электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.436.4+519.178

Метод поиска шаблонов проектирования в UML-диаграммах классов на основе алгоритма поиска изоморфных подграфов

***Сиромаха Р.В.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Рудаков И.В., к.т.н.
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
irudakov@bmstu.ru

Введение

Существует задача поиска шаблонов проектирования [1]. Объектом поиска может являться любой программный продукт, который разработан с использованием парадигмы ООП. Структуру такой программы или программного комплекса можно представить в виде UML-диаграммы классов [2]. Шаблоны проектирования можно представить в виде UML-диаграмм классов. Таким образом, задача сводится к поиску одной UML-диаграммы классов в другой.

Основным подходом решения данной задачи является сведение её к поиску изоморфного подграфа [3][4]. Как правило, граф строится на основе связей наследования. Наиболее используемым является алгоритм определения меры схожести вершин графа [5]. Если требуется детализация представления UML-диаграммы классов, например, сопоставление свойств и операций классов, данный подход неприменим. В качестве результата мы получаем набор коэффициентов соответствия в диапазоне от 0 до 1, который не позволяет определить точно какому классу в шаблоне соответствует класс целевой UML-диаграммы.

Необходимо определение точного соответствия. Более того, маловероятна ситуация, что какой либо класс UML-диаграммы соответствует только одному классу шаблона, поэтому нужен поиск всех возможных комбинаций таких соответствий. Такую задачу позволяет решить алгоритм поиска изоморфизма подграфов [6], который основан на методе поиска с возвратом.

В данной работе описан метод, который позволяет найти все возможные изоморфизмы UML-диаграммы классов шаблону проектирования. Учитываться элементы UML-диаграммы классов, которые наследуются от элемента Classifier (Class, DataType,

PrimitiveType, Interface), их свойства (Property) с учетом типа и атрибутов и операции (Operation) с учетом возвращаемого значения, параметров и атрибутов; связи: Association, Dependency, Generalization, Substitution, Usage.

Множество графов UML-диаграммы классов

Множество связей, а также различие связываемых элементов требует использования множества графов. Для каждого класса связей нужно построить свой граф. Такое множество графов не учитывает всех возможных связей между элементами UML-диаграммой классов, поэтому определим структуру каждой вершины, учитывая остальные связи вершин. Для этого используем UML-диаграмму классов (рис. 1, 2). Можно сказать, что такие связи тоже образую графы, но нам не требуется выделять их особым образом, так как для них не будет выполняться поиск изоморфизмов, хотя они и будут использоваться для поиска в других графах.

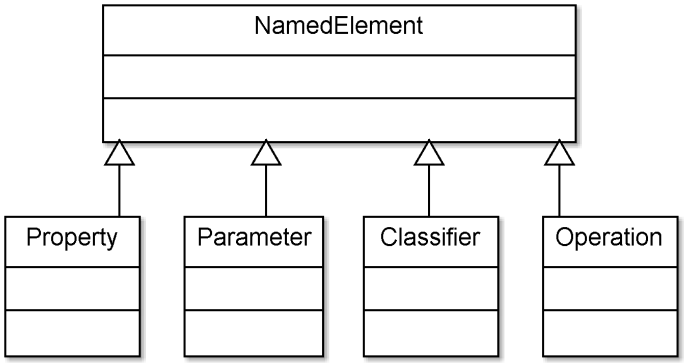

Рис. 1. Структура классов вершин графа UML-диаграммы классов. Класс NamedElement

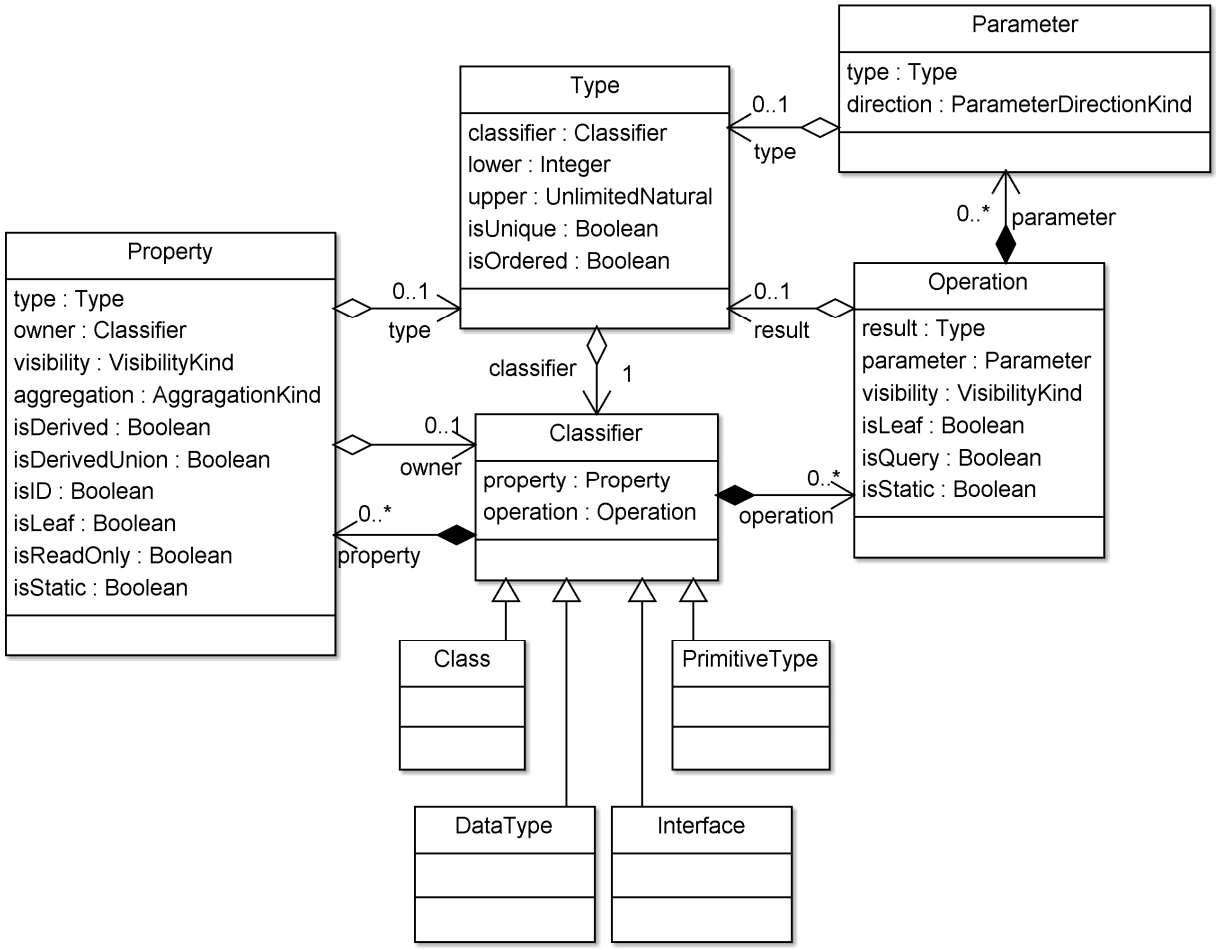

Рис. 2. Структура классов вершин графа UML-диаграммы классов

Определение. $G_{Association} = (\{property_i\}, \{association_j\})$ – граф ассоциаций, где $property_i \in Property$, $association_j = \{property_n, property_m\}$.

Определение. $G_{Dependency} = (\{classifier_i\}, \{dependency_j\})$ – граф зависимостей, где $classifier_i$ – объект класса NamedElement, $dependency_j = (client, supplier)$, $client$, $supplier$ – объекты класса NamedElement.

Определение. $G_{Generatlization} = (\{classifier_i\}, \{generalization_j\})$ – граф наследования, где $classifier_i$ – объект класса Classifier, $generalization_j = (general, specific)$, $general$, $specific$ – объекты класса Classifier.

Определение. $G_{Substitution} = (\{classifier_i\}, \{substitution_j\})$ – граф заменимости, где $classifier_i$ – объект класса Classifier, $substitution_j = (contract, substitutingClassifier)$, $contract$, $substituting$ – объекты класса Classifier.

Определение. $G_{Usage} = (\{classifier_i\}, \{usage_j\})$ – граф использования, где $classifier_i$ – объект класса NamedElement, $generalization_j = (client,\ supplier)$, $client$, $supplier$ – объекты класса NamedElement.

Определение. $UML = \{G_{Association}, G_{Dependency}, G_{Generatlization}, G_{Substitution}, G_{Usage}\}$ – множество графов, представляющих UML-диаграмму классов.

Эквивалентность UML-диаграмм классов

Использование сложной структуры вершины графа требует определения эквивалентности. Эквивалентность классов предполагает эквивалентность их свойств и операций. С другой стороны, эквивалентность свойств означает эквивалентность классов, которым они принадлежат. Возникает бесконечная рекурсия. Данную проблему можно решить введением определения частичной эквивалентности.

Эквивалентность всегда определяется для пары элементов целевой UML-диаграммы и шаблона. Наложим ограничение отсутствия коммутативности, чтобы определять эквивалентность для упорядоченной пары (цель, шаблон). Для обозначения эквивалентности будем использовать оператор ≅ (цель ≅ шаблон), а для частичной – $\cong_p$ (цель $\cong_p$ шаблон).

Определение. Упорядоченная пара $(lower, upper)$ атрибутов класса Type называется множественностью, если $(lower, upper \geq 0)\&(lower \leq upper)|(lower = \emptyset)|(upper = \emptyset)$.

Определение. Множественности $(lower_t, upper_t)$ и $(lower_p, upper_p)$ эквивалентны, если

$$\big(lower_p = \emptyset\big)|\Big(\big((lower_t \neq \emptyset)\&\big(lower_t \geq lower_p\big)\big)\&$$

$$\big(upper_p = \emptyset\big)|\Big(\big(upper_p \neq \emptyset\big)\&\big(upper_t \leq upper_p\big)\Big)$$

Определение. Объекты класса Type частично эквиваленты, если

1) значения атрибута classifier эквивалентны;
2) эквивалентны множественности;
3) значения атрибутов isOrdered и isUnique равны;

Определение. Объекты класса Type равны, если равны значения всех атрибутов.

Определение. Объекты класса Property равны и частично эквивалентны, если равны значения всех атрибутов кроме owner.

Определение. Объекты класса Property эквивалентны, если они равны и

$$(owner_t = \emptyset)|\Big(\big(owner_p \neq \emptyset\big)\&(owner_t \cong owner_p)\Big)$$

Определение. Объекты класса Parameter частично эквивалентны, если значения атрибута direction равны и

$$(type_t = \emptyset) | \left((type_p \neq \emptyset) \& (type_t \cong_p type_p) \right)$$

Определение. Объекты класса Parameter равны, если равны значения всех атрибутов.

Определение. Объекты класса Operation частично эквивалентны, если

1) значения всех атрибутов, кроме result и parameter равны;

2) $(result_t = \emptyset) | \left((result_p \neq \emptyset) \& (result_t \cong_p result_p) \right)$

3) для каждого значения атрибута parameter шаблона существует эквивалентное значение в целевом объекте. Одному значению в целевом объекте может быть эквивалентно только одно значение в шаблоне.

Определение. Объекты класса Operation равны, если равны значения всех атрибутов.

Определение. Объекты, наследуемые от класса Classifier, эквивалентны, если для каждого значения атрибутов property и operation шаблона существует эквивалентное значение в целевом объекте. Одному значению в целевом объекте может быть эквивалентно только одно значение в шаблоне.

Определение. Вершины любого из графа класса связей эквивалентны, если

1) количество дуг, исходящих из вершины, исключая дугу в эту же вершину, не менее количества таких же дуг шаблонной вершины;

2) количество дуг, входящих из вершины, исключая дугу из этой же вершины, не менее количества таких же дуг шаблонной вершины;

3) классы вершин равны;

4) объекты классов вершин эквивалентны.

Алгоритм поиска всех возможных изоморфных подграфов

Используя определение эквивалентности для вершин графа, можно выполнить поиск изоморфного подграфа. Для целевого графа и шаблона нужно найти множество множеств пар (*вершина целевого графа, вершина шаблона*). Будем использовать метод поиска с возвратом, который предполагает наличие состояния поиска – конфигурации.

Определим структуру графа, вершины и конфигурации (рис. 3).

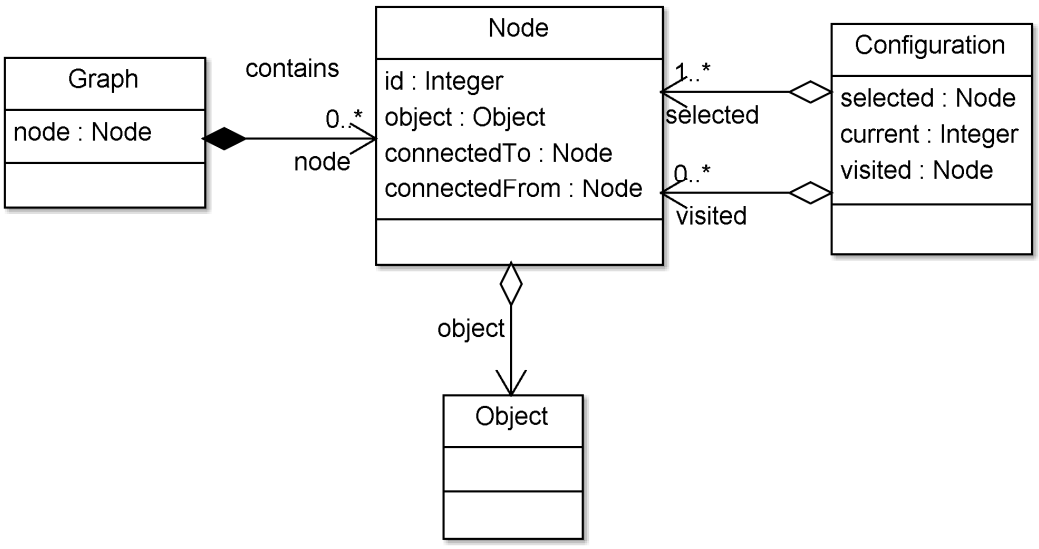

Рис. 3. Структура графа и конфигурации поиска с возвратом

Граф содержит множество вершин node. Каждая вершина имеет уникальный числовой идентификатор id, ссылку на объект object (один из объектов ранее определенных классов), множество вершин connectedTo, в которые проведены дуги, и множество вершин connectedFrom, из которых проведены дуги, оба множества исключают текущую вершину. Конфигурация содержит список пар вершин selected, по которым нужно выполнить обход, индекс текущего элемента current в списке selected и множество пройденных пар вершин visited. В начальном состоянии конфигурации список selected содержит одну пару вершин, current равно нулю, множество visited пусто. Конфигурация находится в конечном состоянии, когда current больше чем максимальный индекс selected.

Алгоритм является обычным поиском в глубину, использующим стек вариантов конфигураций. Инициализируется стек добавлением конфигураций для всех возможных пар вершин целевого графа и шаблона. Конфигурации берутся из стека, пока он не пуст. Если конфигурация находится в конечном состоянии и множество visited содержит все вершины шаблона, то оно выводится в качестве очередного результата, а конфигурация больше не используется. На каждом шаге алгоритма для текущей конфигурации на основе ее копии создаются новые конфигурации, затем текущая пара selected[current] добавляется в множество visited, current инкрементируется, и конфигурация кладется обратно в стек.

Создание конфигураций выполняется следующим образом:

1) Для всех соседей (вершин, соединенных дугой) текущей целевой вершины и шаблонной вершины создается множество всех возможных комбинаций пар эквивалентных вершин. В одной комбинации пар каждой целевой вершине соответствует одна шаблонная вершина.

2) Пары вершин фильтруются: для каждого соседа шаблонной вершины должен существовать эквивалентный сосед целевой вершины. Пары вершин, которые содержатся во множестве visited, считаются эквивалентными.

Описанный алгоритм допускает дубликатов в результате, поэтому в конце необходимо выполнить их удаление.

Выводы

Разработан метод, который позволяет найти все возможные соответствия шаблону проектирования в UML-диаграмме классов. Метод позволяет не только определить, найден шаблон или нет, но и разметить UML-диаграмму классов соответствующими элементами из шаблона. Данный метод может быть использован для исследования существующих программных продуктов на предмет наличия шаблонов проектирования, а также для документирования проектов.

Список литературы

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software Gamma. New Jersey: Pearson Education, 1994. 395 p.
2. OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1, 2011, Available at: http://www.omg.org/spec/UML/2.4.1/Superstructure, accessed 01.12.2014.
3. Tsantalis N., Chatzigeorgiou A., Stephanides G., Halkidis S.T. Design Pattern Detection Using Similarity Scoring. Transactions on Software Engineering. 2006. Vol. 32. Issue 11. P. 896 – 909.
4. Dong J., Sun Y., Zhao Y. Design Pattern Detection by Template Matching. Proceedings of the 2008 ACM symposium on Applied computing. 2008. P. 765 – 769.
5. Blondel V.D., Van Dooren P. A Measure Of Similarity Between Graph Vertices. With Applications To Synonym Extraction And Web Searching. SIAM Review. 2004. Vol. 46. No. 4. P. 647 – 666.
6. Ullmann J.R., An Algorithm for Subgraph Isomorphism. Association for Computing Machinery. 1976. Vol. 23. Issue 1 P. 31 – 42.