УДК 681.3.068

Т.В. Ковалюк, В.В. Гулаков

МОДЕЛІ ТА АЛГОРИТМИ АВТОМАТИЗОВАНОЇ ПОБУДОВИ ОБ'ЄКТНИХ МОДЕЛЕЙ ПРЕДМЕТНОГО СЕРЕДОВИЩА

Анотація: Розглянуто моделі та алгоритми, що формалізують та автоматизують процес побудови об'єктно-орієнтованих моделей предметних середовищ. Специфікація моделі предметного середовища задається природною мовою. На етапі аналізу вербального опису предметного середовища здійснюється розбір мови, який передбачає синтаксичний аналіз з використанням контекстно-вільних граматик, та семантичний аналіз з програмною реалізацією об'єктної моделі.

Ключові слова: Об'єктно-орієнтована модель, предметне середовище, семантичний аналіз, семантичний граф, контекстно-вільні граматики, шаблони проектування.

Вступ

Парадигма об'єктно-орієнтованого підходу спрощує процес розробки складних програмних систем. Однак для того, щоб використовувати усі переваги об'єктно-орієнтованого підходу, необхідно спроектувати програмну систему застосовуючи концепції абстрагування, інкапсуляції, наслідування та поліморфізму. Принципи інкапсуляції та поліморфізму дозволяють розробнику нових функціональних можливостей програмної системи зосередитися на реалізації поставленого завдання і упустити нюанси реалізації вже написаних модулів. А поліморфізм і наслідування дають змогу підвищити коефіцієнт повторного використання коду. Саме з огляду на ці міркування об'єктно-орієнтоване проектування відіграє таку важливу роль в сучасному світі розробки програмного забезпечення. Постає питання, як спроектувати програмну систему якісно та ефективно. Рішення проблеми полягає в застосуванні шаблонів об'єктів із стереотипними функціями і взаємодіями. Шаблони проектування – це елегантні й прості рішення типових завдань, що виникають в процесі розробки системи і які можуть повторно використовуватися.

У цій роботі здійснена спроба формалізувати процес застосування шаблонів проектування програмних систем за допомогою розроблених авторами засобів автоматизованої генерації об'єктно-орієнтованих моделей предметних середовищ.

Загальна ідея полягає в тому, щоб дати можливість розробнику програмної системи задавати специфікацію моделі предметного середовища у вигляді тексту на мові, близькій до природної. Причому розробник системи описує лише загальні риси системи. На етапі аналізу вербального опису предметного середовища здійснюється розбір мови, який передбачає синтаксичний аналіз з використанням контекстно-вільних граматик і побудовою дерева розбору та семантичний аналіз з програмною реалізацією об'єктної моделі. В процесі автоматизованої побудови об'єктної моде-

лі предметного середовища використовуються шаблони проектування, що забезпечує зручність та гнучкість.

У цій статті набули розвитку ідеї, описані раніше автором у статті [2].

Математична модель задачі

Формалізуємо поставлену задачу. Нехай для опису об’єктно-орієнтованої моделі предметного середовища необхідно розробити мову L. На етапі розбору вхідних даних використовується контекстно-вільна граматика G, що визначає мову L:

$$L = L(G). \tag{1}$$

Відомо, що контекстно-вільну граматику можна подати як сукупність чотирьох об’єктів [6]:

$$G = (N, T, P, S), \tag{2}$$

де N – скінченна множина нетермінальних символів мови; T – скінченна множина термінальних символів мови;P – скінченна множина продукцій $\alpha \to \beta, \alpha = (N \cup T)^+$, $\beta = (N \cup T)^*$, S – початковий символ граматики із множини нетермінальних символів N.

Розглянемо етап семантичного аналізу. Задамо множину допустимих ідентифікаторів I, до якої входять ланцюжки із символів латинського алфавіту та цифр, довжина яких не перевищує 255 символів. Перший символ у ланцюжку обов’язково повинен бути літерою.

На першому етапі семантичного аналізу відбувається побудова семантичного графа S:

$$S = (V, E, \varphi), \tag{3}$$

де V – множина вершин графа, E – множина ребер графа, φ – функція інцидентності, яка кожному елементу множини E ставить у відповідність пару елементів з множини V.

Кожна вершина може описувати або сутність предметного середовища, або ж операцію чи процес, що протікає в предметному середовищі. Кожне ребро графа описує зв’язки між сутностями предметного середовища або пов’язує сутності із певними операціями.

Для кожної вершини графа визначимо функції:

$$VertexName : V \to I \tag{4}$$

$$VertexType : V \to \{EssenceType, OperationType\} \tag{5}$$

Вираз (4) ставить у відповідність кожній вершині графа унікальний ідентифікатор. Вираз (5) задає тип вершини. Відповідно до типу вершини, виділимо два підкласи множини вершин:

$$\begin{gathered} Essences = \{v \in V | VertexType(v) = EssenceType\} \\ Operations = \{v \in V | VertexType(v) = OperationType\} \\ Essences \cup Operations = V \\ Essences \cup Operations = \emptyset \end{gathered} \tag{6}$$

 ISSN 1560-8956

Визначимо функцію $EdgeType$ для ребер графа:

. (7)

$$EdgeTypes = \left\{ \begin{array}{c} ContainsType, CanType, UsesType, \\ RealIsType, IsType, DelegatesType, \\ DelegatesToType \end{array} \right\} \quad (8)$$

Вираз (7) задає для кожного ребра графа тип. З виразу (8) видно, що множина типів ребер включає в себе значну кількість елементів. Розіб’ємо множину ребер на підкласи за типом відповідно до (9):

$$Relations_i = \{e \in E | EdgeType(e) = EdgeTypes_i, i = \overline{1, EdgeType}\} \quad (9)$$

де $\bigcup\limits_i Relations_i = E$, $\bigcap\limits_i Relations_i = \emptyset$

Тип ребра накладає певні обмеження на тип вершин, що інцидентні з ним:

$$RoleA = EdgeTypes \rightarrow VertexType \quad (10)$$

$$RoleB = EdgeTypes \rightarrow VertexType \quad (11)$$

$$\begin{array}{l} IsValid(e) = EdgeType(e) \equiv type \wedge \langle v_1 | v_2 \rangle \equiv \varphi(e) \wedge \\ RoleA(type) \equiv VertexType(v_1) \wedge RoleB(type) \equiv VertexType(v_2) \end{array} \quad (12)$$

Функції (10) і (11) для кожного типу відношень визначають типи вершин, до яких ці відношення можуть бути застосовані. Булева функція (12) приймає на вхід відношення і повертає логічне значення істини, якщо відношення пройшло валідацію. Якщо (12) повертає хибне значення, відношення задано некоректно і його потрібно виключити із графа.

Розбір відбувається за класичною схемою синтаксично-керованої трансляції (syntax-directed translation) [1]. Отже, синтаксичний аналізатор – це складова транслятора, яка за необхідності викликає лексичний аналізатор, щоб отримати наступний нетермінал, та семантичні процедури нетерміналів, скеровуючи таким чином семантичний аналіз.

Зокрема у нашому випадку семантичні процедури доповнюють семантичний граф (3) новими вершинами та ребрами. Кожне ребро перед додаванням у граф проходить валідацію згідно з (12).

Після побудови семантичного графа, створюється об’єктно-орієнтована модель. Нехай M – множина усіх об’єктно-орієнтованих моделей. Тоді на виході необхідно скомпонувати модель $Model$, яку можна описати (13):

$$Model = \{Class\}, Model \notin M. \quad (13)$$

Отже, об’єктна модель – це множина класів. Кожен клас може бути представлений у вигляді впорядкованої трійки (14):

$$Class = \langle Name | \{Attribute\} | \{Method\} \rangle, Name \in I, \quad (14)$$

де $Name$ – ім'я класу із множини допустимих ідентифікаторів I; $\{Attribute\}$ – множина атрибутів класу; $\{Method\}$ – множина методів класу.

Визначимо атрибут класу як двійку елементів(15):

$$Attribute = \langle Name|Type, ..., Name|Type\rangle\,, Name \in I, Type \in Model, \quad (15)$$

де $Name$ – ім'я атрибуту із множини допустимих ідентифікаторів I; $Type$ – тип даних атрибута, тобто один із класів, які описані в рамках моделі $Model$.

Метод класу представимо у вигляді трійки елементів (16):

$$Method = \langle Name|ReturnType|\,\{\ \langle ArgName|ArgType\rangle\ \}\rangle\,, \quad (16)$$

де $Name$ – ім'я метода із множини допустимих ідентифікаторів I; $ReturnType$ – тип значення, яке повертає метод; $\{\ \langle ArgName|ArgType\rangle\ \}$ – множина аргументів метода, кожен з яких представлений парою, до складу якої входять ім'я атрибута $ArgName$ та його тип даних $ArgType$.

Алгоритм побудови об'єктно-орієнтованої моделі

На базі семантичного графу побудуємо об'єктно-орієнтовану модель. Для початку визначимо допоміжні функції й відношення. Розглянемо функцію:

$$ApplyRelation : E \otimes M \to M. \quad (17)$$

Функція (17) приймає на вхід дугу семантичного графа $e \in E$ та модель $m \in M$ і повертає на виході модель $m' \in M$, доповнену додатковими класами, методами, атрибутами та відношеннями між класами.

На множині $EdgeTypes$ визначимо відношення часткового порядку "$\leqslant$", яке дозволить визначити послідовність, в якій необхідно опрацьовувати ребра, що відходять від даної вершини.

Наведемо алгоритм побудови об'єктно-орієнтованої моделі.

1. Побудувати множину $Model$:

$$Model = \{\langle\ VertexName(v), \emptyset, \emptyset\ \rangle\,|v \in Essences\}. \quad (18)$$

1. Для кожної вершини $v \in V$:
   (a) Отримати усі ребра E', які суміжні з вершиною v.
   (b) Побудувати частково впорядковану множину $\langle E', \leqslant\rangle$.
   (c) Для кожного ребра e із множини $\langle E', \leqslant\rangle$ визначити
      i. $Model = ApplyRelation(e, Model)$

Етап семантичного аналізу

Як відомо, етап семантичного аналізу дуже складно формалізувати [1]. У попередньому розділі наведена спрощена схема семантичного аналізу, де були упущені важливі деталі. Зокрема не були розглянуті семантичні процедури граматики (2), а також було оголошено, проте не

визначено, функцію $ApplyRelation$ (17). У цьому розділі будуть розглянуті деякі мовні конструкції мови $gen2$, а також будуть описані семантичні дії, що відповідають за їхнє оброблення.

Розглянемо найпростішу конструкцію мови (19):

$$AisB. \tag{19}$$

Це означає, що предметне середовище включає дві сутності з іменами A і B, причому A є підкласом B. При обробленні цієї конструкції семантичний граф буде доповнений вершинами v_1, v_2 такими, що:

$$v_1, v_2 : VertexName(v_1) = A, VertexName(v_2) = B,$$
$$VertexType(v_1) = VertexType(v_2) = Essence$$

та дугою $e : \varphi(e) = (v_1, v_2), EdgeType(e) = IsType$.

На наступному етапі семантичного аналізу до об’єктно-орієнтованої моделі додаються два класи A і B, та визначається наслідування A від B:

$$Acanopu\sin gB, C. \tag{20}$$

Це означає, що предметне середовище включає сутності, що мають імена A, B і C, причому сутність A підтримує операцію op, яка виконується за участі сутностей B і C. При обробленні цієї конструкції семантичний граф буде доповнений вершинами v_1, v_2, v_3, v_4:

$$v_1, v_2, v_3, v_4 : VertexName(v_1) = A, VertexName(v_2) = B,$$
$$VertexName(v_3) = C, VertexName(v_4) = op,$$
$$VertexType(v_1) = VertexType(v_2) = VertexType(v_3) = Essence,$$
$$VertexType(v_4) = Operation$$

та дугами:

$$e1, e2, e3, e4 : \varphi(e1) = (v1, v4), EdgeType(e1) = CanType,$$
$$\varphi(e2) = (v4, v2), \varphi(e3) = (v4, v3),\ EdgeType(e3) = UsesType$$

На другому етапі семантичного аналізу до об’єктно-орієнтованої моделі будуть додані три класи A, B, C. У клас A буде доданий метод op, який приймає два аргументи типів B і C.

Розглянемо складніший приклад.

$$\begin{gathered} Ais\ leafofcompositionB, \\ Cis\ leafofcompositionB, \\ Dis\ compositeofcompositionB, \\ Bcanop1, \\ Bcanop2 \end{gathered} \tag{21}$$

При розборі цієї конструкції буде виконана побудова об’єктно-орієнтованої моделі згідно із шаблоном проектування “композиція” [2]. Семантичний граф, що відповідає (21), у графічному вигляді наведено на рисунку 1.

Побудована об’єктно-орієнтована модель має класи A, B, C, D. Клас B має віртуальні методи $add()$, $remove()$, $children()$, які додаються згідно із структурою шаблону проектування. Також клас B має віртуальні методи $op1()$ та $op2()$, які додані відповідно до специфікації (21). У класах A, C, D, що успадковуються від B, перевизначені усі віртуальні методи із класу B.

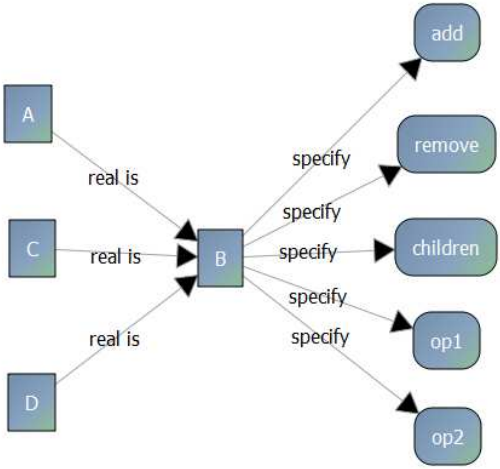

Рис. 1 – Семантичний граф, що реалізує шаблон проектування “композиція”

Засоби реалізації

Авторами даної статті розроблюється програмний продукт $gen2$, у якому реалізовані описані вище ідеї. Розглянемо інструменти, які застосовуються для реалізації.

В якості мови програмування була обрана мова C++, яка часто використовується при розгляді технік об’єктно-орієнтованого проектування [7]. Крім того, існує багато бібліотек на C++, призначених для побудови синтаксичних аналізаторів на основі контекстно-вільних граматик. Автором розглянуто кілька із них, зокрема, Yacc, Bison, Coco/R. Зрештою, була обрана бібліотека boost::spirit, яка дає змогу описувати граматику разом з семантичними діями прямо в програмному коді на C++.

Програмний продукт $gen2$ можна використовувати як бібліотеку, консольне застосування або як застосування із графічним інтерфейсом користувача.

Для реалізації графічного інтерфейсу була обрана бібліотека Qt, оскільки вона об’єктно-орієнтована і кросплатформенна [5]. Також для візуалізації семантичного графа використовується бібліотека $GraphViz$. Програмний продукт незалежний від платформи на рівні програмного

коду, може бути скомпільований для роботи під будь-якою операційною системою.

Висновки

У даній статті описано формалізований підхід до побудови об’єктно-орієнтованих моделей з реалізацією за допомогою програмного продукту $gen2$. Можливості $gen2$ позволяють будувати об’єктно-орієнтовані моделі середньої складності.

Створено основу, яка буде збільшувати свою функціональність, а отже буде розширено коло задач, які можна розв’язувати за допомогою цього інструмента. Зокрема планується:

- додати можливість виконувати розробку в поєднанні з популярними пакетами для об’єктно-орієнтованого проектування;
- додати можливість генерувати об’єктно-орієнтовану модель у форматі UML;
- реалізувати можливість генерувати не тільки об’єктно-орієнтовану модель, а й тіло методів, оскільки це необхідно для того, щоб більш повно реалізувати деякі шаблони проектування;
- розширити бібліотеку шаблонів проектування;
- реалізувати зворотну розробку (reverse engineering).

Література

1. Ахо А., Сети Р., Ульман Дж. Компиляторы: принципы, технологии и инструменты. – М.: Издательский дом “Вильямс”, 2003. – 768 с.
2. Гулаков В.В., Ковалюк Т.В. Моделювання опису предметного середовища засобами синтаксично-орієнтованої трансляції //Адаптивні системи автоматичного управління. – 2008. – Вип. 12 (32). С. 54-61.
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированонного проектирования. Паттерны проектирования. – СПб.: Питер, 2008. – 366 с.
4. Ларман К. Применение UML и шаблонов проектирования. – М.: Издательский дом “Вильямс”, 2001. – 496 с.
5. The Book of Qt 4: The Art of Building Qt Applications by Daniel Molkentin Paperback, 440 Pages, Published 2007, 1st Edition
6. Рейуорд-Смит В. Дж. Теория формальных языков. Вводный курс. – М.: Радио и связь, 1988. – 128 с.
7. Bjarne Stroustrup. Evolving a language in and for the real world: C++ 1991-2006.

Отримано 08.03.2011 р.