Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи аналізу текстів програм на основі семантичних моделей

(тема)

Виконав:

студент II курсу, групи СПзм-20-1

Бочарова О.О.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: зав. каф. Коваленко А.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бочаровій Ользі Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Методи аналізу текстів програм на основі семантичних моделей

затверджена наказом по університету від " 25 " березня 2022 р. № 33 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи
аналіз тексту
семантична модель
тестовий набір даних
вихідний текст програм

4. Перелік питань, що потрібно опрацювати у роботі
Проблематика аналізу віхідного тексту програм
Метод аналізу вихідних текстів програм
Програмна реалізація

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 19 слайдів

6. Консультанти розділів роботи (заповнюється 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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання.Аналіз предметної області	25.03.2022–09.04.2022	
2	Аналіз існуючих моделей та методів	10.04.2022–26.04.2022	
3	Розробка моделі	27.04.2022–30.04.2022	
4	Описм семантичної моделі програми	01.05.2022–04.05.2022	
5	Отримання та аналіз результатів	05.05.2022–08.05.2022	
6	Оформлення пояснювальної записки	09.05.2022–13.05.2022	

Дата видачі завдання 25 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ зав.каф. Коваленко А.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 67 с., 14 рис., 1 табл., 1 дод., 19 джерел.

ПРОГРАМНІЙ ПРОДУКТ, СЕМАНТИЧНА МОДЕЛЬ, МЕТОД, КОМПІЛЯЦІЯ, IDE

Метою кваліфікаційної роботи є досліження існуючих методів аналізу текстів програм на основі семантичних моделей.

У ході виконання кваліфікаційної роботи розроблено метод предметно-орієнтованого аналізу на основі ітеративних перетворень семантичних моделей програм, що дозволяє здійснювати семантичний аналіз текстів програм, що розробляються із застосуванням кількох мов програмування. Розроблено програмну архітектуру та структури даних для подання тексту та семантичних, що динамічно змінюються моделей програм, що дозволяє інтегрувати реалізацію розробленого методу предметно-орієнтованого аналізу в різні середовища розробки.

ABSTRACT

Master’s thesis: 67 pages, 14 figures, 1 tables, 1 appendices, 19 sources.

SOFTWARE PRODUCT, SEMANTIC MODEL, METHOD, COMPILATION, IDE.

The major goal of this thesis is to investigate the existing methods of analyzing program texts based on semantic models.

In order to a method of subject-oriented analysis was developed based on iterative transformations of semantic models of programs, which allows for semantic analysis of program texts developed using several programming languages. The software architecture and data structures for the presentation of text and semantic dynamically changing program models have been developed, which allows to integrate the implementation of the developed method of subject-oriented analysis into various development environments.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

EMF – Eclipse Modeling Framework

IDE – Integrated Development Environment

ПЗ – програмне забезпечення

ВСТУП

У галузі розробки програм та програмних систем велике значення має інструментальне програмне забезпечення, призначене для підтримки процесу розробки. Основним інструментом є інтегроване середовище розробки (Integrated Development Environment, IDE) – комплекс програм, що надає розробнику засоби редагування вихідного коду, контролю версій, налагодження, трансляції, організації командної роботи та інші можливості. Однією з основних функцій інтегрованих середовищ розробки є редагування вихідного програмного коду.

Сучасні текстові редактори здатні частково автоматизувати процес розробки, що суттєво підвищує продуктивність праці програміста. Можливості текстових редакторів дозволяють виділяти кольором синтаксичні елементи (т.зв. «підсвічування синтаксису» від англ. «syntax highlight»), повідомляти програміста про помилки, автоматично доповнювати синтаксичні конструкції при наборі, виконувати навігацію за вихідним кодом, візуалізувати програми.

Особливістю сучасної індустрії розробки програмного забезпечення є застосування кількох мов програмування у в одному програмному проекті. Ця особливість породжує проблему перевірки узгодженості компонентів, реалізованих різними мовами і застосовуваних у складі однієї програмної системи. Як правило, неузгодженість компонентів відстежується тільки на етапі налагодження або тестування. Наприклад, якщо в програмі на мові C # використовується функція, реалізована в окремому модулі на мові Сі, і сигнатура цієї функції змінилася, то відстежити помилку, що виникла внаслідок цього, в програмній системі можливо тільки під час виконання. Такий стан справ істотно знижує ефективність процесу розробки.

Дана робота спрямована на розвиток підходів у галузі аналізу вихідного коду програм, написаних кількома мовами програмування, з метою

виявлення подібних помилок на етапі кодування, що скоротить кількість циклів редагування-налагодження-тестування і в цілому позитивно вплине на процес розробки програмного продукту.

Об'єкт дослідження в даній роботі – вихідні тексти та семантичні моделі програм, написаних з використанням кількох мов програмування.

Мета кваліфікаційної роботи – виявлення помилок у програмному коді, написаному з використанням декількох мов програмування, етап редагування вихідних текстів програм.

Відповідно до мети в даній роботі ставляться і вирішуються такі завдання:

- аналіз способів внутрішнього подання та обробки редагованого тексту програм в інтегрованих середовищах розробки з метою виявлення недоліків існуючих засобів підтримки мультимовних програмних проектів.
- розробка методу предметно-орієнтованого аналізу вихідних текстів програм у процесі редагування на основі семантичних моделей.
- розробка структур даних та програмної архітектури для внутрішнього подання редагованого тексту та семантичних моделей програм;
- розробка та реалізація програмного засобу аналізу вихідних текстів програм;
- апробація розробленого методу.

1 ПРОБЛЕМАТИКА АНАЛІЗУ ВІХІДНОГО ТЕКСТУ ПРОГРАМ

1.1 Семантичний аналіз програм

У цій роботі поняття семантика розглядається стосовно мов програмування, трансляторів і аналізаторів вихідних текстів програм, як спосіб представлення конструкцій мов програмування за допомогою формальних математичних моделей, для побудови яких може використовуватися апарат математичної логіки, λ-числення, теоріографи, теорія множин, реляційна алгебра.

Якщо лексичний та синтаксичний аналіз мають справу зі структурними особливостями програми – зовнішніми текстовими конструкціями мови, то семантика орієнтована на змістовну інтерпретацію «сенсу» об'єктів, описаних у програмі, таких як типи даних, змінні, функції, і у внутрішньому уявленні виглядає як система взаємопов'язаних об'єктів. Наприклад, у мовах програмування C/C++ змінна «посилається» на опис типу даних, а похідний тип даних «посилається» на базовий тип. Для кожної безлічі об'єктів одного типу створюються звані семантичні таблиці. При цьому елементи різних таблиць пов'язані між собою (об'єкти різного типу «посилаються» один на одного) і логічно утворюють структуру, подібну до бази даних, схема якої описується в термінах предметної області мови програмування.

Таке внутрішнє представлення семантики програми як безлічі іменованих об'єктів, із якими працює програма, з описом їх властивостей, показників і зв'язків називається семантичної моделлю програми. Таким чином, для реалізації трансляторів або аналізаторів будь-якої мови програмування крім синтаксису має значення семантика, а саме семантична модель програми, що формується вихідним кодом на даній мові програмування. Семантичний аналіз визначає, чи має створена в тексті програми синтаксична структура припустиме значення, і виявляє такі

помилки, як невідповідність типів і параметрів, використання зарезервованих слів, багаторазове оголошення змінних в області видимості, спроба доступу до змінних поза області видимості та інші.

Для програмних проектів, у яких задіяно кілька мов програмування, має значення узгодженість між частинами програмного коду, написаними різними мовами.

Наприклад, для веб-застосунку, реалізованого відповідно до архітектури SPA (Single-Page Application) [1], має значення узгодженість програмного інтерфейсу (Application Programming Interface, API) веб-сервісу, написаного мовою програмування Java, з клієнтським веб-застосунком, написаним мовою JavaScript, яка взаємодіє з даним веб-сервісом.

Для аналізу такої узгодженості необхідно мати правила відображення об'єктів семантичної моделі в термінах однієї мови програмування на об'єкти семантичної моделі в термінах іншої мови програмування або об'єктно-орієнтованої мови.

Можна виділити два основні сценарії виконання такого аналізу. Перший сценарій має на увазі виконання аналізу на етапі складання проекту або в процесі статичного аналізу коду програми окремим автономним інструментом - аналізатором. Другий - виконання аналізу в процесі редагування тексту програми текстовим редактором або інтегрованим середовищем розробки.

У першому випадку результатом роботи аналізу є звіт про структуру семантичної моделі зі статусом перевірених в ході аналізу обмежень і списком невідповідності між моделлю програми, побудованої на основі її тексту, і заданими параметрами аналізу. Для застосування результатів аналізу необхідна нова ітерація - повторне редагування тексту програми, в процесі якого можуть бути внесені некоректні зміни, з подальшим повторним аналізом і отриманням нового звіту.

У другому випадку результатом роботи аналізатора є візуальний зворотний зв'язок з розробником, що редагує код програми, яка полягає в

підсвічуванні синтаксису, підказках і діалогових вікнах середовища розробки.

Другий сценарій є кращим, так як результати аналізу, що виконується у відповідь на кожну внесену в текст програми зміну, візуалізуються в процесі редагування тексту програми, вказуючи на місце у програмному коді, що вимагає уваги розробника.

Таким чином, можна виділити два основні підходи до організації аналізу семантичних моделей програм:

- послідовна побудова всіх необхідних семантичних моделей на основі тексту програми з подальшим аналізом узгодженості;
- гранулярна побудова семантичних моделей з одночасним аналізом їх сумісності та коректності в міру побудови в процесі редагування тексту програми.

Розглянемо деякі способи подання семантичної інформації про програми. Перед цим перерахуємо ряд завдань, вирішення яких вимагає використання семантичної інформації.

Створення компіляторів, невід'ємною частиною яких є аналіз текстів програм, що породжує артефакти, що містять інформацію про програму, її алгоритми і структури даних у формі, що відрізняється від вихідного тексту.

Створення інтерпретаторів та динамічних середовищ виконання для мов високого рівня.

Створення інструментів статичного аналізу та верифікації програм, що аналізують тексти програм на відповідність специфікаціям та обмеженням. Такі інструменти не входять до складу текстових редакторів та інтегрованих середовищ розробки.

Дані завдання пов'язані з переформулюванням інформації про аналізовану програмі чи алгоритмами з предметної області вихідної мови в предметну область аналізу. Подальший аналіз виконується в термінах і поняттях спеціальної моделі, не пов'язаної з вихідною мовою або програмним проектом.

Створення інтегрованих середовищ розробки, що передбачають можливості автоматизації дій розробника, таких як підсвічування синтаксису, автодоповнення, елементи статичного аналізу – операції над текстом програми та перевірка його коректності щодо деяких обмежень у процесі редагування.

Рішення вищезазначених завдань пов'язане з необхідністю формалізованого подання семантичної інформації, що витягується з тексту програми, і вимагає від розробника розуміння онтології предметної області – концептуального опису безлічі об'єктів зв'язків між ними в термінах конкретної мови програмування. Це необхідно для побудови структур даних, що містять семантичну інформацію.

Для представлення семантичної інформації про текст програми можуть використовуватися різні способи представлення даних. Це залежить від конкретних вимог програмного проекту, обраного варіанта програмної архітектури та інструментарію, що використовується.

Спеціалізовані структури даних [2] та об'єктні моделі, елементи яких відповідають елементам предметної області тексту програми. Елементи семантичної інформації у разі представлені у пам'яті екземплярами конкретних сутностей, наприклад, структур чи класів.

Узагальнені структури даних та способи їх подання, спочатку розроблені для представлення різних моделей даних, таких як реляційні або у вигляді графа.

Елементи відповідного представлення даних (таблиці чи графи) можуть розміщуватися як усередині програми, і у зовнішніх базах даних. У другому випадку можливе використання як уявлень, що спираються у явному вигляді на схему даних [3], так і матеріалізованих уявлень, що відрізняються матеріалізацією даних у формі структур, внутрішня організація яких будується на основі онтології предметної області та відповідної семантичної інформації [4].

При вирішенні конкретної задачі незалежно від обраного уявлення

(об'єктна модель, узагальнена модель у вигляді графа або реляційна модель), обсяг семантичної інформації буде один і той же. На рівні смислових зв'язків він буде ідентичний через вимоги самого завдання. Це не залежить від використання явних посилань між об'єктами в структурі даних, між елементами реляційних відносин або відносин сутностей семантичної мережі. При цьому для розробника, зацікавленого в семантичній інформації, важливий не так спосіб її представлення, як спосіб опису допустимих над семантичної інформацією дій.

Спосіб подання семантичної інформації впливає на алгоритми семантичного аналізу, а способи опису специфікацій, кроків аналізу та обмежень впливають на можливості застосування розробником аналізатора для аналізу семантичної інформації, що витягується з тексту програми.

Тепер розглянемо етапи життєвого циклу програмного забезпечення, на яких може знадобитися аналіз вихідних текстів програм. Центральне місце у процесі розробки ПЗ займає написання самого тексту програми, і навіть складання програмного проекту з вихідних текстів. На рисунку 1.1 показані важливі етапи життєвого циклу ПЗ, в яких торкається текст програми. Зокрема, до них належать тестування та виправлення виявлених помилок.

Рисунок 1.1 – Життєвий цикл програмного продукту

Кожен з цих етапів нерозривно пов'язаний з виявленням та виправленням помилок у тексті програми. На етапі написання тексту програми для цього застосовуються засоби, інтегровані в середу розробки та аналізують текст програми у процесі редагування. На етапі збирання проекту для цього застосовуються транслятори, зокрема компілятори або генератори коду [5], що аналізують текст програми у процесі складання.

Далі слідує етап тестування, в ході якого виявляються помилки, пов'язані з впливом структури програми на її функціональність, що є предметом тестування. Незалежно від способу організації процесу розробки ПЗ процес тестування вимагає попередньої складання програмного проекту, чому свого часу передує написання вихідного коду, а значить, цикл виявлення та виправлення помилок пов'язаний з декількома різними етапами життєвого циклу ПЗ, що вимагає різного часу виправлення помилок залежно від їх етапу виявлення. Чим раніше помилка буде виявлена та виправлена, тим менше часу займе розробка проекту загалом.

У програмних проектах поява помилок, пов'язаних із написанням програмного коду кількома мовами програмування, залежить від узгодженості елементів тексту програми, які відповідають за передачу даних або управління між частинами програми, написаними на різних мовами. Наприклад, під час виклику з програми, написаної мовою C#, функції, написаної мовою C, сигнатура цієї функції має бути відображена за строго визначеними правилами, що гарантують коректну обробку такого виклику за допомогою механізмів p-invoke [8].

Це справедливо та за наявності додаткових вимог до організації вихідний текст програми. Наприклад, використання автоматизованих засобів серіалізації може вимагати додаткової розмітки описів типів даних спеціальними атрибутами, наявності конструкторів по замовчування і т.п.

Те саме стосується і випадків використання предметноорієнтованих мов (Domain Specific Language – DSL) як вбудованих, і зовнішніх. Вбудовані предметно-орієнтовані мови можуть накладати суттєві обмеження на

коректність тексту програми з погляду його структури. Такі обмеження не перевіряються засобами розробки, орієнтованими на семантичний аналіз вихідного коду, написаного мовою програмування загального призначення.

Розробка та аналіз зовнішніх DSL на сьогоднішній день можливі за допомогою створення спеціалізованих аналізаторів або за допомогою використання засобів проектування мов та мовних аналізаторів. Вони також мають ряд обмежень щодо підтримки інтеграції у існуючі процеси розробки. Таким чином, процедура аналізу вихідних текстів програм специфічна для кожного програмного проекту, а саме залежить від складу використовуваних у проекті мов програмування загального призначення та предметно-орієнтованих мов.

Також на процедуру аналізу впливає спосіб представлення елементів прикладної предметної області у вихідному коді. Для адаптації процедури аналізу особливо програмних проектів необхідний інструментарій, що відповідає за семантичний аналіз. У рамках виконання цієї роботи питання аналізу розглядається ширше та представлений як предметно-орієнтований аналіз текстів програм чи семантичних моделей програм з урахуванням вимог та обмежень, що накладаються семантикою програмних проектів.

1.2 Дослідження існуючих підходів до семантичного аналізу текстів програм

При дослідженні існуючих підходів до семантичного аналізу Тексти програм було розглянуто ряд наукових праць та інженерних рішень. У статті [7] порівнюються три підходи до реалізації зовнішніх та вбудованих предметно-орієнтованих мов, жодна з яких не передбачає можливості перевірки коректності семантичної інформації, що витягується з аналізованих текстів у процесі їх редагування. У роботі [8] розглядається перевірка коректності предметноорієнтованих моделей за допомогою їх трансляції в моделі Event-B одночасним аналізом спеціалізованими

інструментами. Такий підхід виключає можливість перевірки коректності аналізованого тексту у його редагування. Він також вимагає, щоб всі аналізовані аспекти тексту програми чи програмної системи могли бути сформульовані у термінах Event-B моделей. Всі решта мови програмування, що використовуються в програмному проекті (у тому числі загального призначення) повинні бути відображені в ті ж Event-B моделі, що у випадку неможливо. У роботі [9] викладено метамодельний підхід для подання інформації про предметні галузі.

Пропонується використання спеціальної мови KM3 для опису метамоделей аналізованих мов, включаючи їх семантику, та подальше відображення зібраної з вихідних тексти інформації в тексти іншими мовами. Така трансляція виконується за допомогою зовнішніх по відношенню до редактора інструментів та передбачає подальший аналіз цільових моделей незалежним від вихідних мов у спосіб, що унеможливлює зв'язування інформації про помилки з первинним вихідним текстом.

При дослідженні існуючих підходів до семантичного аналізу Тексти програм було розглянуто низку наукових праць та інженерних рішень. У статті [10] порівнюються три підходи до реалізації зовнішніх та вбудованих предметно-орієнтованих мов, жодна з яких не передбачає можливості перевірки коректності семантичної інформації, що витягується з аналізованих текстів у процесі їх редагування.

У роботі [11] розглядається перевірка коректності предметноорієнтованих моделей за допомогою їх трансляції у моделі Event-B одночасним аналізом спеціалізованими інструментами. Такий підхід виключає можливість перевірки коректності аналізованого тексту у його редагування Він також вимагає, щоб всі аналізовані аспекти тексту програми чи програмної системи бути сформульовані в термінах Event-B моделей. Усч решта мов програмування, що використовують у програмному проекті (у тому числі загального призначення) повинні бути відображені у ті ж Event-B моделі, що у випадку неможливо.

У роботі [12] викладено метамодельний підхід для подання інформації про предметні галузі. Пропонується використання спеціальної мови KM3 для опису метамоделей аналізованих мов, включаючи їх семантику, та подальше відображення зібраної з вихідних тексти інформації в тексти на інших мовах. Така трансляція виконується за допомогою зовнішніх по відношенню до редактора інструментів та передбачає подальший аналіз цільових моделей незалежним від вихідних мов у спосіб, що унеможливлює зв'язування інформації про помилки з первинним вихідним текстом У роботі [13] описується підхід до аналізу операційних та трансляційних семантик мов програмування, призначених для описи алгоритмів. Наведений метод діє в рамках Eclipse Modeling Framework, що додатково обмежує його способи використання також, як і KM3.

Стаття [14] розглядає способи побудови редакторів, керованих специфікаціями моделей, яким має підкорятися редагований текст. З точки зору роботи користувача з редактором, виділяється два підходи. Перший – підхід на основі фіксованого набору візуальних шаблонів, у формі ієрархії яких користувачу надається можливість маніпуляцій у термінах структури об'єктної моделі. Другий – підхід на основі текстових розпізнавачів (синтаксичних аналізаторів), де користувачеві для роботи пропонується традиційне текстове уявлення, не обмежене візуально та структурно. Внутрішня модель будується під час аналізу редагованого тексту окремо від процесу редагування. Розглядаються також шляхи поєднання цих способів, але не розглядається шляхи підтримки актуальності семантичних моделей редагованого тексту, необхідних для забезпечення функціональності середовищ розробки.

Слід уточнити, що предметом дослідження є саме способи виконання аналізу семантичної інформації, отриманої на основі текстів програм, а не тільки синтаксичних структур у вихідному текст, що виключає з розгляду засоби генерації парсерів, що вимагають самостійної реалізації логіки семантичного аналізу.

Проведене дослідження показало, що представлені в розглянутих роботах кошти аналізу можуть бути розбиті на три групи:

а) не призначені для використання в реальному часі процесі редагування тексту;

б) призначені для використання у процесі редагування тексту програми всередині спеціалізованого середовища розробки або спеціалізованого режиму редагування;

в) призначені для використання у процесі редагування моделей програм у термінах, близьких до моделі абстрактного синтаксичного дерева, які не розглядають аналіз текстів програм.

Таким чином, жоден з існуючих інструментів, призначених для виконання семантичного аналізу, не може бути вбудований в інтегровані середовища розробки для виконання аналізу текстів програм у процесі їх редагування, якщо спочатку він не створювався для цього інтегрованого середовища розробки. Крім того, написання специфікацій, що задають правила семантичного аналізу, у всіх випадках передбачає вивчення спеціалізованих предметних областей, таких як мови та середовища метамоделювання, синтаксичний та семантичний аналіз, онтологічне моделювання.

Висновок – вирішення завдання розробки аналізатора, необхідного для конкретного програмного проекту, вимагає від розробника спеціальної підготовки та відволікає від роботи по основному проекту.

1.3 Сучасна реалізація аналізаторів

Для дослідження особливостей програмної реалізації існуючих підходів до аналізу текстів програм, що застосовуються в сучасних інструментальних засобів розробки програмного забезпечення забезпечення, були вибрані найбільш популярні за даними Google Trends інтегровані середовища розробки, основні з яких представлені в таблиці 1.1.

Таблиця 1.1 – Популярні IDE

	Назва IDE	Сайт проекту
1	Visual Studio	https://visualstudio.microsoft.com/
2	Eclipse	https://www.eclipse.org/
3	Visual Studio Code	https://code.visualstudio.com/
4	IDEA	https://www.jetbrains.com/idea/
5	NetBeans	https://netbeans.org/
6	Xcode	https://developer.apple.com/xcode/

Всі ці середовища розробки мають технічні можливості інтеграції семантичних аналізаторів та можуть використовувати результати роботи аналізаторів, надаючи розробнику функціональні можливості, що частково автоматизують процес розробки програмного забезпечення.

Реалізація таких функціональних можливостей у різних середовищах розробки відрізняється особливостями програмної архітектури та методами внутрішнього подання текстів програм.

До таких можливостям відносяться:

а) підсвічування синтаксису – відображення тексту з використанням різних стилів в залежності від класифікації лексем, синтаксичних конструкцій та навколишнього контексту;

б) навігація по тексту програм за допомогою спеціальних команд, таких як «перехід до визначення» або «перехід до використання ідентифікатора»;

в) форматування – розстановка прогалин та розподіл тексту за рядків залежно від роду синтаксичних конструкцій та їх призначення;

г) автодоповнення – підказки щодо можливостей коректного введення в тих чи інших позиціях тексту, що спираються на інформацію про структурі контексту та трактування його елементів;

ґ) контекстні підказки, що містять різні повідомлення, наприклад, інформацію про помилки, що асоціюється з тим чи іншим елементом

структури тексту;

д) перевірки та підказки за стилем кодування залежно від класифікації лексем, синтаксичних конструкцій та навколишнього контексту.

У всіх випадках єдиним способом включення до процесу редагування специфічних для окремого програмного проекту перевірок вихідного коду є реалізація розширення середовища розробки.

Єдиним способом, що автоматично вирішує питання сумісності двох частин програмного проекту, є генерація коду цих частин на основу загальної специфікації. Це є еквівалентом створення спеціалізованого розширення інтегрованого середовища розробки, але без можливості аналізу текстів програм у процесі редагування. Реалізація такої можливості здійсненна за умови інтеграції створюваного розширення з текстовим редактором інтегрованим середовищем розробки для спільного використання семантичної інформації, що в існуючих середовищах розробки або передбачено, або утруднено.

Таким чином, за результатами дослідження можливостей семантичних аналізаторів найпопулярніших за даними Google Trends інтегрованих середовищ розробки було виділено п'ять груп функціональність.

Мовна підтримка – підтримка розробки різними мовами програмування.

Управління процесом аналізу – можливість налаштування вбудованих аналізаторів.

Організація доступу до моделей програм – можливість використання інформації про структуру програм аналізаторами.

Робота з мультимовними програмами – можливість вбудованих аналізаторів виконувати аналіз текстів програм, написаних з використанням кількох мов програмування.

Розширюваність системи – можливість створення розширень, функціональність яких потребує модифікації внутрішньої моделі подання даних.

Основним недоліком існуючих інструментальних засобів семантичного аналізу є їх непридатність для аналізу вихідних текстів, написаних із використанням кількох мов програмування. Істотний вплив на вирішення цього завдання надають особливості реалізації аналізаторів, що застосовуються у складі текстових редакторів. Дані особливості пов'язані з тим, як відбувається редагування тексту програми.

Редагування текстів програм зазвичай виконується з використанням засобів текстового редактора, що дозволяє вносити складні зміни за кілька простих кроків, в результаті виконання яких текст програми залишається коректним. Прикладом є засіб автоматичного рефакторингу. Так, для перейменування будь-якої сутності у тексті програми, наприклад, підпрограми, достатньо змінити ім'я у місці її оголошення. Нове ім'я у виразах виклику підпрограми буде замінено автоматично. Це дозволяє зменшувати ймовірність внесення помилок за рахунок обмеження розміру фрагмента тексту, який редагується вручну.

Під час редагування вручну необхідно перевіряти коректність зміненого тексту якнайчастіше, щоб виключити можливість внесення великої кількості синтаксичних помилок, через які стане складно помітити та швидко виправити більше суттєві логічні помилки у тексті програми.

Можна виділити два сценарії аналізу вихідних текстів програм:

- глобальний, тобто. аналіз повного обсягу тексту програми, представленою набором файлів із вихідним кодом;

- локальний, тобто. аналіз послідовності операцій із зміни окремих частин тексту (наприклад, додавання чи видалення окремих символів).

Глобальний аналіз пов'язаний з повним повторним синтаксичним розбір тексту програми для кожної ітерації аналізу. Історично саме такий підхід сприймається як основа для побудови трансляторів. При цьому, як правило, не розглядається повторне використання результатів попереднього виклику аналізатора. Використання такого підходу у процесі редагування тексту пов'язано з необхідністю повного зіставлення синтаксичних моделей

текстів програм для виявлення змінених фрагментів. Це підвищує складність аналізу, поширюючи його на обсяг всього тексту програми або програмної системи загалом як етапу синтаксичного аналізу, так етапу зіставлення синтаксичних уявлень при виявленні змінених фрагментів.

Під час виконання локального аналізу виявлення змінених фрагментів робити не потрібно, оскільки дані про зміни тексту програми містять інформацію про те, до яких частин тексту вносилися зміни, а також зміст цих змін. У процесі аналізу програмної архітектури інтегрованих середовищ розробки були досліджені структури даних, що використовуються для подання тексту під час реалізації текстових редакторів. Це такі структури даних, як "Gap buffer", "Piece table", "Rope" та їх незмінні модифікації, що відрізняються від класичних тим, що вузли, складові їх структуру не можуть змінюватися. Для подання супутньої інформації про текст застосовуються асоціативні структури даних, які використовують як ключі позиції в тексті, такі як «Interval Tree», буфера атрибутів, тощо. Під супутньою інформацією маються на увазі дані про області підсвічування синтаксису, що спливають підказки про стилі, що використовуються при відображенні. Однією з основних проблем при реалізації текстових редакторів для мов програмування є завдання інтеграції аналізаторів, результати роботи яких редактор має відображати.

Інтеграція полягає у передачі інформації про вихідні тексти між внутрішніми уявленнями даних текстового редактора та аналізатора. На рисунку 1.2 наведено загальну структуру взаємодії текстового редактора та аналізатора, що визначає простори імен та рівні візуального, логічного, синтаксичного та семантичного представлення моделей вихідний текст програми. Перетворення уявлень даних текстового редактора здійснюється програмним компонентом, відповідальним за синтаксичний аналіз моделей програм та аналіз тексту програмного коду. Під час редагування тексту модель програми змінюється всіх рівнях її уявлення, а результати аналізу візуалізуються у текстовому редакторі у вигляді підсвічених змін та

повідомлень відповідно до набору правил та обмежень, яким має відповідати мову, що використовується в даному програмному проекті програмування. Передача інформації про зміну тексту із редактора в аналізатор та результатів аналізу назад у структури даних текстового редактора пов'язана з накладними витратами на відображення даних між цими структурами даних.

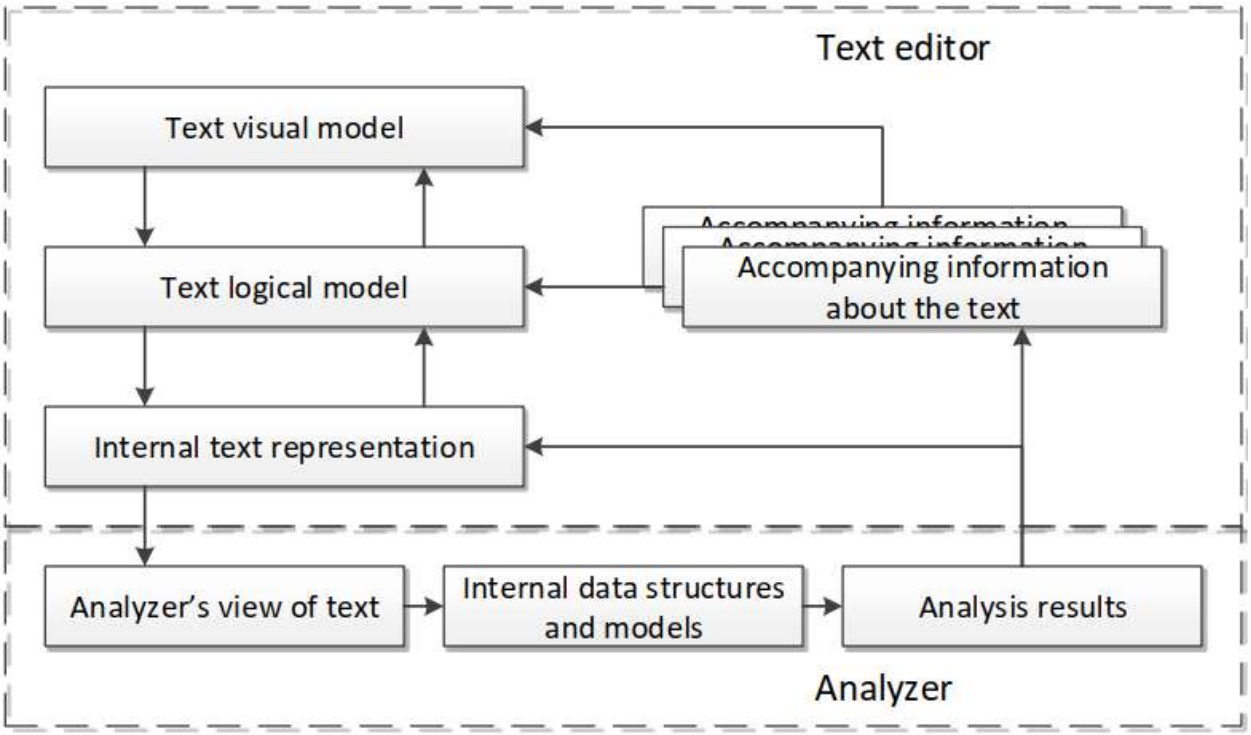

Рисунок 1.2 – Схема процесу передачі інформації в текстових редакторах IDE

При передачі інформації до аналізатора накладні витрати полягають у наступних операціях:

- формування «знімка» стану тексту на момент початку процедури аналізу;
- дублювання текстової інформації зміненого фрагмента при передачу в аналізатор;
- відображення позиції в тексті між знімками різних станів версій при доступі аналізатора до незайманих фрагментів тексту.

При зворотній передачі інформації з аналізатора до текстового

редактор, накладні витрати полягають у зміні структур даних, що забезпечують візуальне подання тексту. Це зв'язано з тим що необхідно зіставити результати аналізу тексту його місцезнаходження в редакторі.

Можна узагальнити та відзначити такі недоліки існуючих рішень.

Наперед визначений набір мов, для яких можливий семантичний аналіз. Додавання підтримки нових мов можливе лише за допомогою спеціалізованих розширень.

Особливості програмної архітектури аналізаторів та текстових редакторів не гарантують узгодженість внутрішнього подання та візуального відображення результатів аналізу текстів програм з актуальним станом аналізованого тексту. Приклад наведено на рисунку 1.3. Підсвічування потенційної помилки неправильно вказує її місце розташування, а текст підказки не відповідає актуальним станом тексту програми.

Мультимовний аналіз обмежений вбудованими поєднаннями мов та певними типами програмних проектів Наприклад, застосування семантичного аналізатора мови шаблонів Razor обмежено проектами ASP.NET.

Рисунок 1.3 – Приклад некоректного аналізу тексту програми

При мультимовному аналізі відсутні налаштування, доступні при аналіз проектів з однією мовою.

Використання наявних способів подання семантичної інформації для предметно-орієнтованого аналізу обмежено або неможливо.

Внаслідок дослідження програмної архітектури інтегрованих середовищ розробки було виявлено, що їх можливості аналізу текстів програм зводяться до реалізації індексуючих модулів, здатних будувати семантичні моделі та обробляти правила лише однієї мови програмування.

У випадках, коли потрібно виконувати аналіз текстів програм, що розробляються із застосуванням кількох мов програмування або предметно-орієнтованих мов, необхідне використання кількох семантичних моделей. Прикладом може бути програмний проект, що розробляється з використанням двох мов програмування – Сі та C#.

Зміна сигнатури функції мовою Сі зробить некоректною частину коду мовою C#, що залежить від цієї функції. Така помилка у порушеній частини програми може бути виявлена лише під час тестування програмної системи, що розробляється.

Однак частини програми на Сі та C# можуть розроблятися різними програмістами, які вирішують не пов'язані між собою завдання. Для усунення такої помилки може знадобитися додаткове узгодження між групами Сі та C# розробників. На сьогоднішній день інтегровані середовища розробки не виконують такого аналізу.

Необхідне створення спеціалізованих модулів сполучення семантичних моделей, що розширюють можливості існуючих середовищ розробки. При цьому можливості налаштування таких розширень часто сильно обмежені, а спільне використання кількох семантичних моделей часто неможливо через особливості програмної архітектури середовища розробки, зокрема у вигляді ізольованості та обмеженості програмних інтерфейсів середовища розробки. На рисунку 1.4 наведено типову схему розширюваності інтегрованих середовищ розробки щодо семантичного аналізу текстів програм.

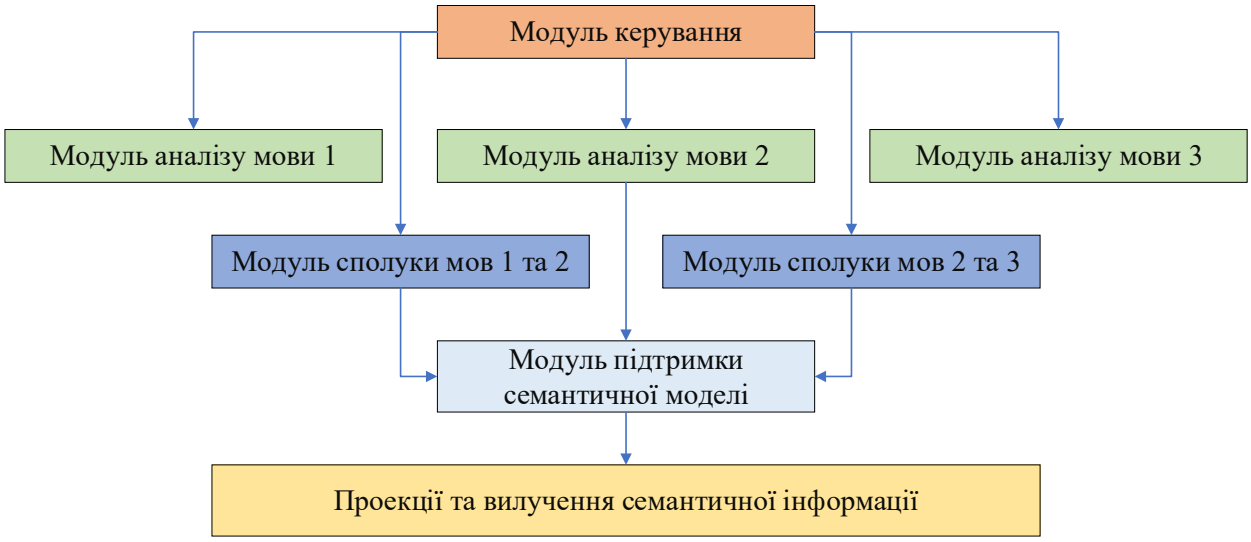

Рисунок 1.4 – Типова схема розширюваності інтегрованих середовищ розробки щодо семантичного аналізу текстів програм

З рисунку 1.4 видно, що при додаванні чергової мови реалізація аналізатора суттєво ускладнюється і є унікальною для кожної групи, що поєднуються в програмному проекті мов. Завдання створення таких розширень є для програміста вторинною по відношенню до основного програмного проекту, що робить їх реалізацію специфічною та часто обмежує застосування таких вузькоспрямованих розширень одним проектом, зважаючи на неможливість використовувати для вирішення тієї ж проблеми у іншому програмному проекті.

Таким чином, завдання розробки методів та засобів, що дозволяють виконувати предметно-орієнтований аналіз над семантичними моделями програм є актуальною. Також актуальність цього питання обумовлена відсутністю на сьогоднішній день засобів виконання такого аналізу безпосередньо у процесі редагування програмного коду, що є одним із основних способів виявлення помилок у тексті програм.

2 МЕТОД АНАЛІЗУ ВИХІДНИХ ТЕКСТІВ ПРОГРАМ

2.1 Загальні відомості

Ключовими поняттями у роботі є семантична модель програми та предметно-орієнтований аналіз. Модель – це форма відображення певного фрагмента вихідної системи, що містить її суттєві властивості та є об'єктом дослідження чи аналізу. Предметна область - це система взаємопов'язаних термінів або понять. Наприклад, система понять мови програмування, система понять прикладної області або спеціальна система понять, що вводиться для розв'язання певної задачі.

Семантична модель програми – це модель, яка описує елементи програми, їх властивості, характеристики та зв'язки у термінах мови програмування чи іншої предметної області. На рівні семантичних моделей виконується смислове поєднання між частинами тексту програмної системи, представленими в термінах різних предметних областей. У даній роботі вихідною системою є текст програми, а предметом аналізу – елементи семантичної моделі, є смисловими одиницями, що формують модель програми.

Аналізована інформація представлена на рівні вихідного тексту, рівні дерева синтаксичного розбору (синтаксичної моделі програми) та на рівні семантичної моделі програми. Для представлення інформації про елементи моделі та взаємозв'язки між ними можуть використовуватись семантичні мережі. формальним поданням семантичної мережі є граф, у якому кожна вершина відповідає елементу безлічі сутностей чи понять, а дуги96 відповідають відносинами між цими сутностями.

Формальні мови, мови програмування та моделювання даних, що розробляються по суворо певним правилам із опорою на суворо організовану систему понять та визначень. Це дозволяє будувати семантичні моделі

вихідних текстів програм у термінах, визначених семантичними специфікаціями, що розробляються для кожного з використовуваних у програмний проект мов програмування. Такий набір термінів буде бути метамоделлю для екземпляра семантичної мережі.

Предметно-орієнтований аналіз – аналіз властивостей семантичної моделі, специфічні для способу її застосування. Предметноорієнтований аналіз тексту програми – аналіз властивостей тексту програми, специфічні для способу застосування синтаксичних конструкцій мови, якою написана програма. Такою мовою може бути мовою загального призначення, предметно-орієнтованою мовою (DomainSpecific Language, DSL) або мовою запитів, наприклад мовою структурованих запитів (Structured Query Language, SQL) та інші.

У процесі виконання предметно-орієнтованого аналізу моделі зазнають перетворень за заздалегідь визначеними правилами – трансляцій. Для формального опису трансляцій та самих моделей програм використовуються специфікації трансляції, синтаксичні та семантичні специфікації. Специфікації синтаксичних моделей, терміни яких необхідно прочитати вихідні тексти на першому кроці аналізу, представляються у вигляді граматик, що описують структуру тексту.

Правила цих граматик трактуються також як схеми абстрактних вузлів. синтаксичного графа, що матеріалізується на базі тієї ж семантичної мережі, як і семантичні моделі. Специфікації семантичних моделей описують схему елементів предметних областей, у термінах яких необхідно здійснити аналіз. Специфікації семантичних трансляцій задають відображення між семантичними моделями.

Аналізована інформація може бути представлена як у вигляді об'єктних моделей, і у реляційної формі, де кожному типу об'єкта реляційної моделі відповідатиме таблиця зі схемою, яка описує поля, які відповідають властивостям об'єкта. Те саме дійсно і для подання даних у вигляді графа, забезпечених аналогічною схемою з класифікацією вузлів графа, що

відповідає набору типів елементів об'єктної моделі. З цієї точки зору семантична трансляція даних з однієї предметної області в іншу відповідатиме набору реляційних запитів (відображень) із однієї схеми даних до іншої.

Використання при цьому гібридного або графового представлення даних для матеріалізації семантичних моделей дозволяє використовувати графові примітиви для опису нереляційних відносин, таких як шляхи Анотація синтаксичної графіки. При цьому для опису трансляцій графове подання може бути використане спільно з об'єктно-реляційним.

2.2 Формалізація розробленого методу

Специфікація семантичної моделі потрібна для опису семантики тексту програми з погляду деякої предметної області. Вона фактично є метаданими для семантичної моделі, формується з вихідного коду. У випадку формально специфікацію Семантичні моделі програми можна представити записом виду:

$$S = (T_s, P_s, V_s, C_s, D_s), \tag{2.1}$$

де:

$t_s \in T_s$ – ідентифікатори типів;

$p_s \in P_s$ – ідентифікатори властивостей;

V_s – схеми значень, що записуються у вигляді:

$$V_s = [v_s = (t,q) \tag{2.2}$$

На рисунку 2.1 зображено уявлення семантичної моделі M(5), метамоделлю якої є специфікація S (1).

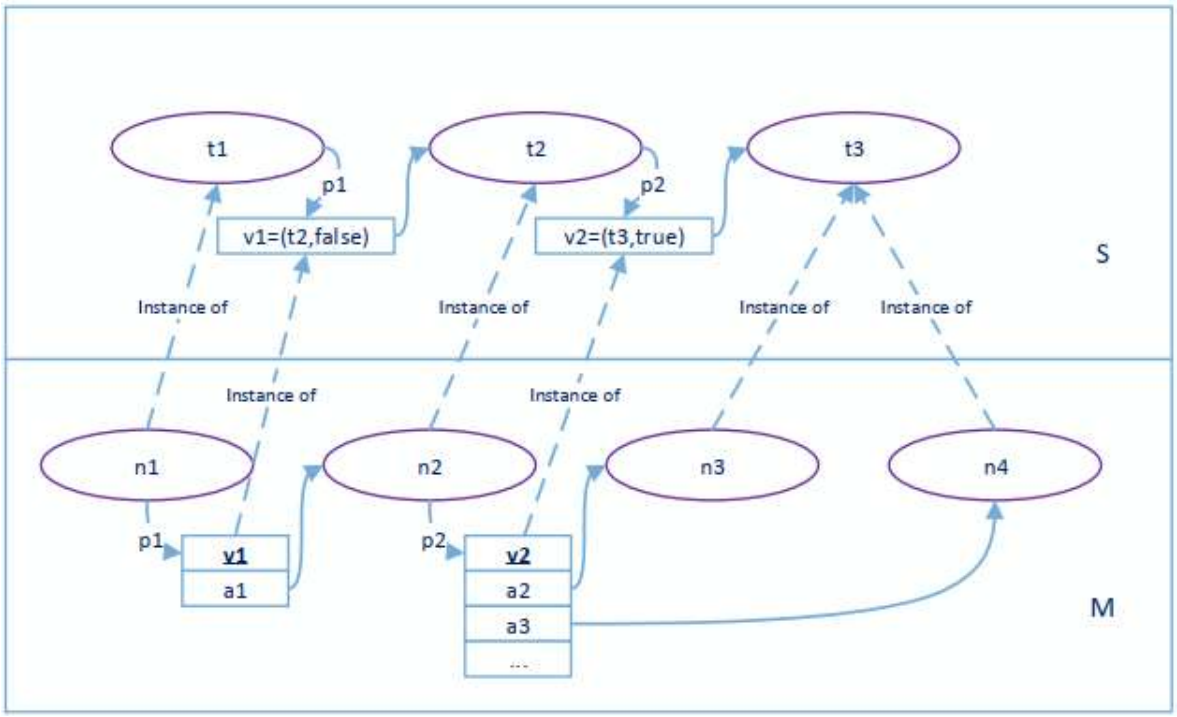

Рисунок 2.1 – Ілюстрація формального уявлення семантичної моделі

Сутності семантичної мережі n мають вигляд t пов'язані відносинами p один з одним. При цьому кожен екземпляр відношення заданий атрибутом a, несе значення у вигляді зв'язку з цільовою сутністю даного відношення. Кількість атрибутів для однієї властивості вузла відповідає кратності з схеми значення якості: один – якщо false, кілька – якщо true. Сутність n1 має вигляд t1 і властивість p1, як значення якого виступає атрибут a1, що представляє відношення до сутності n2. Сутність n2 виду t2, у свою черга має властивість p2, значеннями якого є атрибути a2 і a3, які представляють відносини з сутностями n3 і n4.

Специфікація синтаксичної моделі – це окремий випадок специфікації семантичної моделі (2.1), розширеної контекстно-вільної граматикою:

$$S = (T\text{s}, P\text{s}, V\text{s}, C\text{s}, D\text{s}, G\text{s}) \tag{2.3}$$

Тоді синтаксична модель – окремий випадок семантичної моделі зі специфікацією синтаксичної моделі як SX (5). Додатково до визначення

специфікації семантичної моделі, на специфікацію синтаксичної моделі накладається низка обмежень.

$$
\begin{aligned}
&T_s = N_{G_s} \\
&P_s = N_{G_s} \cup \Sigma_{G_s} \\
&C_s = \left\{c_s = t_s \to P_{t_s} : P_{t_s} \subseteq P_s \wedge t_s \longrightarrow w \in P_{G_s} \wedge P_{t_s} \subseteq w\right\} \\
&D_s = \{d_s = (t_s, p_s) \to v_s : \\
&\qquad t_s \longrightarrow w \in P_{G_s}, \\
&\qquad p_s \in w, \\
&\qquad v_s = (t, q) \in V_s \\
&\qquad \begin{cases} q = false, \text{если } \left|\sigma_{p_s}(w)\right| = 1 \\ q = true, \text{если } \left|\sigma_{p_s}(w)\right| > 1 \end{cases} \\
&\}.
\end{aligned}
$$

Рисунок 2.2 – Синтаксична модель

Типи семантичної моделі T_s ставляться у відповідність до нетерміналів граматики N_{GS} , властивості типів P_s ставляться у відповідність можливим дочірнім вузлам у дереві розбору з граматики, які можуть бути представлені нетерміналами N_{GS} або терміналами Σ_{GS}, кратність q значень цих властивостей відповідає кратності дочірніх вузлів даного виду $|\sigma_p s(w)|$, де $\sigma_p s(w)$ – кількість входжень елемента p_s в ланцюжок w правила $t_s \longrightarrow w$ граматики. Таким чином забезпечується відображення дерева розбору. для граматики G_s на семантичну модель відповідно до графової інтерпретацією як абстрактного семантичного графа.

Початкові відображення $l_{ts1 \to ts2}$ задають кореневі трансляції, з яких починається формування залежностей між елементами моделей. Вони можуть включати звернення до вільних відображень $p_{ts1 \to ts2}$ та контекстним відображенням $q_{ts1 \to ts2}$, що задає прямі залежності між одним вихідним

елементом та одним цільовим без додаткових аргументів з урахуванням додаткових аргументів q’ відповідно.

$$S_1 = (T_1, P_1, V_1, C_1, D_1), m_{s1} = (S_1, N_1, A_1, V_1, E_1, F_1),$$
$$S_2 = (T_2, P_2, V_2, C_2, D_2), m_{s2} = (S_2, N_2, A_2, V_2, E_2, F_2),$$
$$m'_{s2} = (S_2, N'_2, A'_2, V'_2, E'_2, F'_2),$$
$$t_{s1} \in T_1,\ t_{s2} \in T_2$$
$$N'_2 = \{n' \in N_2\},$$
$$A'_2 = \{a' \in A_2 : \exists(n' \to (t_{s1}, \{(\_, a')\})) \in E_2, n' \in N'_2\},$$
$$V'_2 = \{v' \in V_2 : \exists((n', \_) \to (a', v')) \in F_2, n' \in N'_2, a \in A'_2\},$$
$$E'_2 = \{(n' \to \_) \in E_2 : n' \in N'_2\},$$
$$F'_2 = \{((n', \_) \to \_) \in F_2 : n' \in N'_2\}.$$

Рисунок 2.3 – Часткова синтаксична модель

Rожне відображення зв'язує між собою структурні елементи моделей m_{s1} та m_{s2} за допомогою реляційних відображень. Початкові відображення не вимагають додаткових аргументів, крім вихідної моделі, і можуть виступати в як джерела інформації для ініціації алгоритму, що виконує семантичну трансляцію. Вільні відображення, що застосовуються до конкретним елементам вихідної моделі, та контекстні відображення, які додатково вимагають інформації з контексту, можуть використовуватися для виконання вторинних кроків алгоритму семантичної трансляції, ініційованих опосередковано під час виконання попередніх кроків.

В якості вузла аргументу та контексту при цьому будуть використовуватися дані, отримані на попередніх кроках, наприклад, при застосуванні первинних відображення. Крім безпосередньої імперативної інтерпретації, функції відображень можуть розглядатися як декларативні джерела інформації про залежність між елементами різних моделей.

Реляційне відображення – композиція операцій реляційної алгебри над множинами та операцій над окремими елементами або значеннями, описує перетворення між елементами даних. У рамках реляційного подання як елементи даних виступають записи таблиць. Кожна таблиця сприймається як безліч записів, кожна запис – як кортеж полів. Відповідно до розглянутої раніше реляційною інтерпретацією семантичної моделі до описуваних нею даним можуть бути застосовані реляційні операції, результатом чого буде формування нових семантичних моделей.

2.3 Опис методу

У формалізованому вигляді розроблений метод можна записати наступним чином.

Крок 1. Нехай безліч Ψ – тексти τ програмної системи, написані на мовами з синтаксичними специфікаціями S_τ:

$$(\tau_n, S_{\tau_n}) \in \Psi, 1 \leq n \leq |\Psi|,$$

де S_{τ_n} – синтаксична специфікація тексту програми τ_n, n – номер тексту τ програмної системи. Тоді результатом синтаксичного аналізу цих текстів буде безліч синтаксичних моделей Υ:

$$\Upsilon = \{\upsilon_{S_\tau} \forall (\tau, S_\tau) \in \Psi\}, |\Psi| = |\Upsilon|,$$

де безліч Ψ – тексти τ програмної системи, υ_{S_τ} – синтаксична модель за специфікацією S_τ для тексту програми τ.

Крок 2. Нехай Θ – предметні галузі, у межах яких виконується аналіз текстів програмної системи, задані набором специфікацій семантичних трансляцій:

$$\Theta = \{R_n = (S_{1_n}, S_{2_n}, \_, \_, \_)\}, 1 \le n \le |\Theta|,$$

де Rn – специфікація семантичної трансляції, S_{1_n} – специфікація вихідної семантичної моделі, S_{2_n} – специфікація цільової семантичної моделі.

Крок 3. Предметно-орієнтований аналіз на основі семантичних моделей для тексту τ дає набір семантичних моделей Ω_τ, що є набором всіх семантичних моделей, отриманих у результаті трансляцій $f(m, r)$, починаючи з синтаксичної моделі υ_{S_τ} прямо чи побічно для даної семантичної моделі m = (S, _, _, _, _, _) при наявності відповідної специфікації семантичної трансляції r = (S, _, _, _, _, _) $\in \Theta$ для специфікації семантичної моделі S:

$$\Omega_\tau = \bigcup_{\substack{m_0 = \upsilon_{S_\tau} \\ m_n = (S, \_, \_, \_, \_, \_) \\ \exists r_{n \to n+1} = (S, \_, \_, \_, \_, \_) \in \Theta}} \{m_n\}, m_{n+1} = f(m_n, r_{n \to n+1})$$

де m_{n} – семантична модель (2.5), специфікація семантичної трансляції $r_{n \to n+1}$, $f(m_n, r_{n \to n+1})$ – функція, що здійснює трансляцію:

$$f(m_{S_a}, r_{a \to b}) = \varsigma(S_b, \{m' = \iota(m_{S_a}) \forall \iota \in I,\ m_{S_a} = (S_a, \_, \_, \_, \_, \_)$$
$$r_{a \to b} = (S_a, S_b, Q, P, I)\}),$$

де $r_{a \to b}$ – специфікація семантичної трансляції з семантичної моделі m_{S_a} за специфікацією S_{a} у семантичну модель за специфікацією S_{b}, m' – часткова цільова модель, що отримується в результаті застосування трансляції $\iota \in I$ з специфікації $r_{a \to b}$ до моделі m_{S_a}, ς – функція, що поєднує набір семантичних моделей m'_Σ із загальною специфікацією S в одну загальну модель:

$$\varsigma = (S, m'_{\Sigma}) \to (S,$$
$$N_b = \varsigma'(m'_{\Sigma}, (S, N_{m'_{\Sigma}}, \_, \_, \_, \_) \to N_{m'_{\Sigma}}),$$
$$A_b = \varsigma'(m'_{\Sigma}, (S, \_, A_{m'_{\Sigma}}, \_, \_, \_) \to A_{m'_{\Sigma}}),$$
$$V_b = \varsigma'(m'_{\Sigma}, (S, \_, \_, V_{m'_{\Sigma}}, \_, \_) \to V_{m'_{\Sigma}}),$$
$$E_b = \varsigma'(m'_{\Sigma}, (S, \_, \_, \_, E_{m'_{\Sigma}}, \_) \to E_{m'_{\Sigma}}),$$
$$F_b = \varsigma'(m'_{\Sigma}, (S, \_, \_, \_, \_, F_{m'_{\Sigma}}) \to F_{m'_{\Sigma}})$$
$$),$$

де (S, N_b, A_b, V_b, E_b, F_b) – семантична модель (5), ς' – функція, що об'єднує результати застосування аргументу z до кожного з членів множини, заданої аргументом Y:

$$\varsigma' = (Y, z) \to \bigcup z(y) \forall y \in Y$$

M_Γ – всі семантичні моделі для всіх текстів Y (первинні синтаксичні та отримані з них трансляціями в сукупності):

$$M_\Gamma = \bigcup \Omega_\tau \forall \upsilon_{S_\tau} \in Y,$$

а Γ – підсумковий набір об'єднаних семантичних моделей:

$$\Gamma = \{\gamma_S = \varsigma(s, X) : \forall x = (s, \_, \_, \_, \_, \_) \in X \subseteq M_\Gamma\},$$

що складається з об'єднаних функцією ς семантичних моделей x (5) для кожної з доступних специфікацій s(2.1). Специфікації семантичних моделей можуть бути доповнені предикатами, заданими в будь-якій інтерпретації семантичних моделей. Побудова простору Γ не вимагає виконання всіх обмежень та предикатів семантичних та синтаксичних моделей, оскільки

вони описують формальні залежності між елементами даних та їх членами. за задоволеності цих обмежень та предикатів розробник ПЗ може судити про відповідність текстів програмної системи даним специфікаціям.

2.4 Можливості застосування

Після визначення формального базису методу, що розробляється можна розпочати визначення вимог щодо його реалізації з урахуванням особливостей програмних проектів. Вимоги до структури тексту програми, специфічні для програмного проекту у конкретній галузі за визначенням формулюються у термінах цієї предметної галузі.

Розглянемо приклад. Нехай дана програма мовою C#, яка використовує відображення даних із реляційної бази даних на класи мови C#. При використання бібліотек Linq2Database або EntityFramework можливість виконання таких відображень залежить від правильної розмітки класів спеціальними атрибутами, які керують логікою об'єктно-реляційного відображення. При цьому семантика мови C# не включає свою предметну мову область правила об'єктно-реляційних відображень, специфічних для тієї чи іншої реалізації таких відображень.

Тобто необхідний семантичний аналіз, який має бути виконаний для статичної перевірки коректності атрибутів об'єктно-реляційного відображення, виконаємо засобами аналізу коду загального призначення для програм мовою C#. Такий аналіз є предметно-орієнтованим і здійснимо або спеціалізованими інструментами, або бібліотекою, що виконує об'єктно-реляційне відображення під час виконання програми. У даному контексті мається на увазі, що предметною областю є синтаксис та семантика мови програмування, так як вони є самодостатньою системою понять Специфічну розмітку класів для об'єктно-реляційного відображення, що використовується при описі схеми бази даних, можна вважати предметно-орієнтованою мовою. Така мова залежить не тільки від бібліотек, що

здійснюють об'єктно-реляційне відображення, а також способу, яким конкретний програмний проект використовує ці бібліотеки. Наприклад, засоби розробки Linq2Database не дозволяють автоматично створювати окремі таблиці в існуючій базі даних, а також не мають підтримки міграцій.

Така функціональність може бути реалізована у різних програмних проектах по-різному. При цьому вона вимагає додаткової інформації, що описує об'єктно-реляційне відображення за допомогою класів та властивостей, що відображаються на поняття таблиць та записів за допомогою атрибутів мови C#.

У момент початку роботи з кодом програми, має бути завантажена реалізація аналізованого методу та сформовані необхідні семантичні моделі відповідно до принципів, описаних раніше. У даному контексті як предметні області для аналізу виступають синтаксис і семантика мови програмування, якою написаний текст програми та предметна область схеми бази даних, описаної з допомогою вихідної мови. Рисунок 2.4 ілюструє послідовність формування моделей програми.

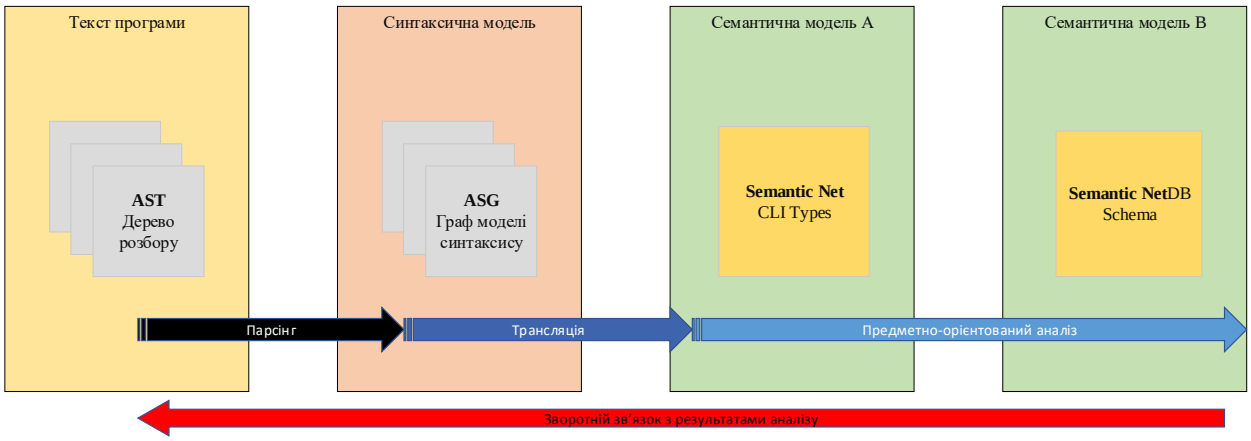

Рисунок 2.4 – Приклад ходу аналізу для моделі БД у програмі C#

На першому етапі виконується синтаксичний аналіз (парсинг) тексту на мовою C#, з урахуванням інформації внутрішніх структур текстових даних редактора. Наприклад, може використовуватися вже існуюче дерево розбору.

Формується синтаксична модель. На наступних етапах аналізу виконується семантична трансляція з абстрактного синтаксичного графа (ASG) синтаксичної моделі в семантичну модель мови C#, а потім у семантичну модель схеми бази даних. Зібрана під час аналізу інформація візуалізується. Сформовані моделі зберігають взаємозв'язки з текстовим редактором у складі середовища програмування та продовжують використовуватись під час редагування тексту.

Врахування залежностей між елементами даних, описаних відображеннями у складі специфікації семантичної трансляції, дозволяє оновлювати лише ті частини моделей, дані яких були порушені у процесі редагування раніше проаналізованого тексту. Так забезпечується гранулярність та інкрементальність аналізу. На етапі синтаксичного аналізу для цього методу аналізу текстів програм немає значення, якого класу аналізаторів належить синтаксичний аналізатор, який використовується для аналізу вихідних текстів.

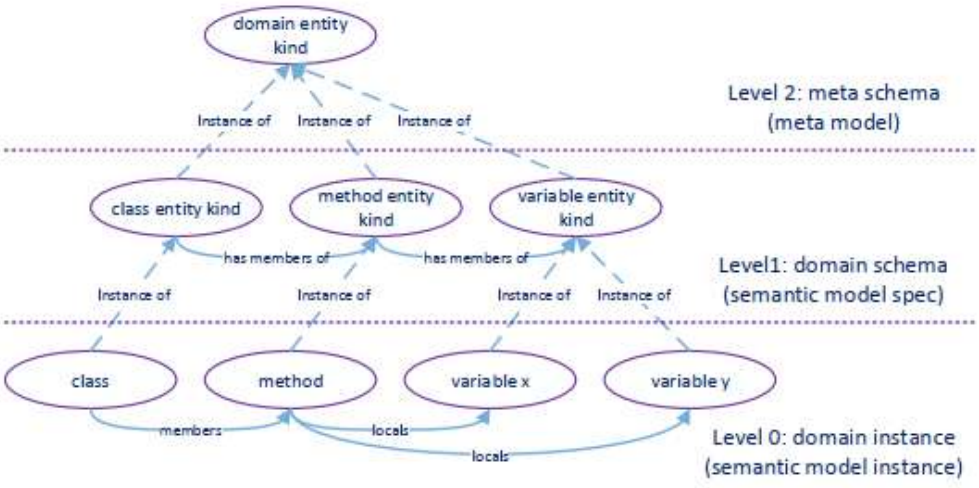

Рисунок 2.5 – Приклад семантичної метамоделі (level 1) та моделі (level 0) програмного коду для мови C#

Важливим є відбиток дерева аналізу на семантичну мережу так, що її фрагмент буде абстрактним синтаксичним графом. Вузли цього графа надалі

трактуються як сутність предметної області синтаксичної моделі вихідного тексту. На наступних етапах робота відбувається з інформацією, представленої за допомогою семантичної мережі. З погляду реалізації, вона містить сукупність моделей. Аналізована інформація є семантичною моделлю – екземпляром предметної області, сформульованою в термінах, заданих специфікацією семантичної моделі – схемою предметної галузі. Приклад такої семантичної мережі представлений на рисунку 2.5.

У процесі виконання семантичних трансляцій між різними семантичними моделями та при врахуванні залежностей між їх елементами, використовуються реляційні відображення. Для семантичної мережі вони є запитами до графа, що представляє цю мережу. В ході експерименту, що проводиться на прикладі подання інформації про програмної системи за допомогою графової бази даних стосовно статичного аналізу веб-додатків, були проаналізовані деякі методи реалізації запитів над графами.

Дослідження проводилося на прикладі представлення інформації про програмну систему за допомогою графової бази даних стосовно статичного аналізу веб-застосунків. Інтерпретація запитів до граф при роботі з графовими уявленнями може бути сформульована як завдання пошуку шляхів.

2.5 Вимоги до програмного рішення

Сформулюємо вимоги до рішення, що дозволяє виконувати предметно-орієнтований аналіз вихідних текстів програм у процесі їх редагування на основі семантичних моделей програм та предметних областей, які стосуються рамках окремих проектів.

Рішення має підтримувати аналіз текстів програм на різних мовами програмування одночасно.

Рішення має підтримувати аналіз повного обсягу тексту програми або програмної системи на початку своєї роботи.

Рішення має підтримувати гранулярність та ітеративність аналізу для якнайшвидшого формування актуального результату аналізу.

Рішення не повинне вносити додаткових накладних витрат на перетворення даних між різними формами уявлення.

Рішення не повинно вимагати від розробника, що його використовує. вивчення додаткових предметних областей, не пов'язаних з розв'язуваними ним завданнями щодо діяльності у програмному проекті.

Для досягнення поставленої в роботі мети з урахуванням перерахованих вище вимог було розроблено метод предметно-орієнтованого аналізу вихідних текстів програм. Метод заснований на використанні семантичних моделей, отриманих в результаті аналізу текстів програм, що розробляються з використанням кількох мов програмування. Основними операціями, що виконуються в процесі аналізу над семантичними моделями програм, є операції семантичної трансляції – відображення однієї семантичної моделі на іншу семантичну модель.

Семантична трансляція задається відображенням елементів даних, що надають інформацію в термінах вихідної предметної області однієї мови програмування, елементи даних в термінах цільової предметної області іншої мови програмування або предметно-орієнтованої мови.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Для перевірки працездатності запропонованого методу та способів описи різних специфікацій були створені необхідні алгоритми та структури даних Розглянемо деякі аспекти програмної реалізації пропонованих методів.

3.1 Розробка структур даних

Застосування методу предметно-орієнтованого аналізу вихідних текстів програм на основі семантичних моделей вимагає автоматизованого виконання семантичних трансляцій відповідно до принципів, що обговорюються у другому та третьому розділах. У свою чергу, щоб розпочати семантичні трансляції, необхідно спочатку прочитати задані специфікації синтаксичних та семантичних моделей. Для цього були розроблені та реалізовані об'єктні моделі, призначені для розміщення інформації, що витягується зі специфікацій у формі текстів розробленою мовою опису семантичних моделей.

На рисунку 3.1 представлена діаграма класів об'єктної моделі специфікацій синтаксичних моделей. Для роботи синтаксичного аналізатора необхідна синтаксична специфікація. У момент початку роботи прочитання синтаксичних специфікацій неможливо, тому що для цього має бути завантажено їх синтаксична модель. Тому ініціалізація аналізатора потребує іншого джерела специфікацій. Для цього було реалізовано підтримку подання синтаксичних моделей у формі XML-документів. Такий XML-документ включено до складу програмної бібліотеки як ресурс. При ініціалізації бібліотеки під час першої спроби її використання відбувається десеріалізація цього XML-документа, з якого відновлюється об'єктна модель граматики. Надалі вона138 використовується для ініціалізації синтаксичного аналізатора та читання специфікацій синтаксичних моделей

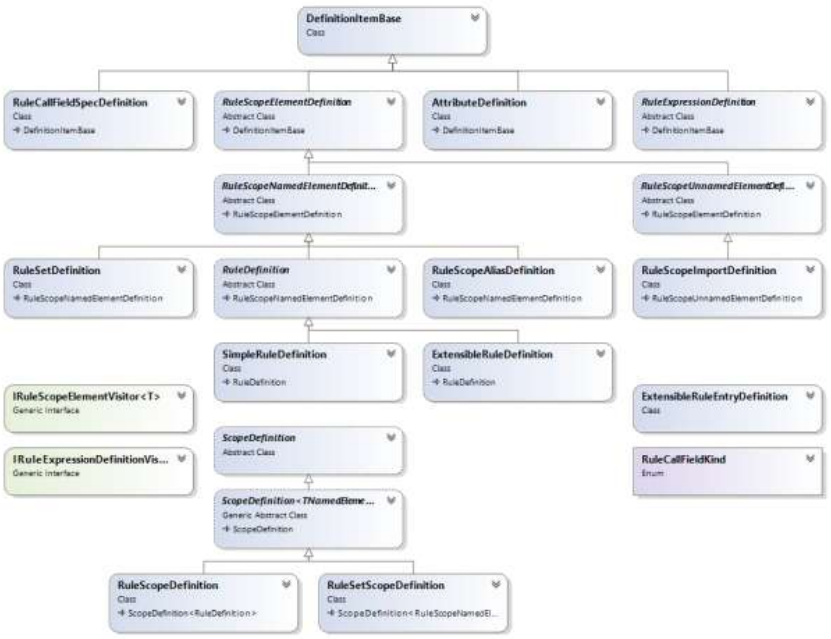

Рисунок 3.1 – Діаграма класів об'єктної моделі специфікацій синтаксичних моделей

Семантична мережа ініціалізується з використанням наборів схем, що задаються за допомогою специфікацій семантичних моделей поетапно. на На основі специфікацій будується об'єктна модель схеми предметної області. Вона містить схеми для кожної сутності, що входить до складу предметної області. Усі необхідні схеми вносяться до структур даних семантичної мережі, що відповідають за рівень метамоделі, де на їх основі формується словник схем предметних галузей. Далі завантажуються специфікації необхідних семантичних трансляцій. Тепер семантична мережа готова до використання для предметно-орієнтованого аналізу у вигляді ітеративного перетворення семантичних моделей.

Розглянемо алгоритм ітеративних перетворень.

На основі результатів синтаксичного аналізу тексту або керуючих команд текстового редактора формується набір мутацій, що описують вироблені над станом семантичної мережі зміни. На цьому кроці за допомогою мутацій формується абстрактний синтаксичний граф для кожного вхідного файлу терміни відповідної йому синтаксичної специфікації.

Набір мутацій застосовується до семантичної мережі, ітеративно застосовуючи кожну мутацію та накопичуючи інформацію про фактично внесених змін. Стан фрагмента семантичної мережі при це змінюється.

В результаті формується набір класифікованих змін, що відображають фактично внесені до стану семантичної мережі зміни. Усі мутації, що потрапили в цей набір, належать вихідній предметної галузі.

З відомих запитів семантичних трансляцій вибираються ті, вихідна предметна область яких відповідає мутаціям в поточний набір змін.

З даної множини запитів за допомогою перетину множини зачіпаних ними сутностей-джерел і безлічі сутностей наявного набору змін, порушених мутаціями, виділяється підмножина запитів, результати виконання яких у нинішньому стан семантичної мережі повинен відрізнятися від попереднього.

Виділені запити застосовуються на підставі їх результатів:

- накопичується набір мутацій для внесення змін до фрагмента семантичної мережі, що відповідає цільовій семантичній моделі цього етапу трансляції;

- формується підмножина запитів, параметри або вихідні дані яких були порушені застосованими на цьому кроці перетвореннями семантичної моделі.

Для нового підмножини запитів кроки п'ять і шість повторюються до до тих пір, поки підмножина відповідних запитів у поточній трансляція не порожня.

Кроки з другого до сьомого повторюються для накопиченого набору мутацій.

В результаті застосування даного алгоритму здійснюється ряд перетворень семантичних моделей. У ході першої ітерації цього алгоритму виконується перенесення даних від вихідного абстрактного синтаксичного графа в термінах синтаксичної моделі до семантичної моделі, яка є предметним аналізом.

У ході наступних ітерацій виконується перенесення даних між

семантичними моделями, для послідовності яких є специфікації семантичних трансляцій. Оскільки перенесення інформації завжди здійснюється у напрямку семантичної трансляції від попереднього екземпляра семантичної моделі до наступного, зациклювання на рівні кроків з другого по сьомий виключено. Зациклювання на рівні кроків п'ять і шість виключено у силу логіки обробки здійснює вибірку даних запиту. Інтеграція даного алгоритму у процес редагування тексту залежить від способу представлення тексту текстовому редакторі.

Як структура даних, що відповідає за внутрішнє подання тексту, була реалізована представлена на рисунку 3.2 гібридна структура даних, що є комбінацією кількох різних структур.

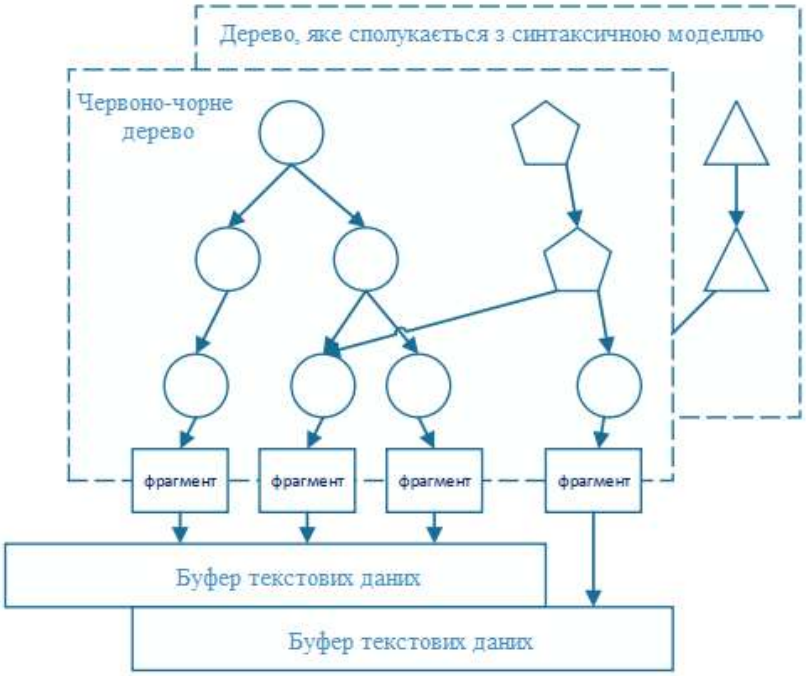

Рисунок 3.2 – Структура даних текстового документа усередині редактора

Для розміщення символьних даних, з яких складено редагований текст, використані виділені буфери аналогічно структура даних Piece-Table. У міру редагування та залучення нових фрагментів буфера в момент розміщення нового фрагмента, зліва та праворуч від нього резервується вільне місце, яке

далі використовується при вставці символів у суміжні з цим фрагментом позиції. Це забезпечує складність вставки символу в текст порядку O(для змін тексту, які впливають з його синтаксичну структуру. Дроблення тексту на фрагменти виконується з урахуванням лексем заданої синтаксичної моделі так, щоб межі фрагментів відповідали межам лексем. Це забезпечує можливість прив'язки стильової інформації до сайтів фрагментів тексту.

Так усувається потреба перебудови цієї інформації під час візуалізації тексту. Фрагменти тексту об'єднані в єдину структуру даних одночасно іммутабельним червоно-чорним деревом та деревом на основі синтаксичної моделі.

Ключем для сортування у червоно-чорному дереві виступає позиція у формі рядок-колонка. Це дозволяє зберегти швидку навігацію текстом великого об'єму. Імутабельні структури даних мають недолік, що полягає у необхідності частого клонування їх частин під час внесення змін. Це компенсується за рахунок використання «лінивого» клонування шляху. Положення курсора відповідає матеріалізований паралельно основному дереву фрагмент шляху від коня дерева до вузла, що містить фрагмент, де зараз розташований курсор.

При внесенні зміни у текст, якщо був запит на знімок поточної версії тексту, породжується нове стан фрагмента, включаючи лист дерева, причому весь шлях до кореня не матеріалізується. Якщо запиту знімка не було, змінюється стан поточного фрагмента і, можливо, листа дерева (якщо необхідна зміна його структури). Клонування частини дерева при цьому не потрібне. Матеріалізація шляху, піддерево якого було змінено, виконується по мірою переміщення курсору межі відредагованого фрагмента, наприклад, у момент переходу курсору від фрагмента поточної гілки до фрагменту тексту, поданому листом іншої гілки, залежно від поточні деревини структури.

Цей же підхід застосовується до дерева, відповідає синтаксичної структури тексту.

3.2 Інтеграція рішення в середовища розробки

Було розроблено два основні варіанти програмної архітектури для інтеграції розроблених алгоритмів у середовищі розробки. Як вбудовані модуль та як зовнішній сервіс. При інтеграції як вбудований модуль реалізація рішення повністю завантажується в процес середовища розробки або текстового редактора та функціонує у ньому як in-proc. Такий підхід ефективніший з точки зору продуктивності, оскільки не потребує перетворення даних для передачі між процесом, що імпортує функціональність, та процесом, що експортує цю функціональність.

Недоліком цього варіанта програмної архітектури є необхідність використання незалежних екземплярів синтаксичного аналізатора як у процесі з редактором, і у процесі з семантичним аналізатором. Даний варіант архітектури представлений рисунку 3.2. Основна відмінність від існуючих рішень полягає у впливі аналізатора на внутрішнє уявлення тексту в текстовому редакторі. Така інтеграція пов'язана з двома особливостями.

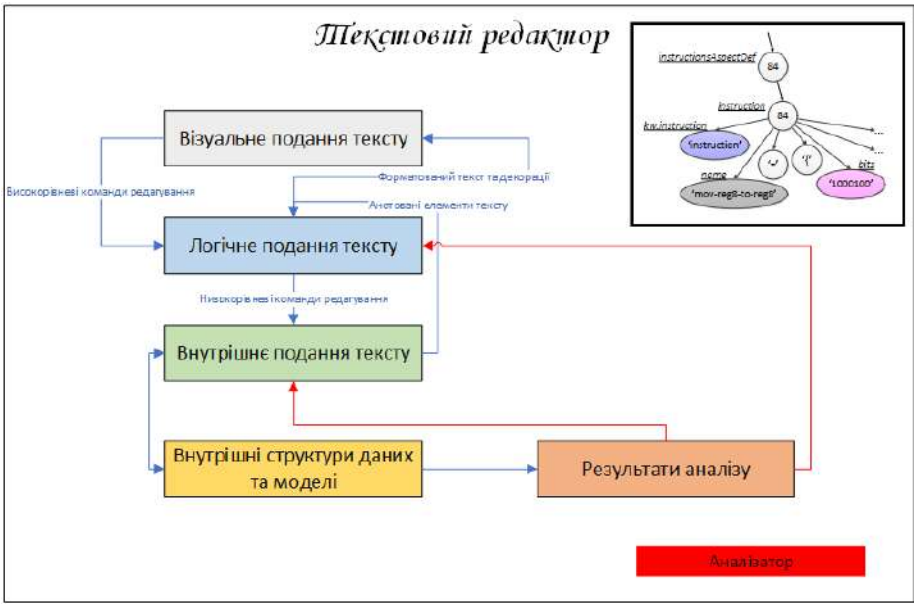

Рисунок 3.3 – Запропонована архітектура текстового редактору

Перша особливість полягає в модифікації внутрішнього подання тексту у складі текстового редактора. Згадані у другому розділі структури даних, які використовуються для представлення тексту, мають краща асимптотика мутацій, ніж текстові рядки, представлені масивами символів. Це досягається за рахунок розбиття тексту на фрагменти та організацію цих фрагментів у структуру, операції редагування якої можна здійснити швидше. З погляду процесу синтаксичного аналізу, довільне розбиття тексту на фрагменти є причиною накладних витрат. Воно призводить до необхідності використання програмного інтерфейсу, текстового редактора, що відображає внутрішнє подання, на лінійний текст, який аналізатор має розбити на лексеми, виділити синтаксичну структуру.

При цьому та сама структура, що формується у процесі аналізу, відображається назад у структури даних, оскільки підсвічування синтаксису, блоки коду та аналогічна функціональність спираються на інформацію про синтаксичну та семантичну структуру тексту. З цієї причини доцільно використовувати аналізатор для розбиття тексту на фрагменти відповідно до правил, що визначаються лексичною та синтаксичною моделями редагованого тексту. Це реалізовано під час використання інформації про зміни тексту аналізатором виявлення того, як має змінитися внутрішнє уявлення, щоб залишитися відповідним логічній структурі тексту.

Друга особливість полягає в тому, що результати семантичного аналізу асоціюються з тими ж синтаксичними одиницями, що були виділені в тексті на перших етапах аналізу. Доцільно асоціювати їх не з додатковими структурами даних, а з атрибутами, прив'язаними до елементів вже існуючих уявлень тексту. Так, інформація про форматування та підсвічування тексту може асоціюватися з внутрішнім поданням тексту. В результаті зникає необхідність у повторний запит цієї інформації або її повторне обчислення в процесі виведення тексту на екран. Те саме стосується й інших операцій, пов'язаним із вилученням даних із результатів семантичного аналізу тексту одночасно з урахуванням позицій у тексті. Наприклад, до операцій навігації

за кодом програми. При інтеграції як зовнішній сервіс, середовище розробки або текстовий редактор взаємодіє із семантичним аналізатором за допомогою механізмів міжпроцесної взаємодії (через пам'ять, мережеве з'єднання або канал). Приклад програмної архітектури представлено рисунку 3.4.

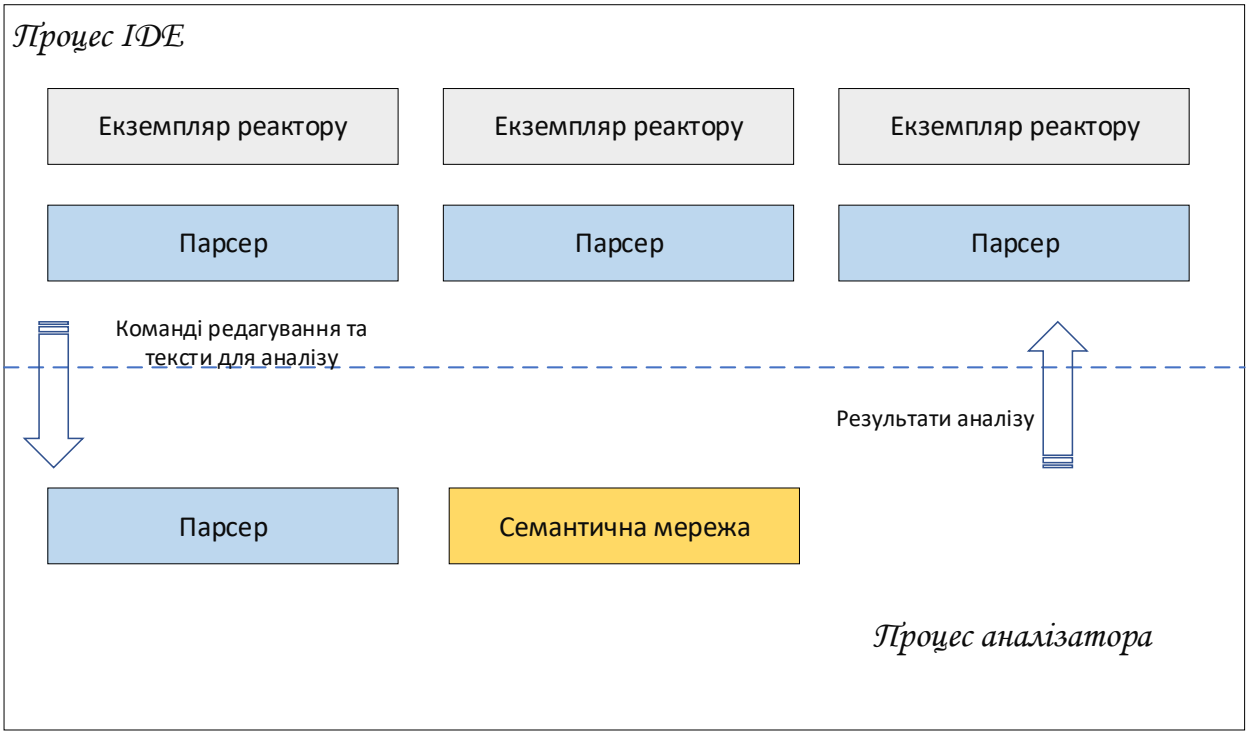

Рисунок 3.4 – Ілюстрація програмної архітектури для випадку інтеграції як зовнішнього сервісу з текстовим редактором на основі пропонованих структур даних

Головною перевагою є те, що така програмна архітектура дозволяє використовувати один екземпляр семантичного аналізатора з кількома екземплярами середовища розробки одночасно. Це ефективніше з погляду споживаної пам'яті, оскільки відсутня необхідність дублювання уявлень моделей програм. Усі зацікавлені в їх аналізі програми (наприклад, екземпляри середовища можуть використовувати загальну семантичну мережу. Крім цього, такий підхід дозволяє виключити повторний аналіз

тексту незміненим програми у разі, коли процес середовища розробки був аварійно завершено. Оцінка принципової можливості застосування пропонованого методу залежить від обсягу аналізованих даних. Варіант із аналізатором в одному процесі простіше і не обтяжений накладними витратами від міжпроцесної взаємодії, тому для створення експериментальної аналізатора було обрано саме він.

3.3 Програмна реалізація

Для розробки експериментального засобу предметно-орієнтованого аналізу вихідних текстів програм було обрано мову програмування C# та IDE Visual Studio 2017 з тієї причини, що вони входять до списку найбільш широко використовуваних відповідно до даними Google Trends мов програмування та інтегрованих середовищ розробки.

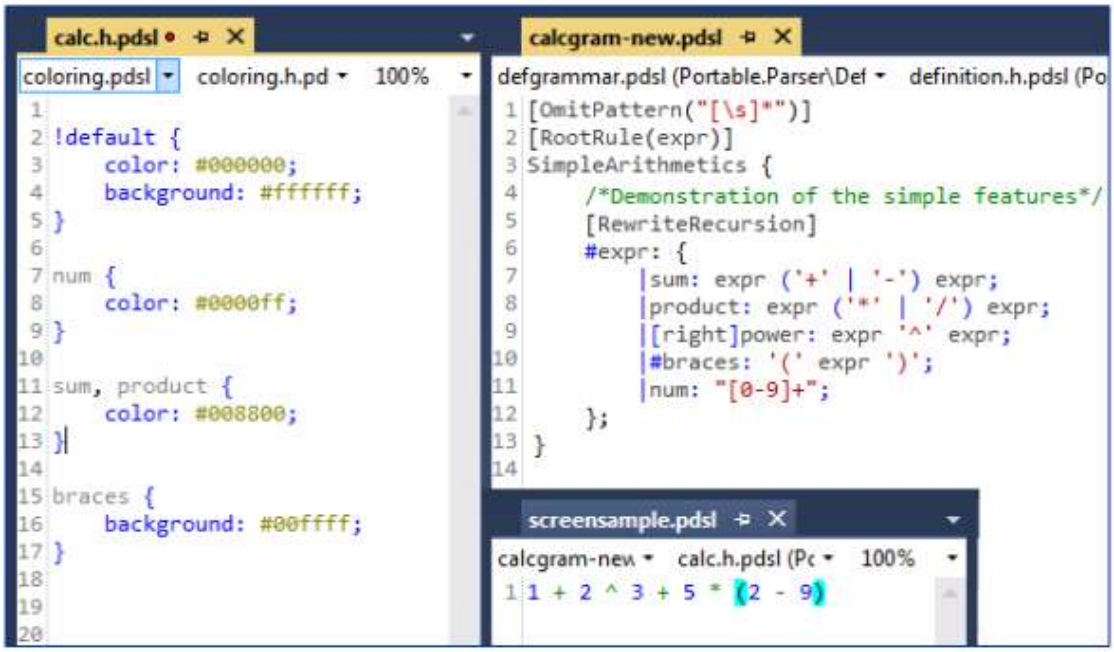

Рисунок 3.5 – Створений текстовий редактор у середовищі Visual Studio 2017

На базі представлених методів, алгоритмів та структур даних був створено конфігурований текстовий редактор (рисунок 3.5) з підтримкою

аналізу вихідних текстів програм у процесі їх редагування. Даний редактор був інтегрований у середу IDE Visual Studio 2017 за допомогою створення розширення (рисунок 3.6).

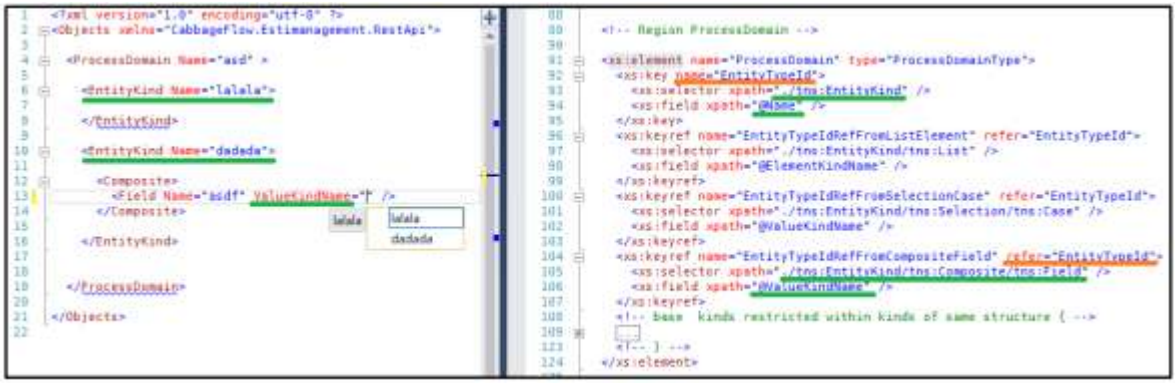

Рисунок 3.6 – Інтеграція окремої реалізації предметноорієнтованого аналізу на базі семантичних моделей із вбудованим редактором середовища Visual Studio 2017

У ході апробації було виконано порівняння середнього часу між зміною тексту та відображенням результатів аналізу на комп'ютері оснащений Xeon E5-1620 3.6GHz 16GB RAM для наявного та запропонованого рішень у двох сценаріях використання.

Перевірка відповідності типів XML-схеми класам мови C#, використовується для автоматичної серіалізації відповідно до даною схемою.

Перевірка коректності ключів XML-документу щодо keyref-обмежень відповідної XML-схеми.

Нижче наведено порядок та умови проведення експериментів:

- у середовищі розробки створюється проект із експериментальними даними;

- включається відеозапис з екрану у форматі 60 кадрів/с та запускається скрипт, що емулює сценарій внесення зміни до вихідний текст програми 1000 разів;

- під час експерименту на екрані у певних місцях змінюється

підсвічування синтаксису, з'являються і зникають повідомлення про помилки;

- після завершення експерименту відеозапис покадровий аналізується та обчислюється інтервал часу між введенням, запускає аналіз, і візуальним зворотним зв'язком, що формується в результаті аналізу та відображається на екрані.

Для першого сценарію проведення експерименту користувачем виконувались такі дії:

- додавання до коду опису класу, відповідного типу XML-схеми;
- складання проекту.

Для другого сценарію проведення експерименту користувачем виконувалися зміни в XML-документі значення для атрибуту, схема якого має keyref-обмеження. В обох сценаріях оцінювався час між початком введення та появою повідомлення про помилку у тексті програми.

На рисунку 3.7 показано середній час між зміною тексту та відображенням результатів аналізу. Все це робить доцільним зменшення часу між введенням та отриманням результатів аналізу.

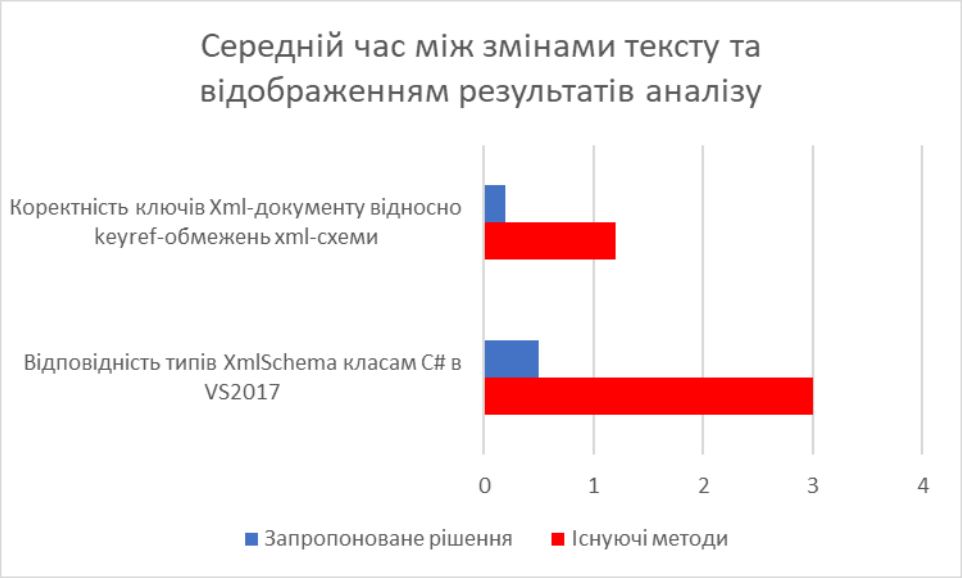

Рисунок 3.7 – Середній час між зміною тексту та відображенням результатів аналізу

Таким чином, результати порівнянь та апробації показали, що для розглянутих випадків:

- час перевірки узгодженості коду програм у мультимовних програмні проекти зменшено;
- вирішено проблему неактуального зворотного зв'язку від аналізу;
- розширений спектр доступного розробнику аналізу семантики вихідних текстів програм. У цілому нині, апробація показала працездатність запропонованих методів.

ВИСНОВКИ

Розроблено метод предметно-орієнтованого аналізу на основі ітеративних перетворень семантичних моделей програм, що дозволяє здійснювати семантичний аналіз текстів програм, що розробляються із застосуванням кількох мов програмування. Розроблено програмну архітектуру та структури даних для подання тексту та семантичних, що динамічно змінюються моделей програм, що дозволяє інтегрувати реалізацію розробленого методу предметно-орієнтованого аналізу в різні середовища розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бочарова О.О., Дяченко В.О., Коваленко А.А. Аналіз текстів програм на основі семантичних моделей // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей дванадцятої міжнародної науково-технічної конференції 27-28 квітня 2022 р., т.1. Баку-Харків-Жиліна. 2022 р. С.69.

2. Хомский Н., Милле Д. Введение в формальный анализ естественных языков /Пер. с англ. ЕВ Падучевой //М.: Едиториал УРСС. – 2003.

3. Adrion W. R., Branstad M. A., Cherniavsky J. C. Validation, verification, and testing of computer software //ACM Computing Surveys (CSUR). – 1982. –Т. 14. – №. 2. – С. 159-192.

4. AlbahariJ.,AlbahariB.C# 7.0in a nutshell: The definitive reference. "O'Reilly Media, Inc.", 2017.

5. Barrasa J., Corcho Ó., Gómez-Pérez A. R2O, an extensible and semantically based database-to-ontology mapping language. – SWDB, 2004.161

6. .Barzdins J. et al. Domain specific languages for business process management: a case study //Proceedings of DSM. – 2009. – Т. 9. – С. 34-40.

7. .Batini C., Lenzerini M. A methodology for data schema integration in the entity relationship model //IEEE transactions on software engineering. – 1984. – №. 6. – С. 650-664.

8. Becchi M. Data structures, algorithms and architectures for efficient regular expression evaluation. – Washington University in St. Louis, 2009.

9. Beelders T., du Plessis J. P. The influence of syntax highlighting on scanning and reading behaviour for source code //Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists. – 2016. – С. 1-10.

10. Binkley D. Source code analysis: A road map //Future of Software

Engineering (FOSE'07). – IEEE, 2007. – С. 104-119.

11. .Biron P. V. et al. XML schema part 2: Datatypes. – 2004.

12. .Breslav A. DSL development based on target meta-models. Using AST transformations for automating semantic analysis in a textual DSL framework //arXiv preprint arXiv:0801.1219. – 2008.

13. .Chebotko A. et al. Semantics preserving SPARQL-to-SQL query translation for optional graph patterns //Wayne State University, Tech. Rep. TR-DB- 052006-CLJF. – 2006.

14. .Chen H., Luo Y. Object Automatic Serialization and De-serialization in C++ //proceedings of 2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010 no. 2). – 2012.

15. Chisnall D. The challenge of cross-language interoperability /Communications of the ACM. – 2013. – Т. 56. – №. 12. – С. 50-56.

16. Chomsky N. Syntactic structures. – Walter de Gruyter, 2002.

17. Chowdhury K. Mastering Visual Studio 2017. – Packt Publishing Ltd, 2017.162

18. Codd E. F. et al. Relational completeness of data base sublanguages. – IBM Corporation, 1972. – С. 65-98.

19. Codd E. F. Extending the database relational model to capture more meaning //ACM Transactions on Database Systems (TODS). – 1979. – Т. 4. – №. 4. – С. 397-434.