Технічні науки

УДК 004.43

Булах Богдан Вікторович

кандидат технічних наук,

доцент кафедри системного проектування

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Булах Богдан Викторович

кандидат технических наук,

доцент кафедры системного проектирования

Национальный технический университет Украины

"Киевский политехнический институт имени Игоря Сикорского"

Bulakh Bogdan

Candidate of Technical Sciences,

Associate Professor of the System Design Department

National Technical University of Ukraine

"Igor Sikorsky Kyiv Polytechnic Institute"

ЗАСТОСУВАННЯ СЕМАНТИЧНИХ ТЕХНОЛОГІЙ ДЛЯ АНАЛІЗУ ТА РЕФАКТОРИНГУ ПРОГРАМНОГО КОДУ
ПРИМЕНЕНИЕ СЕМАНТИЧЕСКИХ ТЕХНОЛОГИЙ ДЛЯ АНАЛИЗА И РЕФАКТОРИНГА ПРОГРАММНОГО КОДА
APPLICATION OF SEMANTIC TECHNOLOGIES FOR SOURCE CODE ANALYSIS AND REFACTORING

Анотація. *Досліджено проблему створення інструментарію для поглибленого аналізу програмного коду та його рефакторингу. Запропоновано спосіб використання семантичних технологій для реалізації пошуку складних шаблонів в коді програм.*

Ключові слова: *семантичні технології, онтологія, патерн.*

Аннотация. *Исследована проблема создания инструментария для углубленного анализа программного кода и его рефакторинга. Предложен способ использования семантических технологий для реализации поиска сложных шаблонов в коде программ.*

Ключевые слова: *семантические технологии, онтология, патерн.*

Summary. *The problem of development of the deep source code analysis and refactoring tools was investigated. The way of application of the semantic technologies to implement the complex source code patterns search was proposed.*

Key words: *semantic technologies, ontology, pattern.*

Постановка задачі. Задача аналізу та рефакторингу програмного коду постійно постає перед розробниками програмного забезпечення. Аналіз програмного коду є необхідною умовою для контролю якості програмного продукту, в той час як рефакторинг є, по суті, оптимізацією коду для підвищення якості продукту. Нині більшість інтегрованих середовищ розробки пропонують певний набір інструментів для аналізу коду та здійснення його рефакторингу. Серед них як прості, не прив'язані до мови програмування засоби пошуку та заміни фрагментів у коді (за допомогою регулярних виразів), так і засоби, що використовують особливості конкретної мови програмування (наприклад, заміна імен класів чи методів в усіх файлах проекту в об'єктно-орієнтованих мовах). Однак цих базових інструментів часто недостатньо для нетривіальних операцій з виявлення складних конструкцій коду (шаблонів) та їх модифікації. Причина в тому, що пошук з використанням регулярних виразів не враховує семантики мови (результати пошуку потребують уточнення вручну). Крім того, цими простими засобами неможливо врахувати знання розробників програмного продукту про призначення та

правила коректного використання окремих фрагментів в коді, для розуміння яких недостатньо семантики самої мови програмування (найпростіший приклад - коментарі. Смисл, закладений у них для людей-програмістів, неможливо обробити машинно).

Проблема машинного "розуміння смислу" коду може бути вирішена аналогічно до ширшої проблеми машинного "розуміння смислу" інформації, доступної у мережі Інтернет (проблема побудови "семантичного веб"). Тобто, для вирішення цієї проблеми доцільно скористатися технологіями семантичного веб з деякими уточненнями. Постановка задачі: необхідно описати знання про код програмного продукту таким чином, щоб уможливити автоматичний пошук довільних фрагментів коду задаючи в якості вхідних даних для пошуку певні факти, поняття, зв'язки між поняттями, логічні правила. Опис семантики коду для його аналізу сам по собі не є новим підходом, й існує ряд публікацій на цю тематику [1; 2; 3]. Метою даної публікації є пропозиція інфраструктури для аналізу та рефакторингу коду, що може бути реалізована з використанням останніх розробок в області семантичних технологій. Причому ця інфраструктура повинна допомагати в пошуку специфічних рішень у коді (загальноприйнятих або прийнятих на конкретному проектів анти-патернів) незалежно від кількості файлів у проекті, що містять фрагменти таких рішень (іншими словами, шукані анти-патерни можуть виходити за рамки файлу). На даний момент стандартних (де-факто) засобів вирішення цієї задачі сильно бракує.

Стек семантичних технологій

Технології семантичного веб все ще не перебувають на етапі зрілості та не досягли широкого впровадження, хоча існує чимало стандартів та інструментів для створення семантичних описів та їх подальшої обробки. Нині можна окреслити таке коло найбільш типових засобів для реалізації

семантичних додатків: опис онтологій на мовах родини OWL, зберігання даних у вигляді триплетів "суб'єкт - предикат - об'єкт" у форматі RDF, запити до бази знань (сховища триплетів) за допомогою мови SPARQL, використання логічних правил на мові SWRL для доповнення онтологій, використання OWL/SWRL-різонерів (програм логічного виведення фактів з онтологій) для пошуку усіх можливих прихованих взаємозв'язків для усіх сутностей в онтології.

Для вирішення проблеми пошуку фрагментів коду за їх абстрактним описом через певні поняття, зв'язки слід, відповідно, побудувати систему понять та їх зв'язків, відому як онтологію, для предметної області конкретної мови програмування. Тобто для об'єктно-орієнтованих мов слід визначити поняття класу, підкласу, об'єкту, області видимості, оператору та складеного оператору, інтерфейсу, методу, поля, наслідування та багато іншого для того, щоб мати змогу описати код та шукати у ньому фрагменти, описані на високому рівні абстракції. До того ж необхідно описати проектні рішення, прийняти на даному конкретному проекті (стандарти де-факто в рамках проекту).

Пошук патернів та антипатернів у коді

Патерни проектування є умовно-кращими узагальненими рішеннями типових задач на етапі проектування архітектури програмного продукту. Наприклад, в об'єктно-орієнтованих мовах визнають набір т.зв. GoF-патернів [4], що можуть бути успішно застосовані на етапі проектування ієрархії класів для вирішення задач коректного створення та знищення об'єктів, їх взаємодії, зменшення взаємозв'язків та спрощення розширюваності ієрархії класів тощо. Так, одним з найпростіших патернів є "Одинак": клас, для якого гарантовано не буде створено більше одного об'єкта. Логічно це можна описати таким "семантчним псевдокодом" [3]:

Тип "Одинак" є об'єднання:

- тип "Клас"

- обмеження: властивість "має конструктор" повинна мати хоч одне значення типу "закритий конструктор":

- обмеження: властивість "має конструктор" повинна мати лише значення типу "закритий конструктор".

Тобто Одинак - це розширення Класу за рахунок додаткових обмежень (лише закриті конструктори - хоча б один). Слід додати, що вносячи додаткові правила у базу знань, такі як "відсутність відкритого конструктора означає неможливість створення екземплярів класу зовні методів класу", "статичний метод може викликатися без створення екземплярів класу", "новий екземпляр створюється у статичному методі тоді і тільки тоді, коли екземплярів не було створено раніше", можна описати патерн Одинак детальніше та на вищому рівні абстрагування.

Антипатерни є протилежністю до патернів в тому сенсі, що вони є прикладом не найкращих, а помилкових практик при проектуванні програмного забезпечення. Серед таких антипатернів: переобтяжені об'єкти (англ. God object), змішані інтерфейси, "спагеті-код" тощо [2]. Цікаво, що деякі дослідники та теоретики технологій розробки програмного забезпечення відносять Одинак до антипатернів. Це є добрим прикладом відносності в оцінках певних типових рішень, але також дає змогу проілюструвати той факт, що в межах проекту можуть існувати власні кращі або гірші практики. Припустимо, що всередині певного проекту вирішили вважати неприпустимим використання Одинаків. Тоді можна скористатися описом патерну одинак для пошуку таких класів в проекті та їх подальшого усунення програмістами.

Структура семантичного інструментарію аналізу коду

Загальна структура для аналізу коду пропонується такою (рис.1). Код проходить етап синтаксичного розбору, результатом якого є абстрактне дерево синтаксису (AST). Це дерево дозволяє коректно описати основні елементи мови програмування, вжиті в програмному коді, та їх взаємозв'язки, та абстрагуватися від стилю написання коду (відступи, переноси рядків, коментарі тощо, які б ускладнювали пошук антипатернів за допомогою виключно регулярних виразів). Для аналізу коду і отримання AST доцільно скористатися генератором аналізаторів формальних мов ANTLR.

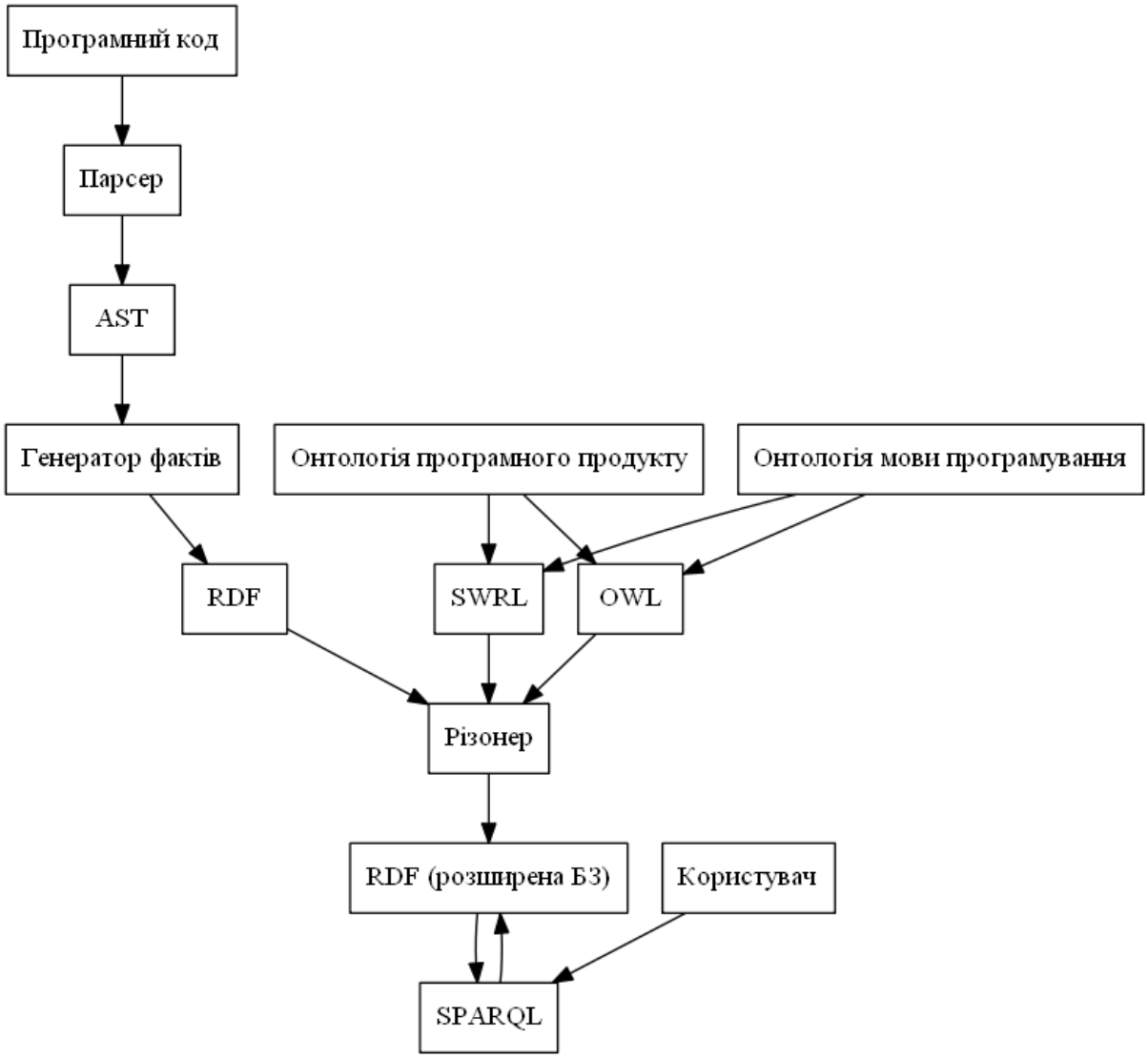

Рис. 1. Послідовність аналізу коду

Далі генератор знань приймає на вхід синтаксичне дерево та описує усі вжиті елементи мови та відношення між ними у формі триплетів з урахуванням понять OWL-онтології для конкретної мови програмування. Якщо дерево описане на мові XML, то простий генератор знань про код може бути реалізований як XSLT-шаблон, що перетворює опис дерева на новий XML-документ (RDF-триплети). Згадана ж онтологія мови програмування також містить базові факти про семантику мови програмування, а також складніші правила на мові SWRL, що дозволяє розширювати згенерований по дереву синтаксичного розбору набір триплетів прихованими в структурі онтології фактами за допомогою т.зв. різонерів. Наприклад, ланцюг наслідування, описаний фактами "ClassB є наслідує ClassA", "ClassC наслідує ClassB", "відношення 'наслідує' є транзитивним" з використанням різонера дасть новий (прихований) факт "ClassC наслідує ClassA". А це дасть змогу включити ClassC у відповідь на SPARQL-запит "*знайти усі X, для яких X наслідує ClassA*" до сховища триплетів (інакше відповідь включала б лише ClassB). Різонер, очевидно, слід обирати таким, що підтримує як OWL, так і SWRL.

Висновки. В даній статті розглядається проблема створення інструментарію для поглибленого аналізу програмного коду та рефакторингу. Запропоновано набір стандартів та програмних засобів для реалізації пошуку фрагментів коду, а також описано послідовність дій для аналізу коду з їх використанням. Головна ідея - застосувати мови опису онтологій OWL та мову опису логічних правил SWRL для опису семантики мови програмування та семантики функцій програмного продукту, з тим, щоб потім ставити запити до сховища фактів про наявність в коді певних конструкцій. Недоліком рішення є чимала робота з опису семантики мови (одноразова) та постійне оновлення семантичного опису продукту (що може бути автоматизоване лише частково за рахунок результатів синтаксичного та семантичного аналізу під час компіляції).

Надалі планується реалізувати запропоновану структуру аналізатора коду для об'єктно-орієнтованих мов C++ та Java, та перевірити можливість та ефективність пошуку як базових, так і специфічних патернів та антипатернів в коді.

Література

1. Mattia Atzeni and Maurizio Atzori. CodeOntology: RDF-ization of Source Code. The semantic web - ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings. Part II. - P.20-28.
2. Tappolet, Jonas & Kiefer, Christoph & Bernstein, Abraham.. Semantic web enabled software analysis. Web Semantics: Science, Services and Agents on the World Wide Web. 8. - 2010. - P. 225-240.
3. Kirasić, Damir & Basch, Danko. Ontology-Based Design Pattern Recognition. Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I. - 2010. - P. 384-393.
4. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. - 1994. - 395 p.