МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)

Кафедра Програмної інженерії
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

рівень вищої освіти другий (магістерський)

**Дослідження методів створення сервіс-орієнтованих програмних систем**
**(тема)**

Виконав:
Студент 2 курсу, групи ІПЗздм-19-1
Горобець Д.Г.
(прізвище, ініціали)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник проф. Шостак І.В.
(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
« ____ » _____ 2021 р.

**ЗАВДАННЯ**
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Горобця Дмитра Григоровича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів створення сервіс-орієнтованих програмних систем

затверджена наказом університету від 26.03.2021 № 34 Стз

2. Термін подання роботи до екзаменаційної комісії 12 05 2021р.

3. Вихідні дані до роботи *проаналізувати існуючі алгоритми, що використовуються для вимог підтримки прийняття рішень, мови розробки програмного забезпечення*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, опис запропонованих варіантів оптимізації, використовувані методи та алгоритми, опис розробленої програмної системи, опис застосованих програмних рішень, аналіз можливих застосувань*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

*Мета завдання, обґрунтування доцільності розробки, постановка задачі, базові моделі, методи й алгоритми, структурно-логічна схема взаємодії даних, інтерфейс програмної системи, результати дослідної експлуатації програмної*

*системи, висновки*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| спецчастина | проф. Шостак І.В. | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Аналіз предметної галузі | 26 березня 2021 р. | виконано |
| 2. | Огляд існуючих методів | 31 березня 2021 р. | виконано |
| 3. | Розробка алгоритмів, проектування та розробка ПЗ | 15 квітня 2021 р. | виконано |
| 4. | Підготовка пояснювальної записки | 28 квітня 2021 р. | виконано |
| 5. | Спецчастина | 30 квітня 2021 р. | виконано |
| 6. | Підготовка презентації та доповіді | 05 травня 2021 р. | виконано |
| 7. | Попередній захист | 10 травня 2021 р. | виконано |
| 8. | Нормоконтроль, рецензування | 10 травня 2021 р. | виконано |
| 9. | Занесення роботи в електронний | 11 травня 2021 р. | виконано |
| 10. | Допуск до захисту в зав. кафедри | 12 травня 2021 р. | виконано |

Дата видачі завдання ____ _____ 2021р.

Студент _____
(підпис)

Керівник роботи _____ проф. Шостак І.В.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ /ABSTRACT

Пояснювальна записка до кваліфікаційної роботи магістра: 101 с, 46 рис., 6 дод., 37 джерел

ОНТОЛОГІЇ, СЕРВІС-ОРІЄНТОВАНІ АРХІТЕКТУРИ, ІНТЕЛЕКТУАЛЬНІЙ АНАЛІЗ, СЕМАНТИЧНИЙ АНАЛІЗ, ДОСТУП ДО ДАННИХ.

Об’єкт досліджень – сервіс-орієнтовані архітектури та онтологічні моделі.

Метою роботи є створення формалізму опису взаємодії сервісів і алгоритмів побудови інтерфейсів веб-сервісів для реалізації ефективної SOA-системи із застосуванням парадигми, що заснована на онтологіях доступу до даних.

Методи дослідження – сучасні підходи до питань онтологій, аналізу даних, оцінки Інтернет-сервісів.

Результат – отримані алгоритми автоматизованої побудови інтерфейсів веб-сервісів для SOA-архітектури, що дають можливість автоматизовано будувати інтерфейси веб-сервісів.

ONTOLOGIES, SERVICE-ORIENTED ARCHITECTURES, INTELLECTUAL ANALYSIS, SEMANTIC ANALYSIS, ACCESS TO DATA.

The object of research is service-oriented architectures and ontological models.

The aim of the work is to create a formalism for describing the interaction of services and algorithms for building web service interfaces for the implementation of an effective SOA-system using a paradigm based on ontologies of data access.

Research methods – modern approaches to ontologies, data analysis, evaluation of Internet services.

The result is algorithms for automated construction of web service interfaces for SOA-architecture, which allow to automatically build web-service interfaces.

Я, Горобець Дмитро Григорович, студент гр. ІПЗздм-19-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів створення сервіс-орієнтованих програмних систем», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ЗМІСТ

# ВСТУП

Зростання популярності за останні роки в мережі Internet таких технологій, як хмарні обчислення (cloud computing), семантичні мережі й Semantic Веб, демонструє актуальність задачі організації взаємодії Internet-Ресурсів між собою. Забезпечити таку взаємодію дозволяє сервіс-орієнтована архітектура (SOA) розробки програмного забезпечення. SOA-системи легко масштабуються, а їх сервісні структури полегшують повторне використання компонентів. SOA-архітектура забезпечує створення прикладних програмних систем, побудованих за допомогою розподілених взаємодіючих сервісів.

На даний момент існує множина систем для створення веб-додатків з SOA архітектурою: IBM, Microsoft .NET, Oracle SOA Suite. У той же час не існує ефективних систем, здатних автоматично будувати інтерфейси й перетворювати популярні сайти колишнього покоління в елементи композитних служб або сервісних кластерів. Необхідність таких систем обумовлюється проблемами реалізації EAI (Enterprise Application Integration – інтеграції корпоративних додатків) [2].

EAI – це інтеграційна архітектура, що складається з набору технологій і методів для інтеграції систем і додатків у масштабах корпоративних архітектур. Додатки керування ланцюжками поставок, документообігу, керування взаємовідносинами із клієнтами, бізнес-аналітики й інші, як правило, не можуть взаємодіяти один з одним з метою обміну даними або правилами бізнес-процесів без створення додаткових програмних рішень. Отже, з'являються проблеми з автоматизацією процесів і надлишковими даними, що зберігаються в декількох місцях. Інтеграція корпоративних додатків – засіб, що забезпечує процес зв'язування таких додатків у межах однієї організації разом для спрощення й автоматизації бізнес-процесів. Процес зв'язування повинен протікати, не вносячи радикальних змін в існуючі додатки або структури даних. Внаслідок цих факторів

побудова SOA-системи з композитною структурою звичайно зводиться до методів EAI.

В 2003 році стало відомо, що 70 % усіх EAI-проектів зазнало невдачі. Голова європейського консорціуму по інтеграції Стів Краггс відзначив проблеми реалізації EAI-систем. Головна проблема полягає в тому, що елементи EAI-системи постійно змінюються. Потрібні ресурси на постійне перероблення EAI-системи та її розширення. Крім проблем реалізації EAI-систем, підтримці SOA-систем, що активно працюють із мережею Internet, заважає фактор постійного розвитку всесвітньої павутини. При цьому постійна зміна архітектури й розробка заново вже існуючих інтерфейсів вимагають безперервної роботи фахівців.

Побудову ефективної SOA-системи, позбавленої або частково позбавленої проблем реалізації EAI-систем, доцільно розв'язати за допомогою застосування парадигми заснованого на онтологіях доступу до даних (OBDA – Ontology-Based Data Access), що просуваються завдяки вченому з Італії Діего Кальванезе. OBDA – доступ до даних через високорівневий інтерфейс онтології. Центральним процесом заснованого на онтологіях доступу до даних є представлення бази даних мовою дескриптивної логіки (DL). Такий підхід дозволяє використовувати переваги онтологій і DL. Внаслідок застосування OBDA-технологій спростилася робота зі зміни структури SOA-системи в порівнянні із провідними SOA-системами, зазначеними раніше. Для зміни структури SOA-системи досить відредагувати онтологію системи, що не складає труднощів для людини без фахової освіти, за допомогою редактора онтологій. При цьому відкривається можливість побудови інтерфейсів на базі семантичної сервіс- орієнтованої архітектури (SSOA), що просувається творцем всесвітньої павутини Тімоті Бернерс-Лі, ніж забезпечується подальший ефективний розвиток технології Semantic Веб.

Метою роботи є створення алгоритмів опису взаємодії сервісів і алгоритмів побудови інтерфейсів веб-сервісів для реалізації ефективної SOA-системи із застосуванням парадигми, що заснована на онтологіях доступу до даних.

Реалізація OBDA-системи дозволяє отримувати дані з баз даних на основі логічного аналізу вирішувача над базою знань. При цьому ефективність доступу до даних досягається за рахунок відображення запитів у середовище реляційної СУБД, що розташовує ефективними засобами обробки запитів і іншими перевагами. TBox (набір аксіом) повинен використовуватися для забезпечення необхідної семантичної інформації на більш високому концептуальному рівні [2].

Для реалізації SOA-системи необхідно створити формалізм для опису процесів системи за допомогою онтології й ефективні алгоритми або вдосконалити існуючі OBDA-алгоритми під роботу з SOA-системами.

Для досягнення мети роботи необхідно розв’язати наступні завдання.

Провести аналіз основних видів структур взаємодії SOA-сервісів. Навести короткий огляд експериментальних реалізацій систем OBDA, що працюють із сімейством мов дескриптивної логіки Dl-lite. Підтвердити практичне застосування дескриптивної логіки і мов веб-онтологій з метою надання онтологічного доступу до даних.

Розробити алгоритми опису й аналізу концептуальної моделі SOA-системи, орієнтований на онтологічний підхід, що й дозволяє адекватно описувати взаємодію веб-сервісів.

Спроектувати алгоритми автоматизованої побудови інтерфейсів веб-сервісів для SOA-архітектури, що дають можливість автоматизовано будувати інтерфейси веб-сервісів на основіонтології.

Розробити програмну реалізацію SOA-системи, що використовує отримані алгоритми автоматизованого побудови інтерфейсів взаємодії веб-сервісів. Для переходу системи на SSOA-архітектуру структура онтології, необхідної для роботи системи, повинна мати OWL-розмітку стандарту Semantic Веб.

Для розв’язку поставлених завдань використовувалися методи загальної алгебри, теорії алгоритмів, об'єктно-орієнтованого проектування, теорії уніфікації.

Отримані алгоритми автоматизованої побудови інтерфейсів веб-сервісів для SOA-архітектури, що дають можливість автоматизовано будувати інтерфейси веб-

сервісів. Алгоритми відрізняються більш загальною моделлю й швидкістю роботи вище аналогів на 24 – 41% залежно від вхідних параметрів алгоритмів.

Отриманий метод оптимізації алгоритмів трансформації запитів під SIR-систему, що відрізняється від сторонніх алгоритмів трансформації запитів більш швидкою роботою за рахунок ігнорування несуттєвих елементів вхідних даних. Різниця у швидкості роботи між отриманим алгоритмом трансформації запитів і аналогами залежить від кількості несуттєвих елементів в онтології.

Доведена коректність і оцінки складності побудованих алгоритмів

Формалізм для опису й аналізу концептуальної моделі SOA-системи, орієнтований на онтологічний підхід, що й дозволяє описувати взаємодію веб-сервісів.

Нові алгоритми автоматизованої побудови інтерфейсів веб-сервісів для SOA-архітектури, що дають можливість автоматизовано будувати інтерфейси веб-сервісів.

Програмна реалізація SOA-системи, призначена для використання в основі систем автоматизованої побудови інтерфейсів взаємодії веб-сервісів.

Структура онтології, необхідної для роботи SOA-системи, що має OWLE-розмітку стандарту Semantic Веб.

Отримана точна семантика SOA-системи, що дає можливість описувати SOA-системи в мовах дескриптивної логіки. Показана ефективність математичного формалізму для завдань логічного аналізу прикладних онтологій і опису сервіс-орієнтованих систем.

Показана необхідність застосування алгоритмів автоматичної побудови веб-інтерфейсів.

Доведені теореми про обчислювальну складність, можливості розв'язання й кінцівки SIR-алгоритмів.

Отримана система SIR-System, що показує працездатність систем автоматизованого побудови інтерфейсів взаємодії веб-сервісів. Структура онтології, необхідної для роботи системи, має OWLE-розмітку стандарту Semantic

Веб. Цей фактор відкриває можливість переходу системи на базу семантичної сервіс-орієнтованої архітектури (SSOA), орієнтованої на Semantic Веб сервіси.

Використання дескриптивних логік гарантує динамічність і простоту масштабування системи, розробленої на базі представлених алгоритмів. Невелика кількість часу, необхідне для роботи SIR алгоритму, і прийнятна кількість породжених запитів доводять ефективність алгоритму системи автоматизованої побудови інтерфейсів взаємодії веб-сервісів.

# 1 АНАЛІЗ СТАНУ РОЗВ'ЯЗАННЯ ПРОБЛЕМИ ТА ОБҐРУНТУВАННЯ

## 1.1 Загальні поняття SOA-систем

Зі зростанням популярності в мережі Internet таких технологій, як хмарні обчислення (cloud computing), розподілені системи й Semantic Веб демонструє актуальність завдання організації взаємодії Internet-ресурсів між собою. Забезпечити таку взаємодію дозволяє сервіс-орієнтована архітектура (SOA) розробки програмного забезпечення [3].

Перевага SOA-технологій полягає в багаторазовому використанні програмного коду для створення складених розподілених програмних систем. Сервіс-орієнтована архітектура забезпечує кросплатформеність і незалежність від засобів розробки, дозволяючи легко управляти й інтегрувати створювані програмні системи. Наприклад, програмні реалізації компонентів SOA-системи, написані мовою програмування Java, можуть бути викликані компонентами, написаними мовою програмування C# у рамках однієї або декількох SOA-систем [4]. Основною перевагою SOA-архітектури є здатність до масштабованості – керування зростанням корпоративних систем, здатність надання й використання послуг у масштабах мережі Internet, здатність до скорочення витрат на організацію взаємодії систем. Нижче наведений приклад проекту, для якого підходить SOA-архітектура. Великій компанії, яка здобуває невеликі компанії, необхідно визначити, як інтегрувати придбані IT-інфраструктури у свій проект. Завдяки здатності до масштабування й інтеграції SOA-архітектура дозволяє проектам адаптуватися до потреб конкретної предметної області або процесам, що проходять у рамках цієї предметної області. Інфраструктура SOA сприяє також більш гнучкій і оперативній роботі системи, саме тому цільна система, побудована на великій кількості парних інтерфейсів. Таким чином, SOA-архітектура забезпечує міцну основу для систем, що відповідають умовам бізнес-гнучкості й адаптивності.

Центральним поняттям SOA-архітектура є сервіс (service – послуга) [3]. Іменник «послуга» визначається в словниках як «виконання робіт (функцій) друг для друга». Однак треба пам'ятати, що термін поєднує в собі наступні взаємозалежні ідеї:

– можливість виконувати роботу для клієнтів;

– чітку формалізацію робіт, наданих клієнтам;

– пропозиція виконати роботу для клієнта.

Сервіси є центральним поняттям SOA-архітектура, тому що вони реалізують концепцію масштабування завдяки ефектам прозорості й легкої інтеграції. Прозорість виражається через опис сервісу, що містить інформацію, необхідну для взаємодії із сервісом, і описує це в таких термінах, як вхідні дані служб, вихідні дані служб і пов'язана з ними інформація.

Опис сервісів також передає інформацію про завдання служби й умови для використання сервісу. Ключовим компонентом сервісів є строго детерміновані інтерфейси. Інтерфейси можуть бути викликані стандартними способами, навіть якщо дані про додаток, що їх викликає, не відомі сервісу. У той же час об'єктам, взаємодіючим із сервісами, не відомі дані про механізми виконання службами їх цільового завдання. Ці ефекти досягаються за рахунок інкапсуляції деталей реалізації від інших частин системи [2].

Далі наведено три основні структури, які характеризують SOA-системи по внутрішньому складу й способам взаємодії з обчислювальним середовищем [4].

– атомарна служба (atomic service) – неподільна й проста по складу служба;

– композитна служба (composite service) – батьківська служба зі складною будовою, яка охоплює одну або більш прості по складу дочірні служби, сховані від зовнішнього миру – ця формація може бути представлена у вигляді ієрархічної структури;

– сервісний кластер (service cluster) – група розподілених по призначенню й технологічно пов'язаних служб, взаємодіючих для розв'язку поставленого завдання.

Слід звернути увагу на вплив обчислювальних середовищ і на вибір структури SOA-систем (див. рисунок 1.1) [4].

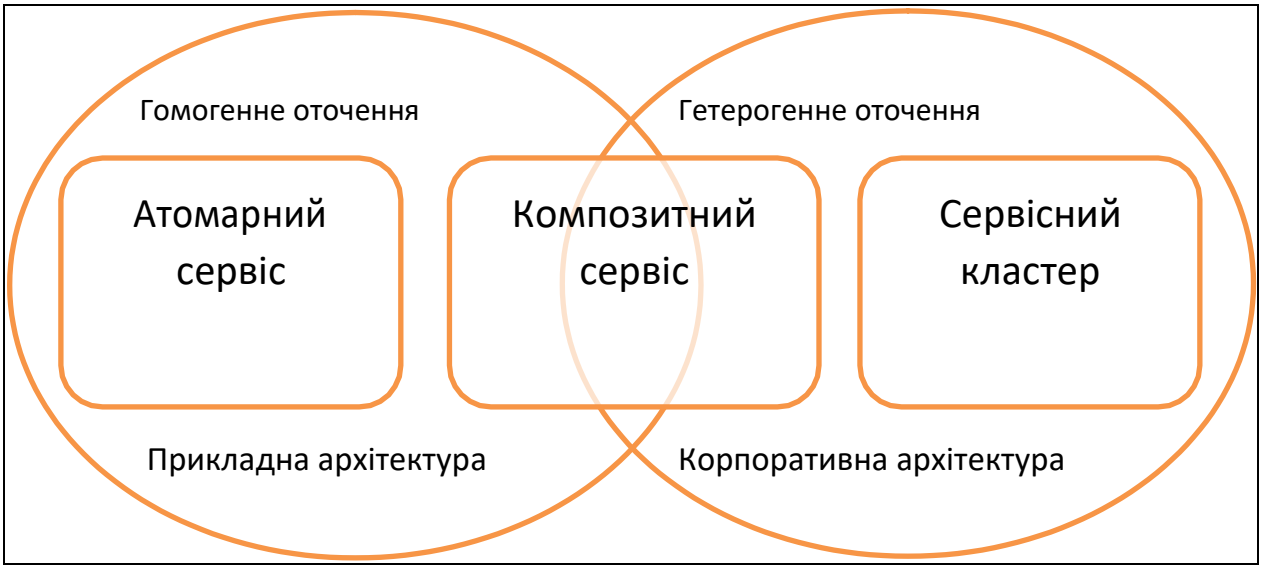


Рисунок 1.1 – Вплив обчислювального середовища на структуру SOA-систем

Атомарні й композитні служби ефективно працюють в умовах однорідного обчислювального середовища (Homogeneous Computing Environment). Однорідне обчислювальне середовище підпадає під категорію прикладних архітектур (Application Architecture). У той же час композитні служби й сервісні кластери часто зустрічаються в гетерогенних обчислювальних середовищах (Heterogeneous Computing Environment) під впливом задач корпоративних архітектур (Enterprise Architecture). Отже, композитні служби відіграють важливу роль в обох обчислювальних середовищах.

SOA-системи легко масштабуються, а їх сервісні структури полегшують повторне використання компонентів. SOA-архітектура забезпечує створення прикладних програмних систем, побудованих за допомогою розподілених взаємодіючих сервісів [6].

У цей час існують SOA-системи для створення веб- додатків з SOA архітектурою: IBM Lotus Notes, Microsoft .NET, Oracle SOA Suite. У той же час невідомі ефективні системи, здатні автоматично будувати інтерфейси й перетворювати звичайні сайти старого покоління в елементи композитних служб або сервісних кластерів.

Необхідність таких систем обумовлюється проблемами реалізації EAI (Enterprise Application Integration – інтеграція корпоративних додатків) [1].

EAI – це інтеграційна архітектура, що складається з набору технологій і методів для інтеграції систем і додатків у масштабах корпоративних архітектур. Додатки керування ланцюжками поставок, документообігу, керування взаємовідносинами із клієнтами, бізнес-аналітики й інші, як правило, не можуть взаємодіяти один з одним з метою обміну даними або правилами бізнес-процесів. Отже, з'являються проблеми з автоматизацією процесів і надлишкові дані, що зберігаються в декількох місцях. Інтеграція корпоративних додатків – це засіб, що забезпечує процес зв'язування таких додатків у межах однієї організації разом для спрощення й автоматизації бізнес-процесів. Процес зв'язування повинен протікати, не вносячи радикальних змін в існуючі додатки або структури даних [7]. Внаслідок цих факторів побудова SOA-системи з композитною структурою зводиться до методів EAI.

Відомо, що до 2003 року 70% усіх EAI-проектів зазнало невдачі. Голова європейського консорціуму по інтеграції Стів Краггс відзначив сім основних проблем реалізації EAI-систем.

Постійні зміни. Елементи EAI-системи постійно змінюються. Потрібні ресурси на постійне перероблення EAI-системи і її розширення.

Нестача EAI-фахівців. Процес реалізації EAI-систем вимагає знання багатьох проектних нюансів і технічних аспектів.

Конкуруючі стандарти. В області EAI існує парадокс, що полягає в тому, що EAI-стандарти не є універсальними.

EAI – це парадигма. EAI не є інструментом, не існує універсальної архітектури EAI.

Побудова інтерфейсів це мистецтво. Інженерні рішення повинні бути обговорені з усіма основними користувачами системи для досягнення загального консенсусу відносно остаточного результату. Відсутність консенсусу по конструкції інтерфейсів приводить до витрат на перевірку дотримання вимог між різними системами.

Втрата деталей. Інформація, що здається неважливої на більш ранній стадії, може стати важливою пізніше.

Підзвітність. Через наявність великої кількості відомств, що відрізняються бізнес-процесами, що працюють із однієї EAI-системою, виникають суперечливі вимоги. Складно створити чітку звітність для остаточної структури системи.

У той же час, підтримці SOA-систем, що активно працюють із мережею Internet, заважає фактор постійного розвитку всесвітньої павутини. При цьому постійна зміна архітектури й розробка заново вже існуючих інтерфейсів вимагають безперервної роботи фахівців [8].

## 1.2 Веб-сервіси як сучасні засоби глобальних мереж

При розробці системи для мережі Internet основним об'єктом концепції SOA виступає веб-сервіс – веб-адресою програмна система, яка може бути ідентифікована веб-адресою, що оснащена стандартизованими інтерфейсами для взаємодії по стандартизованим протоколам [8].

Існує множина реалізацій SOA-архітектури: REST, RPC, DCOM, CORBA, SOAP. Зараз більшість програмних рішень, розроблених відповідно до сервіс-орієнтованої архітектури, найчастіше використовують комплекс веб-служб, взаємодіючих по протоколу SOAP.

SOAP (Simple Object Access Protocol – простий протокол доступу до об'єктів) – протокол обміну структурованою інформацією в розподільному обчислювальному середовищі для реалізацій веб-служб у комп'ютерних мережах. Протокол SOAP використовується для обміну повідомленнями формату XML і для віддаленого виклику процедур (RPC – Remote Procedure Call). Внаслідок чого є розширенням протоколу XML-RPC. SOAP взаємодіє з усіма протоколами прикладного рівня й найчастіше використовується поверх протоколу HTTP [8]. Структура протоколу SOAP [9] представлена на рис. 1.2.

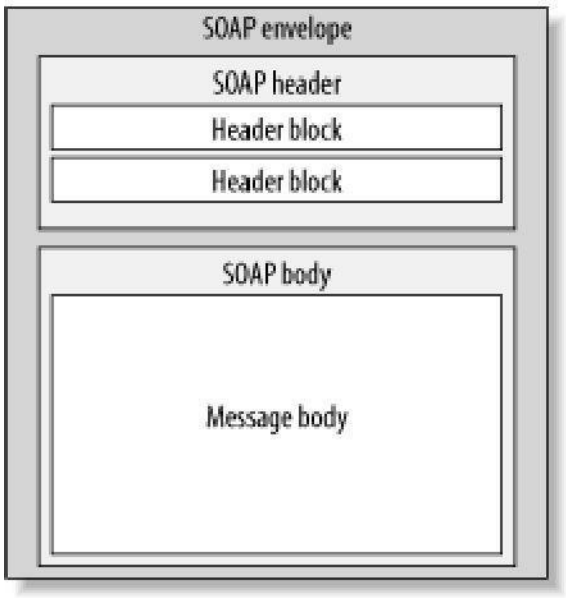


Рисунок 1.2 – Структура протоколу SOAP

WSDL (Веб Services Description Language – мова опису веб-сервісів) – дана специфікація мови XML призначена для створення документів, що докладно описують веб-сервіси й способи доступу до них (див. рисунок 1.3) [10].

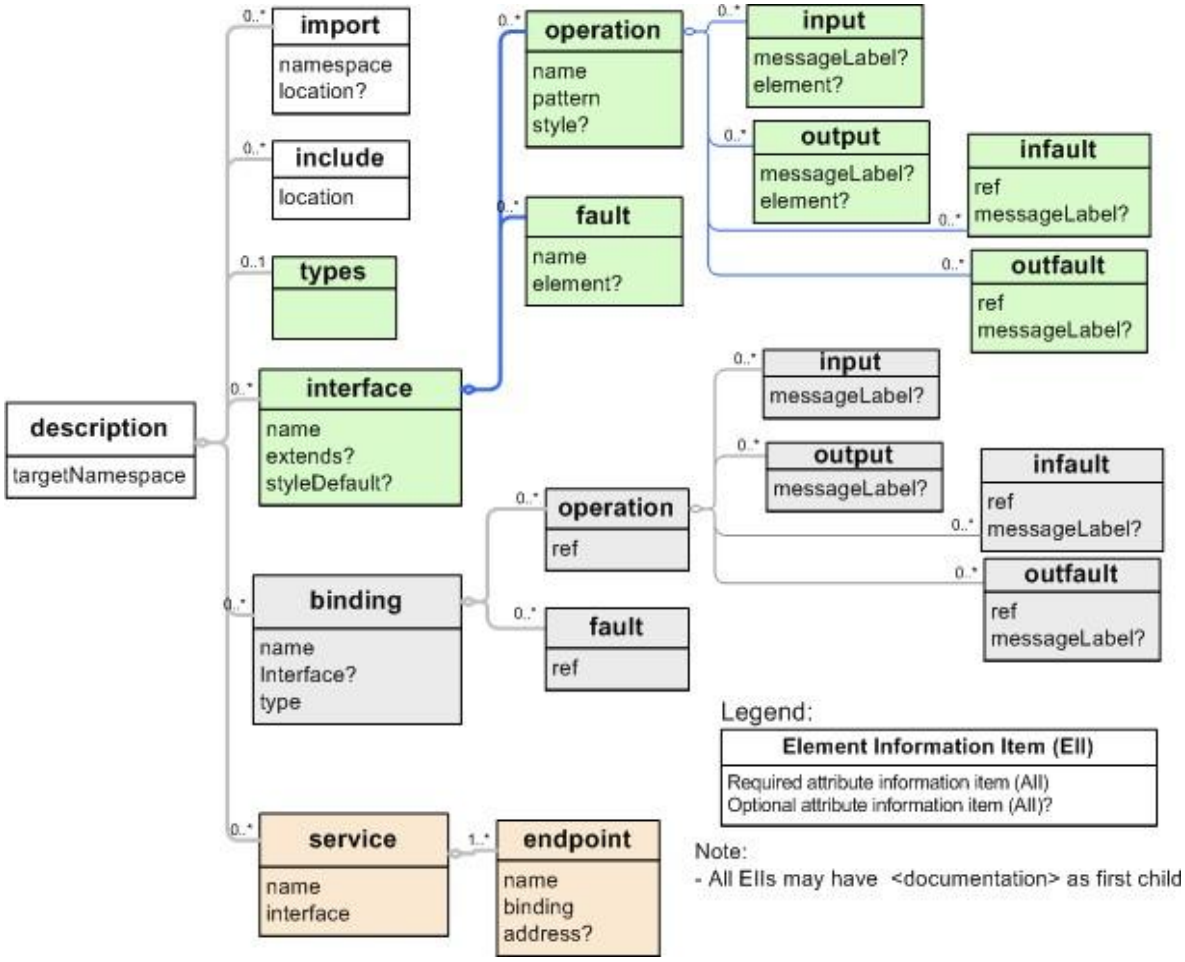


Рисунок 1.3. – Структура WSDL-документа у форматі XML [10]

Кожний документ WSDL містить наступну основну інформацію про веб-сервіси:

– Types – типи, що відправляються й отримується сервісом XML повідомлень;

– message – повідомлення, що посилають веб-сервісом;

– binding – спосіб, яким повідомлення буде доставлено.

Як недоліки варто відзначити, що застосування SOAP збільшує об'єм повідомлень і час обробки даних, необхідний час на побудову веб-сервісу на базі SOAP або інших інструментів [11].

## 1.3 Аналіз технологій Semantic Веб

Зі зростанням обсягу інформації в мережі Internet потрібно спростити доступ до даних веб-сервісів для наступної автоматизації процесів структурування й представлення інформації. Такий інтерфейс можна забезпечити за рахунок представлення знань про предметну область у формі онтології, доступної як інструмент технологій Semantic Веб.

Semantic Веб (семантична павутина) – це перспективна галузь в області розвитку Internet [12]. Основне завдання Semantic Веб – представлення інформації у вигляді, придатному для машинної обробки. Програмний агент може безпосередньо отримувати з павутини факти й виводити з них логічні висновки. Основними інструментами семантичної павутини є семантичні мережі, універсальні ідентифікатори ресурсів (URI – Uniform Resource Identifiers) і онтології [13].

Далі наведена докладна інформація про основні інструменти Semantic Веб [14]. Структуру Semantic Веб складається з множинаі мов опису, схвалених консорціумом W3C. Ця множина, крім іншого, містить у собі URI, XML, RDF,

RDF-S, OWL [15] (див. рисунок 1.4). Нижче наведений опис основних стандартів, що становлять базу понять Semantic Веб.

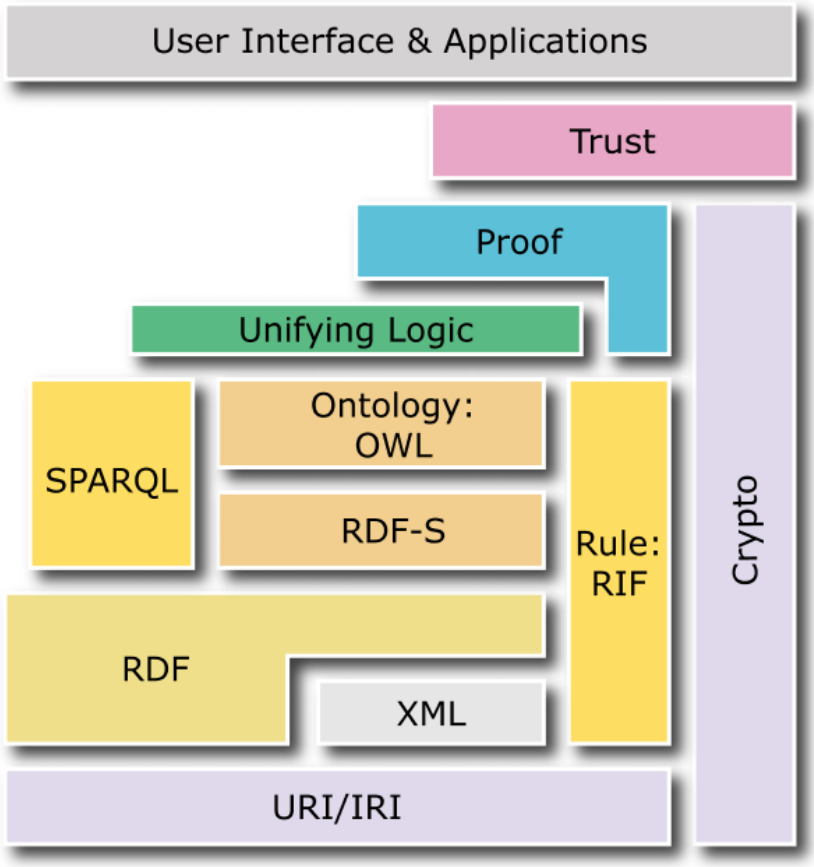


Рисунок 1.4 – Стек понять Semantic Веб [15]

URI – універсальний ідентифікатор ресурсів, звичайно використовується для посилання на об'єкт за допомогою протоколу HTTP у мережі Internet. У той же час в Semantic Веб універсальний ідентифікатор ресурсів використовується для позначення об'єктів. У тому числі й об'єктів матеріального світу, таких як країни, компанії, люди та інші, або абстрактних об'єктів: найменування компанії, ім'я, вік. Внаслідок того, що URI глобально унікальні, вони дозволяють ідентифікувати ті самі об'єкти в різних місцях середовища Semantic Веб [14].

XML – це розширювана мова розмітки, що не несе семантичного навантаження. Синтаксис мови XML дозволяє визначати стандарти опису документа для полегшення машинної обробки. Структура XML-документа визначається за допомогою специфікації XML Schema. Завдяки схемі документа,

сформованій специфікацією XML Schema, стає можливим проаналізувати будь-який XML-документ на правильність його структури [16].

RDF – стандарт, що описує графи, у яких дуги й вузли містять URI. Формат опису RDF-даних має вигляд триплету «суб'єкт-відношення-об'єкт» (Рисунок 1.5). Елементами RDF-триплету є URI [17].

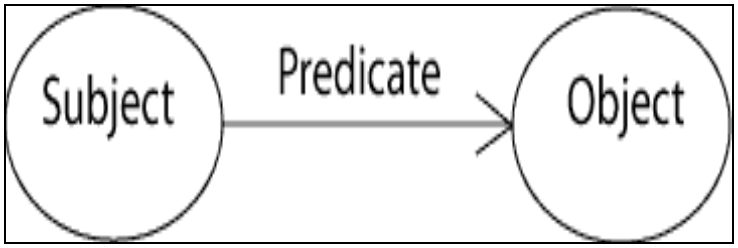


Рисунок 1.5 – Схема формату «суб'єкт-відношення-об'єкт»

Консорціум W3C створив XML-схему для опису RDF-документів. У той же час для RDF існує аналог XML Schema – RDF-S. Нижче представлений приклад графа RDF-документа, що описує частину бізнес-процесів Internet-магазину [14] (див. рисунок 1.6).

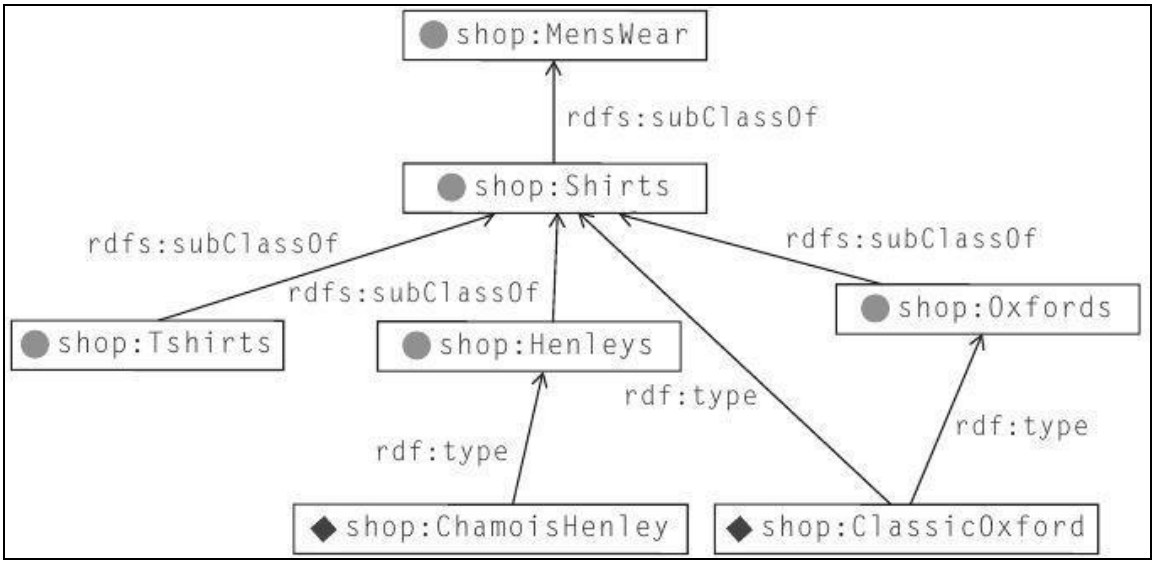


Рисунок 1.6 – Приклад графа RDF-документа

RDF-графи можна використовувати для роботи з онтологіями [19], описаними за допомогою RDF-S і OWL. Цей підхід забезпечить можливість отримувати з RDF-документів логічні висновки. Внаслідок чого є можливість використовувати стандарти Semantic Веб для взаємодії баз знань і веб-сервісів.

## 1.4 Бази знань і дескриптивні логіки для опису концептуальних моделей

База знань (Knowledge Base, KB) – це опис, що покриває певну предметну область. Звичайно KB зберігає у собі не тільки фактичну інформацію, але й правила відображення для інтелектуального аналізу інформації. Під інтелектуальним аналізом інформації маються на увазі автоматичний умовивід про факти, що вводяться, і відображення нових фактів. Бази знань найчастіше застосовуються для побудови інтелектуальних систем [20]. У той же час бази знань можуть використовуватися для створення експертних систем зберігання даних, таких як документації й керівництва організацій, статті технічного забезпечення.

Інтелектуальна система – це технічна або програмна система, яка здатна вирішувати завдання, що традиційно вважаються творчими, що належать до конкретної предметної області, знання про яку зберігаються в пам'яті такої системи [20].

Головна відмінність бази даних (Database, DB) [21] від бази знань – це відповідно припущення про закритість і про відкритість світу. Те, на що DB може дати відповідь «ні», KB може відповісти «невідомо». Крім того, реляційні бази даних працюють із мовами запитів, рівні за потужністю логіці предикатів.

Онтологією називається ієрархічний спосіб представлення в базі знань набору понять і їх відносин [22]. Основні складові частини онтологій – це екземпляри, поняття, атрибути й відношення. Далі – складові розглядаються докладніше.

Поняття або класи – абстрактні набори об'єктів. Вони містять у собі екземпляри або підкласи. Наприклад, у поняття «Людство» може бути вкладене поняття «Люди». Класи в онтологіях створюють таксономічну ієрархію понять [23].

Індивіди або екземпляри являють собою конкретні матеріальні або ідеальні об'єкти. Екземпляром поняття «Людей» може бути «Сергій». У той же час «Сергій», залежно від завдань онтології, може бути вкладеним поняттям [24].

Класи й індивіди в онтології можуть містити атрибути, що використовуються для зберігання інформації. Інформація специфічна для об'єктів і прив'язана до них. Кожний атрибут має ім'я й значення. Наприклад, поняття «Чоловік» має наступні атрибути: Назва: Чоловік; Клас: Людей; Стать: Чоловіча.

Атрибути визначають відносини між об'єктами онтології. У більшості випадків відношенням стає атрибут об'єкта зі значенням у вигляді іншого об'єкта.

Опис двомісних відносин, що утворюється між об'єктами, називається ролями. Нижче наведений приклад об'єктів в онтології і єднальних ролей. Припустимо, в онтології перебувають два екземпляри класу «Марія» і «Сергій». Нехай Сергій – брат Марії, тоді роль між «Сергій» і «Марія» визначимо як атрибут «брат» зі значенням «Сергій» для об'єкта «Марія».

За допомогою онтологій є можливість створювати загальні й приватні твердження. Наприклад, «будь-який Чоловік є Людиною», «існують Викладачі, що є Професорами».

Онтології підрозділяються на загальні й прикладні. Загальні або базисні онтології містять універсальні для більшості предметних областей поняття. Такий тип онтологій має на увазі наявність базового набору термінів. Ці терміни є правилами для наступного опису більш конкретних об'єктів предметних областей. Існують практичні реалізації технологій уніфікації онтологій, де використовуються загальні онтології [24]. Прикладні онтології містять знання про об'єкти якої-небудь конкретної предметної області. Такі онтології є більш модифіковані стосовно загальних онтологій. Прикладні онтології містять у собі термінологію зі спеціальними для цієї області термінами, що не враховують омоніми з інших предметних областей. Прикладом омонімів може служити слово «провідник». У предметній області «Фізика», «Провідник» відрізняється за значенням від того ж слова предметної області «Транспорт».

У якості мови опису онтологій використовуються дескриптивні логіки (Description Logics, DL) [24]. Дескриптивні логіки містять у собі багаті виразні можливості логіки предикатів і прийнятні обчислювальні властивості: можливість розв'язання й невисоку обчислювальну складність завдань логічного аналізу.

Ці фактори забезпечують можливість застосовувати DL на практиці для роботи з SOA-системами. Дескриптивні логіки є розв'язними, але менш виразними фрагментами логіки предикатів. По синтаксису DL схожі на модальні логіки. Мінімальними вимогами до виразної сили мови для побудови моделі SOA-системи є опис концептуальної моделі даних у базах даних і складових їх відносин.

Дескриптивні логіки створювалися з метою розширити семантичні мережі й фреймові структури механізмами формальної логіки. Споконвічно DL мали назву «термінологічні системи», але в 19809 отримали назву – «дескриптивні логіки».

Дескриптивні логіки користуються поняттями онтологій, такими як «ролі» і «концепти». В інших розділах математичної логіки поняття «концепт» відповідає поняттю «одномісний предикат». Поняття «роль» відповідає поняттю «двомісний предикат» або «бінарне відношення» [25].

У поняттях DL знання KB діляться на:

– Tbox – набір термінів загального виду, званий термінологією, що здатні описати предметну область або концептуальну модель даних, наприклад модель «сутність-зв'язок» (Entity-Relationship, ER) [25]. ER-модель даних використовується при проектуванні реляційних баз даних [25];

– Abox – набір тверджень про індивідів приватного виду.

Розглянуто кілька типів аксіом на прикладі синтаксису дескриптивної логіки *ALC*. Припустимо, *A* і *B* – деякі концепти або ролі. Аксіома вкладеності – це вираз виду $A \in B$. Цей вираз буквальне означає, що концепт або роль A включені в концепт або роль B. Аксіома еквівалентності – це вираз виду $A \equiv B$, еквівалентність можна представити як дві аксіоми $A \in B$ і $B \in A$.

Аксіома кон'юнктивності – це вираз виду $A \wedge B$. Аксіома кон'юнктивності відповідає за операцію перетинання з теорії множин.

Для отримання знань із онтологій, написаних за допомогою дескриптивних логік, використовуються алгоритми логічного аналізу (Reasoners – машини висновку). Машини висновку дозволяють автоматизовано виводити знання з онтологій і робити інші операції з онтологіями. Прикладом логічного аналізу є перевірка відсутності протиріч, відображення нових знань із уже існуючих, відповідь на запити до баз знань. Семантика й синтаксис дескриптивних логік створювалися з метою можливості розв'язання основних логічних проблем.

Розглянуто докладніше основні види логічного аналізу [24].

Аналіз сумісності бази знань – перевіряє наявність мінімум однієї моделі в наборі КВ (Tbox, Abox). По такому ж принципу можна перевіряти окремо сумісність термінології, аналізуючи Tbox без Abox.

Аналіз виконуваності концепту – забезпечує перевірку на виконуваність певного концепту в рамках цільового Tbox.

Аналіз вкладеності концептів – перевіряє включення певного концепту в аналізований концепт щодо цільового Tbox.

Класифікація – будує ієрархію концептів від певного концепту до всіх вкладених атомарних концептів у рамках цільового Tbox.

Відповідь на кон'юнктивні запити до бази знань – знаходить існуючі індивіди, що задовольняють певному запиту щодо цільової КВ (Tbox, Abox). Кон'юнктивні запити схожі на запити до баз даних. На теперішній момент не існує ефективних машин висновку, здатних давати відповідь на більш складні запити до бази знань, ніж кон'юнктивні.

Є множина реалізацій машин висновку, що різняться по наступних основних параметрах:

– формат вхідних і вихідних даних;

– тип алгоритму логічного аналізу;

– підтримувані логіки;

– мова програмування й інші інструменти реалізації;

– можливість інтегрування з редакторами онтологій і реалізаціями онтологічних систем.

Таблиця 1.1 – Характеристики машин висновку

| Найменування машини висновку | Формат даних | Алгоритм логічного аналізу | Підтриму-вані логіки | Мова програмування | Інтеграція |
|---|---|---|---|---|---|
| Hermit | OWL 2 | hypertableau algorithm | SROIQ | Java Редактор | Онтологій Protégé |
| Pellet | OWL 2 | tableau algorithm | SROIQ | Java Редактор | Онтологій Protégé |
| Fact++ | OWL 2 | tableau algorithm | SROIQ | C++ Редактор | Онтологій Protégé |
| Racerpro | OWL 1 | tableau algorithm | SHIQ | LISP Редактор | Онтологій Protégé |

Можливість логічного аналізу виникає завдяки формалізації онтологій за допомогою дескриптивних логік. Складність логічного аналізу залежить від виразності, що роздільної здатності і інших характеристик. Нижче наведені фундаментальні характеристики тієї або іншої дескриптивної логіки, по яких можна аналізувати витрати на розробку машин висновку.

Повнота – це властивість скінченної моделі (finite model property) [26]. Наявність властивості повноти в дескриптивної логіки означає, що для цієї логіки легко проводиться логічний аналіз на основі розв'язаних процедур. Обчислювальна складність логічного аналізу дескриптивних логік – характеристика, що визначає залежність необхідного об'єма пам'яті й часу для виконання алгоритму від розміру вхідних даних.

Обчислювальна складність – одна з головних проблем у розвитку дескриптивних логік. Дослідження оптимізації алгоритмів починається із визначення класу обчислювальної складності завдання, яке алгоритм повинен вирішувати [27]. Розуміння принципів зменшення складності в дескриптивних логіках дозволяє не тільки вибрати відповідну логіку для опису онтологій, але й використовувати ці знання при проектуванні бази знань [28].

Завдання побудови SOA-системи доцільно вирішувати за допомогою застосування парадигми заснованої на онтологіях доступу до даних (OBDA – Ontology-Based Data Access). OBDA – це доступ до даних через високорівневий інтерфейс онтології [11]. Центральним процесом заснованим на онтологіях доступу до даних є представлення бази даних мовою дескриптивної логіки [29]. Цей підхід дозволяє використовувати переваги онтологій і технологій дескриптивних логік. При цьому відкривається можливість побудови інтерфейсів на базі семантичної сервіс-орієнтованої архітектури (SSOA) [30], чим забезпечується подальший ефективний розвиток технології Semantic Веб.

Реалізація OBDA-системи повинна дозволяти отримувати дані з Abox, що зберігаються в базах даних. При цьому ефективність доступу до даних досягається за рахунок відображення запитів у середовищі реляційної СУБД, що має ефективні засоби для обробки запитів і інші переваги. Tbox повинен використовуватися для забезпечення необхідної семантичної інформації на більш високому концептуальному рівні. Запропонована схема дозволяє здійснювати міркування на Tbox і Abox незалежно друг від друга [31]. Потрібно враховувати фактори відмінності бази даних і бази знань, наведені вище, при розробці інтерфейсних OBDA-засобів, здатних отримувати базу знань на основі традиційної бази даних [32].

Для того щоб додати формальну семантику властивостям стандартних концептуальних моделей, таких як ER- і UML-діаграми, і забезпечити ефективний доступ до великого обсягу даних, що зберігаються в базах даних, було введене сімейство мов дескриптивної логіки Dl-lite [33]. Мови дескриптивної логіки Dl-lite спеціалізовані під завдання заснованого на онтологіях доступу до даних. Обчислювальна складність усіх завдань логічного аналізу в основних логіках сімейства Dl-lite не перевершує поліноміальної за рахунок низької виразної сили, достатньої для опису ER-моделей. Для відповіді на запити складність не перевищує Logspace за рахунок алгоритмів переписування запитів. Під переписуванням запитів мається на увазі перетворення запиту в об'єднання кон'юнктивних запитів на відносинах реляційної бази даних.

Відповідь на кон'юнктивний запит CQA($A$,$T$,$q_{з}$), де $q_{з}$ – кон'юнктивний запит до даних з Abox $A$ деякої KB, що залежить від Tbox $T$, зводиться до завдання виконання запиту до бази даних: QE($A$, $q_0$), де $q_0$ – запит у рамках логіки предикатів, що не залежить від Abox і здатний бути реалізованим у мові реляційних баз даних, наприклад SQL. Проблема даного підходу полягає в тому, що запит може суттєво розростися.

Фахівцями ведеться активна робота з поліпшення цих алгоритмів у наступних напрямках: спрощення первинного запиту за допомогою редукції; виключення несуттєвих або не заданих у схемі баз даних кон'юнктивних запитів [34].

Нижче наведена інформація про варіанти реалізації онтологій і OBDA-технологій. OWL (Веб Ontology Language – мова веб-онтологій) – мова опису онтологій для технології Semantic Веб [35]. Мова OWL дозволяє описувати веб-документи й об'єкти дійсності у форматі дескриптивних логік (див. табл. 1.2). Кожному об'єкту онтології мовою OWL прив'язується URI. У той же час існують методи для опису веб-сервісів за допомогою OWL [36]. Мова OWL заснована мовою DAML+OIL, має XML формат, отже, вірне твердження, що OWL є XML-синтаксисом деяких дескриптивних логік.

Таблиця 1.2 – Відповідність понять в OWL і дескриптивних логіках

| Дескриптивні логіки | OWL |
|---|---|
| Концепт | Клас |
| Роль | Властивість |
| Індивід | Об'єкт |
| База знань | Онтологія |

Будь-який OWL-документ є RDF-документом. У той же час OWL дозволяє створювати нові схеми RDF на основі існуючих [35].

Специфікація OWL містить кілька профілів, різних по виразності. OWL DL містить велику виразну силу, при цьому має властивості повноти й прийнятної

обчислювальної складності. OWL DL дозволяє описувати онтології в рамках дескриптивної логіки SHOIN. Нижче наведені можливості OWL DL [35], крім стандартних можливостей дескриптивної логіки ***ALC***:

- аксіоми діз'юнктності, імплікації, еквівалентності, інверсії класів;
- булеві комбінації виражень класів;
- обмеження спільності й існування властивостей;
- обмеження кардинальності властивостей;
- транзитивні, симетричні, функціональні властивості.

OWL Lite і OWL Full відповідно – урізані й розширені версії OWL DL [30]. OWL Lite не містить складних обмежень і менш виразний, ніж OWL DL. Отже, OWL Lite має більш низьку обчислювальну складність. OWL Full містить величезну виразну силу й синтаксис, близький до RDF. У той же час OWL Full не гарантує можливість проведення логічного аналізу.

Розроблена нова версія мови веб-онтологій OWL 2 [36]. OWL 2 має схожу структуру й сумісний з OWL 1. Ці фактори забезпечують можливість інтегрувати онтології формату OWL 1 в онтології формату OWL 2, але не навпаки. У той же час OWL 2 додає нові функціональні можливості й кілька нових профілів. OWL 2 розширює виразну силу до формату дескриптивної логіки SROIQ. Поява цього фактора призвело до ослаблення обмежень стосовно OWL DL. Нижче наведені нові можливості OWL 2 у порівнянні з OWL 1 [37]:

- кваліфікації обмежень на число елементів;
- розширений набір типів і діапазонів даних;
- ключі;
- більш пророблені описи анотацій;
- ланцюжок властивостей;
- непересічні властивості;
- асиметричні властивості;
- рефлексивні властивості.

Для OBDA-систем консорціумом W3C запропонований стандарт OWL 2 QL як профіль мови веб-онтологій OWL 2, що входить до складу технологій Semantic

Веб і заснований на дескриптивній логіці Dl-lite R. Важлива відмінність між сімейством Dl-lite і OWL 2 QL полягає в статусі припущення унікального імені (UNA): UNA прийняте в рамках логіки Dl-lite, але не прийняте в OWL 2. Замість цього синтаксис OWL 2 забезпечує явні засоби для того, щоб сказати, що a і b це той самий індивід (same as) або ж навпаки (different from).

Нижче наведені короткі описи деяких реалізацій OBDA-систем, працюючих із сімейством Dl-lite [33].

Система Quonto (Querying Ontologies) допускає підключення популярних зовнішніх реляційних СУБД, використовуючи для опису онтологій логіку сімейства Dl-lite. Вона має оптимізовані засоби міркувань за допомогою онтологій і обробки запитів даних. Система здатна взаємодіяти з популярними СУБД і редактором онтологій Protege. Крім того, тут є графічний інтерфейс – Qtoolkit . На основі Quonto створені наступні продукти: Rowlkit для онтологій, написаних мовою OWL 2 QL; The Mastro для онтологій мовою дескриптивної логіки Dl-lite.

Requiem (Resolution-based Query rewriting for Expressive Models) – реалізація алгоритму переписування запитів до виразних моделей. Алгоритм генерує менш надлишкові запити до DB, ніж Quonto. Requiem можна також використовувати для більш виразних дескриптивних логік, ніж сімейство Dl-lite. Однак Requiem усе ще демонструє низьку ефективність на більших реальних обсягах даних. Він не захищений від зациклення й вимагає додаткової інформації для підвищення швидкості переписування запитів.

## 1.5 Постанова завдань дослідження

Результати проведеного аналізу цієї області дозволяють вибрати мову веб-онтологій OWL 2 QL для створення методів побудови SOA-систем на базі заснованого на онтологіях доступу до даних [31].

Таким чином сформульоване завдання кваліфікаційної роботи. Розробити методи створення SOA-систем на базі переваг, що використовують OBDA, онтологій і технологій дескриптивних логік. Розробити алгоритм, що здатний автоматично будувати інтерфейси взаємодії веб-сервісів за допомогою логічного аналізу.

Забезпечити можливість змінювати структуру SOA-системи без втручання спеціальної тривалої підготовки. Уможливити побудову інтерфейсів на базі семантичної сервіс-орієнтованої архітектури (SSOA), забезпечуючи подальший ефективний розвиток технології Semantic Веб.

Наведений аналіз сучасних технологій для проектування програмного забезпечення в рамках парадигми SOA. Наведена класифікація аналогічних технологій і підходів. Розглянуті мови опису взаємодії елементів SOA-систем.

Виявлені недоліки існуючих підходів до проектування інтерфейсів веб-сервісів і EAI-систем.

Проаналізовані основні види структур взаємодії SOA-сервісів.

Наведений короткий огляд експериментальних реалізацій систем OBDA, що працюють із сімейством дескриптивних логік Dl-lite. Наявність таких реалізацій підтверджує практичне застосування дескриптивних логік і мов веб-онтологій з метою надання онтологічного доступу до даних.

Необхідно створити формалізм опису взаємодії сервісів і алгоритмів побудови інтерфейсів веб-сервісів для реалізації ефективної SOA-системи із застосуванням парадигми заснованого на онтологіях доступу до даних та розробити програмну систему, що моделює такі процеси.

# 2 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ДОСЛІДЖЕНЬ

## 2.1 Опис моделей SOA-систем

У якості основної теоретичної концепції використовується сервіс-орієнтована архітектура (SOA), що є парадигмою розробки програмного забезпечення за допомогою застосування розподілених слабко пов'язаних компонентів [2]. При розробці системи для мережі Internet основним об'єктом концепції SOA виступає веб-сервіс – програмна система, що може бути ідентифікована веб-адресою, оснащена стандартизованими інтерфейсами для взаємодії по стандартизованих протоколах. Основою роботи веб-сервісів може послужити представлення певних даних DB веб-сервісу. У тому випадку якщо сервіс пропонує послуги з обробки даних, він може запитувати дані в іншого сервісу й обробляти їх.

Для побудови інтерфейсу $I_{SOA}$ між двома веб-сервісами необхідно мати наступну інформацію у вигляді трійки значень:

$$I_{SOA} = <C_s,Q_s,S_s>,$$

де $C_s$ – параметр, що містить адресу веб-сервісу, що запитує інформацію (адреса клієнтської частини);

$Q_s$ – множина запитуваних інформаційних ресурсів;

$S_s$ – параметр, що містить адресу веб-сервісу, що запитує інформацію (адреса серверної частини).

Нехай $C_s$, $S_s$ – це параметри, що містять інформацію про адреси веб-сервісів, $Q_s$ – множина переданої між цими веб-сервісами інформації. Тоді $Service(C_s, Q_s, S_s)$ – відповідність інтерфейсів взаємодії веб-сервісів $C_s$ і $S_s$, що виражає структуру інтерфейсів, реалізованих між сервісом за адресою $C_s$ і сервісом за адресою $S_s$ інформації, що й передає, $Q_s$ між ними.

Для роботи системи автоматичної побудови інтерфейсів веб-сервісів необхідно описати концептуальну модель взаємодії веб-сервісів.

На рис. 2.1 наведено приклад концептуальної моделі композитних сервісів і сервісних кластерів в SOA-системі [2].

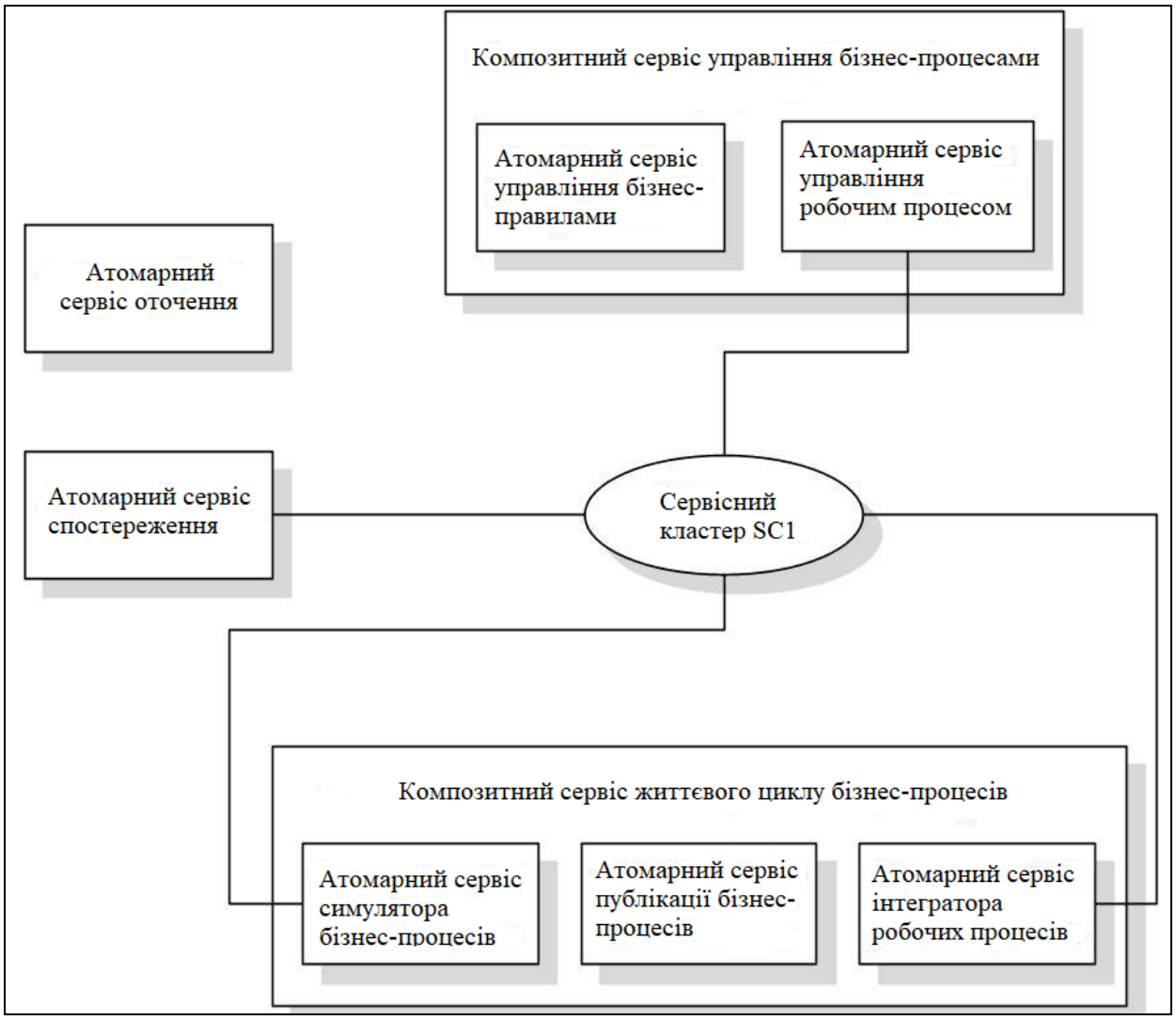


Рисунок 2.1 – Приклад концептуальної моделі композитних сервісів і сервісних кластерів керування бізнес-процесами

Виходячи з наведеного прикладу, можна описати ER-діаграму [30] взаємодії сервісів в SOA-системі (див. рисунок 2.2).

При автоматизації побудови веб-інтерфейсів на основі вхідних параметрів для SOA-архітектури потрібно побудувати цього математичну формалізацію елементів сервіс-орієнтованої архітектури побудови веб-інтерфейсів.

Сервіси усередині SOA-системи можуть належати до кількох категорій [2], певними розроблювачами, що наприклад розподіляють сервіси по територіальних і функціональних ознаках.

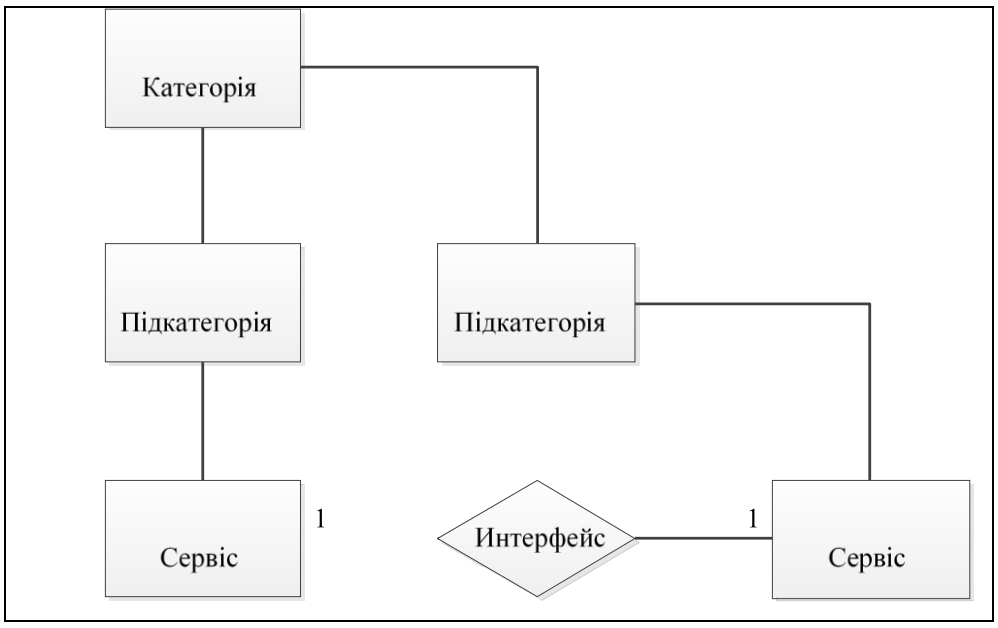


Рисунок 2.2 – Приклад ER-діаграми взаємодії веб-сервісів

Категорії, у свою чергу, можуть включатися в інші категорії. Кожний сервіс може мати інтерфейс взаємодії з іншими сервісами. Отже, категоризацію композитних сервісів і сервісних кластерів можна представити у вигляді наступної формалізації в рамках теорії множин.

Якщо сервіси $b_s$, $b_{s1}$, $b_{s2}$ є елементами множини $B_s$, певні категорії $B_s$, $B_{s1}$, $B_{s2}$, $B_{s3}$ є множинами, що входять в універсальну множину $U$, інтерфейси належать множині відносин $R_s$, то система множин, що описує SOA-архітектуру, підпадає під наступні обмеження.

Сервіс bs повинен входити в певну категорію ($bs \in Bs$).

Вкладення підкатегорії в категорію повинне відповідати наступній вимозі:

$$B_{s1} \subseteq B_{s2}.$$

Вкладення підкатегорії в кілька категорій повинне відповідати наступній вимозі:

$$Bs1 \subseteq Bs2 \cap Bs3.$$

Вкладення декількох підкатегорій в одну категорію повинне відповідати наступній вимозі: $Bs1 \cup Bs2 \subseteq Bs3$.

Непересічні категорії повинні відповідати наступній вимозі: $Bs1 \cup Bs2 = \varnothing$.

Інтерфейс повинен відповідати наступній вимозі: $(bs1, bs2) \in Rs$.

Тоді автоматизована побудова інтерфейсів виражається у вигляді наступної відповідності.

Відповідність є відображенням області визначення *Service(Cs, Qs, Ss)* в область значень

$$Servi(c_i, q, s_i), ISOA$$

де $C_s$, $Q_s$, $S_s$ – параметри для множинного створення сервісів;

$I_{SOA}$ – елемент множини *ISOA*;

*ISOA* – множина інтерфейсів, які підходять по даним параметрам.

## 2.2 Аналіз методів дескриптивної логіки

Мова дескриптивної логіки DL так само, як і інші мови логік, представляються за допомогою синтаксису й семантики. Синтаксис логіки – це правила, що описують вирази, правильно побудовані в рамках конкретної дескриптивної логіки [24].

Для формулювання синтаксису будь-якої мови DL задається набір символів (алфавіт). За допомогою цього набору створюються вирази в рамках конкретної мови. Крім того, кожна мова дескриптивної логіки містить набір символів, що дозволяють побудувати складені концепти й ролі з атомарних концептів і ролей [24].

Прикладами складових синтаксису для побудови складених концептів можуть бути наступні символи в таблиці 2.1, де *R* – атомарна роль; *C, D* –

Наведені операції дозволяють виконати композиції концептів, наприклад у дескриптивній логіці операції об'єднання й перетинання схожі на диз'юнкцію й кон'юнкцію обчислення предикатів першого порядку, хоча позначаються інакше, відрізняючись від обчислення вищих порядків. Типізація концептів надає можливість використовувати їх поліморфічне (багатозначне) тлумачення [36].

Наведено приклад синтаксису одного з основних [24] сімейств мови дескриптивної логіки – ***ALC*** [3]:

<B, ⊤, ⊥, ¬C, ∃R.C, ∀R.C, C ⊓ D, C ⊔ D>, де B – атомарний концепт. У логіці ***ALC*** концептами є наступні й тільки наступні вирази:

– будь-який атомарний концепт B;

– вирази ⊤ і ⊥;

– доповнення ¬C концепту C;

– вирази ∀R.C і ∃R.C концепту C і ролі R;

– перетинання C ⊓ D і об'єднання C ⊔ D концепту C і D.

Для інтерпретації синтаксичних виразів використовується семантика дескриптивної логіки, що надає виразам формальне значення, що й указує, які вирази вважаються дійсними й неправильними.

Семантика DL задається в термінах теорії множин на підставі того, що будь-який концепт представляється множиною денотатів, а роль суть бінарне відношення, тобто множина пар [36].

Інтерпретація *I* у дескриптивній логіці – двійка:

$$I = <\Delta, \bullet I>,$$

де Δ – домен даної інтерпретації, що є непустою множиною;

• I – інтерпретуюча функція, що зіставляє будь-якому атомарному концепту A деяка підмножину $A^I \subseteq \Delta$ і будь-якої атомарної ролі R якусь підмножину

$$R^I \subseteq \Delta \times \Delta.$$

Якщо всі твердження Abox є здійсненними в інтерпретації *I*, те Abox Формалізація моделі стосовно бази знань виглядає в такий спосіб. I ⊨ α, де I – це модель α, α – твердження або аксіоми. Моделлю I ⊨ K бази знань K = <T, A> є інтерпретація I, якщо I ⊨ T і I ⊨ A. K, що містить α, що записується в такий спосіб:

K ⊨ α, якщо всі моделі K также є моделями α. У той же час Tbox T, що включає в себе α, записується як T ⊨ α, якщо всі моделі T є моделями α.

Інтерпретація більш специфічних складових у кожній мові DL своя

Інтерпретацію *I* для цих виразів. Область Δ – множина усіх біологічних особин. Концепти «Людина», «Чоловіча Стать» – відповідно велика кількість людей і сутностей чоловічої статі. Інтерпретація ролей «єДитина» і «єБрат» проводиться за допомогою двумісних відносин, що зв'язують усяку сутність із її дитиною й усяку сутність із її братом відповідно. Отже, пара <*e,d*> належить відношенню єБрат, якщо d є братом e.

Чим більше елементів конструкції в синтаксисі мови дескриптивної логіки, тим більше виразність цієї мови. Збільшення виразності мови може приводити до зростання обчислювальної складності завдань логічного аналізу [14]. Через багату виразність логіка може стати нерозв'язною [24].

Розширення логіки – це невеликі зміни базової мови дескриптивної логіки з метою посилення виразності логіки. У табл. 2.3 представлено кілька базових мов DL і їх розширення [24].

Розширення логік найчастіше іменуються за допомогою додавання до найменування базової мови першої букви з найменування доданого елемента. Назви розширень сімейства логік *DL-Lite* будуються за допомогою застосування надрядкових і підрядкових знаків.

Підрядкові знаки зазвичай показують, які фрагменти вирахування предикатів першого порядку вони реалізують. Наприклад, назва *Dl-lite* $_{krom}$ походить від Кромовського фрагменту [16] вирахування предикатів першого порядку. Надрядкові знаки показують, які елементи конструкції містять у собі дане розширення. Необдумане розширення дескриптивної логіки може привести до появи проблем можливості розв'язання, якщо в розширенні дескриптивної логіки ***ALC*** присутні елементи конструкцій ***S*** і ***H***. При цьому додається елемент конструкції ***Q*** або ***N***. Якщо не накладати обмеження на використання транзитивних підролей, то розширення стає нерозв'язним [24]. При додаванні нових елементів конструкції в мову, зростає обчислювальна складність алгоритмів логічного аналізу. Складність ***SHOIN*** знаходиться в класі EXPTIME. Це означає, що час обробки онтології алгоритмами логічного аналізу

експоненціально залежить від обсягу онтології та інших вхідних параметрів. Це значення є негативним чинником при виборі даної мови для розробки прикладної онтології. З іншого боку, OWL-DL, заснований на цьому розширенні, надає високі виразні можливості для опису онтологій. Після виходу нової версії мови веб-онтологій – OWL 2, виразність виросла до ***SROIQ*** і обчислювальна складність зросла до NEXPTIME [32]. У той же час відомо, що основні завдання при такій складності вирішуються за прийнятну кількість часу й за допомогою невеликого обсягу затрачуваної комп'ютерної пам'яті. Для стабільної швидкої обробки онтологій потрібні полегшені мови дескриптивної логіки.

Наприклад, OWL-lite використовує розширення логіки $\boldsymbol{ALCSHIF}^{(D)}$. Мова онтологій OWL 2 QL використовує логіку DL-Lite ***H***core, псевдонімом якої є Dl-lite ***R***. Основні завдання Dl-lite ***H***core мають логарифмічну складність [11]. Цей фактор – одна із причин того, що для опису онтології SOA-системи використовувалася логіка Dl-lite$\boldsymbol{REL}^{++}$, яка є основою для мови веб-онтологій OWL 2 EL. Логіка EL за рахунок низької виразності дозволяє зберігати й обробляти велику кількість аксіом. Прикладом є медична онтологія SNOMED, що нараховує більш ніж 300 000 аксіом.

Для зменшення проблем з обробкою знань онтології необхідно вибирати ту або іншу мову або його розширення, виходячи з мінімально необхідних елементів конструкції для опису цільової предметної області.

## 2.3 Формалізація онтологій веб-сервісів

Потрібно встановити, чи можна описати модель роботи SOA-системи засобами DL [32]. Для цього співставлено інтерпретації конструкцій дескриптивної логіки, еквівалентні конструкціям мови теорії множин для елементів SOA-архітектури, наведеним раніше.

Потрібно встановити необхідну виразність мови дескриптивної логіки для формального опису SOA-системи на основі необхідних елементів конструкції:

- включення концептів;
- перетинання з правої сторони аксіоми;
- об'єднання з лівої сторони аксіоми;
- диз'юнкція концептів;
- повне екзистенціальне обмеження.

Необхідна виразність формального опису SOA-системи відповідає виразності логіки ***ALUE***. У той же час використання дескриптивної логіки ***ALUE*** надає додаткову виразність. Найбільш значуща з можливостей – це елемент конструкції заперечення концепту. Заперечення концепту дозволяє позначити, які інтерфейси й сервіси не повинні створюватися, а спроба їх створення є помилкою.

## 2.4 Запити в рамках архітектури OBDA

При побудові онтологічної моделі SOA-системи слід урахувати активну роботу SOA-архітектури з базами даних (БД) і можливість використання БД для обробки більших обсягів даних [1]. Таку систему можна побудувати за допомогою технології OBDA [21]. Цей підхід використовує переваги DL для роботи з онтологіями з метою витягу запитів до БД [1]. Основний принцип роботи систем OBDA полягає в трансформації (переписуванні) запитів до бази знань у відповідні запити до БД . Бази даних у технології OBDA є структурою Abox дескриптивної логіки [21].

Внаслідок того, що формування реляційних БД описується вирахуванням предикатів першого порядку, необхідно дати визначення запитів вирахування предикатів першого порядку. Запит вирахування предикатів першого порядку являє собою формулу вирахування предикатів першого порядку з рівністю. Позначимо запит $q$ як формулу вирахування предикатів першого порядку з

вільними змінними *x*. Розмір *x* є арністю запиту *q*. Інтерпретація *I* для *qi* є множиначю кортежів елементів. Ці кортежі, при присвоюванні значення вільним змінним, роблять формулу $\varphi$ вірною в *I* [32].

Булевим запитом називається запит, який не включає вільних змінних, інакше кажучи, це замкнена формула. Внаслідок того, щобулевий запит виглядає в такий спосіб: $=\theta\{|\varphi\}$, можливо позначити його як $\varphi$. Серед різних видів запитів найпростіше працювати з кон'юнктивними запитами [33] і диз'юнкцією кон'юнктивних запитів Кон'юнктивний запит до бази знань *K* – кон'юнктивний запит, атоми якого мають вигляд *A*(z) або $P(z_1, z_2)$, де *A* и *P* – відповідно атомарні концепти й атомарні ролі в *ДО*, z, $z_1$, $z_2$ – константи в *ДО* або змінні.

Нехай запит *q* – кон'юнктивний запит або об'єднання кон'юнктивних запитів. Відповідь на *q* до бази знань *K* є множина *ans*(*q*, *K*) кортежів констант, що входять у *K*, для кожної моделі $M_c$ у $K_c$. За визначенням зазвичай *ans*(*q*, *K*), тому що база знань *K* кінцева і число констант, що входять у *K*, скінченно. Кортеж може бути порожнім у випадку, коли *q* є булевим кон'юнктивним запитом. У цьому випадку множина *ans*(*q*, *K*) складається з єдиного порожнього кортежу, тоді й тільки тоді, коли формула *q* вірна в кожній моделі *K*.

Відповідь на кон'юнктивний запит CQA(*A*,*T*,$q_c$), де $q_з$ – кон'юнктивний запит до даних з Abox *A* деякої бази знань, що залежить від Tbox *T*, зводиться до завдання витягу запиту до БД: QE(*A*, $q_0$), де $q_0$ – запит у рамках вирахування предикатів першого порядку, що не залежить від Abox, і здатний бути відбитим у мові реляційних БД, наприклад SQL. Існує множина алгоритмів відповідей на запити в рамках технології OBDA [33]. Проблема більшості підходів до відповіді на запити полягає в тому, що запит може суттєво розростися [34].

Застосування запитів до бази знань для реалізації представлення *Servirend*. Трійка аргументів *Servi*(c, *q*, *s*) має статичну структуру. Цю структуру зручно зберігати у вигляді схеми бази даних. У цьому випадку параметри *C*, *Q*, *S* будуть Obdao. Запитами до Abox, що дозволяє зберігати велику кількість структурованої інформації про сервіси в Abox і використовувати швидкість БД для доступу до неї.

Значення для області визначення відповідності називаються вхідними параметрами із залежними умовами, якщо x ∈ xC', x ∈ xs', де x – деякий елемент умови запиту. В інших випадках значення для відповідності називаються вхідними параметрами без залежних умов.

Приклад для параметрів без залежних умов. Потрібно побудувати інтерфейси (Клієнт $C_s$, Дані $Q_s$, Сервер $S_s$), де $C_s$ – сайт навчального порталу (Educationportal) міста Рівне (Rivnecity). В веб-сервісі *Cs* з веб-сервісів *Ss* будуть відправлятися дані *Qs*, що містять інформацію про номери телефонів установ (Phone). Веб-сервіси $S_s$, у свою чергу – це утворювальні сайти навчальних закладів (Education) міста Рівне (Rivnencity).

$$Cs(x_1, x_2) \leftarrow \text{Educationportal}(x_1), \text{Rivnecity}(x_2);$$
$$Qs(x_3) \leftarrow \text{Phone}(x_3);$$
$$Ss(x_4, x_5) \leftarrow \text{Education}(x_4), \text{Rivnearea}(x_5).$$

Далі наведений приклад для параметрів із залежними умовами. Запит адреси сервера й запити адреси клієнта можуть містити однакові елементи умови, такі параметри називаються параметрами із залежними умовами.

Потрібно побудувати інтерфейси (Клієнт $C_s$, Дані $Q_s$, Сервер $S_s$), де на всіх сайтах навчальних порталів (Educationportal) кожного міста (City) центрального регіону (Centralregion) відобразяться дані про персонал (Personal) з веб-сервісу $S_s$.Веб-сервіси $S_s$, у свою чергу, – це навчальні сайти міста (City), зазначеного в $C_s$. Залежні умови позначаються однаковими змінними елементів умови запиту, у цьому прикладі така змінна – $x_3$.

$$Cs(x_1, x_2, x_3) \leftarrow \text{Educationportal}(x_1), \text{Centralregion}(x_2), \text{City}(x_3);$$
$$Qs(x_4) \leftarrow \text{Personal}(x_4);$$
$$Ss(x_5, x_3) \leftarrow \text{Education}(x_5), \text{City}(x_3).$$

Далі представлена оптимізація множинаі *ans* для відповідності *Servirend*. Ролі в онтологічній SOA-моделі використовуються для реєстрації інформації про інтерфейси, що додаються. Множина аксіом і тверджень бази знань визначена *K*, що містять ролі, як *Rbox*. При обробці вхідних параметрів відповідності *Servirend* інформація про ролі не потрібна. Отже, немає необхідності враховувати *Rbox* при виконанні реалізації представлення *Servirend*.

Для інтерпретації відповіді на запит до бази даних без обліку *Rbox*. Нехай запит $q$ – кон'юнктивний запит або об'єднання кон'юнктивних запитів, а база знань $K_c$ = *K/Rbox*, тоді відповіддю на $q$ до $K_c$ є множина *servans*($q$, $K_c$) кортежів констант, що входять у $K_c$,.

Відсутність *Rbox* зменшує розмір *Tbox* і *Abox* що частково вирішує проблему розростання запиту при відповіді на OBDA запит.

Особливості опису моделі обраними мовами дескриптивної логіки

Для опису Obda-онтологій доцільно використовувати логіку *DL- Lite**R*** сімейства логік *Dl-lite*. Ця мова підходить для завдань OBDA.

На логіці *Dl-lite* ***R*** заснована мова OWL 2 QL. Це – профіль мови OWL 2, розроблений консорціумом W3C, що входить до складу формального інструментарію технологій Semantic Веб. Характерною рисою сімейства *Dl-lite* ***R*** є відсутність можливості визначення функціональних залежностей і числових обмежень між концептами. Дані інструменти виразності не можуть бути використані в OWL 2 QL з- за припущення про унікальність імен (UNA – Unique Name Assumption). UNA прийнято у базовій *Dl-lite*, але не прийняте в OWL2. Це, у свою чергу, може підвищити клас обчислювальної складності завдань логічного висновку [33].

Розглянуто синтаксис і семантику *Dl-lite* ***R*** псевдоніма логіки *DL- Lite**H**core*. Якщо $A_d$ – атомарні концепти, $P_d$ – атомарні ролі, $P^{-d}$ – інверсія атомарної ролі $P_d$, то концепти й ролі в мові *Dl-lite* ***R*** формуються за допомогою синтаксису:

$$Bd \rightarrow Ad \mid \exists R,\ Cd \rightarrow Bd \mid \neg BdR_d \rightarrow P_d \mid P^{-}_d,$$

Концепт Bd називається базовим концептом. Базовий концепт складається з атомарного концепту або концепту у формі ролі ∃Rd, де ∃ – елемент конструкції неповного екзистенціального обмеження. Rd – базова роль, яка може містити атомарну роль і інверсію атомарної ролі.

Концепт $C_d$ – загальний концепт, який може бути базовим концептом або його запереченням. *Ed* – загальна роль, яка може бути базовою роллю або її інверсією.

Tbox *T* у логіці *Dl-lite* ***R*** формується за допомогою кінцевої множинаі аксіом вкладеності виду:

$$Bd \sqsubseteq Cd;\ Rd \sqsubseteq Ed.$$

Для вкладеності концептів у лівій частині аксіоми може перебувати тільки базовий концепт, у правій частині аксіоми може перебувати головний концепт. Для вкладеності ролей у лівій частині аксіоми може перебувати тільки базова роль, у правій частині аксіоми може перебувати головна роль.

Abox формується з кінцевого набору тверджень приналежності до атомарних концептів і атомарним ролям: *Ad*(*ad*); *Pd*(*ad, bd*).

Інтерпретація I є моделлю Cd1 ⊑ Cd2, де Cd1, Cd2 – головні концепти, якщо Cd1I ⊆ Cd2I. Аналогічно I є моделлю E1 ⊑ E2, де E1, E2 – головні ролі, якщо Ed1I ⊆ E d2I. Для того щоб описати семантику тверджень, необхідно розширити інтерпретуючу функцію константами за допомогою присвоєння кожній константі ad окремої конструкції ad I ∈ ΔI. Цей фактор має на увазі дотримання припущення про унікальність імен констант. Інтерпретація I є моделлю твердження Ad(ad), якщо ai ∈ AI. Інтерпретація I є моделлю твердження Pd(ad, bd), якщо (ad I, bd I) ∈ Pid.

Незважаючи на невелику виразність Dl-lite ***R*** ця мова здатна описувати основні елементи концептуального моделювання формалізмів, використовуваних у базах даних і для розробки програмного забезпечення, таких як ER-моделі й UML діаграми класів.

Далі наводиться приклад опису ER-моделі обмеження доступу користувачів до тем або іншим видам послуг вебисервісів засобами логіки Dl-lite***R*** (див. рис. 2.3).

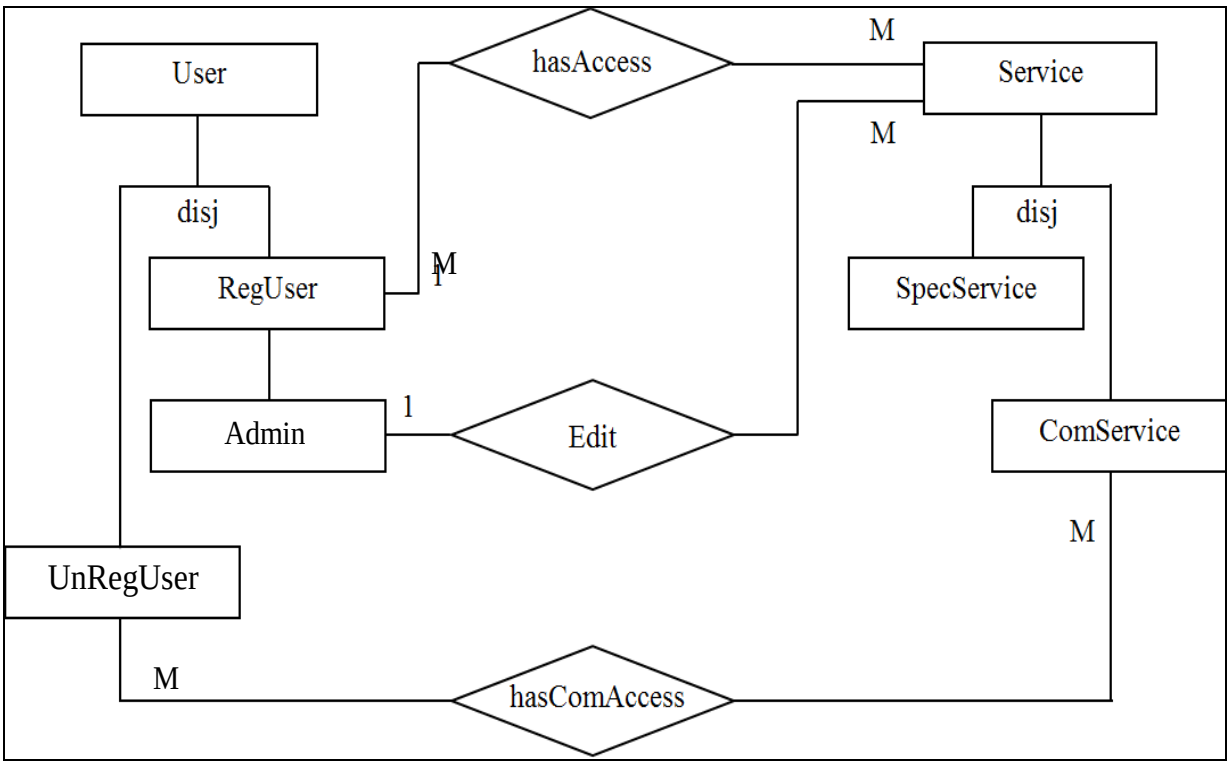


Рисунок 2.3 – ER-діаграма доступу до послуг веб-сервісу

У розглянутій ER-моделі користувачі діляться на зареєстрованих (Reguser) і незареєстрованих (Unreguser). Послуги (Service) діляться на спеціальні (Specservise) і загальні (Comservice). Незареєстровані користувачі мають доступ до загальних послуг (hascomaccess). Зареєстровані користувачі мають доступ до всіх послуг (hasaccess), включаючи доступ до спеціальних послуг. Зареєстрований користувач може бути адміністратором (Admin), що редагують (Edit) послугу.

Підсутності ER-моделі описуються за допомогою включення концептів.

Reguser $\sqsubseteq$ User, Unreguser $\sqsubseteq$ User, Admin $\sqsubseteq$ Reguser,

Comservice $\sqsubseteq$ Service, Specservice $\sqsubseteq$ Service.

Аналогічно описуються підзв’язки за допомогою включення ролей:

$$\exists \text{hascomacces} \sqsubseteq \exists \text{hasaccess}.$$

У логіці Dl-lite ***R*** немає можливості повністю виразити диз'юнкцію. Проте, є можливість описати диз'юнкцію концептів і ролей за рахунок негативного включення:

Reguser ⊑¬ Unreguser, Comservice ⊑¬specservice.

Для зазначення зв'язку між двома сутностями, що беруть участь у відношенні, використовуються еквівалентність ролі з концептом, що виражають першу сутність, і еквівалентність інверсії цієї ролі з концептом, що виражають другу сутність. Еквівалентність в Dl-lite ***R*** реалізується за допомогою взаємної імплікації двох сигнатур.

Клас обчислювальної складності логіки *Dl-lite* ***R*** менше, ніж логіки ***ALUE***. Внаслідок цих факторів для опису онтологічної SOA-моделі ефективніше використовувати логіку *Dl-lite* ***R***.

Не зважаючи на те, що *Dl-lite* ***R*** має низьку виразність, для редукції до підтримуваних мовою конструкцій можна використовувати приведення до доступних елементів конструкції без збільшення обчислювальної складності.

# 3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

## 3.1 Логічний аналіз онтологій SOA-систем

Для створення алгоритмів логічного аналізу онтологічної SOA- системи потрібно розглянути основні алгоритмічні завдання, пов'язані з обраними мовами дескриптивної логіки й парадигмою OBDA. Визначення цих завдань [29].

Несуперечність бази знань – перевірка на існування хоча б однієї моделі в KB $K$ = <Tbox, Abox>.

Вкладеність концептів або ролей – перевірка на існування $B^I \subseteq C^I$ у довільній інтерпретації I для заданого Tbox, де C, B – концепти або ролі.

Перевірка екземпляра класу – перевірка на існування $a^I \in A^I$ у довільній моделі I бази знань K, де A – концепт; a – індивід. Перевірка екземпляра ролі визначається відповідно.

Повернення відповіді на запити до KB.

Щоб уникнути поганих результатів роботи алгоритмів логічного аналізу необхідно використовувати явні критерії вибору мови дескриптивної логіки для рішення завдань конкретної предметної області. Такими критеріями можуть послужити обчислювальна складність і завершуваність алгоритмів.

При наявності фактора завершуваності алгоритм видає результат своєї роботи через кінцевий час.

Обчислювальна складність алгоритму визначає залежність необхідного об'єма пам'яті й часу для виконання цього алгоритму від розміру вхідних даних [30]. Це – одна з головних проблем у розвитку DL. Дослідження оптимізації алгоритмів починається з витягу класу обчислювальної складності завдання, яке алгоритм повинен вирішувати.

Далі наведені класи складності, розповсюджені в області розробки алгоритмів для дескриптивної логіки:

$$\text{Logspace} \subseteq \text{Nlogspace} \subseteq \text{P} \subseteq \text{np} \subseteq \text{exptime} \subseteq \text{Nexptime}.$$

Клас Logspace містить завдання, для рішення яких детермінованій машині Тьюрінга [30] буде потрібно обсяг пам'яті, що перебуває в логарифмічній залежності від вхідних значень.

Приставка N у назвах класів Nlogspace, NP і Nexptime означає, що ці класи містять завдання, які можуть розв'язатися за витрати ресурсів, що відповідають їхньому класу складності з назвою без приставки на недетермінованій машині Тьюрінга. Тому що недетермінована машина Тьюрінга – усього лише абстрактна модель, то алгоритм цього класу навряд чи зможе розв'язатися на реальному комп'ютері, відповідному до детермінованої машини, за такий же час.

Клас P містить всі ті проблеми, рішення яких детермінованою машиною Тьюрінга поліноміально залежить від розміру входу.

Exptime – клас, що містить завдання, які детермінована машина Тьюрінга вирішує за кількість часу, що перебуває в експонентній залежності від вхідних значень.

Спочатку дослідження складності висновку в дескриптивній логіці були більш зосереджені на класі P без розгляду нерозв'язних NP-проблем. Це давало гарантовану ймовірність того, що алгоритм буде працювати швидко. Однак, надалі було визнано [30], що бази знань реального розміру можуть бути оброблені в розумний термін, незважаючи на високий клас складності – Exptime або вище. Обчислювальна складність гіршого випадку логічного аналізу логіки ***ALUE*** на значенню Exptime, логіки *Dl-lite* ***R*** – Ptime .

Розуміння принципів зменшення складності в дескриптивних логіках дозволяє не тільки вибрати підходящий язик для опису веб-онтології, але й використовувати ці знання при проектуванні КВ.

Визначено КВ *K* як двійку: *K* = <*Tbox*, *Abox*>, де *Tbox* – множина аксіом; *Abox* – множина тверджень. Вхідними параметрами, що впливають на обчислювальну складність завдань логічного висновку є: розмір *Tbox*, розмір *Abox*, розмір запиту – число параметрів запиту *q*.

Для завдань логічного висновку розрізняють три види обчислювальної складності:

– складність відносно даних: складність оцінюється як функція від розміру *Abox*. Це використовується для мов, що спеціалізуються на більших розмірах *Abox*. Наприклад, для завдань онтологічного доступу до БД;

– складність щодо онтології й даних: складність оцінюється як функція від розміру *Tbox* і *Abox*. Це використовується в тому випадку, коли *Tbox* може бути дуже більшим і його розміром не можна зневажати;

– комбінована складність: складність оцінюється як функція від розміру *q*, *Tbox*, і *Abox*. Це звичайно застосовується для дуже виразних мов.

Зниження обчислювальної складності алгоритмів звичайно досягається за рахунок обмеження виразності мови. Також складність можна знизити за рахунок винаходу алгоритмів для конкретної предметної області.

Розглянуто ці випадки на прикладі логіки *Dl-lite* ***R***. Складність щодо онтології й даних для стандартних завдань висновку досягає Ptime за рахунок відсутності можливості прив'язки ролей до конкретних концептів і обмеження можливості виразити диз'юнкцію концептів [30]. Складність відносно даних для алгоритмів відповіді на запити в *Dl-lite* ***R*** досягає Logspace за рахунок методів трансформації запитів. Вирахування предикатів першого порядку є фактично логікою реляційних баз даних. Проблема оцінки запитів першого порядку до БД входить у клас завдань $AC^0$ [31]:

$$AC^0 \subsetneq \text{Logspace},$$

де $AC^0$ – клас обчислювальної складності схем [31].

У теоретичній інформатиці обчислювальна складність схем є підкласом теорії складності обчислень, у якій булеві функції класифікуються залежно від розміру й глибини булевих схем – математичних моделей, які обчислюють ці функції. $AC^0$ – клас завдань, обумовлений за допомогою сімейства булевих схем постійної глибини й поліноміального розміру.

Цей клас може бути породжений детермінованою машиною Тьюрінга за час, що логарифмічно залежить від розміру вхідних даних [30].

Алгоритми логічного аналізу для логіки *Dl-lite*. ролі *Pd*.

Інтерпретація *db*(*A*) є мінімальною моделлю Abox *A*. Існує можливість зводити завдання перевірки несуперечності бази знань і відповіді на запити до оцінки запиту логіки першого порядку до Abox. Для цього розглядається запит до реляційних БД, інакше кажучи, запит до *db*(*A*).

Здійснимість у мові дескриптивної логіки *L* зводиться до вирахування предикатів першого порядку, якщо для кожного Tbox *T*, вираженого в *L*, існує логічна приводимість запиту *q* в *T*. Крім того, для будь-якого непустого Abox *A* – <*T, A*> повинна бути здійсненна тоді й тільки тоді, коли *q* = «неправда» в *db*(*A*)

Необхідно взяти до уваги взаємодію між позитивними й негативними включеннями. Для цього будується спеціальна Tbox, що замикає негативні включення стосовно множинаі позитивних включень:

– *can(K)* – модель *K* тоді й тільки тоді, коли *K* здійсненна.

– *K* здійсненна тоді й тільки тоді, коли *db(A)* являє собою модель <*cln*(*T*), *A*>.

– інтерпретація *db*(A) являє собою модель усіх тверджень в *cln*(*T*) тоді й тільки тоді, коли *K* здійсненна.

Таким чином, на основі наведених логічних рівнянь запропоновано алгоритм, що використовує зведення до вирахування предикатів першого порядку. Схема алгоритму *Consistent* представлена на рис. 3.1.

Вхідним значенням алгоритму є база знань логіки *Dl-lite* $\boldsymbol{R}$ *K* = <*T,A*>, вихідним значенням є «істина», якщо база знань *K* несуперечлива.

Далі описується принцип роботи розробленого алгоритму. Для перекладу із тверджень в *cln*(*T*) до формул вирахування предикатів першого порядку використовується функція перекладу δ. Алгоритм обчислює *db*(*A*) і *cln*(*T*), витягає над *db*(*A*) булевий запит вирахування предикатів першого порядку, отриманий за допомогою об'єднання всіх формул вирахування предикатів першого порядку й повернутих після застосування функції *δ* до кожного твердження в *cln*(*T*). Таким чином, у випадку відсутності негативних тверджень включення, що входять у *K*, qdb(A)unsat = ⊥ db(A), де qunsat – запит, що перевіряє існування тверджень. Отже, *Consistent*(*K*) повертає значення «істина».

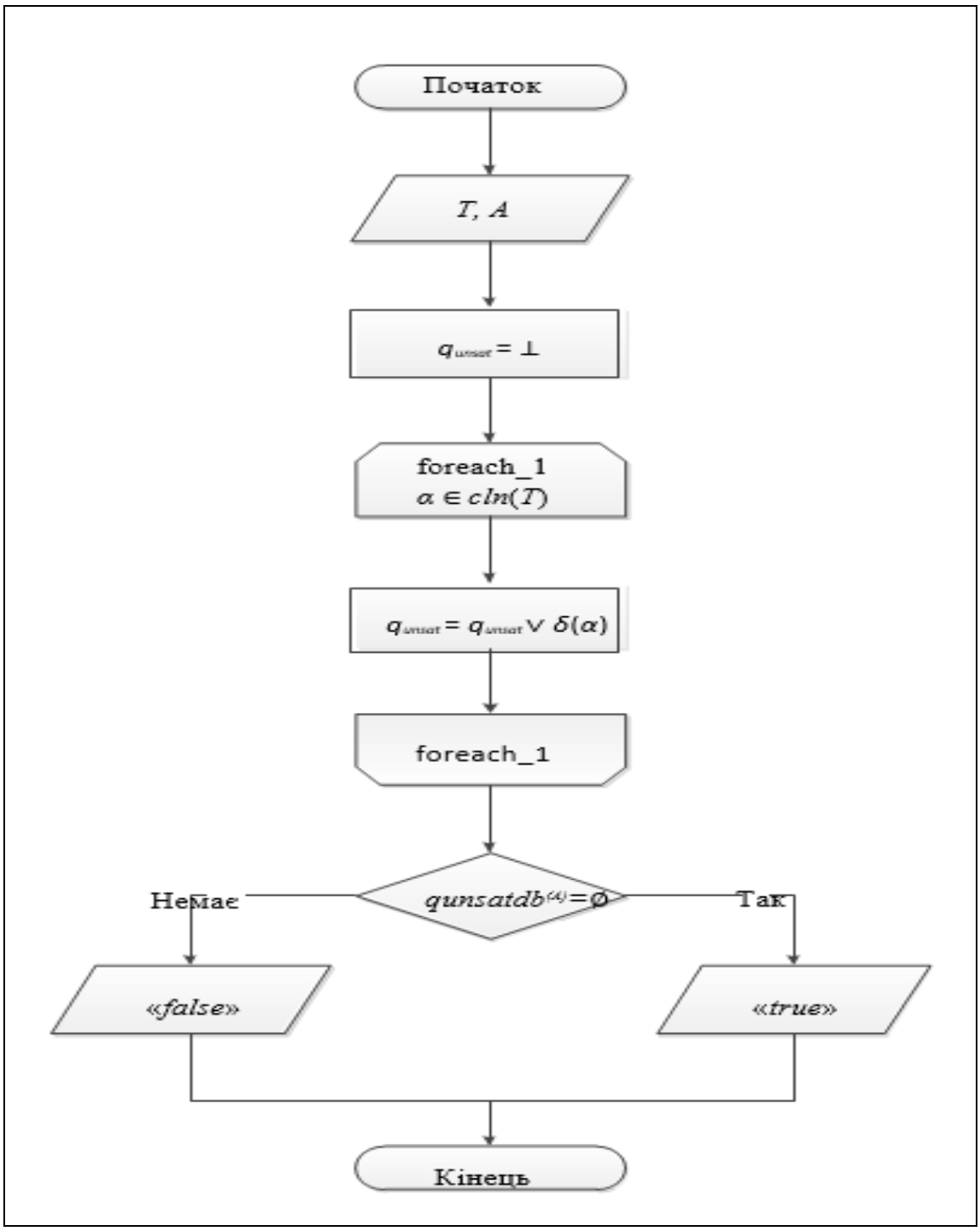


Рисунок 3.1 – Схема алгоритму зведення до вирахування предикатів першого порядку

Коли $K$ є суперечливою базою знань, відповідь на об'єднання кон’юнктивних запитів $Q$ визначається як кінцева множина кортежів [24], що не є достовірною відповіддю на запит. Внаслідок сказаного перед використанням специфічних алгоритмів логічного аналізу необхідно перевіряти базу знань на суперечливість.

Внаслідок того, що вкладеність концептів C ⊑ D еквівалентна нездійсненності концепту C⊓ ¬D достатнє рішення інших завдань зводити до завдань здійснимості [24].

Внаслідок зведення завдань включення й перевірки екземпляра до завдання перевірки несуперечності бази знань показник обчислювальної складності для завдання перевірки несуперечності бази знань поширюється й на завдання включення й перевірки екземпляра класу.

Завдання вкладеності концептів або ролей перебуває в класі Ptime залежно від розміру Tbox. Завдання перевірки екземпляра перебуває в класі Logspace залежно від розміру Abox і в Ptime залежно від розміру всієї бази знань.

Для алгоритмів відповіді на запити до бази знань *K* логіки *Dl-lite**R***, де *Q* – об'єднання кон'юнктивних запитів над *K*, *q* – кон'юнктивний запит, вірні наступні властивості.

$$\text{ans(Q, K)} = \text{Qcan}^{(K)}.$$

Для кожної кінцевої інтерпретації $I_K$ існує таке, що для кожного *q* до *K* *ans*(*q*, *K*) = *qik*.

$$ans(Q, K) = \bigcup ans(qi, K); \; qi \rightarrow q .$$

## 3.2 Застосування алгоритмів онтологічного підходу

Потрібно виявити переваги онтологічного підходу до побудови SOA-системи в порівнянні з існуючими аналогами.

Швидкість виконання алгоритмів для завдань логічного висновку в технологій дескриптивної логіки вище, ніж у структурованих алгоритмів.

Можливість взаємодії SOA-системи з іншими алгоритмами логічного висновку, онтологіями і язиками DL. На прикладі використання логік *Dl-lite* ***R*** і

***ALC*** різних завдань, завдяки схожості принципів їх організації можна використовувати ефективність певних інструментаріїв цих логік для певних завдань.

Інтеграція SOA-системи з реалізаціями технології Semantic Веб і наступний їхній розвиток. Інтеграція можлива завдяки подобі мови веб-онтологій OWL 2 QL і логіки *Dl-lite* ***R***.

## 3.3 Розробка алгоритму трансформації запитів

Зручність й простота розширення SOA-системи – при збільшенні кількості елементів сервіс-орієнтованої системи – це не потрібне розширення алгоритмів новими структурами, потрібно тільки розширення онтології необхідними аксіомами й твердженнями.

Необхідна розробка алгоритмів для реалізації відповідності Serviren внаслідок того, що таких алгоритмів на базі дескриптивної логіки в цей час не існує. У той же час структуровані алгоритми, як показано раніше, не мають перевагу технологій DL.

Необхідно зменшити число породжених запитів у порівнянні з іншими алгоритмами логічного висновку. На відміну від стандартних алгоритмів логічного висновку, існує можливість створення алгоритму запиту до бази даних, оптимізованого під SOA-завдання. Цей алгоритм буде відрізнятися від інших алгоритмів меншим розміром породжуваних запитів до бази даних.

Для онтологічної SOA-системи необхідні специфічні алгоритми-CRUD на базі дескриптивної логіки.

Оптимізований алгоритм трансформації запитів до бази знань. Класичне рішення завдання відповіді на запити до бази знань логіки *DL- Lite* ***R*** складається з наступних пунктів: запит $q$ обробляється й формулюється на основі позитивних

включень в Tbox. Отриманий запит до Abox трансформується в запит $Q'$ до бази даних. Тут $Q'$ – об'єднання кон'юнктивних запитів, що звертаються до $db(A)$.

Нехай існують $q_i$ – запит вирахування предикатів першого порядку, $a_i$ – кортеж констант, A – Abox Dl-liteR, T – TBox DL-Lite R, Q – об'єднання кон'юнктивних запитів. Відповіді на запити в дескриптивній логіці Dl-lite R, що зводиться до виражень вирахування предикатів першого порядку. Цей фактор дійсний внаслідок того, що для кожного $Q$ і кожного $T$ існує запит $q_1$ до $T$

На рис. 3.2 представлено схему алгоритму с модифікованою функцією *gr Rewwr* (Rewriting Without Roles) алгоритму кон'юнктивного запиту, що формулює, з урахуванням включень *I з* Tbox T.

Принцип роботи алгоритму складається з наступних етапів.

Перевірити зміст у запиті q інформації про ролі.

Переформулювати атоми кожного кон'юнктивного запиту q ∈ PR' і одержати новий запит для кожного атома переформульованої формули.

Для кожної пари атомів $g_1$ і $g_2$, яка уніфікується й перебуває в тілі запиту $q$, обчислити кон'юнктивний запит $q' = reduce\ (q, g_1, g_2)$.

Загальна змінна – це змінна, що з'являється мінімум два рази в тілі запиту або константа. У той же час аргумент атома запиту називається незалежним, якщо він є нерозпізнаваною незагальною змінною. Незалежний аргумент атома позначається символом «_».

Функція *findr* видає результат «1», якщо в запиті перебувають атоми із двома аргументами, інакше результат рівний «0». Тут $q[g/g']$ означає кон'юнктивний запит, отриманий від $q$ атома, *що* замінює g *новим* атомом g'. Функція $\tau$ ухвалює в якості вхідного значення кон'юнктивний запит $q$ і повертає новий кон'юнктивний запит, одержуваний заміною кожного входження незалежної змінної в $q$ *символом* «_». Аргумент атома в запиті є залежним, якщо він відповідає розпізнаваній змінній або загальній змінній.

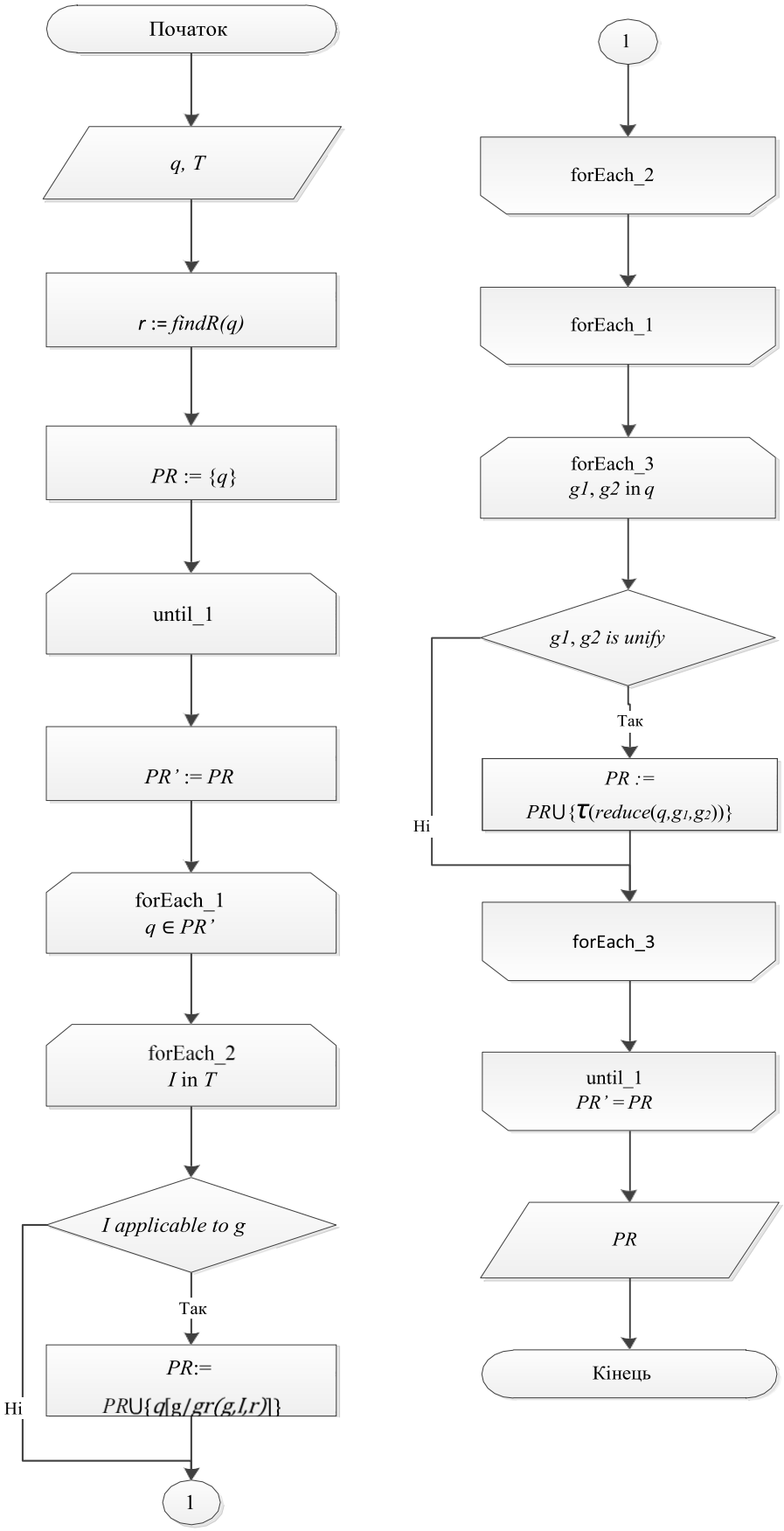


Рисунок 3.2 – Схема алгоритму Rewwr

Функція *reduce* ухвалює в якості вхідних параметрів кон'юнктивний запит $q$ і два атоми $g_1$ і $g_2$, що перебувають у тілі запиту $q$, і повертає кон'юнктивний запит $q'$ за допомогою застосування до $q$ уніфікатора MGU (Most General Unifier) для $g_1$ і $g_2$. При уніфікації $g_1$ і $g_2$ кожне входження символу «_» повинне бути розглянуте як присутність незалежної змінної. Уніфікатор MGU заміняє кожний символ «_» в $g_1$ на відповідний аргумент в $g_2$ і кожний символ «_» в $g_2$ на відповідний аргумент в $g_1$. Завдяки уніфікації, виконуваної за допомогою функції *reduce*, зв'язані змінні в $q$ можуть стати незв'язаними в $q'$. Таким чином, позитивні включення, які незастосовні до атомів $q$, пізніше можуть стати застосовними до атомів $q'$.

Далі представлена модифікація функції *gr* стандартного алгоритму перезапису запитів логіки *Dl-lite* $\boldsymbol{R}$ *Perfectref*. Модифікація функції спрямована на трансформацію запитів під створення інтерфейсів SOA-взаємодії, необхідну для збільшення швидкості роботи й зменшення розміру зайвої інформації в порівнянні із класичними алгоритмами відповідей на запити виражень логіки *Dl-lite* $\boldsymbol{R}$.

Модифікована функція *gr* представлена на рис. 3.3. Нехай $I$ – позитивне включення (розділ 2.6), $P$ – роль $P_d$ . Якщо запит не містить інформацію про ролі, то позитивне включення $I$ застосовне до атома $A(x)$, якщо є в правій частині $I$.

Якщо запит містить інформацію про ролі, то використовуються способи переформулювання атомів, наведені у функції *gr* для алгоритму трансформації запитів *Dl-lite* $\boldsymbol{R}$ Позитивне включення $I$ застосовне до атома $P$, залежно від наявності залежних або незалежних змінних в атомі $g$ або ролей в аксіомах $I$.

Далі наведені леми, необхідні надалі для доказу ефективності Sir-алгоритмів.

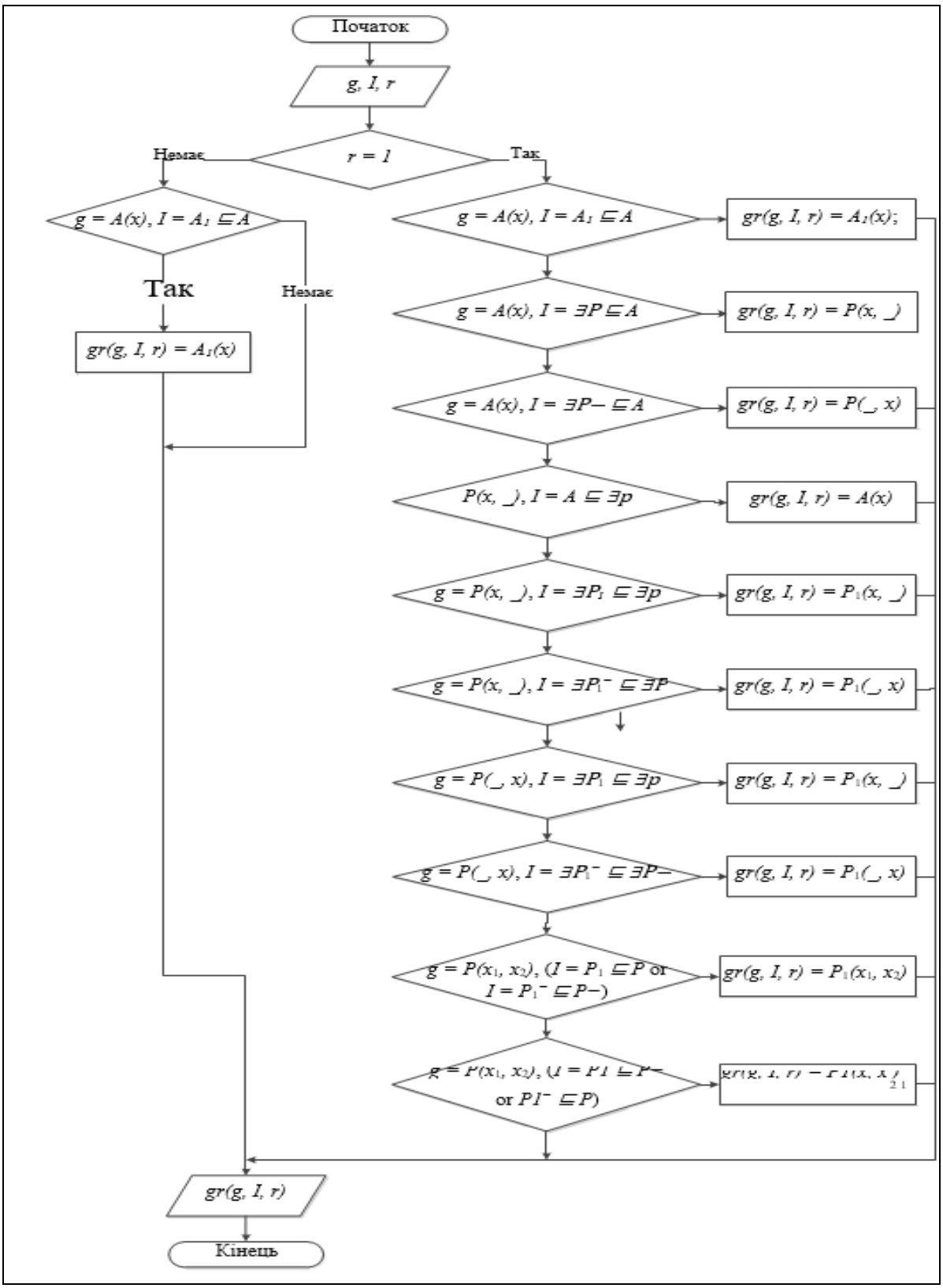


Рисунок 3.3 – Модифікація функції *gr*

Максимальне число атомів у тілі кон'юнктивного запиту, що генерується за допомогою алгоритму *Rewwr*, дорівнює довжині початкового запиту *q*. Довжина

запиту менше або рівна $n$, де $n$ – розмір запиту, тобто $n$ пропорційний числу атомів і числу компонентів запиту.

Кількість термів, що зустрічаються в кон'юнктивних запитах, сгенерованих за допомогою алгоритму *Rewwr*, дорівнює кількості змінних і констант, що входять в $q$, плюс символ «_». Отже, кардинальність такого набору менше або рівна $n + 1$, де $n$ – розмір запиту.

Число різних атомів, які можуть виникнути в кон'юнктивних запитах, що генеруються за алгоритмом, менше або рівно $m \cdot (n + 1)^2$, $m=m_1+m_2$, де $m_1$ – число концептів чи ролей у запиті. У той же час $m_2$ – число концептів чи ролей в TBox або, у випадку відсутності інформації про ролі у запиті, в TBox/RBox.

Алгоритм ураховує запити, які він згенерував.

Кількість різних кон'юнктивних запитів, сгенерованих алгоритмом, звичайно. З пункту 4 випливає, що алгоритм не генерує запит більш одного разу. Отже, алгоритм *Rewwr* завжди завершується.

Нехай $T$ – Tbox логіки *Dl-lite* $\boldsymbol{R}$, A – Abox логіки *Dl-lite* $\boldsymbol{R}$, а $q_{wr}$ – довільний кон'юнктивний запит без інформації про ролі до $T$, *RWR* – об'єднання кон'юнктивних запитів, одержуваних після роботи алгоритму *Rewwr*($q_{wr}$, $T$). Тоді для кожної несуперечливої бази знань $K_c$ = <$T$, $A$> дескриптивної логіки *Dl-lite* $\boldsymbol{R}$ вірно наступне: *servans*(q, $K_c$) = $RWR^{db(A)}$.

Число різних кон'юнктивних запитів, сгенерованих за допомогою алгоритму, менше або рівно $(m \bullet (n+1)^{2)\ n}$ і відповідає максимальній кількості виконань циклу until_1(рис 2.1). Тоді m перебуває в лінійній залежності від розміру Tbox у найгіршому разі, і в найкращому разі – від розміру Tbox/Rbox. У той же час n не залежить від розміру $T$. Отже, час виконання алгоритму *Rewwr* поліноміально залежить від розміру $T$.

## 3.4 Алгоритм трансформації параметрів

Алгоритм Dtrew для трансформації параметрів із залежними змінними в об'єднання параметрів без залежних змінних

Алгоритм *Dtrew* використовується, якщо вхідні параметри містять залежні елементи. Алгоритм *Dtrew* трансформує параметри із залежними змінними в об'єднання параметрів без залежних змінних. Принцип роботи алгоритму *Dtrew* представлений на рис. 3.4.

Алгоритм *Dtrew* включає підрахунок операцій, де $o_1$ – кількість операцій за один прохід циклу; $o_2$ – кількість операцій для кожного елементу колекції; $o_3$ – прості операції.

Алгоритм *Dtrew* працює наступним чином: якщо існують змінні (*varqc*) елементів умов запиту C, ідентичні змінним (*varqs*) умов запиту S, додати ці змінні в колекцію змінних залежних елементів (*Dtset*).

Задати значенню *C'S* значення *S*. Якщо в об'єднанні трансформованих запитів (*uc*) є змінні елементів умов запиту (*uc'*) колекції, що входять до складу, *Dtset*, то дописати цей елемент в *C'S*.

Помістити *C'S* в асоціативний масив *C'SDict[uc']*. Найгіршим випадком для алгоритму *Dtrew* є випадок, коли всі перевірки умови «if» будуть дійсні.

Далі представлений опис нового алгоритму, що повертає *Servii(ci, q, si)*, для наступної побудови інтерфейсів i, p для взаємодії веб-сервісів. Вхідними параметрами алгоритму запитів до онтології є Tbox *T* і трійка $i_{SOA}(C, Q, S)$, що називається далі Sir-триплет. Вихідними даними алгоритму є об'єднання *Sirset*, Sire Триплетів (*Sirtrip*) із запитами до баз даних. На рис. 3.5 представлена схема алгоритму *SIR1*.

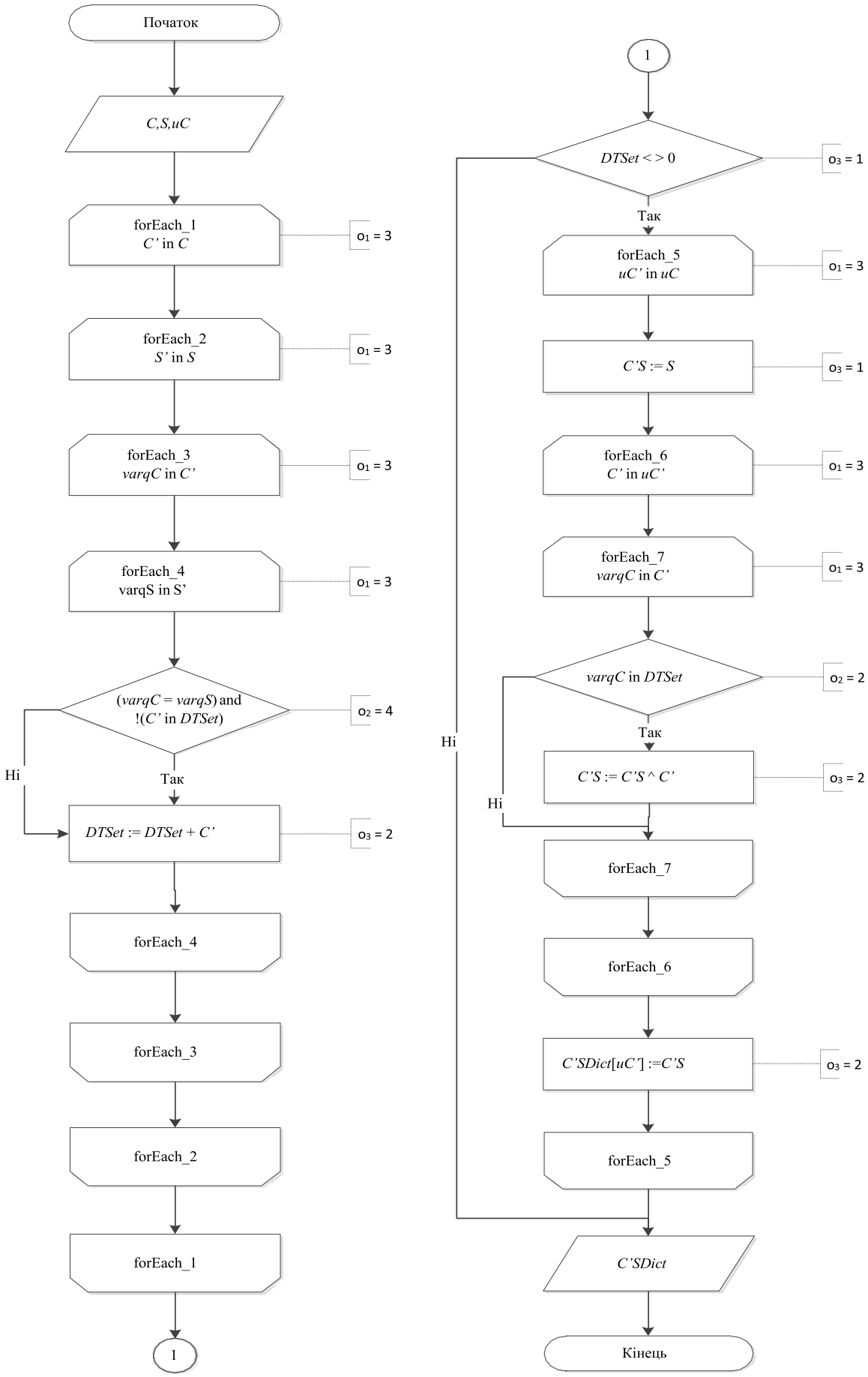


Рисунок 3.4 – Схема алгоритму *Dtrew*

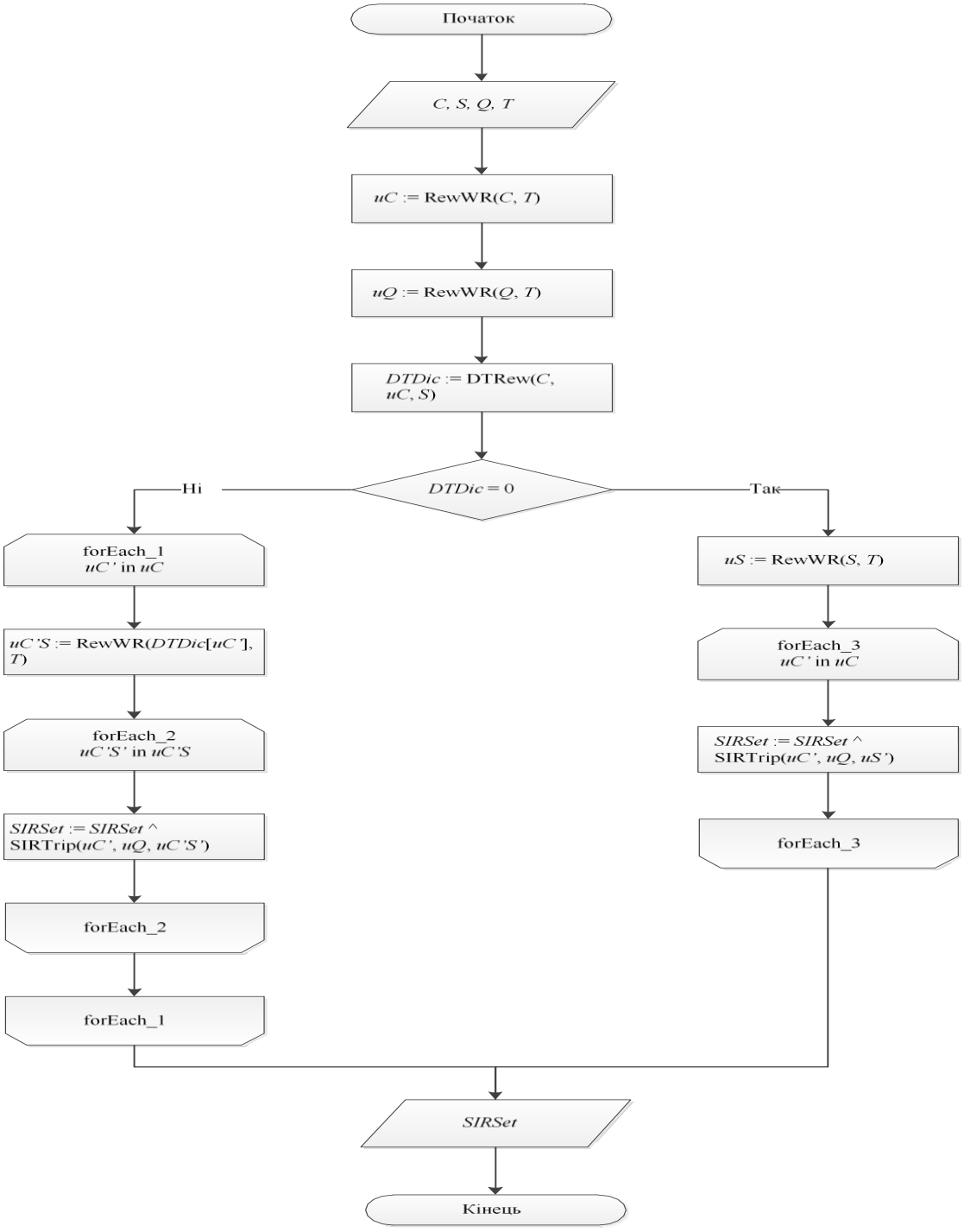


Рисунок 3.5 –Схема алгоритму *SIR1*

Послідовність роботи алгоритму *SIR1*. За трансформацію запитів відповідає описаний раніше алгоритм *Rewwr*. Алгоритм *SIR1* складається з наступних дій:

За допомогою алгоритму трансформації запитів переписати *C* і помістити об'єднання запитів у множину *uc*.

Переписати запит Q і помістити об'єднання запитів у множину *uq*.

Якщо вхідні параметри не містять залежних елементів, то проводяться наступні дії:

Переписати запит *S* і помістити об'єднання запитів у множину *us*. Для кожного *uc'* з *uc* створити Sir-триплет, для якого виконується умова про клієнта Abox = *uc'*, умова про передану інформацію Abox = *uq'* з множини *uq*, умова про сервіс в Abox = *us'* з множини *us*.

Помістити триплет в *Sirset*. Якщо вхідні параметри включають залежні елементи, то виконуються наступні дії:

Для обробки параметрів із залежними елементами буде використана функція *Dtrew*, представлена раніше.

Після завершення роботи функції для кожного *uc'* з *uc* переписати запит *Dtdic[uc']* і записати результат у множину *uc'S*.

Для кожного *uc'S'* з *uc'S* створити Sir-Триплет, для яких виконується умова про клієнта Abox = *uc'*; умова про передану інформацію Abox = *uq'* з множини *uq*, умова про сервіс Abox = uc'S' з множини *uc'S*.

## 3.5 Алгоритм трансформації запитів

Трансформація запитів (відповідь на кон'юнктивні запити), відноситься до класу складності Ptime. Операція додавання відноситься до задач із лінійною залежністю, що входять у клас P. Трансформація запитів в Dl-lite поліноміально залежить від розміру *T*.

При використанні параметрів із залежними змінними структура алгоритму буде складатися із трансформації запитів і функції обробки пов'язаних запитів. Ключове значенням в алгоритмі з параметрами із залежних елементів має функція обробки пов'язаних запитів *Dtrew*.

Опис нового алгоритму, що повертає множину $\bigcup_{isoa} Servi(ci, q, si)$, для наступного видалення інтерфейсів для взаємодії веб-сервісів. Вхідними параметрами алгоритму запитів до онтології є Tbox *T* і Sir-триплет. Вихідними даними алгоритму є об'єднання *Sirset*, Sir-триплетів (*Sirtrip*) з запитами до баз даних. На рис. 3.6 представлена схема алгоритму *SIRR*.

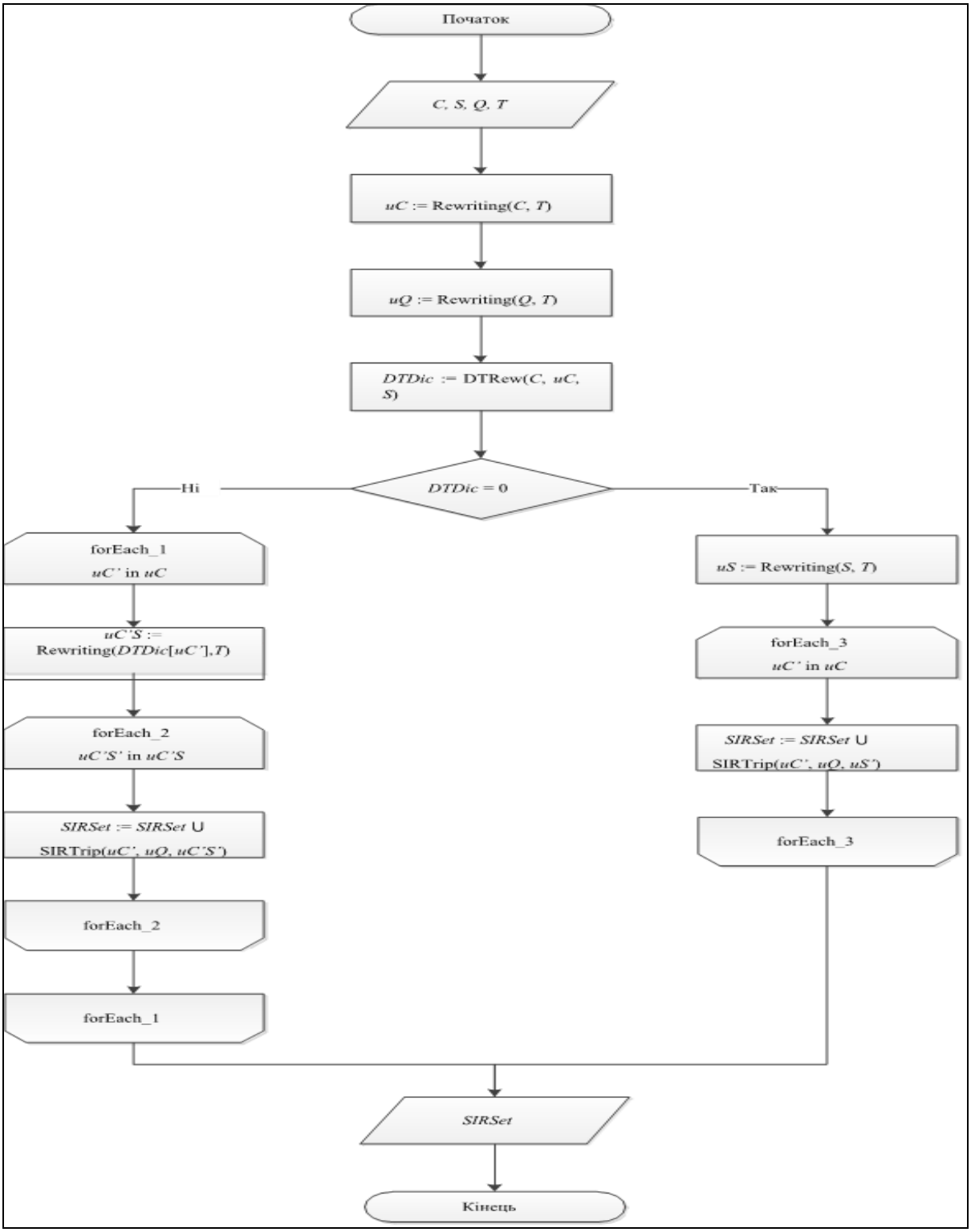


Рисунок 3.6 – Блок-схема алгоритму *SIRR*

За трансформацію запитів відповідає функція *Rewriting*. Під функцією *Rewriting* мається на увазі один з алгоритмів відповідей на запит у рамках технологій OBDA для *Dl-lite*. Застосування функції *Rewriting* необхідно для зберігання даних про ролі в Tbox. Загалом, алгоритм *SIRR* схожий із алгоритмом *SIR1*.

# 4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПОБУДОВИ ІНТЕРФЕЙСІВ ВЕБ-СЕРВІСІВ

## 4.1 Архітектура системи

Виходячи з формалізму й алгоритмів, розроблена інтелектуальна SOA-система для веб-сервісів, веб-додатків і веб-сайтів. Програмний код системи Sirsystem написаний мовою програмування Java. Він був протестований на ОС Centos, 3 GB RAM, AMD  II N850 Triple-Core Processor 2,20 ГГц. Дані з Abox у цій реалізації зберігаються у системі управління базами даних Mysql. Для опису інтерфейсів веб-сервісів використані стандарти SOAP і WSDL. Для машини-сервера необхідна наступна мінімальна конфігурація апаратного забезпечення:

− процесор Intel Pentium 4 або Amda;

− обсяг оперативної пам’яті 1 GB;

− обсяг дискової пам'яті 100 GB і вище (при збільшенні об’єму даних у базі даних);

− будь-який монітор;

− наявність інтернет з’єднання зі швидкістю не менш ніж 5 Мбіт/с.

Набір програмних засобів сервера:

− операційні системи сімейства Windows, Unix-системи;

− веб-сервер Apache версії 2.2.14;

− інтерпретатор PHP версії не нижче 5.1;

− сервер СУБД Mysql не нижче версії 5.1;

− Java Runtime Environment 6.0.

Існує онтологія O1, що описує взаємодію елементів SOA-системи, що й передбачає ієрархічну структуру за наступними ознаках: локаційне місце розташування і тип установи. Ця онтологія написана за допомогою редактора онтологій Protégé мовою веб-онтологій OWL 2 QL. В онтології SOA-системи зберігаються дані навчальних закладів регіону.

Система Sir-system забезпечує зручне створення інтерфейсів між веб-сервісами на основі онтології O1 і перегляд існуючих інтерфейсів веб-сервісів.

Автоматична створення інтерфейсів веб-сервісів для SOA-системи.

Зберігання онтологічної моделі взаємодії сервісів.

Простота розширення онтологічної моделі взаємодії елементів системи.

Гнучке оновлення веб-сервісів.

Облік існуючих інтерфейсів веб-сервісів з даними про сервіси, між якими вони встановлені.

Виходячи з отриманих результатів розроблена програмна реалізація SOA-системи Sirsystem v.1.0. На рис. 4.1 наведено UML-діаграму бізнес-процесів системи Sirsystem.

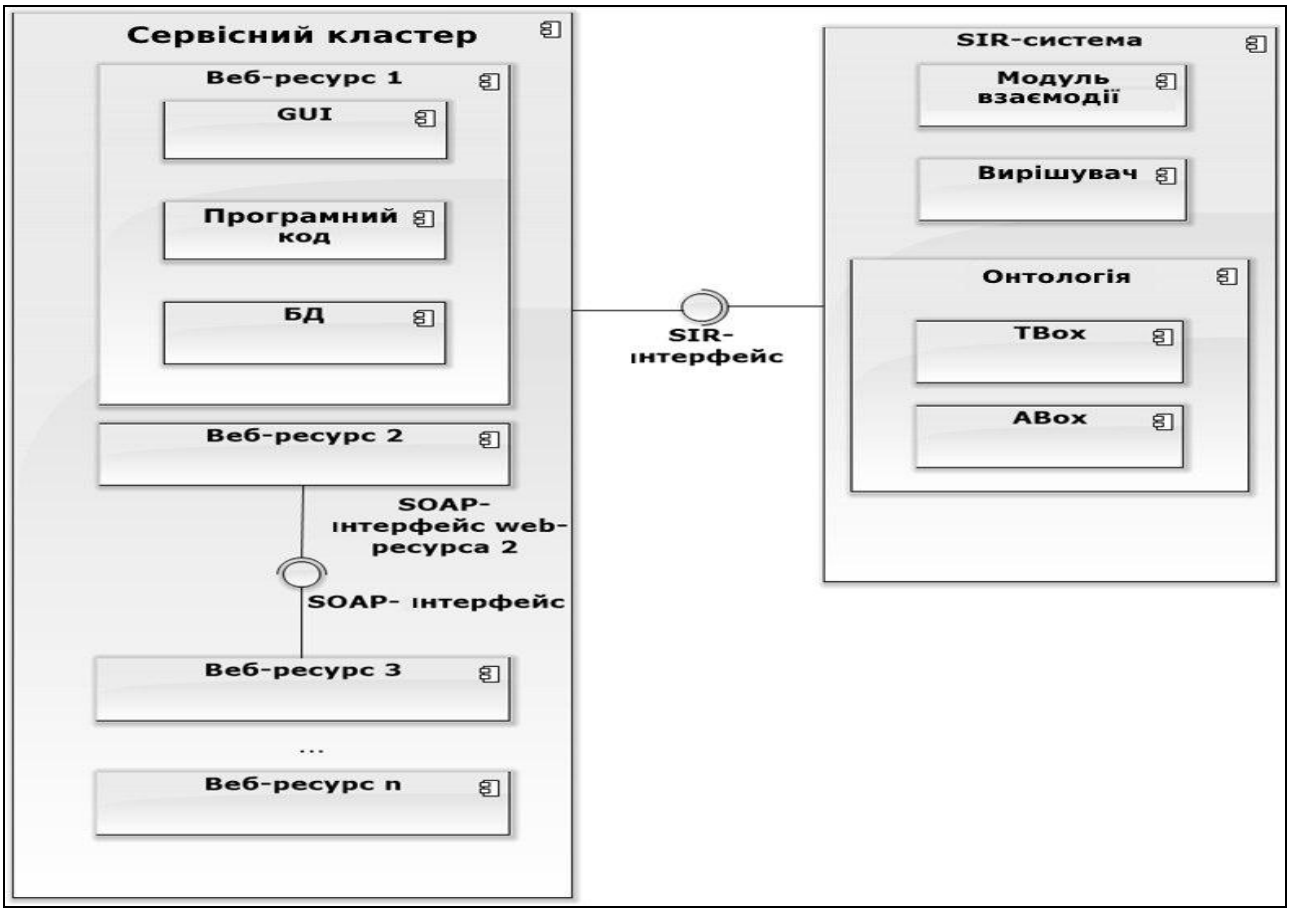


Рисунок 4.1 – UML-діаграма розробленого ПЗ

UML-діаграма реалізована за допомогою програмного забезпечення Software Ideas Modeler.

Розглянуто систему на прикладі побудови веб-сервісів між усіма школами м. Рязані й створеним порталом для надання інформації про кадровий склад

організації. Тому що онтологія системи Sirsystem включає поки одне місто, можна використовувати більш загальні параметри. Отже, потрібно побудувати інтерфейси (Клієнт $C_S$, Дані $Q_S$, Сервер $S_s$), де на всіх сайтах створених установ (Educationportal) кожного міста (City) центрального регіону (Centralregion) відобразяться дані про персонал (Personal) з веб-сервісу $S_s$. Веб-сервіси $S_s$, у свою чергу, – це створені сайти міста (City), вказаного в $C_S$. Залежні умови позначаються однаковими змінними елементів умови запиту, цей приклад використовує змінну – $x_3$.

$$C_s(x_1, x_2, x_3) \leftarrow \text{Educationportal}(x_1), \text{Centralregion}(x_2), \text{City}(x_3)\text{б}$$

$$Q_s(x_4) \leftarrow \text{Personal}(x_4)\text{б}$$

$$S_s(x_5, x_6) \leftarrow \text{Education}(x_5), \text{City}(x_6).$$

Інтерфейси повинні створитися тільки для м. Рязані за відсутності інших міст у системі.

На рис. 4.2 представлений графічний інтерфейс програми з описаним раніше прикладом вхідних параметрів системи.

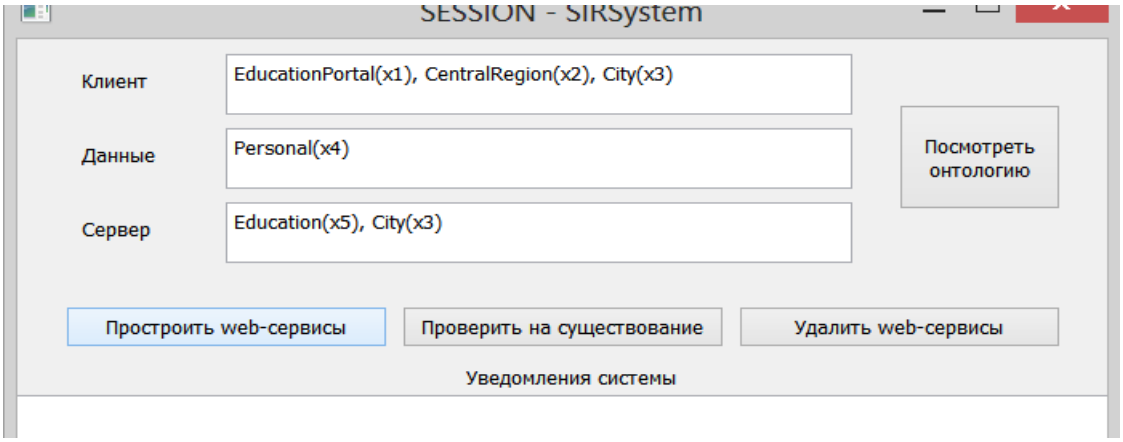


Рисунок 4.2 – Графічний інтерфейс Sirsystem

Система Sirsystem включає наступні модулі:

– модуль взаємодії, який відповідає за відправлення згенерованого програмного коду інтерфейсів веб-сервісів;

– модуль розрахунку – модуль, що включає алгоритми логічного аналізу й роботи з програмним модулем;

– онтологія, яка включає Tbox у форматі OWL 2 QL і OWL 2, у той же час онтологія включає Abox у форматі бази даних мовою Mysql;

– інтерфейс взаємодії Sir-системи з веб-сервісами: SIR-інтерфейс, що визначається за допомогою протоколів SOAP і HTTP, а також за допомогою мов OWL і XML;

– кластер веб-ресурсів, що створений за допомогою пошуку ресурсів у мережі Internet;

– веб-ресурс, що розташовується в мережі Internet, що має персональний URI. Веб-ресурс може включати в себе веб-сервіс або веб-сайт с графічним користувацьким інтерфейсом (GUI) у вигляді гіперсторінок. У той же час веб-ресурс використовує бази даних і СУБД для роботи з ними.

Далі представлена розроблена онтологія O1, це онтологія навчальних закладів, що розроблена спеціально для системи автоматичної побудови інтерфейсів взаємодії веб-сервісів. Набір аксіом (Tbox) онтології O1 зберігає дані про установи і їх територіальне розташування. Набір тверджень (Abox) онтології зберігає дані про адреси веб-сервісів навчальних закладів і дані, що надані веб-сервісами.

Необхідно описати обмеження, що вказують на те, що в системі існують створені заклади, в які входять створенісфілії.

Нижче представлений уривок з онтології O1 синтаксисом язика Dl-Lite.

```
Typeofestablishment⊑think,
Education⊑typeofestablishment,
Gouvpo⊑education,
Generaleducationestablishment⊑education,
Preschooleducationestablishment⊑education,
Highschool⊑generaleducationestablishment,
…
                                                    ,
```

Для зберігання і представлення Tbox онтології O1 [11] використовується мова веб-онтологій OWL 2 QL. На рис. 4.3 наведено фрагмент OWL-документа, що описує онтологію у форматі OWL.

```
<!-- http://www.semanticweb.org/ontologies/2012/11/Ontology1356005502435.owl#FederalDistrict -->

<owl:Class rdf:about="&Ontology1356005502435;FederalDistrict">
    <rdfs:subClassOf rdf:resource="&Ontology1356005502435;RegionOfEstablishment"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2012/11/Ontology1356005502435.owl#GOUVPO -->

<owl:Class rdf:about="&Ontology1356005502435;GOUVPO">
    <rdfs:subClassOf rdf:resource="&Ontology1356005502435;Education"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2012/11/Ontology1356005502435.owl#GeneralEducationalEstablishment -->

<owl:Class rdf:about="&Ontology1356005502435;GeneralEducationalEstablishment">
    <rdfs:subClassOf rdf:resource="&Ontology1356005502435;Education"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2012/11/Ontology1356005502435.owl#Gymnasy -->

<owl:Class rdf:about="&Ontology1356005502435;Gymnasy">
    <rdfs:subClassOf rdf:resource="&Ontology1356005502435;GeneralEducationalEstablishment"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2012/11/Ontology1356005502435.owl#HealthCare -->

<owl:Class rdf:about="&Ontology1356005502435;HealthCare">
    <rdfs:subClassOf rdf:resource="&Ontology1356005502435;TypeOfEstablishment"/>
</owl:Class>
```

Рисунок 4.3 – Фрагмент OWL-документа

Отриманий опис можна обробляти й переглядати за допомогою редакторів онтологій.

На рис. 4.4 наведено візуалізацію фрагмента онтології за допомогою редактора онтологій Protégé.

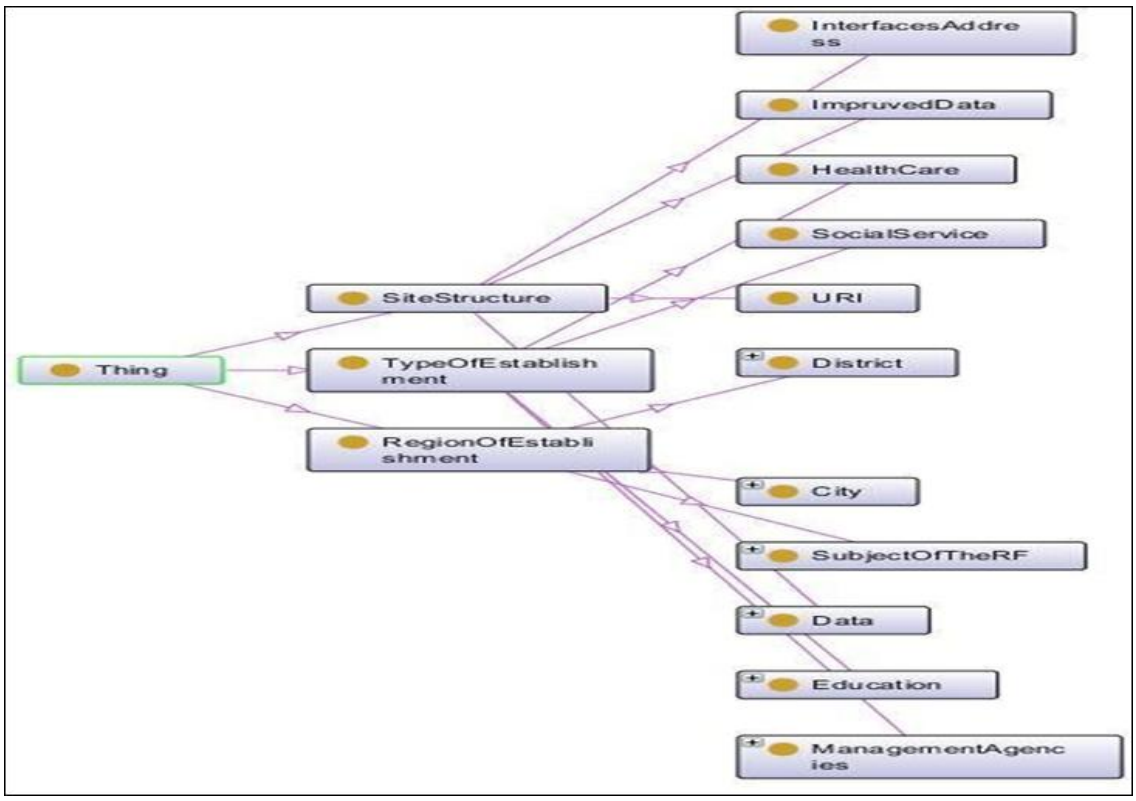


Рисунок 4.4 – Представлення онтології O1 за допомогою Protégé

Для зберігання Abox онтології O1 використовуються бази даних, що зберігаються в рамках інструментарію СУБД Mysql. Далі представлений приклад глобальної Abox системи, реалізованої у вигляді бази даних Aboxbd.

Таблиця сервісів s бази даних зберігає дані про існуючі веб-ресурси. Зразок даних про сайт умовного навчального закладу міста Барселона з таблиці Tbresources представлено в таблиці 4.1.

З таблиці Tbresources, згідно із прикладом, веб-сервіс цього ресурсу буде відправляти дані про викладачів на веб-сервіс створеного порталу.

Таблиця 4.1 – Дані таблиці

| ID | Serviceuri | Regionofestab | Typeofestab | Lgtype | Email |
|---|---|---|---|---|---|
| 2 | http:// Highschool1.eu | Barcelonacity | Highschool | PHP | info@highschool1.eu |

Виконано оптимізацію структури онтології O1. При проектуванні системи Sirsystem стало відомо, що система завжди буде масштабуватися за глобальними

категоріями місця розташування типам закладів. Ухвалено рішення вивести інформацію про ці два елементи.

У базі даних є присутнім таблиця синонімів . Таблиця синонімів необхідна для роботи з базами даних веб-ресурсів. Дані, що зберігаються в цій таблиці, необхідні для адекватної автоматичної генерації коду, що посилається на веб-ресурси.

Дані про створені інтерфейси і згенерований код записуються в таблицю Tbinterfaces бази даних. У табл. 4.2 представлений приклад даних таблиці Tbinterfaces.

Таблиця 4.2 – Дані таблиці

| ID | Rolemembership | Clienturi | Servicedata | Serveruri |
|---|---|---|---|---|
| 3 | Int3 | Highschool8.edu | <?php<br>class Quoteservice { private $anketa = $Globanketa;<br>function querystatement($anketa){<br>If ((isset($this-> anketa[$lastname])) & (isset($this-> anketa[$name])) & (isset($this->anketa[$middlename])) & (isset($this->anketa[$login]))) {<br>return $this->anketa[$name];<br>… | edu.admrzn.ru |

Дані з таблиці Tbinterfaces можна представити у форматі RDF-трійок і обробляти за допомогою Sparql-запитів.

## 4.2 Використання об'єктно-орієнтованого проектування

Якщо немає відповідних даних для наповнення коду, то необхідна інформація з веб-ресурсів SOA-системи Sirsystem відправляється обслуговуючому персоналу веб-ресурсу, наприклад веб-сайту навчального закладу, за адресою з поля Email таблиці Tbresources для ручного введення веб- інтерфейсів.

Наведено частини програмного коду, що демонструють способи реалізації алгоритмів, що працюють із дескриптивними логіками, за допомогою об'єктно-орієнтованих мов програмування. Далі демонструється використання типів даних, розроблених за допомогою ООП-підходу, на прикладі реалізації алгоритму Dtref. Представлений програмний код реалізує перетворення параметрів із залежними змінними в параметри без залежних змінних. У програмному коді Term – тип елементів запитів. Далі представлений фрагмент програмного коду функції мовою Java. Представлений фрагмент коду формує запис зв'язних змінних у спеціальний масив.

```
        int ·ii=0;¶
        int ·nope=0;¶
        Term[] ·clientbody ·= ·client.getbody(); ·Term[] ·serverbody ·= ·
server.getbody();¶
        Term[] ·curterm1 ·= ·new ·Term[clientbody.length];¶
        for(int ·i=0; ·i<clientbody.length; ·i++){ ·// ·для ·кожного ·елемента ·
запиту ·на ·клієнт¶
              for(int ·j=0; ·j< ·serverbody.length; ·j++){ ·// ·для ·кожного ·елемента ·
        запиту ·на ·сервер¶
                    // ·якщо ·елементи ·клієнта ·й ·сервера ·рівні, ·то ·записуємо ·
              елемент ·у ·масив ·curterm1¶
                    if(clientbody[i].tostring().equals(serverbody[j].tostring()
                    )){ ·¶
                          curterm1[ii] ·= ·serverbody[j].getvariableorconstant(); ·¶
                          nope++;¶
                          ii++;¶
        ·¶
                    }¶
              }¶
        }¶
        Term[] ·curterm2 ·= ·new ·Term[ii];¶
        for(int ·i=0; ·i<ii; ·i++){¶
              curterm2[i]=curterm1[i];¶
        }¶
        if(nope!=0){¶
              for(int ·i=0; ·i<curterm2.length; ·i++){ ·¶
                    curterm ·+= ·curterm2[i].tostring();¶
              }¶
        }¶
        ¶
```

Потім у знайдених атомах знаходяться змінні для запитів до клієнта й додаються до серверних запитів. У представленому програмному коді використовуються наступні типи змінних.

Variable – змінні умов запиту, Clause – використовується для зберігання запитів.

В представленому фрагменті коду за допомогою пошуку елементів, які дорівнюють елементам вхідних у масив curterm, знаходяться спільні атоми.

```
int cursatint = 0;
for(Clause currentsaturation: clientsaturation){
      nope=0;
      ii=0;
      Term[] cursatbody = currentsaturation.getbody();
      Term[] nonterm1 = new Term[cursatbody.length];
      Term[] finterm1 = new Term[cursatbody.length];
      for(int i=0; i<cursatbody.length; i++){
            for(int j=0; j<curterm2.length; j++){
                  //якщо існує в запиті елемент, дорівнює будь-якому елементу
                  з масиву curterm, тоді додати терми в стек
                  if(cursatbody[i].getvariableorconstant().tostring().equals(
                  curterm2[j].tostring())){
                        nonterm1[ii] = curterm2[j];
                        finterm1[ii] = cursatbody[i]; // додати терми в стек
                        nope++;
                        ii++;
                  }
            }
      }
}
```

Далі необхідно додати елементи до запиту таким чином, щоб сформувати новий масив запитів до сервера.

```
Term[] finterm2 = new Term[ii];
for(int i=0; i<ii; i++)
      finterm2[i] = finterm1[i];
int iii = 0;
//дописати до запиту
Term[] newbody = new Term[querybody.length+finterm2.length];
for(int i=0; i<querybody.length; i++){
      newbody[i]=querybody[i];
      iii = i+1;
}
```

```
for(int i=0; i<finterm2.length; i++)
      newbody[iii+i]=finterm2[i];
server 1.add(new Clause(newbody, query.gethead()));

//переписати масив у розмірний
Term[] nonterm2 = new Term[ii];
for(int i=0; i<ii; i++)
      nonterm2[i] = nonterm1[i];
if(nope!=0){
      for(int i=0; i<nonterm2.length; i++) {
            curterm += nonterm2[i].tostring();
            curterm += "\r\n";
            curterm += "Query     \r\n";
            curterm += server1.toarray()[cursatint]; curterm += "\r\n";
      }
      cursatint++;
}
```

Далі представлений фрагмент програмного коду, що реалізує формування даних для веб-інтерфейсів, запитів до Abox і відправлення згенерованого програмного коду на веб-ресурси. Фрагмент демонструє зручність використання об'єктно-орієнтованого програмування при розробці SOA-системи.

```
m_satur = new Satur(m_termf);
if(query != null){
      saturation = m_satur.saturate(pi, query);
      send(saturation);
} //завершення насичення запитів сервера
else {
      System.err.println("Invalid query.");
}
/* Насичення запитів серверів */
for(Clause curquery: server1) {
      m_satur = new Satur(m_termf);
      if(query1 != null){
                  System.out.println("\nontology loaded ("+ pi.size() +"
                  axioms)"); long begintime = System.currenttimemillis();
```

```
            sumbegin += begintime;
            saturation = m_satur.saturate(pi, curquery);
      }
      send(saturation);
      System.out.println("Query saturation completed ("+ saturation.size()
      +" clauses)");
      long endtime = System.currenttimemillis();
      sumend += endtime;
      System.out.println("Rendering computed in " + Long.tostring(endtime -
      begintime) + "milliseconds");
} //end for clientsaturation
else {
      System.err.println("Invalid query.");
}
```

Далі наведено основні елементи цього фрагмента, що забезпечують коректне виконання програми: `PI` – виділення позитивних значень масиву в окремий масив, `m_Termf` – об'єкт елементів поточного запиту, `m_satur.saturate` – метод трансформації елементів m_Termf обраного запиту, `send(saturation)` – запити до набору даних (Abox) бази знань і відправлення згенерованого коду інтерфейсів веб-сервісів на веб-ресурси використовуючи трансформований запит (saturation), `endtime, begintime` – змінні для підрахунку часу роботи елементів програми.

Для обробки вхідних параметрів використовується модуль Parser. Parser зчитує текстові дані у вигляді структури запиту у текстовому форматі виду «Q(?0,?1) <- Educationportal(?1), Ryazancity(?0)» і перетворить їх у тип Clause за допомогою бібліотеки ANTLR, ця бібліотека є генератором синтаксичних аналізаторів.

# 5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Для тестування системи й алгоритмів використовувалися 3 Sir-алгоритми: SIRR з використанням Perfectref, SIRR з використанням Requiem і алгоритм SIR1, запропонований у кваліфікаційній роботі.

Тестові дані, на яких аналізувався алгоритм, складаються з онтологій, створених або адаптованих під мову OWL 2 QL для тестування алгоритмів логічного аналізу..

VICODI – онтологія, що зберігає інформацію про Європейську історію, розроблену Європейським проектом Євросоюзу. VICODI позначена в табл. 5.1 літерою V.

Stockexchange – онтологія, що зберігає інформацію про фінансовий інститут Євросоюзу. Розроблена для архітектури OBDA. Stockexchange позначена в табл. 4.6 літерою S.

SAMAO – онтологія навчальних закладів, розроблена спеціально для системи автоматичного створення інтерфейсів взаємодії веб-сервісів. SAMAO – позначена літерами SM у табл. 5.1.

Вхідні параметри алгоритму SIR1(C, Q, S, T) і SIRR(C, Q, S, T) для тестових онтологій вибиралися з тестів запитів до онтологій: C = Q2, Q = Q4, S = Q3, T = Tbox онтології.

Вхідні параметри алгоритму SIR1(C, Q, S, T) для онтології SAMAO представлені нижче.

Параметри без залежних умов:

$$C(x_1, x_2) \leftarrow \mathbf{Educationportal}(x_1), \mathbf{l},$$

$$Q(x_3) \leftarrow \mathbf{Phone}(x_3),$$

$$S(x_4, x_5) \leftarrow \mathbf{Education}(x_4), .$$

Параметри із залежними умовами:

$$C(x_1, x_2, x_3) \leftarrow \mathbf{Educationportal}(x_1), \mathbf{Centralregion}(x_2), \mathbf{City}(x_3),$$

$$Q(x_4) \leftarrow \mathbf{Phone}(x_4),$$

$$S(x_5, x_3) \leftarrow \text{Education}(x_5), \text{City}(x_3).$$

При роботі алгоритму створюються нові запити до баз даних. Створені запити зберігаються у текстовому форматі. Для обчислення пам'яті, яку займають результати роботи алгоритму, слід розрахувати кількість породжених запитів. Кількість створених запитів була розрахована по формулі, ідентичній формулі розрахунку часу роботи алгоритму, з різницею в тому, що замість функції розрахунку часу *t(x)* використовувалася функція розрахунку кількості породжених запитів.

Були проведені експерименти по тестуванню роботи функції Dtrew в Sir-алгоритмах, використовуючи різноманітні підходи до перезапису запитів. Алгоритми були протестовані на онтології SAMAO. Результати наведені в табл. 5.1

Таблиця 5.1 – Робота функції Dtrew при тестуванні Sir-алгоритмів

| Тип вхідних параметрів | Кількість породжених запитів | | | Час виконання, мс | | |
|---|---|---|---|---|---|---|
| | Кількість запитів Dtrew | кількість запитів SIRR (Perfectref) | кількість запитів SIR1 | Час Dtrew | Час SIRR (Perfectref) | Час SIR1 |
| Без залежних умов | - | 75 | 68 | - | 19 | 14 |
| С залежними умовами | 3 | 147 | 113 | 1 | 33 | 21 |

У той же час було проведено тестування алгоритмів SIR1 і SIRR з використанням популярних алгоритмів перезапису запитів.

Алгоритми тестувалися на описаних раніше тестових онтологіях.

Результати тестування показали, що виконання Sir-алгоритмів вимагає порівняно невеликої кількості процесорного часу й оперативної пам'яті. Ці результати відповідають тим, що обчислювальна складність Sir-алгоритмів належить класу P, що є класом швидких алгоритмів. Більше того, для онтології, розробленої спеціально для системи на базі Sir-алгоритмів, алгоритми виконуються швидше, чим для довільних онтологій (SM на рис. 5.1).

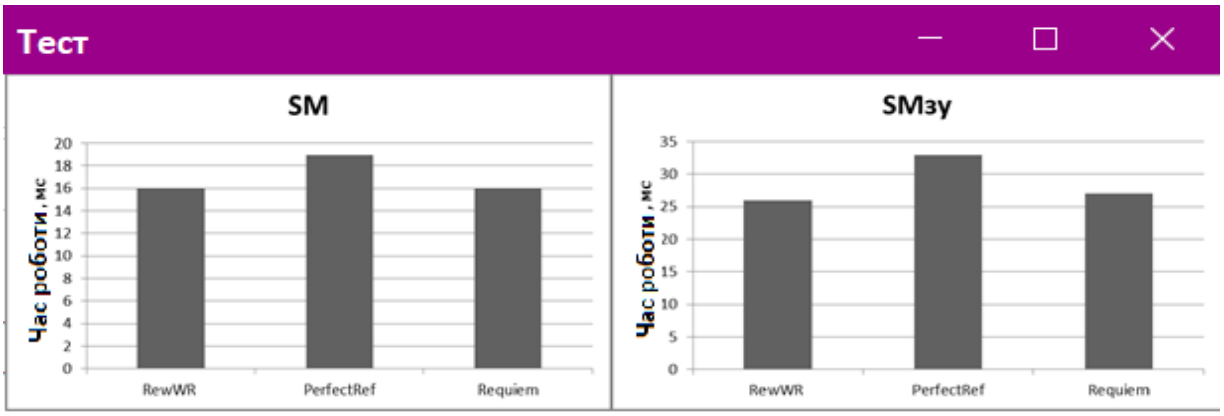


Рисунок 5.1 – Екранна форма порівняння швидкості роботи алгоритмів

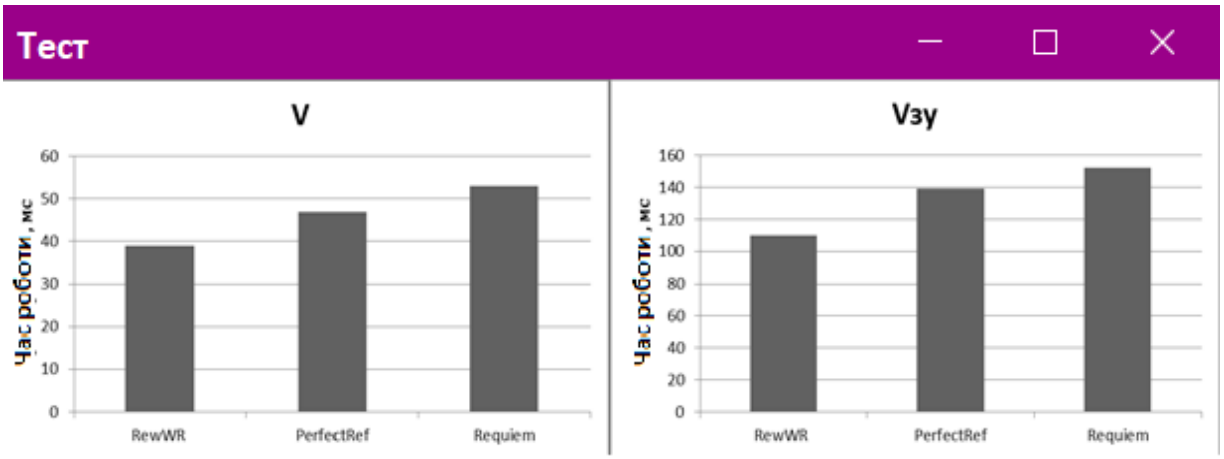


Рисунок 5.2 – Діаграма порівняння швидкості роботи алгоритмів з онтологією VICODI

Розроблене ПЗ може використовуватися у якості основи систем автоматичної побудови інтерфейсів взаємодії веб-сервісів. Структура онтології, яка необхідна для роботи системи, має OWL-розмітку стандарту Semantic Веб. Цей показник надає можливість переходу системи на базу семантичної сервіс-орієнтованої архітектури (SSOA), направленої на Semantic Веб сервіси.

# ВИСНОВКИ

Розроблена SOA-система за допомогою парадигми заснованої на онтологіях доступу до даних. Для реалізації SOA-системи був створений алгоритм для опису процесів системи за допомогою онтології, створені ефективні алгоритми і вдосконалені існуючі Obda-алгоритми для роботи з SOA-системами.

Проведений аналіз сучасних технологій для проектування в рамках парадигми SOA. Представлена класифікація аналогічних технологій і підходів. Розглянуті мови опису взаємодії елементів SOA-систем. Виявлені недоліки існуючих підходів до проектування інтерфейсів веб-сервісів і EAI-систем. Проаналізовані основні види структур взаємодії SOA-сервісів. Представлений огляд експериментальних реалізацій систем OBDA, що працюють із сімейством дескриптивних логік Dl-lite. Наявність таких реалізацій підтверджує практичне застосування дескриптивних логік і мов веб-онтологій з метою надання онтологічного доступу до даних.

Розроблено алгоритми для опису і аналізу концептуальної моделі SOA-системи, орієнтований на онтологічний підхід, що дозволяє описувати взаємодія веб-сервісів.

Розроблено алгоритми автоматизованого побудови інтерфейсів веб-сервісів і CRUD-алгоритми для SOA-архітектури, що надають можливість автоматичного створення інтерфейсів веб-сервісів. Алгоритми відрізняються від аналогів швидкістю побудови і загальнішою моделлю. Новий алгоритм трансформації запитів для SIR-системи відрізняється від більш загальних алгоритмів трансформації запитів швидкістю роботи за рахунок ігнорування неважливих елементів вхідних даних. Різниця у швидкості роботи між отриманим алгоритмом і аналогами залежить від кількості неважливих елементів в онтології.

Отримані ефективні проектні рішення програмної реалізації SIR-алгоритмів, використовуючи ООП підхід.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kienast R. Semantic Data Integration on Biomedical Data using Semantic Web Technology / R. Kienast, C. Baumgartner // Trends and Methodologies. – 2011. – P. 57–76.

2. Hendler A. J. Handbook of Semantic Web Technologies. – Springer, 2019. – 479p.

3. Semantic Web Technologies in Automotive Repair and Diagnostic // URL: http://www.w3.org/2001/ sw/sweo/public/UseCases/Renault/.

4. RIF basic logic dialect // URL:http://www.w3.org/TR/2013/REC-rif-bld-20130205/.

5. Gruber T. Collective Knowledge Systems: Where the Social Web meets the Semantic Web // Journal of Web Semantics. – 2018. – V. 6, № 1. – P. 4–13.

6. The Description Logic Handbook / F. Baader, D. Calvanese, D. Guinness, D. Nardi, P. Schneider. – Cambridge University Press, 2017. – 574 p.

7. OWL Web Ontology Language Semantics and Abstract Syntax // URL: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

8. Babenko A., Lempitsky V. Aggregating Deep Convolutional Features for Image Retrieval // Proceedings of the IEEE International Conference on Computer Vision. – 2018. P. 1269–1277.

9. Berclaz J., Fleuret F., Fua P. Robust people tracking with global trajectory Computer Vision and Pattern Recognition. – 2016. – P. 744–750.

10. Shubin, I., Snisar, S., Zhyrnov, V., Slavhorodskyi, V.// Practical Application of Formal Representation of Information for Intelligent Radar Systems 2018 International Scientific-Practical Conference on Problems of Infocommunications Science and Technology, PIC S and T 2018 - Proceedings, 2019, pp. 433-436, 8632103

11. Drayer B., Brox T. Object Detection, Tracking, and Motion Segmentation for Object-level Video Segmentation // arxiv.org. 2016. – URL: https://arxiv.org/abs/1608.03066.

12. Hinton G. A practical guide to training restricted Boltzmann machines // Momentum. – 2010. – № 9(1).

13. Kim C., Li F. Multiple Hypothesis Tracking Revisited // Proceedings of the IEEE International Conference on Computer Vision. – 2019.

14. Konev A., Chigorin A., Krivovyaz G., Velizhev A., Konushin A. Traffic signs recognition on images with training on synthetic data // Technical vision in computer systems. – 2019. P. 65-66.

15. Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Kar- pathy A., Khosla A., Bernstein M., Berg A.C., Fei-Fei L. Imagenet large scale visual recognition challenge // IJCV. – 2015

16. Ruta A. A New Approach for In-Vehicle Camera Traffic Sign Detection and Recognition // IAPR Conference on Machine vision Applications (MVA). – 2009. – P. 509-513.

17. Аведьян Є.Д., Галушкин А.І., Селиванов С.А. Порівняльний аналіз структур полносвязных і сверточных нейронних мереж і їх алгоритмів навчання // Інформатизація й зв'язок. – 2017. – № 1.

18. Антощук С.Г. Відстеження об'єктів інтересу при побудові автома-тизированных систем відеоспостереження за людьми // Електротехнічні й комп'ютерні системи. – 2018. – №8(84). – С. 151–156.

19. Zhai M., Roshtkhari M., Mori G. Deep Learning of Appearance Models for Online Object Tracking // arxiv.org . 2019. – URL: https://arxiv.org/abs/1607.02568 .

20. Zhang K., Liu Q. Robust Visual Tracking via Convolutional Networks // arxiv.org . 2019. – URL: https://arxiv.org/abs/1501.04505 .

21. Zheng L., Bie Z. MARS: A Video Benchmark for Large-Scale Person Re-identification // Proc. European Conference on Computer Vision (ECCV). – 2016.

22. Xiang Y., Alahi A. Learning to Track: Online Multi-Object Tracking by Deci- sion Making // Proceedings of the IEEE International Conference on Computer Vision. – 2015.

23. Chetverikov G., Puzik O., Vechirska I. Multiple-valued structures of intellectual systems //Proceedings of the with Internations Computer Sciences and

Information Technologies (CSIT). 2016, 7589907. -pp. 204-207

24. Yang M., Wu Y. A Hybrid Data Association Framework for Robust Online Multi-Object Tracking // arxiv.org 2017. – URL: https://arxiv.org/abs/1703.10764 .

25. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition // Proceedings of the Neural Information Processing Sys- tems conference, NIPS. – 2015.

26. Szegedy C., Liu W., Jia Y. Going deeper with convolutions // CVPR. – 2015.

27. Talukder K.H., Harada K. Haar Wavelet Based Approach for Image Compres- sion and Quality Assessment of Compressed Image // IAENG International Journal of Applied Mathematics. – 2007. – 36(1).

28. Viola P., Jones M. Robust Real-Time Face Detection // International Journal of Computer Vision. – 2014. – V. 57. – №2. – P. 137–154.

29. A Neural Network for Machine Translation, at Production Scale. URL: https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

30. Добро пожаловать в блог DeepL! URL: https://www.deepl.com/blog/20180215.html

31. Shostak I., Matyushenko I., Romanenkov Yu., Danova M., Kuznetsova Yu. Computer Support for Decision-Making on Defining the Strategy of Green IT Development at the State Level. In book: Green-IT Engineering: Social, Business and Industrial Applications, Vol. 171. Berlin, Heidelberg: Springer International Publishing, 533–559 (2018), https://doi.org/10.1007/978-3-030-00253-4

32. Shostak I., Kapitan R,. Volobuyeva L., and Danova M., Ontological Approach to the Construction of Multi-Agent Systems for the Maintenance Supporting Processes of Production Equipment. In Proc. : IEEE International Scientific and Practical Conference «Problems of Infocommunications. Science and Technology» (PICS&T-2018). Ukraine, Kharkiv, October 9-12, 2018. P. 209 – 214

33. Кукіер, К. Big Data: A Revolution That Will Transform How We Live, Work, and Think/К. Кукіер, В. Штойнберг, 2018. – 236 с.

34. Кузнецов М., Симдянов И. MySQL на примерах// СПб.: БХВ- Петербург, 2008. 952 с.

35. Eleven SPARQL 1.1 Specifications are W3C Recommendations. : w3.org. 2013-03-21. http://www.w3.org/blog/SW/eleven-sparql-1-specifications-are-w3c-recommendations/

36. DuCharme *B*. Learning SPARQL // O'REILLY, 2018. 235 p.

37. Гома Х. UML-проектирование систем реального времени параллельных и распределенных приложений // ДМК Пресс, 2011. 704 с.