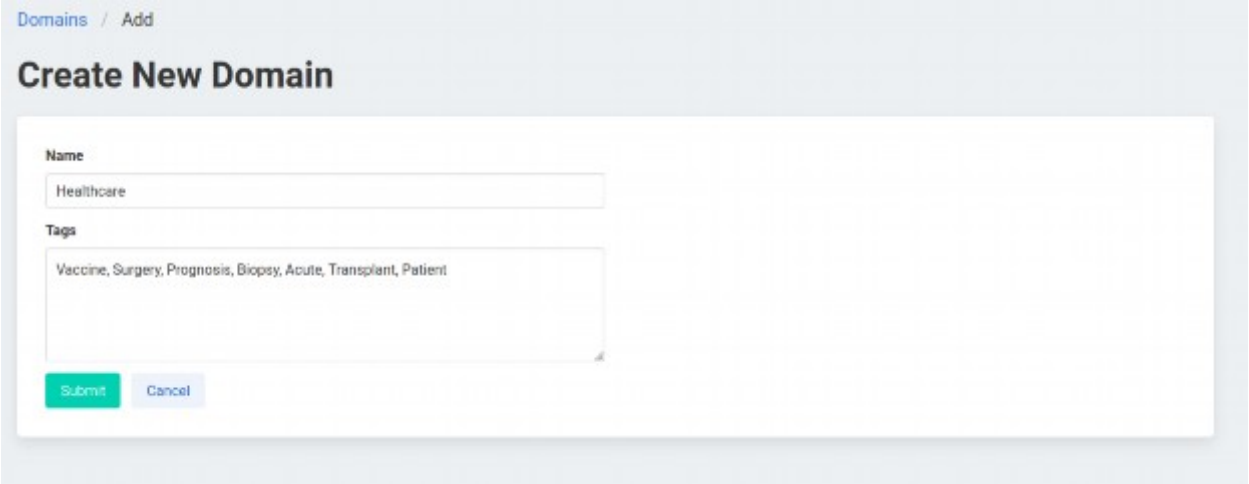

Рис. 3.14. Приклад домену програми, що складається з набору пов'язані теги

Отже, ми використовували лише основні системи Arrowhead для впровадження систем, які формують конвеєр підготовки моделі MAAS. Ця система робить можливим виконання попередньо визначених робочих процесів за допомогою оркестровки та споживання послуг Arrowhead Framework [3]. Кожен робочий процес складається з трьох обов’язкових компонентів: плани, дії та кроки. За допомогою планів можна визначити робочий процес за назвою, що складається з набору дій, що об’єднують кілька кроків. За допомогою цих робочих процесів можна виконувати низку функціональних можливостей, наданих системами, у певній послідовності.

Було реалізовано модель сканера, що використовує багатопотоковість для паралельного виконання завдань сканування. Однак цей дизайн обмежений лише однією машиною/пристроєм. Отже, дизайн можна збільшити лише вертикально, а не горизонтально, додавши більш потужну машину. Для горизонтального масштабування ми рекомендуємо розробити розподільний сканер. По суті, дозволяючи системі справлятися зі збільшеним навантаженням, додаючи більше пристроїв до локальної хмари.

Ми реалізували евристичний механізм відображення доменів, який витягує всі імена вузлів із моделі процесу та обчислює оцінку подібності між визначеними доменами. Однак цей метод не є точним у відображенні доменів. Тому ми рекомендуємо включати більш складні методи в зіставлення доменів з моделями процесів, такими як машинне навчання.

Загалом, виходячи з результатів оцінки, ми рекомендуємо вдосконалити інтерфейс користувача. Зокрема, покращення зручності інтерфейсу шляхом включення рейтингових чітких інструкцій, підказок і візуального зворотного зв’язку . Хоча поточна реалізація ¬інтерфейсу забезпечує необхідну функціональність для роботи з MAAS, вона дуже проста і вимагає від користувача певних технічних знань. Крім того, ми рекомендуємо оптимізувати процес автентифікації користувача сканера GitHub.

Висновки до розділу

У цьому розділі представлено детальний огляд системи автоматизації аналізу моделей (MAAS), вказано мету MAAS, а також вимоги, якими вона обмежена. Також розглядається пояснення архітектури системи та її основних елементів і описується процес інтеграції системи в Arrowhead Framework.

ВИСНОВКИ

В магістерській роботі представлено архітектуру для системи автоматизації аналізу моделей (MAAS) разом із реалізацією працюючого прототипу. MAAS — це частина програмного забезпечення, що працює на основі Arrowhead Framework як набору кількох систем, метою яких є співпраця для виконання завдань автоматизації в аналітичному робочому процесі.

Основні функції MAAS полягають у зборі моделей процесів із сховищ програмного забезпечення через інтелектуальний аналіз репозиторіїв, перевірці зібраних моделей процесів на їхню метамодель, наданні параметрів фільтрації їхніх метаданих і пропонуванні операцій для перетворення моделей у цільовий формат. У представленій архітектурі використовуються концепції локальної хмари на основі SOA Arrowhead Framework, що забезпечує кілька переваг. З одного боку, він додає рівень абстракції, що дозволяє розширити MAAS додатковими службами для автоматизації додаткових кроків у робочому процесі аналітики моделі. З іншого боку, архітектура дозволяла гнучкість, об’єднуючи кілька шаблонів проектування програмного забезпечення. Отже, розробники програмного забезпечення можуть легко розширити основні служби додатковими службами.

Було оцінено MAAS за допомогою тематичного дослідження, використовуючи єдиний випадок цілісного дизайну. Для користувачів, які мають досвід роботи з Arrowhead Framework та керування моделями можуть використовувати MAAS для збору та підготовки низки моделей процесів для аналітики. Описано реалізацію та проектні рішення для прототипу версії MAAS, що реалізовано з використанням багаторівневої архітектури, що складається з рівня керування репозиторієм, даних і представлення, до яких можна отримати доступ через наступний репозиторій.

В роботі детально описано рівень зберігання моделей, який складається з інформації про базову базу даних для зберігання моделей і пов’язаних

метаданих, а також пояснення структури даних і концептуальної схеми системи репозиторію.

Запропоновані методи та системи можуть бути впроваджені в компаніях, що працюють з великими даними, машинним навчанням та штучним інтелектом для полегшення управління та повторного використання моделей.

Створені інструменти автоматизації репозиторіїв дозволять зменшити часові витрати на пошук, адаптацію та впровадження моделей у різні системи, що сприятиме прискоренню інноваційних процесів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ModelBus User Guide. – Режим доступу: https://www.modelbus.org/projects/ModelBus/static/custom/file/docs/modelbus/ModelBus_UserGuide_0983_v1.9.9.pdf
2. Kerstin Altmanninger et al. “AMOR–towards adaptable model versioning”. In: 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS. Vol. 8. 2008, pp. 4–50.
3. Arrowhead Framework. Режим доступу: https://github.com/arrowhead-f/.
4. Axellience: GenMyModel. Режим доступу:https://www.genmymodel.com/.
5. Aaron Bangor et al. “Determining what individual SUS scores mean: adding an adjective rating scale”. In: Journal of usability studies 4.3 (2009), pp. 114–123. issn: 1931-3357.
6. Francesco Basciani et al. “MDEForge: An extensible Web-based modeling platform”. In: CEUR Workshop Proceedings 1242.619583 (2014), pp. 66–75. issn: 16130073.
7. Fredrik Blomstedt et al. “The arrowhead approach for SOA application development and documentation”. In: IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society. 2014, pp. 2631–2637. doi: 10.1109/IECON.2014.7048877.
8. MDEForge: an extensible Web-based modeling platform? Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Amleto Di Salle, Ludovico Iovino, and Alfonso Pierantonio - https://ceur-ws.org/Vol-1242/paper10.pdf
9. Ozren Dabic, Emad Aghajani, and Gabriele Bavota. “Sampling Projects in GitHub for MSR Studies”. In: (2021). arXiv: 2103.04682. Режим доступу: http://arxiv.org/abs/2103.04682.
10. Fred D. Davis. “Perceived usefulness, perceived ease of use, and user acceptance of information technology”. In: MIS Quarterly: Management Information Systems 13.3 (1989), pp. 319–339. issn: 02767783. doi: 10.2307/249008.

11. Jerker T A T T Delsing. IoT automation : arrowhead framework LK. Boca Raton, 2017.
12. Juri Di Rocco et al. “Collaborative repositories in model-driven engineering”. In: IEEE Software 32.3 (2015), pp. 28–34.
13. Steve Easterbrook et al. “Selecting Empirical Methods for Software Engineering Research”. In: Guide to Advanced Empirical Software Engineering. Ed. by Forrest Shull, Janice Singer, and Dag I K Sjøberg. London: Springer London, 2008, pp. 285– 311. isbn: 978-1-84800-044-5. doi: 10.1007/978-1-84800-044-5_11.
14. Elasticsearch. Режим доступу: https : / / github . com / elastic / elasticsearch.
15. Mturi Elias and Paul Johannesson. “A survey of process model reuse repositories”. In: Communications in Computer and Information Science. Vol. 285 CCIS. 2012, pp. 64–76. isbn: 9783642291654.
16. Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. 1st ed. Addison-Wesley Professional, 1994. isbn: 0201633612. Режим доступу: http://www.amazon . com / Design - Patterns - Elements - Reusable - Object - Oriented / dp /0201633612/ref=ntt_at_ep_dpi_1.
17. Georgios Gousios. “The GHTorrent dataset and tool suite”. In: Proceedings of the 10th Working Conference on Mining Software Repositories. MSR ’13. San Francisco, CA, USA: IEEE Press, 2013, pp. 233–236. isbn: 978-1-4673-2936-1. Режим доступу: http://dl.acm.org/citation.cfm?id= 2487085.2487132.
18. Martin Haeusler et al. “ChronoSphere: a graph-based EMF model repository for IT landscape models”. In: Software and Systems Modeling 18.6 (2019), pp. 3487–3526.
19. Regina Hebig et al. “The quest for open source projects that use UML: Mining GitHub”. In: Proceedings - 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems,

MODELS 2016. 2016, pp. 173–183. isbn: 9781450343213. doi: 10.1145/2976767.2976778.

20. Christian Hein, Tom Ritter, and Michael Wagner. “Model-driven tool integration with ModelBus”. In: Future Trends of Model-Driven Development - Proceedings of the 1st International Workshop on Future Trends of Model-Driven Development - FTMDD 2009 In Conjunction with ICEIS 2009. 2009, pp. 35–39.
21. Thomas S. Heinze, Viktor Stefanko, and Wolfram Amme. “Mining BPMN processes on GitHub for tool validation and development”. In: Lecture Notes in Business Information Processing. Vol. 387 LNBIP. Springer, 2020, pp. 193–208. isbn: 9783030494179. doi: 10.1007/978-3-030-49418-6_13.
22. (PDF) Model-based Real-time Synchronization. – Режим доступу: https://www.researchgate.net/publication/267452480_Model-based_Real-time_Synchronization
23. Mo Jamshidi. Systems of systems engineering: principles and applications. CRC press, 2017, pp. 2–3.
24. ChronoSphere: a graph-based EMF model repository for IT landscape models | Software and Systems Modeling. – Режим доступу: https://link.springer.com/article/10.1007/s10270-019-00725-0
25. Zador Daniel Kelemen et al. “Selecting a Process Modeling Language for Process Based Unification of Multiple Standards and Models”. In: June 2014 (2013), pp. 1– 14.
26. Ahsanun Naseh Khudori and Tri Astoto Kurniawan. “Transforming EPC Aris Markup Language into BPMN Metadata”. In: Proceedings of 2019 4th International Conference on Sustainable Information Engineering and Technology, SIET 2019 (2019), pp. 358–363. doi: 10.1109/SIET48054.2019.8986075.
27. Maximilian Koegel and Jonas Helming. “EMFStore - A model repository for EMF models”. In: Proceedings - International Conference on Software Engineering 2 (2010), pp. 307–308.

28. Oleksii Kononenko et al. “Mining modern repositories with Elasticsearch”. In: 11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings. Association for Computing Machinery, Inc, 2014, pp. 328–331. isbn: 9781450328630. doi: 10.1145/2597073.2597091. Режим доступу: http://dx.doi.org/10.1145/2597073.2597091.
29. Vladimir Kotsev, Ivan Stanev, and Katalina Grigorova. “BPMN-EPC-BPMN Converter BPMN-EPC-BPMN Converter”. In: January 2016 (2011).
30. Daniel Kozma, Pal Varga, and Kristof Szabo. “Achieving Flexible Digital Production with the Arrowhead Workflow Choreographer”. In: IECON Proceedings (Industrial Electronics Conference) 2020-October.October (2020), pp. 4503–4510. doi: 10.1109/IECON43393.2020.9254404.
31. Marcello La Rosa et al. “APROMORE: An advanced process model repository”. In: Expert Systems with Applications 38.6 (2011), pp. 7029–7040.
32. Urška Lah, James R Lewis, and Boštjan Šumak. “Perceived Usability and the Modified Technology Acceptance Model”. In: International Journal of Human-Computer Interaction 36.13 (2020), pp. 1216–1230. issn: 15327590. doi: 10.1080/10447318.2020.1727262. Режим доступу: https://www.tandfonline.com/action/journalInformation?journalCode=hihc2.
33. Steve LaValle et al. “Big Data, Analytics and the Path From Insights to Value”. English. In: MIT Sloan Management Review 52.2 (2011). Copyright - Copyright © Massachusetts Institute of Technology, 2011. All rights reserved; Document feature - Charts; Tables; Diagrams; Last updated - 2020-11-17; CODEN - SMRVAO, pp. 21– 32.
34. Ruopeng Lu and Shazia Sadiq. “A survey of comparative business process modeling approaches”. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 4439 LNCS. Springer Verlag, 2007, pp. 82–94. isbn: 9783540720348. doi: 10.1007/978-3-540-72035-5_7.

35. Miklós Maróti et al. “Next generation (Meta)modeling: Web- and cloud-based collaborative tool infrastructure”. In: CEUR Workshop Proceedings. Vol. 1237. 2014, pp. 41–60.
36. Robin J.P. Mennens. “The Philips Remote AI Streaming (PRAIS) platform”. English. PdEng Thesis. PhD thesis. Oct. 2020.
37. Cecilia Titiek Murniati and Ridwan Sanjaya. “Learning technologies in education: Issues and trends”. In: (2017).
38. Pete Chapman Ncr et al. “Step-by-step data mining guide”. In: SPSS inc 78 (2000), pp. 1–78. Режим доступу: http://www.crisp-dm.org/CRISPWP-0800.pdf.
39. Object Management Group. “BPMN Core Elements”. In: httpbpmnorgSamplesElementsCoreBPMNElementshtm (2010). Режим доступу: https://www.omg.org/bpmn/Samples/Elements/Core_BPMN_Elements.htm
40. Saheed Popoola, Jeffrey Carver, and Jeff Gray. “Modeling as a service: A survey of existing tools”. In: CEUR Workshop Proceedings. Vol. 2019. 2017, pp. 360–367. Режим доступу: https://azure.microsoft.com.
41. Qualtrics. Net Promoter Score (NPS®) - The Ultimate Guide | Qualtrics. 2021. Режим доступу: https : / / www . qualtrics . com / uk / experience - management / customer / net- promoter- score/?rid=ip&prevsite=en&newsite=uk&geo=GB&geomatch=
42. Khalid Raza. APPLICATION OF DATA MINING IN BIOINFORMATICS. Tech. rep. 2. 2010, pp. 114–118. Режим доступу: http://multalin.toulouse.inra.fr/multalin/multalin.html.
43. Gregorio Robles et al. “An extensive dataset of UML models in GitHub”. In: IEEE International Working Conference on Mining Software Repositories May (2017), pp. 519–522. issn: 21601860. doi: 10.1109/MSR.2017.48.
44. SPARX Systems: Enterprise Architect. Режим доступу: https://sparxsystems.com/.

45. A.S. Tanenbaum M. van Steen. Distributed Systems. 3rd ed. distributed-systems.net, 2017.
46. “Strategies for Qualitative Interviews”. In: Harward University (2017), pp. 1–4. Режим доступу: http : / / sociology . fas . harvard . edu / files / sociology / files / interview _strategies.pdf.
47. Bedir Tekinerdogan et al. “Introduction to model management and analytics”. In: Model Management and Analytics for Large Scale Systems. 2020, pp. 3–11. isbn: 9780128166499. doi: 10 . 1016 / b978 - 0 - 12 - 816649 - 9 . 00009 - 0. Режим доступу: https://doi.org/10.1016/B978-0-12-816649-9.00009-0.
48. The CDO Model Repository. http://www.eclipse.org/cdo.
49. Willi Tscheschner. Transformation from EPC to BPMN. Tech. rep. 2006.
50. Pal Varga et al. “Making system of systems interoperable – The core components of the arrowhead framework”. In: Journal of Network and Computer Applications 81 (2017), pp. 85–95. issn: 1084-8045. doi: https://doi.org/10.1016/j.jnca.2016.08.028. Режим доступу: https://www.sciencedirect.com/science/article/pii/S1084804516301965.