УДК 004.415.23

М.В. Коваленко, Б.Ф. Трофімов

МЕТОДИКА АВТОМАТИЗОВАНОЇ ВЕРИФІКАЦІЇ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА БАЗІ ОНТОЛОГІЙ

Робота присвячена розробці методики автоматизованої верифікації архітектури програмного забезпечення. Вперше для представлення архітектури програмного забезпечення використовується онтологічний апарат, зо дозволяє описувати процес верифікації архітектури в термінах формалізованих мов і застосовувати до нього апарат математичної логіки. Вперше використовується алгоритм перевірки ізоморфізму графів для верифікації.

Ключові слова: програмне забезпечення, онтології, архітектура, верифікація, ізоморфізм.

Вступ

З підвищенням складності програмного забезпечення постало питання про забезпечення таких його якостей, як надійність, швидкодія, відповідність заявленим вимогам, що неможливо без належного рівня організації процесу його проектування.

Для успішної реалізації проекту об'єкт проектування повинен бути перш за все адекватно описаний, тобто повинні бути побудовані повні і несуперечливі моделі архітектури програмного забезпечення, що обумовлює сукупність структурних елементів системи і зв'язків між ними, поведінку елементів системи в процесі їх взаємодії, а також ієрархію підсистем, об'єднуючих структурні елементи. Вибір архітектури задає спосіб реалізації вимог на високому рівні абстракції. Архітектура визначає більшість характеристик якості програмного забезпечення в цілому. Тому етап проектування архітектури є одним з найважливіших етапів життєвого циклу програмного забезпечення.

Проектування архітектури – це опис системи в термінах її складових модулів. При визначенні структури системи необхідно враховувати вимоги до програмного забезпечення, проводячи верифікацію архітектури, тобто узгодження архітектури з вимогами.

Проте сам етап верифікації є слабо формалізованим, тобто не існує формалізованих методів перевірки відповідності спроектованої архітектури усім встановленим вимогам [1].

Існуючі засоби автоматизованого проектування програмного забезпечення охоплюють усі етапи його проектування, включаючи ета-

пи специфікації вимог і проектування архітектури, який зазвичай зводиться до складання UML-діаграм (Unified Modeling Language) вручну і автоматичної генерації каркасного коду. Хоча деякі з цих CASE-засобів (Computer Aided Software Engineering) і пропонують можливість перевірки коректності спроектованої архітектури, тобто відсутності помилок в складених UML-діаграмах, проте жодне з відомих CASE-засобів не здатне проводити верифікацію архітектури, тобто перевірку її узгодженості з вимогами до інформаційної системи. Для верифікації архітектури існує декілька методів, таких як експертиза, статистичний аналіз, синтетичні методи і деякі інші. Проте ці методи є слабо формалізованими і припускають виконання верифікації вручну, що збільшує сумарний час розробки програмного забезпечення, а людський чинник при цьому привносить вірогідність появи помилок.

У зв'язку з цим виникла задача розробки методики автоматизації процесу верифікації архітектури програмного забезпечення для зниження витрат часу на процес розробки архітектури та підвищення якості архітектури, що проектується.

Однією з перспективних технологій представлення інформації, що отримала розвиток в останні десятиліття, є використання онтологій. Мова UML може припускатися неточностей і передбачає в деякій мірі вільний і неформальнй опис системи; крім того, існує кілька різних моделей і підходів до опису системи. С. Крейнфільд у своїй роботі [2] визначив один із значних поточних недоліків UML: відсутність формального визначення. Семантику UML визначають метамоделі, деякі додаткові обмеження, які виражаються в напівформальній мові обмежень об’єктів, і описи різних елементів мови звичайною англійською.

У структурі даних, що представляють онтології, типи понять, що використовуються, та обмеження на їх використання заявлено декларативно, явно і з використанням формальної мови [3]. Формальне уявлення передбачає, що онтологія повинна бути такою, щоб її могла обробляти машина. Однак, онтологія не є «активною» та не може бути виконаною як програма. Вона декларативно представляє деякі знання, які будуть використовуватися у програмі.

 ISSN 1562-9945

Таким чином, використання онтологій дозволяє описувати процес верифікації архітектури в термінах формалізованих мов і застосовувати до нього апарат математичної логіки [4].

Загальну схему розробленої методики наведено на рис. 1.

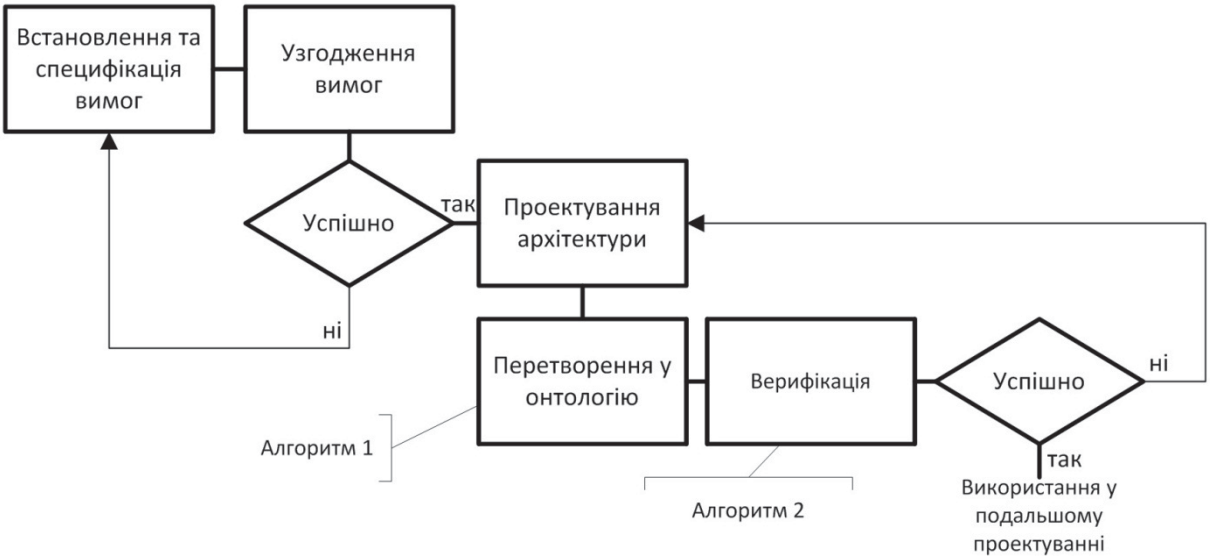

Рисунок 1 – Загальна схема методики

Методика полягає у:

1. Використанні вимог, описаних в термінах формалізованої мови, тобто представлених у вигляді узгоджених онтологій, таких що не містять протиріч, неточних, неясних вимог або вимог, що дублюються.
2. Представленні архітектури програмного забезпечення у вигляді онтологічної моделі.
3. Перевірці проміжного варіанту архітектури, представленого у вигляді онтологічної моделі та запропонованого проектантами, на відповідність вимогам, що пред'являються до системи.
4. Поверненні архітектури на доробку проектантам у разі необхідності.

У даній методиці використовуються два алгоритми: алгоритм перетворення діаграм UML, що описують архітектуру програмного забезпечення, в онтологію; та алгоритм верифікації архітектури.

Алгоритм перетворення архітектури в онтологію

Незалежно від того, являється або не являється інформаційна система Веб-застосунком, представлення його структури у вигляді онтології еквівалентно представленню її у вигляді діаграми класів UML.

Відмінність в тому, що онтології використовують модель орієнтованого поміченого графа для опису схеми.

Діаграма класів UML також може бути представлена у вигляді орієнтованого поміченого графа. Якщо орієнтований помічений граф використовується для моделі діаграми класів UML, то можна показати, що орієнтований помічений граф OWL-схеми (Web Ontology Language) буде ізоморфний підграфові орієнтованого поміченого графа діаграми класів UML. Це можливо завдяки прямому зіставленню конструкціям OWL елементів UML-діаграм класів [5].

Наведемо основні три кроки алгоритму:

1. Для кожного класу
2. Створити опис класу
3. Зробити посилання на підкласи
4. Описати атрибути класу
5. Для кожного підкласу
6. Створити опис підкласу
7. Описати атрибути класу
8. Для кожного зв'язку між класами
9. Описати тип зв'язку
10. Зробити посилання на класи-учасники зв'язку

Алгоритм було програмно реалізовано; в основу програмної реалізації лягла технологія XSL-трансформації документів XML.

Алгоритм верифікації архітектури

В рамках методики було розроблено також алгоритм верифікації архітектури програмного забезпечення. Відомо, що онтології (OWL) використовують модель орієнтованого поміченого графа для опису схеми. Таким чином, одне з можливих рішень проблеми перевірки узгодженості онтології вимог та онтології архітектури програмного забезпечення (тобто її верифікації) зводиться до визначення ізоморфності орієнтованого поміченого графа OWL-схеми архітектури програмного забезпечення підграфу орієнтованого поміченого графа OWL-схеми вимог до програмного забезпечення.

Тут під ізоморфністю будемо розуміти таке взаємно-однозначне відображення $f : V_1 \to V_2$ між множинами вершин-графів (V_1, E_1) і (V_2, E_2), що ребро одного графу переводиться у ребро другого та навпаки:

$$\forall u, v \in V_1 \ (u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$$

Для визначення ізоморфності графів було прийнято рішення використовувати алгоритм послідовного накладення з поверненням [5]. Суть алгоритму полягає в послідовному накладенні (встановленні відповідності) вершин графа A на вершини графа B таким чином, щоб формовані підмножини вершин в одному і іншому графах мали однакові відносини суміжності. Таким чином, в A і B будується загальний, з точністю до ізоморфізму, підграф, число вершин у якому на кожному кроці збільшується. При цьому, якщо на черговому кроці ніяка з решти вершин графа B не підходить для включення до вибраної підмножини, робиться крок назад і виконується спроба замінити останню вершину з включених у вказаній підмножині на іншу (рис. 2). Алгоритм завершується, якщо формується підмножина, що покриває всі вершини графа (з видачею результату "ТАК"), або при вичерпанні всіх можливих варіантів накладення (з видачею результату "НІ").

Тут важливо зауважити, що хоча онтологію можна графічно зобразити у вигляді графа, фактично графом вона не є. Тому, для того, щоб застосовувати до неї алгоритм визначення ізоморфізму, онтологію потрібно спершу перетворити в граф. Консорціумом W3C були визначені правила [6], за якими онтологію, представлену на мові OWL, можна привести до дводольного RDF графу.

На даний момент розроблений алгоритм не здатний виявити всі помилки і неточності, яких що можуть зустрітися при спробі порівняти два графа. Тому, невдале завершення алгоритму виробляється при знаходженні однієї проблеми, з виведенням тих вузлів графів, на яких була виявлена помилка.

Експериментальне дослідження алгоритму виявило його істотний недолік: при малому або навпаки великому відносному реберному заповненні час роботи алгоритму зростає на багато разів (на кілька порядків). На рис. 3 показано залежність середнього часу роботи алгоритму від відносного реберного заповнення для графів з кількістю вершин $n = 50$. У [7] вказується, що в найгіршому випадку, такого роду алгоритм може вимагати виконання порядку $O(n!)$ операцій.

Рисунок 2 – Алгоритм перевірки ізоморфізму

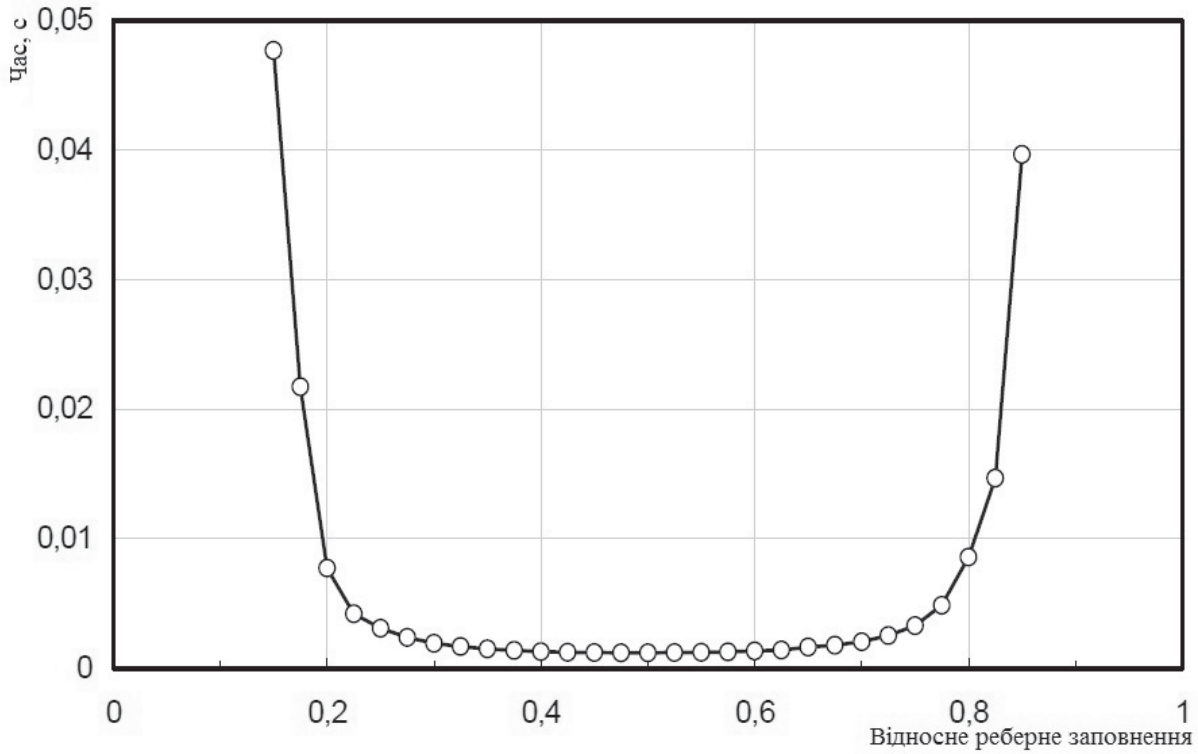

Рисунок 3 – Залежність середнього часу роботи алгоритму від реберного заповнення графів

Слід відмітити, що час роботи алгоритму слабо залежить від того, чи є надані графи ізоморфними.

Окремою проблемою є інтерпретація вихідних даних. У даному випадку алгоритм зупиняється на тому етапі, коли подальші спроби визначити ізоморфізм більше не приносять результатів, з подальшою

 ISSN 1562-9945

видачею того вузла графа, на якому сталася зупинка. Іншими словами, алгоритм працює до першого збою. Таким чином, виникає необхідність запускати алгоритм безліч разів, щоразу визначаючи лише одну проблемну ситуацію.

Очевидно, що верифікація тут є ітеративним процесом. При виявленні помилок або неузгодженостей, архітектура виправлятиметься людиною у вказаному місці, після чого процес верифікації запускатиметься наново. Якщо при верифікації не було виявлено помилок, можна з великою часткою вірогідності стверджувати, що була спроектована коректна архітектура, що реалізовує усі пред'явлені вимоги.

Висновки

Проектування архітектури є важливим етапом розробки програмного забезпечення. Оскільки архітектура безпосередньо залежить від вимог і повинна реалізувати їх, необхідно проводити її верифікацію.

У зв'язку з цим було розроблено методику автоматизації процесу верифікації архітектури програмного забезпечення на базі онтологій. В рамках цієї методики було розроблено алгоритм перетворення архітектури програмного забезпечення в онтологічну модель, а також алгоритм верифікації архітектури програмного забезпечення.

У ході експериментального дослідження було виявлено, що методика повністю вирішає поставлені перед нею задачі, а при порівнянні з іншими аналогічними системами не тільки показала себе на одному з ними рівні, але й перевищила показники деяких з них. Зокрема, у порівнянні з однією з кращих аналогічних методик час на верифікацію знизився на 4%.

Недоліком цієї методики є залежність від наданої онтології вимог і від якості їх узгодження.

Переваги цієї методики :

1. підсумкове зниження тимчасових витрат за рахунок машинного аналізу архітектури;

2. можливість отримання гарантовано коректної і узгодженої архітектури;

3. використання онтологій для представлення архітектури системи, що, за наявності апарату їх формального аналізу, дозволить виявити конфлікти у разі ітеративного процесу розробки програмного забезпечення.

ЛІТЕРАТУРА

1. Clements, P. Evaluating Software Architectures: Methods and Case Studies [текст] / P. Clements, R. Kazman, M. Klein // Addison-Wesley Professional, January 2002.
2. Cranefield, S. “Networked knowledge representation and exchange using UML and RDF” [тест] / Journal of Digital Information, vol. 1, no. 8, article no. 44, 2001-02-15.
3. Cranefield, S. “UML and the Semantic Web” [текст] / Proceedings of the Semantic Web Working Symposium, Stanford University, CA, pp. 113–130.
4. Akerman, A. Using ontology to support development of software architectures [текст] / A. Akerman, J. Tyree // IBM Syst, J. 45, 4 (2006), 813-825.
5. ISO/TC 211/WG 7/PT 19150. Report from Project 19150 Geographic information - Ontology meeting in Molde [текст], Norway, May 28-29, 2009.
6. P. F. Patel-Schneider, P. Hayes, I. Horrocks (eds.). OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 February 2004.
7. Рейнгольд Э, Нівергельт Ю, Део Н. Комбінаторні алгоритми. Теорія й практка.-М.: Мир, 1980.- 476 с.

 ISSN 1562-9945