ЗМІСТ

Вступ 4
1 Опис бази практики 5
1.1 Основні відомості про компанію «Академія СМАРТ» 5
2 Основні поняття, постановка задачі переддипломної практики 6
2.1 Основні поняття та визначення 6
2.2 Постановка задачі переддипломної практики 7
3 Моделювання та розробка основних компонентів програмного забезпечення дипломного проекту 8
3.1Мотивований вибір моделі життєвого циклу програмного забезпечення 8
3.1.1 Ітераційна модель життєвого циклу ПЗ 8
3.1.2 Мотивація вибору ітераційної моделі життєвого циклу ПЗ 9
3.2 Моделювання архітектури майбутнього програмного забезпечення 10
3.3 Розробка моделей основних компонентів проекту 14
3.3.1 Компонент генерації онтології 14
3.3.2 Компонент злиття двох різних онтологій 15
3.3.3 Компонент доступу до джерела даних 16
3.3.4 Компонент виконання користувацького SPARQL запиту 17
3.4 Алгоритм інтеграції даних під час обробки SPARQL запиту 18
4 Тестування розробленого програмного забезпечення 20
4.1 Розробка тест плану проекту за стандартом IEEE 829-2008 20
4.1.1 Вступ 20
4.1.2 Об’єкти, що тестуються 20
4.1.3 Функціональність, що тестується 20
4.1.4 Функціональність, що не тестується 20
4.1.5 Метод тестування 20
4.1.6 Критерій проходження / не проходження тесту 21
4.1.7 Критерій зупинки та продовження тестування 21

4.1.8 Артефакти тестування 21
4.1.9 Середовище тестування 21
4.1.10 Можливі ризики та непередбачені обставини 21
4.2 Модульне тестування елементів розробленої компоненти 22
4.3 Інтеграційне тестування компоненти 22
Висновки 25
Список джерел інформації 26

ВСТУП

Виробнича практика є частиною навчального процесу і проводиться на 4-му курсі. Тривалість практики - 4 тижні. Виробнича практика спрямована для початкового знайомства студентів з основними формами діяльності по спеціальності "Програмна інженерія" і є часткою навчального процесу.

Студент при проходженні практики зобов'язаний:

– повністю виконувати завдання, передбачені програмою практики;

– вивчити і дотримуватися правила охорони праці, техніки безпеки і виробничої санітарії;

– брати участь у суспільному житті колективу підприємства, установи, організації;

– нести відповідальність за роботу ,що виконується нарівні зі штатними робітниками.

Контроль за проведенням практики з боку вищого навчального закладу здійснюється: керівником практики, завідуючим кафедрою, представниками ректора і інспекторською групою вищого навчального закладу. Контролюючий приймає оперативні міри по усуненню виявлених недоліків (про серйозні відхилення контролюючий доповідає керівництву вищого навчального закладу і підприємства - бази практики).

По закінченню практики студент складає письмовий звіт по виробничій практиці. Звіт практики захищається на кафедрі в комісії, призначеної завідуючим кафедрою.

1 ОПИС БАЗИ ПРАКТИКИ

1.1 Основні відомості про компанію «Академія СМАРТ»

Академія СМАРТ – компанія з розробки програмного забезпечення, що має два офіси в Україні та Ізраїлі, що розробляє високотехнологічні програмні додатки і забезпечує комплексні послуги з розробки систем для стартапів і продуктових компаній.

Компанія працює в сфері IT вже 7 років. Також співпрацює з професійними компаніями, наприклад Bitmedia і Gamabssa, що є лідерами в галузі електронного навчання на своїх ринках, CA (ім'я змінено через NDA) - заснований в Ізраїлі стартап і світовий лідер в області управління хмарними платформами, Car2Go - це сервіс аренди машин у будь-якому місці.

Академія Смарт має досвід роботи в таких галузях як фінансові програми, ERP-системи, електронне навчання, EDP, системи бронювання і корпоративні веб-сайти. Компанія забезпечує повний цикл розробки по «Agile» методології, включаючи повне тестування і підтримку

Основний профіль діяльності фокусується на розробці програмних рішень для он-лайн навчання (e-learning) і управління процесом навчання (Learning Management Systems). Компанія надає послуги розробки, використовуючи технології PHP, Java і .NET, тестування, управління проектами і підтримки. Більш того, компанія також надає послуги навчання програмістів різних напрямків [1].

2 ОСНОВНІ ПОНЯТТЯ, ПОСТАНОВКА ЗАДАЧІ ПЕРЕДДИПЛОМНОЇ ПРАКТИКИ

2.1 Основні поняття та визначення

Інтеграція даних включає об'єднання даних, що знаходяться в різних джерелах та надання даних користувачам в уніфікованому вигляді.

Системи інтеграції даних можуть забезпечувати інтеграцію даних на фізичному, логічному і семантичному рівні. Інтеграція даних на фізичному рівні є найбільш простим завданням і зводиться до конверсії даних з різних джерел в необхідний єдиний формат їх фізичного представлення. Інтеграція даних на логічному рівні передбачає можливість доступу до даних, що містяться в різних джерелах, в термінах єдиної глобальної схеми, яка описує їх спільне подання з урахуванням структурних і, можливо, поведінкових (при використанні об'єктних моделей) властивостей даних. Семантичні властивості даних при цьому не враховуються.

Підтримку єдиного представлення даних з урахуванням їх семантичних властивостей в контексті єдиної онтології предметної області забезпечує інтеграція даних на семантичному рівні [2].

Онтологія — це загальноприйнята і загальнодоступна концептуалізація певної області знань (світу, середовища), яка містить базис для моделювання цієї області знань і визначає протоколи для взаємодії між агентами, які використовують знання з цієї області, і, нарешті, включає домовленості про представлення теоретичних основ даної області знань [3].

OWL (англ. Web Ontology Language) — мова опису онтологій для семантичної павутини. Мова OWL дозволяє описувати класи і відносини між ними, властиві для веб-документів і застосунків. OWL заснований на більш ранніх мовах OIL і DAML OIL і в наш час є рекомендованим консорціумом Всесвітньої павутини [3].

Тестування ПЗ - це процес дослідження програмного забезпечення з метою виявлення помилок і перевірки його якості. Також тестування ПЗ можна описати

як процес перевірки відповідності між реальною і очікуваною поведінкою програми. Тестування включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

План тестування (Test Plan) - це документ, що описує весь обсяг робіт по тестуванню, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення.

2.2 Постановка задачі переддипломної практики

Основною ціллю даної переддипломної практики є вибір моделі життєвого циклу майбутнього програмного забезпечення проекту, розробка моделей та розробка одного з основних модулів з подальшим його тестуванням на основі досвіду, здобутого під час проходження практики на підприємстві.

Темою науково-дослідницької роботи, що розглядається є: «Розробка інформаційних моделей та інструментальних засобів для інтеграції даних систем електронного тренінгу персоналу та систем управління корпоративними знаннями».

3 МОДЕЛЮВАННЯ ТА РОЗРОБКА ОСНОВНИХ КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДИПЛОМНОГО ПРОЕКТУ

3.1Мотивований вибір моделі життєвого циклу програмного забезпечення

3.1.1 Ітераційна модель життєвого циклу ПЗ

Альтернативою каскадній моделі є так звана модель ітеративної і інкрементальної розробки (з англ. iterative and incremental development, IID), що отримала також від Т. Гілбі в 70-і рр. назву еволюційної моделі. Також цю модель називають ітеративною моделлю і інкрементальною моделлю (рисунок 3.1).

Модель IID передбачає розбиття життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує «міні-проект», включаючи всі процеси розробки в застосуванні до створення менших фрагментів функціональності, порівняно з проектом в цілому. Мета кожної ітерації - отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим вмістом всіх попередніх і поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації продукт бере приріст - інкремент до його можливостей, які розвиваються еволюційно. Ітеративність, інкрементальність і еволюційність в даному випадку є вираз одного і того ж сенсу різними словами зі злегка різних точок зору.

Підхід IID має і свої негативні сторони, які, по суті, - зворотна сторона достоїнств. По-перше, цілісне розуміння можливостей і обмежень проекту дуже довгий час відсутнє. По-друге, при ітераціях доводиться відкидати частину

Рисунок 3.1 – Інкрементна модель життєвого циклу ПЗ

зробленої раніше роботи. По-третє, сумлінність фахівців при виконанні робіт все ж знижується, що психологічно зрозуміло, адже над ними постійно тяжіє відчуття, що «все одно все можна буде переробити і поліпшити пізніше».

Різні варіанти ітераційного підходу реалізовані в більшості сучасних методологій розробки (RUP, MSF, XP) [5].

3.1.2 Мотивація вибору ітераційної моделі життєвого циклу ПЗ

Оскільки проект, що розробляється є досить великим і включає декілька менших проектів (що, у свою чергу можна розбити на менші етапи) і розробляється однією людиною, то, було б доцільно обрати ітеративну модель розробки ПЗ, оскільки кожна ітерація робиться після попередньої (що підходить для однієї людини), на кожній ітерації додається частина функціональності основного проекту, що дає змогу краще осмислити та розробити дану компоненту, протестувати її. Також, у деяких випадках є неможливим осмислення та моделювання усього проекту одразу, оскільки не ясно, чи можна буде реалізувати

технічну частину деяких компонент (із-за відсутності готових фреймворків, що б дозволяли оцінити можливість розробки деяких компонент).

Саме тому, на перших ітераціях планується розробка проблематичних компонентів, щоб було зрозуміло, чи є доцільними подальші ітерації; також на кожній ітерації буде робитись рішення про можливе переосмислення основних концепцій проекту, та, можливо переробка деякої функціональності готових компонентів з попереднім доопрацюванням відповідних моделей програмних компонент.

Також, специфікою даної роботи є часта зміна / доповнення вимог до розроблюваного ПЗ, тому ітеративна розробка є як ніколи доцільною.

3.2 Моделювання архітектури майбутнього програмного забезпечення

Для того, щоб розпочати моделювання архітектури, ми повинні розуміти основні варіанти використання системи кінцевим користувачем/користувачами, тому ми повинні побудувати діаграму варіантів використання (рисунок 3.2).

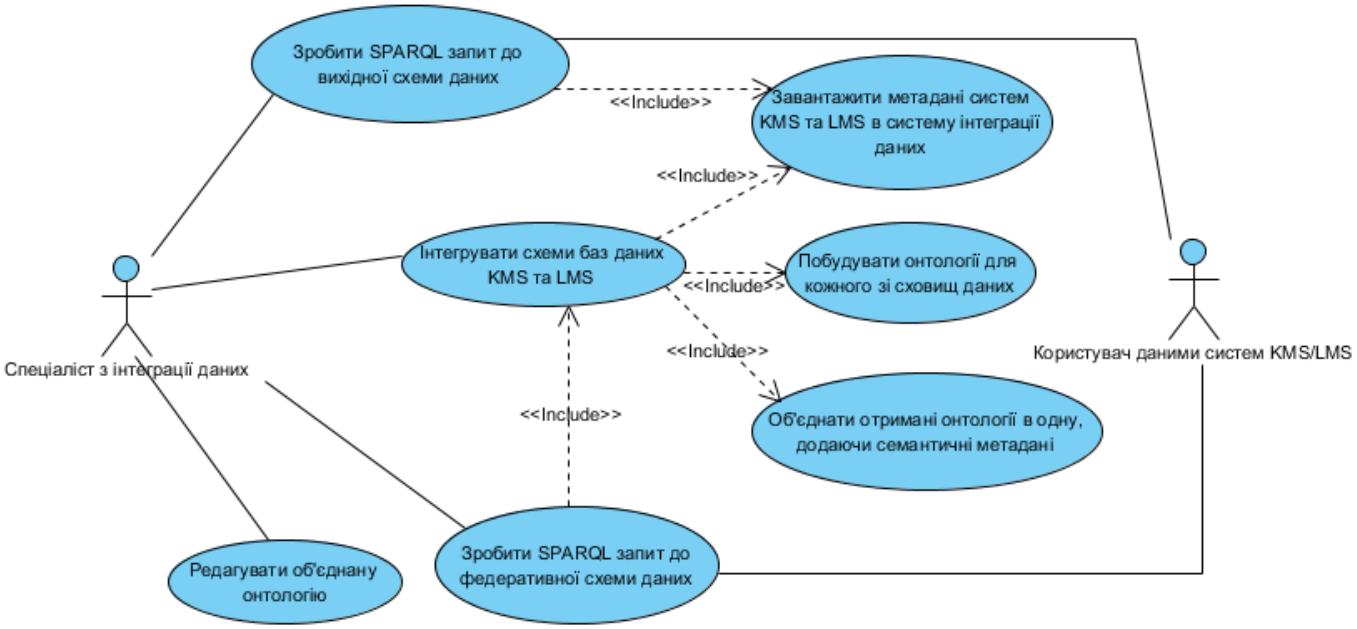

Рисунок 3.2 – Діаграма варіантів використання ПЗ для інтеграції даних

На основі діаграми основних варіантів використання системи, ми можемо побудувати діаграму основних компонентів системи.

На рисунку 3.3 зображена діаграма компонентів майбутнього програмного забезпечення.

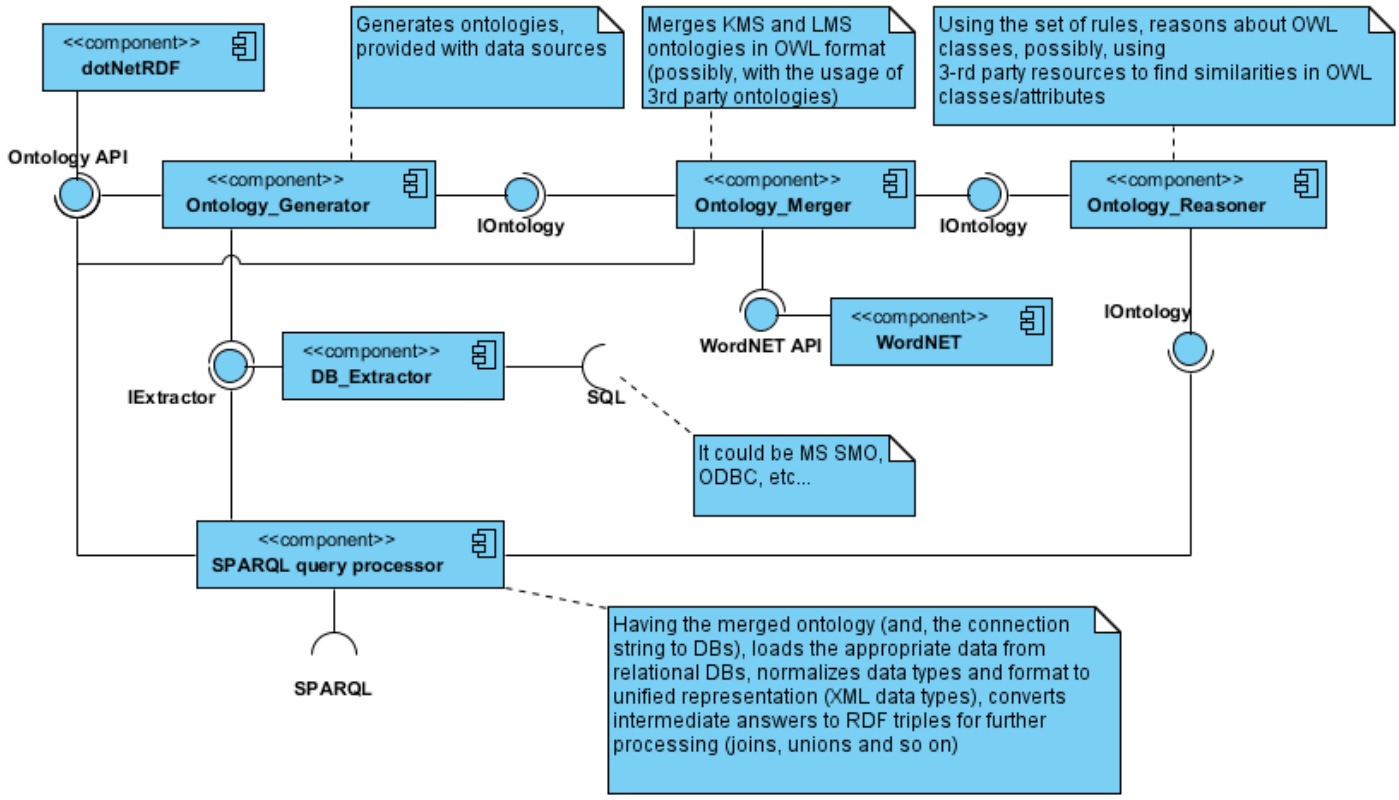

Рисунок 3.3 – Діаграма компонентів програмного забезпечення для інтеграції баз даних систем KMS та LMS

На діаграмі зображені наступні компоненти:

1) dotNetRDF – бібліотека низькорівневого доступу до онтологій / RDF даних;

2) OntologyGenerator – компонент, відповідальний за завантаження метаданих з баз даних KMS та LMS, та побудову їхніх онтологій (компонент будує одну окрему онтологію для одного конкретного джерела даних);

3) OntologyMerger – компонент, що «зливає» онтології, отримані від компонента «OntologyGenerator» у напівавтоматичному режимі та використовуючи API для доступу до системи WordNet (онтологія англомовних слів);

4) WordNet – компонент, що надає доступ до найбільшої онтології англомовних слів, що надає можливість оцінити схожість різних концептів онтології джерел даних;

5) OntologyReasoner – компонент, що, маючи об’єднану онтологію, шляхом логічного виводу, додає нову семантичну інформацію до онтології, таким чином, дозволяючи робити більш складні запити до джерел даних;

6) DBExtractor – компонент, що відповідає за вивантаження матеданних та, безпосередньо, даних із відповідних джерел даних систем KMS та LMS (у даному випадку – реляційні бази даних);

7) SPARQL Query processor – компонент, що відповідає за виконання SPARQL запитів до федеративної схеми даних (підхід віртуалізації джерел даних).

Для цієї програмної системи була обрана клієнт-серверна архітектура з товстим клієнтом (відображення та уся бізнес-логіка інтегратора даних знаходиться на комп’ютері клієнта, що у свою чергу, взаємодіє з двома виділеними серверами баз даних) – рисунок 3.4.

Програмне забезпечення інтегратора має 4 головних контролера:

1) Presentation Controller – контролер користувацького інтерфейсу (використовує для відображення фреймворк Windows Presentation Foundation);

2) Data Integration Controller – контролер, у якому відбувається основна робота по інтеграції схем даних та самих даних; взаємодіє з усіма іншими контролерами безпосередньо через відповідні інтерфейси;

3) Data Access Controller – відповідає за отримання даних із джерела даних (у даному випадку 2 сервера реляційних баз даних); взаємодіє з контролером “Data Integration Controller” через відповідний інтерфейс, що означає, що реалізація цього контролеру може бути будь-яка (не обов’язково взаємодіюча з реляційною базою даних).

4) Data Querying Controller – відповідає за парсинг SPARQL запиту до федеративної схеми даних у об’єкт SPARQL алгебри та за обробку цього запиту відповідним чином.

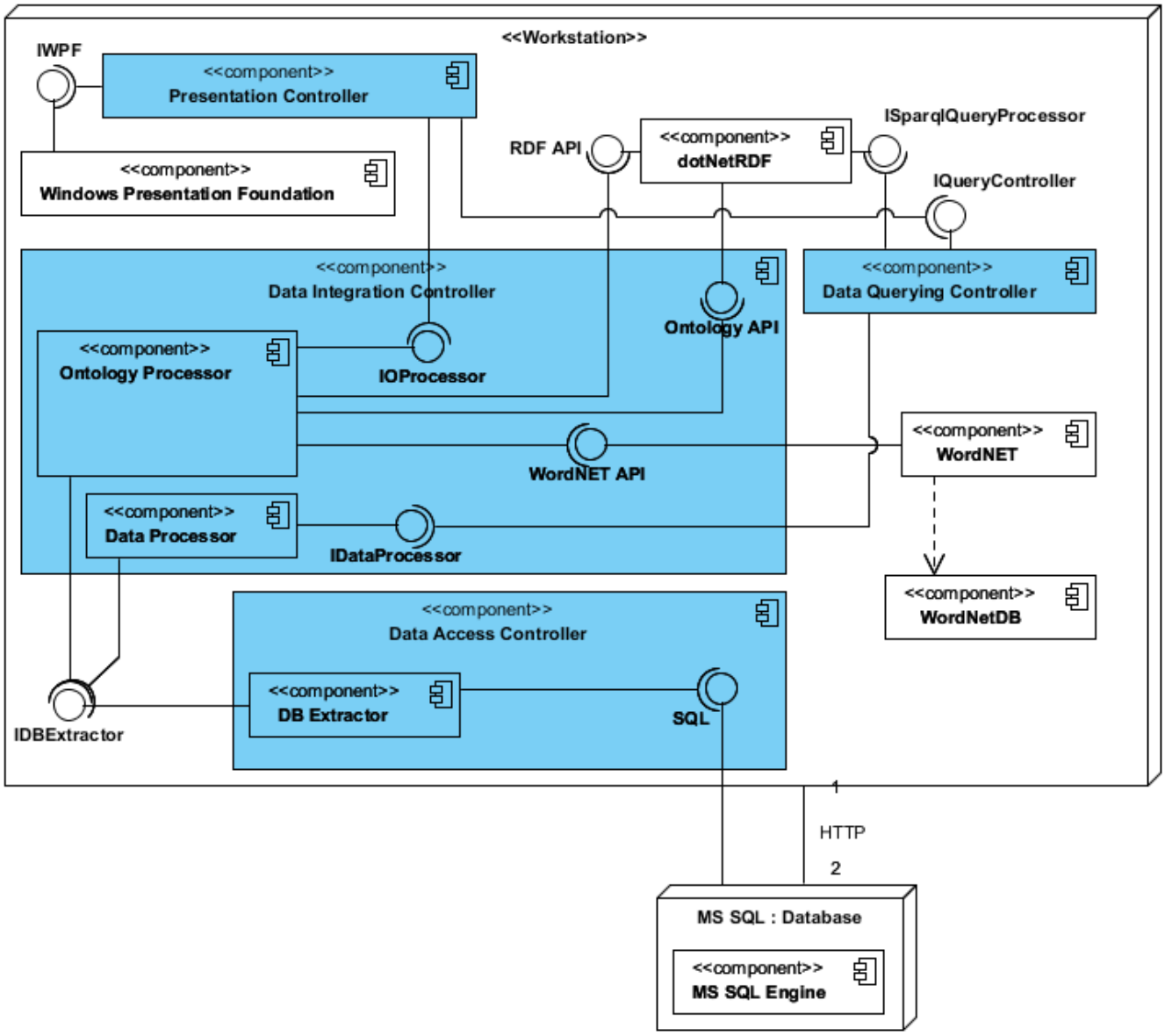

Рисунок 3.4 – Діаграма розгортання інтегратора баз даних систем KMS та LMS

Моделюючи внутрішню структуру програмного забезпечення самого інтегратора, можна виділити наступні пакети (рисунок 3.5): пакет, що включає класи по обороці онтологій (Ontology processing), що у свою чергу включає такі пакети як: ontology generation – містить логіку з генерації онтологій для джерел даних, ontology merging – містить логіку для злиття двох онтологій у напівавтоматичному / автоматичному режимах, пакет «Data integration» - містить класи з логікою інтеграції самих даних у реальному часі (у момент виконання користувацького запиту до федеративної схеми), даний пакет залежить від пакету «DB extractor», що містить класи, відповідальні за взаємодію з самими джерелами

даних напряму, та пакет «SPARQL query processing», що містить класи з логікою обробки, безпосередньо, SPARQL запиту від користувача.

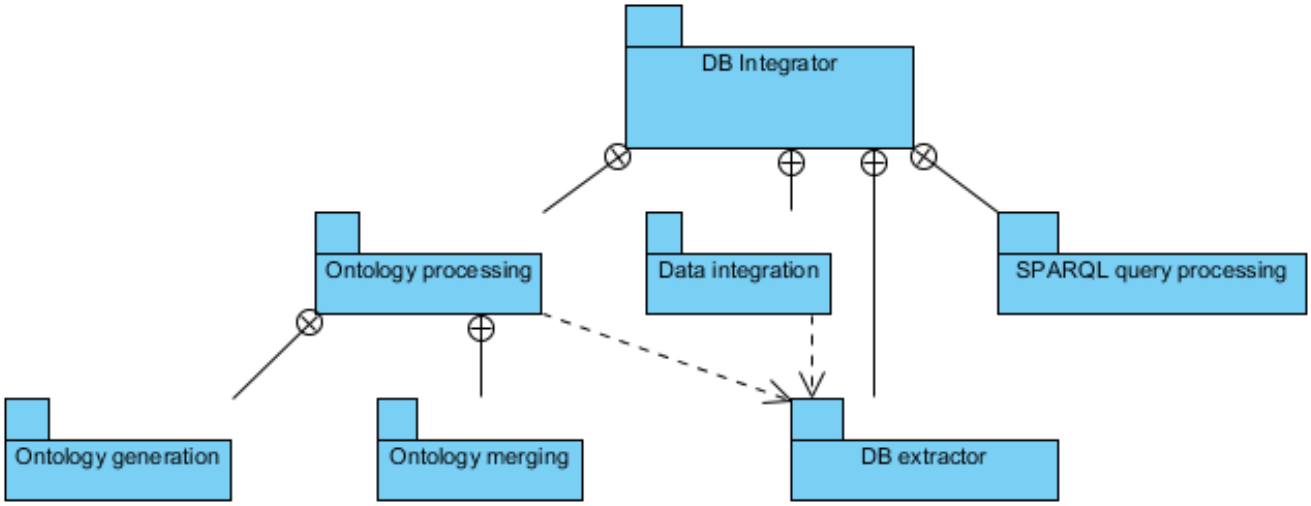

Рисунок 3.5 – Діаграма пакетів для ПЗ інтегратора

3.3 Розробка моделей основних компонентів проекту

Основними компонентами даного програмного забезпечення являються (як можна бачити з попередніх діаграм): компонент генерації онтології для відповідної бази даних, компонент злиття двої різних онтологій, компонент доступу до безпосереднього джерела даних (реляційної бази даних), та компонент виконання користувацького SPARQL запиту (що може містити логіку, або взаємодіяти з компонентом злиття даних у режимі реального часу).

3.3.1 Компонент генерації онтології

Діаграма класів для описаного раніше компоненту генерації онтології (Ontology generation) зображена на рисунку 3.6. Тут клас «DBSemanticsGenerator» - містить логіку безпосередньої конвертації метаданих бази даних до вигляду онтології, клас «OWLWriter» - містить логіку запису коректних концептів, атрибутів онтології, клас «SqlToXsdDtMapper» - відповідає за приведення SQL типів даних до XSD типів даних (і є статичним), та клас «DBLoader» - містить логіку запиту деяких метаданих з реляційної бази даних.

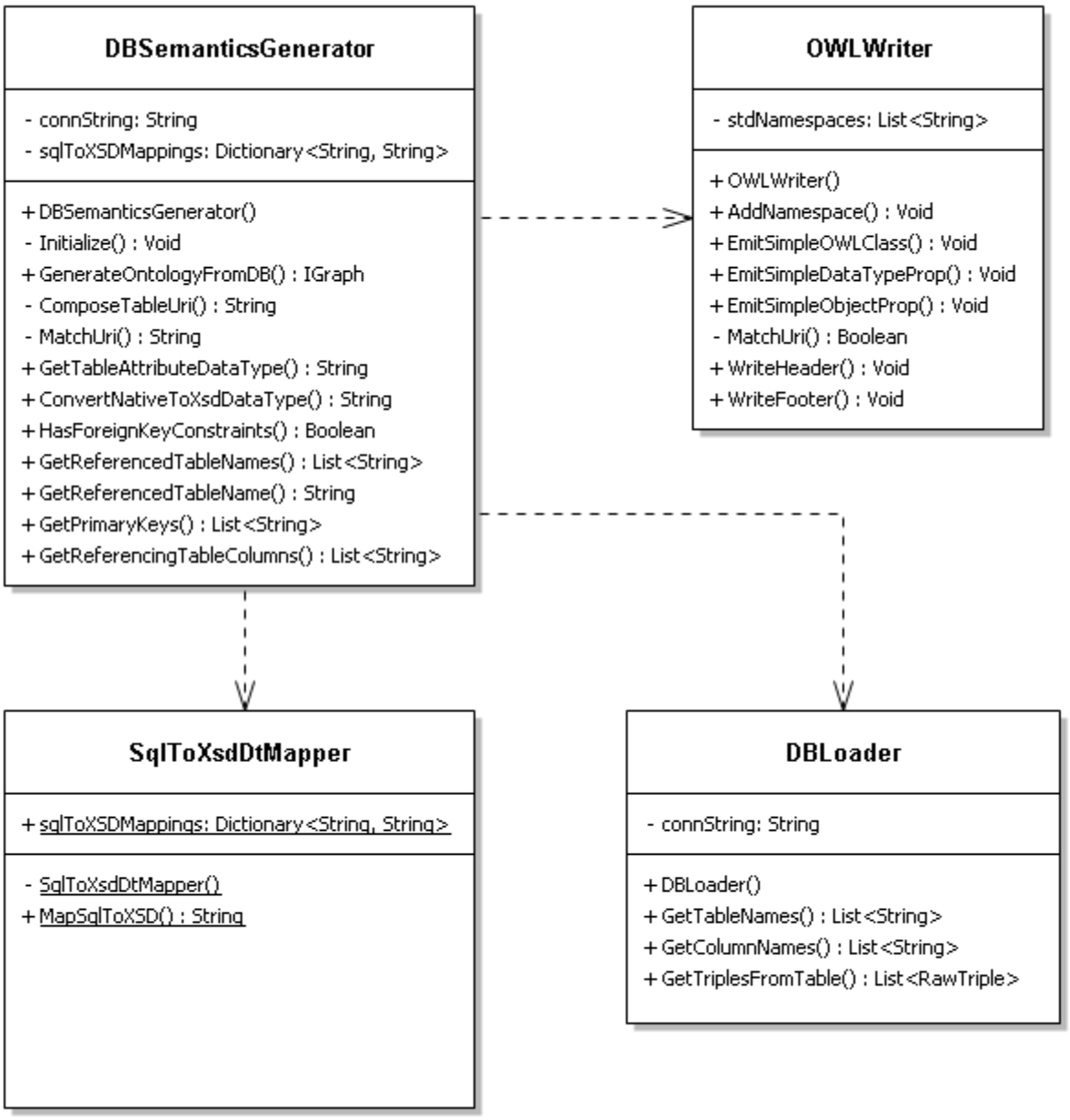

Рисунок 3.6 – Діаграма класів для компоненту генерації онтології

3.3.2 Компонент злиття двох різних онтологій

Компонент злиття двох онтологій приймає на вхід дві онтології (що згенеровані компонентом генерації онтологій), що написані на мові OWL (версії 1.0-2.0), та, використовуючи семантичний словник для англійської мови WordNET та вказівки користувача, намагається злити дві онтології в одну та згенерувати федеративну онтологію. Діаграма класів для цієї компоненти зображена на рисунку 3.7.

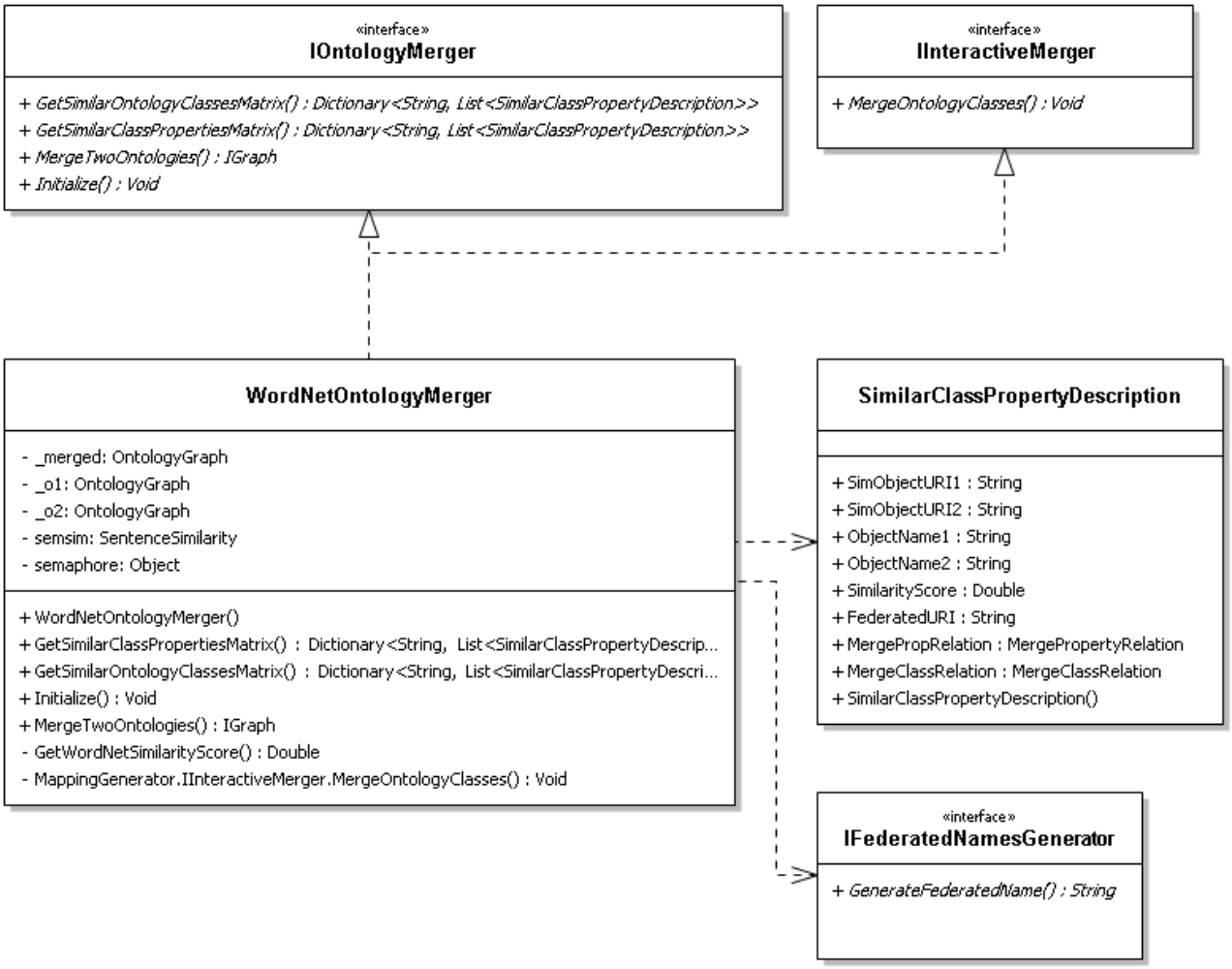

Рисунок 3.7 – Діаграма класів для компоненти злиття онтологій

Тут клас «WordNetOntologyMerger» - містить основну логіку з інтеграції онтологій та надає можливість зливати онтології у двох режимах: автоматичному та напівавтоматичному (хоча, зважаючи, на складність предметної області, рекомендується напівавтоматичний режим). Для генерування імен концептів/атрибутів федеративної онтології використовуються об’єкти, що імплементують інтерфейс «IFederatedNamesGenerator».

3.3.3 Компонент доступу до джерела даних

Даний компонент містить класи з логікою доступу до бази даних, логікою попередньої конвертації отриманих даних до синтаксису «subject-predicate-object» (що знадобиться при обробці користувацького SPARQL запиту). Діаграма класів для цього компоненту зображена на рисунку 3.8.

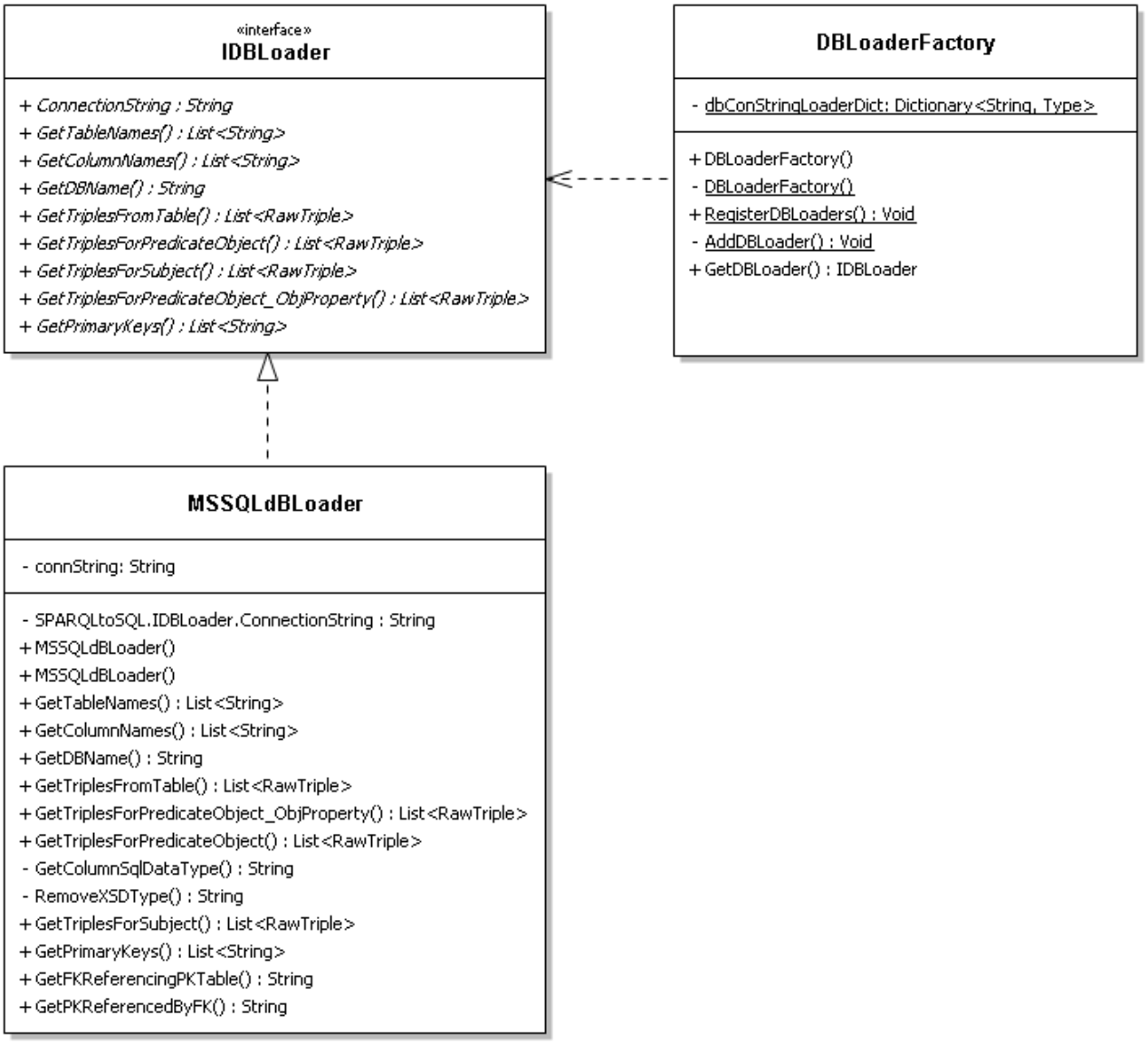

Рисунок 3.8 – Діаграма класів компоненту доступу до джерела даних

3.3.4 Компонент виконання користувацького SPARQL запиту

Компонент виконання SPARQL запиту містить логіку щодо виконання безпосередньго запиту користувача та взаємодіє з компонентом щодо попереднього завантаженя та обробки (інтеграції у реальному часі) даних з джерела даних. Діаграма класів даної компоненти зображена на рисунку 3.9.

На діаграмі також показана взаємодія даного класу (QuantumQueryProcessor) з іншими важливими інтерфейсами та класами. Саме згаданий клас містить логіку обрання розподілених джерел даних для запиту та попередньої обробки та

інтеграції отриманих даних. Після чого використовується інтерфейс «ISparqlQueryProcessor» для виклику обробника запитів бібліотеки dotNetRDF. Отримані дані зберігаються у пам’яті, тож система не оптимізована для нецільових запитів (наприклад, запиту всієї бази даних користувачів системи KMS, що може мати величезні розміри).

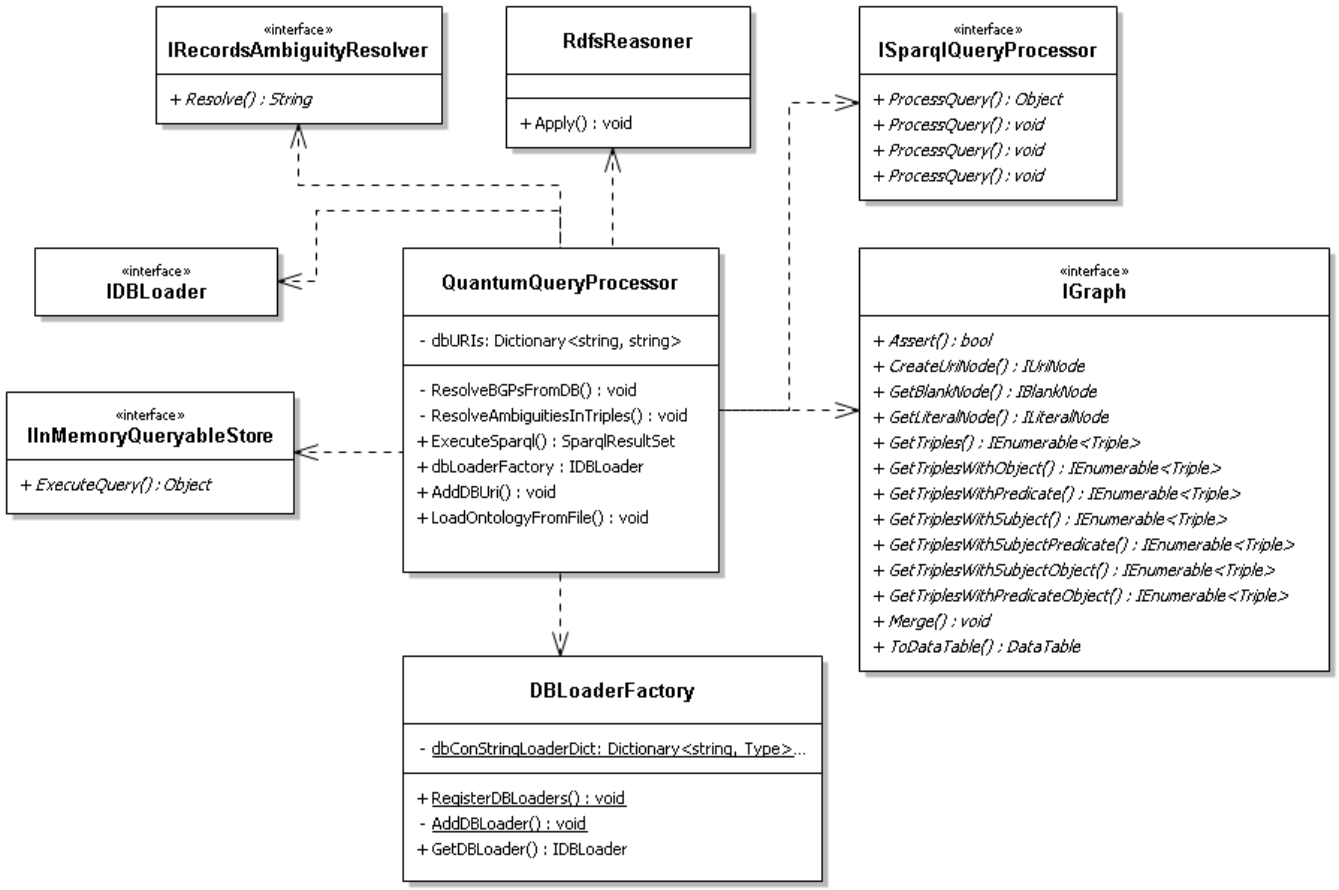

Рисунок 3.9 – Діаграма класів компоненти обробки SPARQL запиту

3.4 Алгоритм інтеграції даних під час обробки SPARQL запиту

Даний алгоритм (рисунок 3.10) містить дві головні частини: процедура «Обробити BGP ноди» (BGP – Basic Graph Pattern), що і є головною логікою виконання запиту до федеративної схеми (отримання даних з кінцевих джерел даних та їх первинна обробка); друга ж частина (після цієї процедури) містить логіку щодо інтеграції отриманих даних та усунення помилок під час інтеграції, далі викликається стандартний процесор виконання SPARQL запитів.

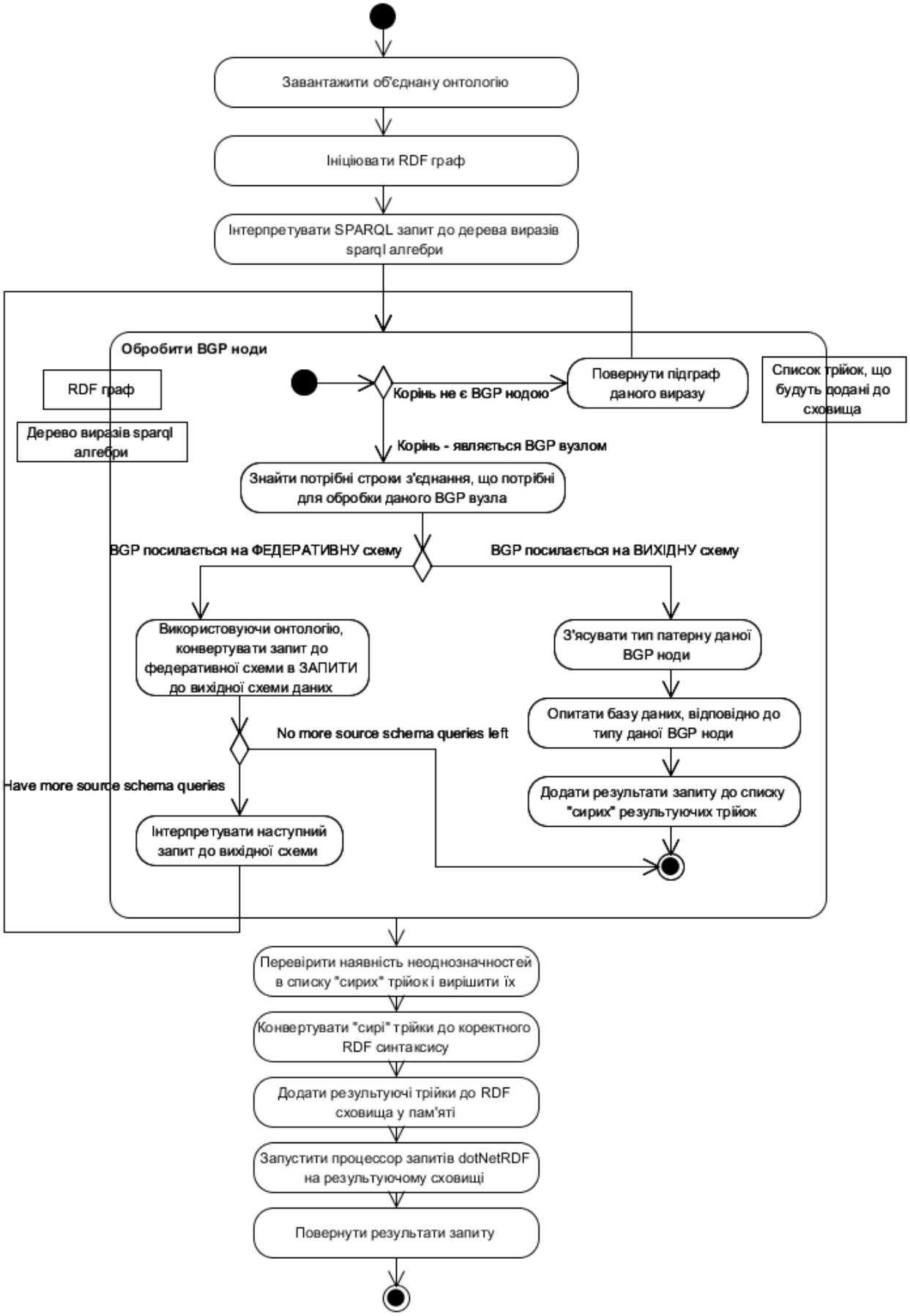

Рисунок 3.10 – Алгоритм виконання запиту

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка тест плану проекту за стандартом IEEE 829-2008

4.1.1 Вступ

Метою даної бакалаврської практики є моделювання програмного забезпечення інтегратора баз даних систем KMS та LMS, розробка однієї з частин даного ПЗ та його тестування (оскільки неможливо за час виробничої практики розробити та протестувати увесь проект). Тому, на даному етапі вже була розроблена компонента генерації онтології, використовуючи метадані реляційної бази даних (компонента «Ontology generator», та головний клас цієї компоненти – «DBSemanticsGenerator»).

4.1.2 Об’єкти, що тестуються

На даному етапі тестується компонента генерації онтологій («Ontology generator»), а саме, її головний клас «DBSemanticsGenerator».

4.1.3 Функціональність, що тестується

Тестується функціональність генерації онтології з реляційної бази даних (використовується база даних MS SQL), функціональність коректного приведення SQL типів до XSD типів даних (для вказання таких властивостей атрибутів, як “range” та “domain”), функціональність отримання метаданих вказаної бази даних (таких метаданих, як назви таблиць, назви первинних ключів таблиць, назви таблиць, на які ссилається конкретний зовнішній ключ (foreign key). Також тестується вихідний формат онтології: він повинен мати концепти (що відповідають назвам таблиць бази даних), атрибути (відповідають атрибутам таблиці бази даних), та властивості атрибутів – «domain», «range».

4.1.4 Функціональність, що не тестується

У даному випадку не тестується взаємодія з різними базами даних (від різних виробників), участь у тестування бере лише база даних MS SQL.

4.1.5 Метод тестування

У даному випадку використовується підхід «білого ящика» під час проведення модульного та інтеграційного тестування.

4.1.6 Критерій проходження / не проходження тесту

Об’єкт проходить тестування, якщо його поведінка (вихідні дані) співпадає з очікуваною – критерій тестування (assertion criteria). У всіх інших випадках, об’єкт не проходить тестування.

4.1.7 Критерій зупинки та продовження тестування

Тестування модулю повинно бути зупинено, якщо хоча б один з модульних тестів показав помилку при тестуванні.

Тестування може бути продовжене, якщо помилка виправлена, та всі модульні тести не виявляють помилки реалізації (всі компоненти поводяться так, як визначено у вимогах).

4.1.8 Артефакти тестування

Даний звіт (із скріншотом проходження тестування) та вихідний код тестів (що не входить у даний звіт) є єдиними артефактами.

4.1.9 Середовище тестування

1) Операційна система: Windows 7, build 7601, (64-bit);

2) процесор: Intel Core I5-3230M, 2.6GHz (4 CPUs);

3) оперативна пам’ять: 8192 MB;

4) .NET Framework 4.5;

5) роздільна здатність екрану: 1366x768 (32bit) (59Hz);

6) Інструменти тестування: MS Visual Studio 2015 Community Edition + Microsoft Unit Testing framework (unit testing).

4.1.10 Можливі ризики та непередбачені обставини

Можливі помилки при під’єднанні до бази даних (непрацюючий сервіс бази даних, наприклад), конфлікт тестуючого фреймворку (Microsoft Unit Testing framework) із включеними до ПЗ компонентами.

4.2 Модульне тестування елементів розробленої компоненти

Модульне (компонентне) тестування (Unit testing) – ізольоване тестування кожного окремого елемента програмного забезпечення (це можуть бути модулі, класи, об’єкти, функції), в штучно створеному середовищі [6].

Згідно з тест планом, був протестований головний клас даної компоненти – “DBSemanticsGenerator”, та виявлено 2 помилки (які були потім виправлені). На рисунку 4.1 зображено кінцевий результат модульного тестування – всі функції даного класу працюють так, як і повинні, згідно з вимогами.

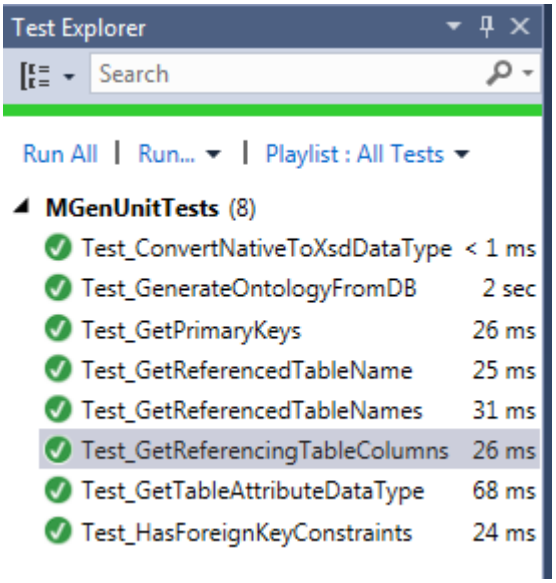

Рисунок 4.1 – Результати модульного тестування

4.3 Інтеграційне тестування компоненти

Інтеграційне тестування тісно пов’язане із модульним тестуванням, воно проводиться з метою перевірки взаємодії компонентів програмного забезпечення між собою. Існує декілька підходів до інтеграційного тестування, серед яких підходи «зверху-вниз» та «знизу-вверх» [7].

У даному випадку потреба в інтеграційному тестуванні мотивується використанням даною компонентою (Ontology generator) ще декількох компонент, що перебирають на себе деяку функціональність з отримання метаданих з бази данних. Результат інтеграційного тесту вже був зображений на рисунку 4.1 (тест-функція «Test_GenerateOntologyFromDB»). Даний тест, використовуючи

спеціально розроблену для цього базу даних, набір таблиць (та атрибутів) яких заздалегідь відомий, тестує загальну поведінку всієї компоненти генерації онтології виходячи з бази даних (тестується, також, взаємодія функцій між собою). Щоб згенерувати таку онтологію компонента повинна правильно отримати метаданні бази даних, коректно сконвертувати SQL типи даних до XSD типів даних, відобразити кожну з таблиць до OWL-класу (окрім таблиць, що зображають відношення N:M (багато до багатьох) – такі таблиці повинні бути відображені до атрибуту типу owl:ObjectProperty відповідного онтологічного класу), правильно записати значення властивостей атрибутів «range» та «domain». Також, компонента звертається до відповідних методів класів бібліотеки dotNetRDF (яка ще знаходиться в активній розробці). Тож, для перевірки коректної поведінки всього модулю, ми повинні провести інтеграційний тест. Код для даного тесту наведений нижче:

```
        [TestMethod]
         public void Test_GenerateOntologyFromDB()
         {
             //test owl file creation, test all classes/attributes to be present
             //Arrange
             string fileName = $"{dbName}_test.owl";

             //Act
             IGraph g = generator.GenerateOntologyFromDB(dbName, $"xmlns: {dbPrefix}
=\"{dbURI}\"", fileName);

             //Assert
             Assert.IsTrue(File.Exists(fileName)); //check file creation

             OntologyGraph ograph = new OntologyGraph();
             ograph.Merge(g);

             //check that each table is present in ontology as class
             IEnumerable<OntologyClass> oclasses = ograph.AllClasses;
             foreach(string tableName in tables.Except(nmRelTables))
             {
                 string classUri = $"{dbURI}{tableName}";
                 OntologyClass oclass = (from c in oclasses where (c.Resource as
UriNode).Uri.ToString() == classUri select c).FirstOrDefault();
                 Assert.IsNotNull(oclass);

                 //check for properties
                 OntologyClass classOfClasses =
ograph.CreateOntologyClass(ograph.CreateUriNode("owl:Class"));
                 var classProps =
ograph.GetTriplesWithPredicateObject(ograph.CreateUriNode("rdfs:domain"),
ograph.CreateUriNode(new Uri(classUri))).ToList();

                 foreach(var propNameExpected in tableAttributesDict[tableName])
                 {
```

```
                    Triple triple = classProps.FirstOrDefault(t =>
t.Subject.ToString().Split('#')[1] == propNameExpected);
                    Assert.IsNotNull(triple); //if null -> we didn't find the property in
ontology that should be there
                }
            }

            //check that all n:m tables are mapped as object properties
            IEnumerable<OntologyProperty> objectProperties = ograph.OwlObjectProperties;
            foreach(var opropName in nmRelTables)
            {
                Assert.IsNotNull(objectProperties.FirstOrDefault(p =>
p.ToString().Split('#')[1] == opropName));
            }
        }
```

ВИСНОВКИ

У процесі проходження практики відбулося ознайомлення з перспективними напрямками розвитку систем обробки інформації на базі практики (компанія “Академія СМАРТ”); методиками та методологіями розробки програмних систем, що використовуються на сьогоднішній день; обов'язками службових осіб підрозділів компанії; стандартами, нормами і іншою нормативно-довідковою документацією, що використовується у підрозділі (стандарти SWEBOK, ISTQB та інші).

Відповідно до завдання дипломного проекту і знань, отриманих в компанії, відбувся вибір моделі розробки ПЗ проекту, моделювання та розробка одної з основних компонентів проекту, розробка тест-плану та саме тестування розробленої компоненти.

Навчальна практика сприяла заглибленню і закріпленню теоретичних знань, отриманих в вищому навчальному закладі зі спеціальності «Програмна інженерія».

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1 Academy Smart // _http://academysmart.com.ua/, 02.05.17

2 Лекція №1 Інтеграція даних // _moodle.onaft.edu.ua/pluginfile.php/1165/mod_resource/content/1/Лекція1 мадо.pdf, 02.05.17

3 What is an Ontology? // _http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 02.05.17

4 Web Ontology Language // _https://www.w3.org/2001/sw/wiki/OWL, 02.05.17

5 Ітеративна модель // _http://lviv.qalight.com.ua/baza-znan/iterativna-model-iterative-model/, 02.05.17

6 Модульне тестування // _http://lviv.qalight.com.ua/baza-znan/modulne-testuvannya, 06.05.17

7 Software testing (SWEBOK) // _http://swebokwiki.org/Chapter_4:_Software_Testing, 06.05.2017