Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Программное обеспечение ЭВМ и информационные технологии**

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к магистерской диссертации на тему:

Метод поиска шаблонов проектирования в объектно-ориентированных программах

Студент _____ _____ Р.В. Сиромаха
(Подпись, дата) (И.О.Фамилия)

Руководитель магистерской диссертации _____ И.В. Рудаков
(Подпись, дата) (И.О.Фамилия)

Москва, 2015

Реферат

В работе предложен метод поиска шаблонов проектирования в объектно-ориентированных программах на основе алгоритма поиска изоморфных подграфов. В первой части рассмотрены существующие методы. Проанализированы алгоритмы поиска изоморфных подграфов. Во второй части предложена модель представления программы на основе UML-диаграммы классов. Представление модели в виде графа и алгоритм построения этого графа. Описан алгоритм поиска изоморфных подграфов в ориентированных графах с множеством типов дуг. В третьей части описана реализация программного комплекса, который позволяет искать шаблоны проектирования в программах, компилирующийся в байт-код виртуальной машины **Java**. В четвертой приведены результаты исследования: поиск шаблонов проектирования в существующих программах и библиотеках.

Отчет содержит 93 с., 4 ч., 27 рис., 4 табл., 15 источников, 6 приложений.

Ключевые слова: ШАБЛОН ПРОЕКТИРОВАНИЯ, UML-ДИАГРАММА КЛАССОВ, ГРАФ, ИЗОМОРФИЗМ ПОДГРАФОВ, ОБЪЕКТНО-ОРИЕТИРОВАННЫЕ ПРОГРАММЫ.

Содержание

Введение

Объектно-ориентированные системы обладают свойством повторяемости конструкций. Решения множества конкретных задач можно обобщить таким образом, что структура части системы может быть описана некоторым шаблоном проектирования. Существуют описания шаблонов, которые рекомендуется использовать при проектировании таких систем. Описание шаблона — структура из классов и связей между ними. Один из наиболее выразительных языков для описания таких структур — UML [1]. Также существуют анти-шаблоны проектирования. Некоторые из них можно описать аналогично шаблонам проектирования.

Задача поиска шаблонов проектирования имеет место, её решают различными методами. 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. Таким образом, задача сводится к поиску одной UML-диаграммы классов в другой. В этой работе будет предложен метод поиска шаблонов проектирования в объектно-ориентированных программах. Основой для модели программы будут UML-диаграммы классов. Метод позволяет найти все возможные изоморфизмы UML-диаграммы классов шаблону проектирования.

1 Аналитический раздел

1.1 Задача поиска шаблонов проектирования

Шаблон проектирования системным образом называет, определяет причины и разъясняет в общем виде проблему повторяемости некоторой конструкции в объектно-ориентированных системах. Описывает не только задачу, но и решение, когда его применить, и к каким последствиям это приведет. Также предоставляет примеры и советы по реализации. Решение — это общая структура объектов и классов, которая решает задачу. Оно варьируется, и его реализация зависит от контекста задачи [2].

Из этого определения можно выделить следующие особенности шаблонов проектирования, которые необходимо учесть при описании структуры:

а) описание конструкции в общем виде — означает, что нужно иметь механизм, который позволит уйти от конкретных типов данных, различного рода идентификаторов;

б) повторимость конструкции — описание шаблона должно определять все возможные конструкции в системе, подпадающие под этот шаблон;

в) описывает решение — шаблоном может быть часть программы написанная на любом объектно-ориентированном языке.

Целью для поиска должна быть некоторым образом формализованная объектно-ориентированная система. Найти готовый проект такой системы, описанный по определенным стандартам довольно сложно. Чтобы убедиться в работоспособности метода можно использовать исходный код программ. Структура таких проектов точно будет работоспособной, сложность будет только в построении модели.

Результатом поиска должно быть множество вариантов соответствий структурных элементов системы элементам шаблона. В самом общем виде метод представлен на рисунке 1.1. Подробнее на рисунке 1.2.

1.2 Анализ существующих методов поиска шаблонов проектирования

Основным подходом решения поставленной задачи является сведение её к поиску изоморфного подграфа. Кратко рассмотрим возможные модификации такого подхода.

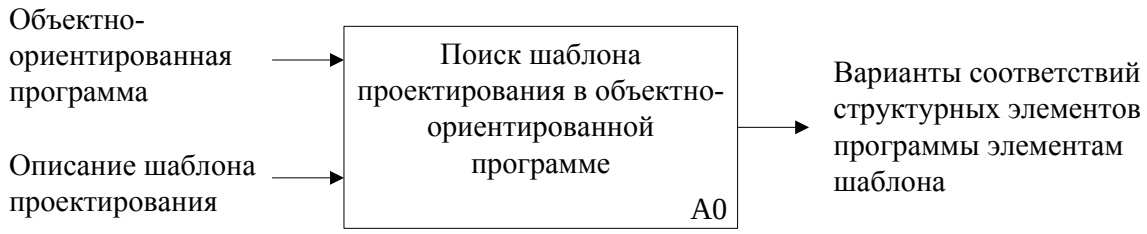

Рисунок 1.1 — Метод поиска шаблонов проектирования в объектно-ориентированных программах

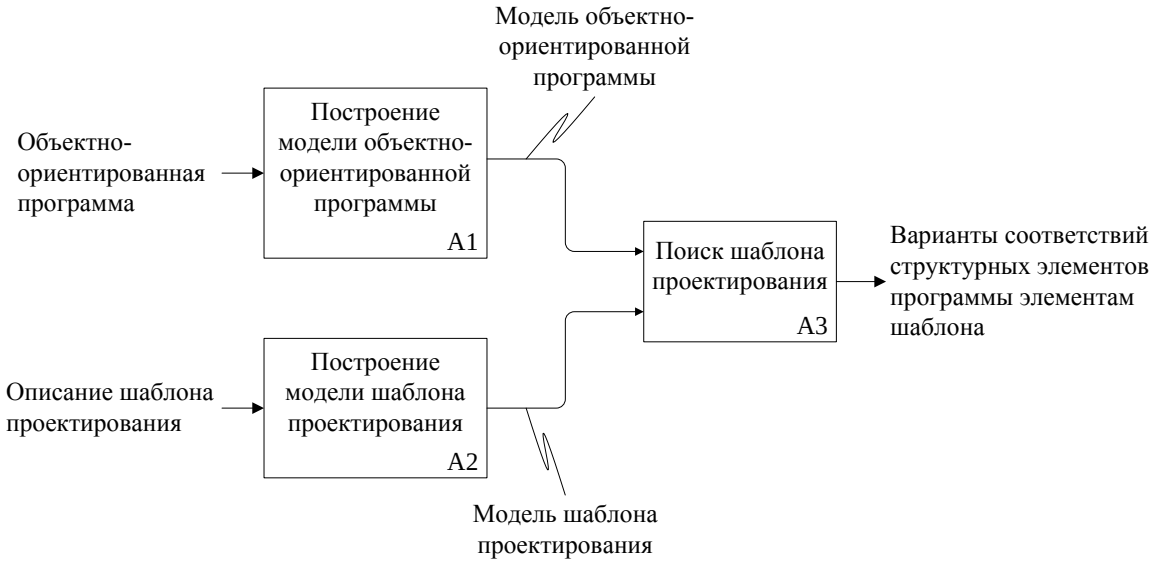

Рисунок 1.2 — Метод поиска шаблонов проектирования в объектно-ориентированных программах с использованием моделей программы и шаблона

1.2.1 Метод поиска шаблонов проектирования с использованием меры схожести

Алгоритм описан в статье «Design Pattern Detection Using Similarity Scoring» [3]. Шаблон и программа представляются с помощью четырех ориентированных графов:

а) ассоциаций — определяет ассоциации между классами;

б) обобщений — определяет связи между базовым и производными классами;

в) абстрактных классов — определяет, какие классы являются абстрактными;

г) вызовов абстрактных методов — определяет вызов из тела метода начального класса такого же абстрактного метода другого класса.

Графы представляются в виде матриц смежности. Недостаток такого подхода в том, что учитываются только классы. Отсутствие свойств и методов ведет к потере точности определения шаблонов.

Для поиска шаблонов используется алгоритм изоморфного подграфа на основе меры схожести вершин [4]. Авторы делают выбор в пользу него как неточного алгоритма исходя из следующих преимуществ:

а) применимость при невозможности найти изоморфизм;

б) возможность вычисления расстояния между графами — количества модификаций, необходимых, чтобы один граф стал другим.

Считается, что в контексте поиска шаблонов проектирования такой подход лучше. Если изоморфизма не существует, не понятно действительно ли это реализация шаблона. Возможно, в реализации допущена ошибка. Расстояние между графами может пригодиться, если нужно найти среди множества конструкций наиболее похожую заданному шаблону. Решение о том, является ли конструкция реализацией шаблона, в этом случае — отдельная задача.

Точный алгоритм не используется из-за следующих недостатков:

а) задача является NP-полной;

б) реализация шаблона может отличаться от его описания.

Экспоненциальное время работы может быть неприемлемо. Описание же шаблона должно быть таким, чтобы его можно было найти в любой его реализации. Это не может быть недостатком.

1.2.2 Метод поиска шаблонов проектирования на основе поиска по образцу

Алгоритм описан в статье «Design Pattern Detection by Template Matching» [5]. В качестве модели используется набор из восьми ориентированных графов, представленных в виде матриц смежности:

а) ассоциаций — определяет ассоциации между классами;
б) обобщений — определяет связи между базовым и производными классами;
в) абстрактных классов — определяет, какие классы являются абстрактными;
г) различные виды вызовов методов.

Все матрицы смежности объединяются в одну с помощью произведения. Предварительной каждый элемент матрицы умножается на простое число, соответствующее этой матрице. В итоге задача сводится к вычислению взаимнокорреляционной функции. Матрицы при этом преобразуются в вектора. Схожесть с шаблоном определяется косинусом угла между векторами, считаемым специальным образом.

Этот подход более точный, чем описанный ранее за счет поиска шаблона как композиции элементов, а не поиска отдельных составляющих. Как и в первом случае метод применим для определения похожести, но не для точного определения структуры системы, как композицию конструкций.

1.3 Анализ существующих алгоритмов поиска изоморфных подграфов

Задача поиска изоморфного подграфа является **NP**-полной. Точное решение задачи в общем случае занимает экспоненциальное время. Можно решить задачу за полиномиальное время, но с некоторой точность. Рассмотрим соответствующие алгоритмы.

1.3.1 Алгоритм поиска изоморфного подграфа на основе метода поиска в глубину

Описан в статье «An Algorithm for Subgraph Isomorphism» [6]. Алгоритм применим к неориентированным графам. Находит точное решение и основан на поиске в глубину. Графы представляются в виде матриц смежности вершин.

Изначально вершина целевого графа считается эквивалентной вершине шаблона, если ее степень не менее степени шаблона. Применяется процедура «отсеивания» лишних пар эквивалентности: если две вершины цели связаны и каждая из них имеет эквивалентную вершину в шаблоне, то эти две вершины шаблона должны быть связаны, иначе считается, что первая вершина цели не имеет эквивалентных в шаблоне. Формально условие представлено формулой 1.1.

$$(\forall x \in [1, p_\alpha])((a_{ix} = 1) \Rightarrow (\exists y \in [1, p_\beta])(m_{xy} \cdot b_{yj} = 1)) \tag{1.1}$$

— p_α — количество вершин цели;

— p_β — количество вершин шаблона;

— $A = (a_{ij} \in \{0, 1\})$ — матрица смежности вершин цели;

— $B = (b_{ij} \in \{0, 1\})$ — матрица смежности вершин шаблона;

— $M = (m_{xy} \in \{0, 1\})$ — матрица эквивалентностей: $m_{ij} = 1$, если вершина цели x эквивалентна вершине шаблона y.

Точное решение единственное достоинство этого алгоритма.

К недостакам можно отмести следующее:

а) экспоненциальная сложность;

б) применим только к неориентированным графам;

в) не учитывает структуру вершины.

Последние два недостатка делают алгоритм неприменимым для поставленной задачи.

1.3.2 Алгоритм поиска изоморфного подграфа на основе меры схожести вершин

Описан в статье «A Measure Of Similarity Between Graph Vertices. With Applications To Synonym Extraction And Web Searching» [4]. Как было показано ранее, алгоритм приводит к неточному результату. В рамках поставленной задачи такой подход считается неприемлемым.

1.4 Метод поиска шаблонов проектирования в объектно-ориентированных программах на основе поиска изоморфных подграфов

Множество ориентированных графов — основной вариант представления структуры объектно-ориентированной программы. Для программы граф можно построить сразу. Задавать шаблон же в виде графа неудобно. К тому же структура графов может меняться. Нужно представление, которое можно построить для программы, с помощью которого можно описать шаблон, и для которого можно построить необходимую структуру графов.

UML-диаграммы классов — один из вариантов представления структуры объектно-ориентированной системы. С помошью него можно описать следующие сущности:

— классы, интерфейсы, перечисления, примитивные типы данных;
— свойства и методы классов;
— связи между классами: обобщения, зависимости, ассоциации.

Шаблон проектирования также может быть описан в виде программы, специальным формальным языком или форматом. Такой язык или формат должен учитывать возможность неопределенности значений некоторых атрибутов элементов UML-диаграммы классов или допустимого множества. Это необходимо, чтобы описание шаблона было более общим.

UML-диаграммы классов довольно подробно могут описывать систему. Для описания шаблонов проектирования используется только часть элементов. Назовем такое подмножество моделью объектно-ориентированной системы.

Итак можно выделить следующие этапы метода:

а) построение модели шаблона;
б) построение модели программы;
в) построение графа модели шаблона;
г) построение графа модели программы;
д) поиск изоморфного подграфа.

Схема метода представлена на рисунке 1.3.

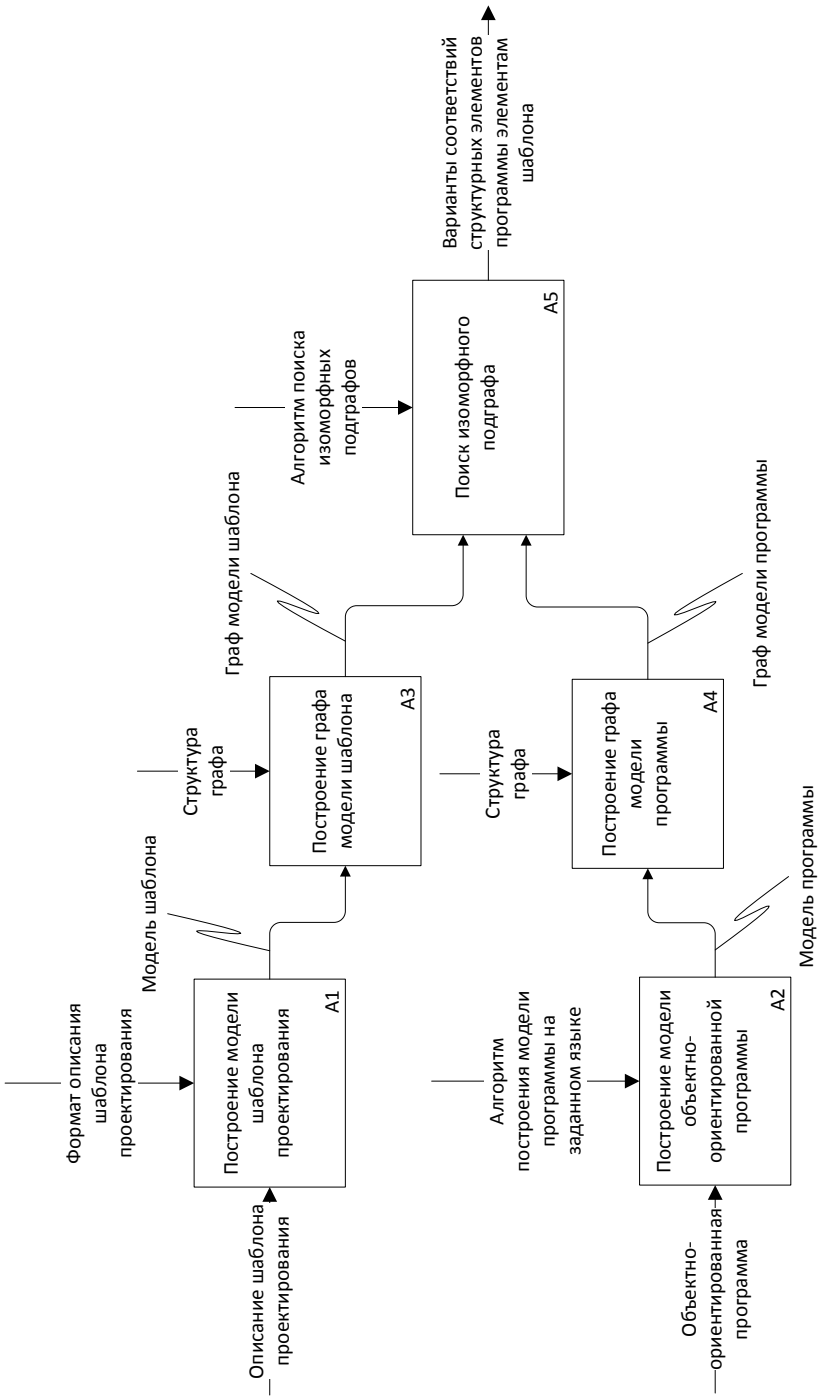

Рисунок 1.3 — Метод поиска шаблонов проектирования в объектно-ориентированных программах на основе поиска изоморфных подграфов

2 Конструкторский раздел

2.1 Модель объектно-ориентированной системы

Для поиска шаблонов проектирования нужно привести программу и шаблон к одному представлению. За основу можно взять UML-диаграммы классов. В них не хватает явных определений некоторых необходимых связей, например, вызовов методов. Такие связи, обычно, указываются в комментариях к элементам диаграммы.

В UML-диаграммах довольно много элементов, при описании шаблонов проектирования используется только часть. Убрав все лишнее и добавив нужные связи получим структуру модели представленную на рисунке 2.1.

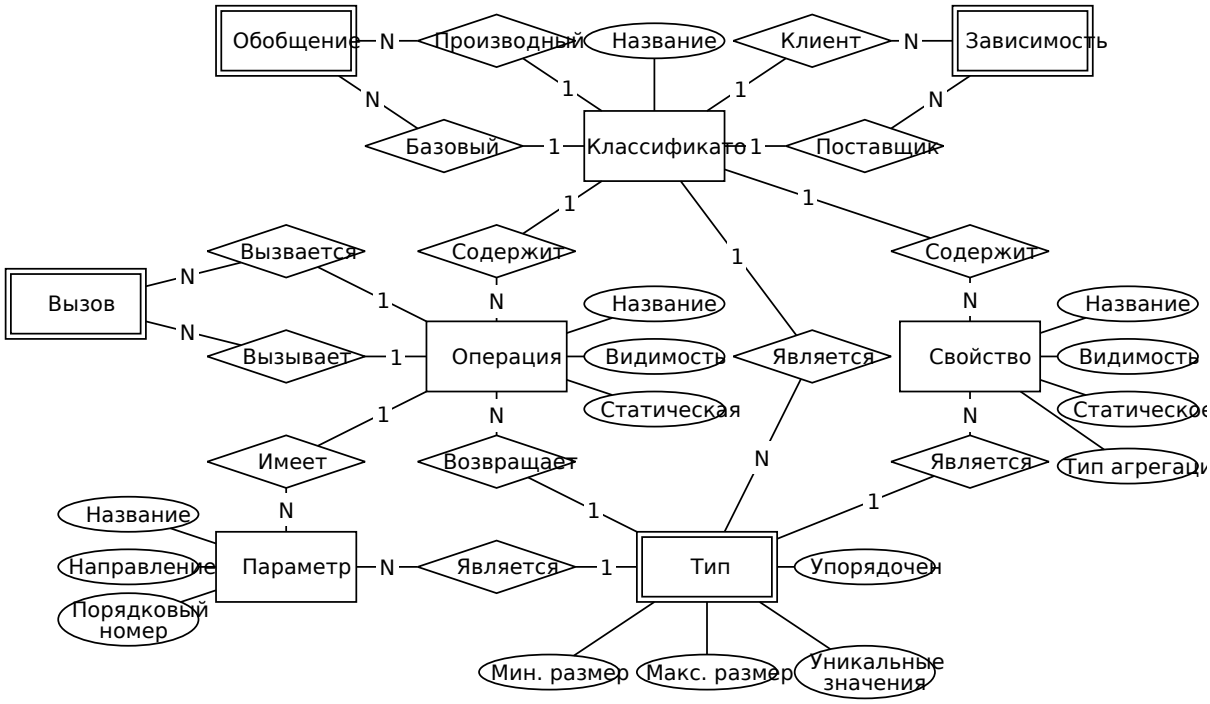

Рисунок 2.1 — Структура модели объектно-ориентированной системы

2.2 Граф со множеством типов дуг

Поиск шаблонов проектирования можно выполнять, выполняя поиск изоморфного подграфа в графе. Типы вершин можно не различать, если считать, что в графе не может существовать двух эквивалентных вершин. Описанная модель содержит множество типов связей. Может существовать несколько типов связей между вершинами. Введем множество типов дуг, обозначив каждый тип меткой.

Определение. $G = (V, L, E)$ — граф с множеством типов дуг, где

— $V = \{v : v = (object)\}$ — множество вершин;

— $object$ — значение вершины;

— $L = \{l\}$ — множество меток дуг;

— $E = \{e = (u, v, l) : u, v \in V \wedge l \in L \wedge (\forall l' \in L, \forall u', v' \in V)(|\{(u', v', l')\}| \leq 1))\}$ — множество дуг.

Структура графа представлена на рисунке 2.2.

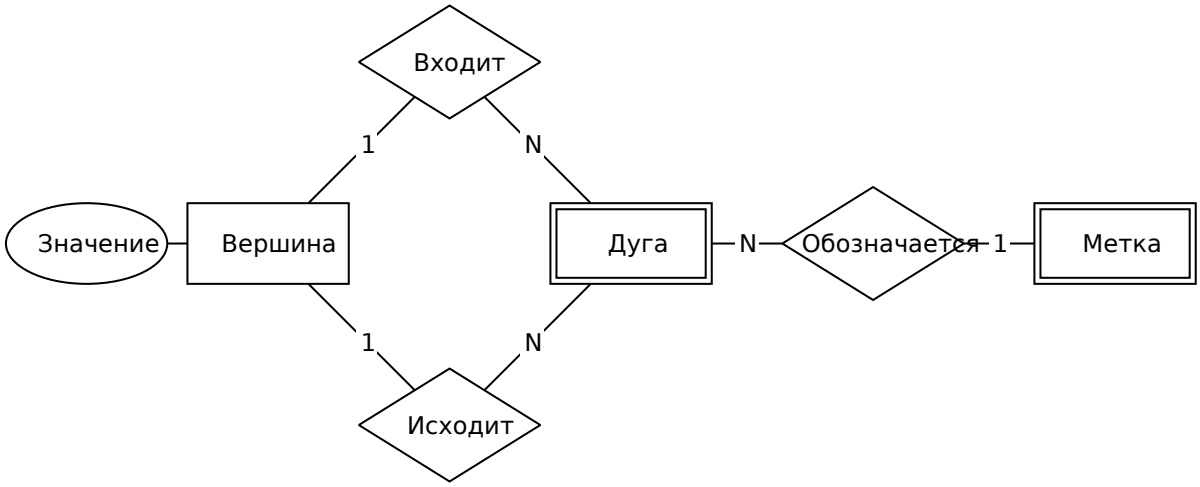

Рисунок 2.2 — Структура графа с множеством типов дуг

2.3 Алгоритм поиска изоморфных подграфов с множеством типов дуг

Рассмотренный ранее алгоритм Ульмана для поиска изоморфных подграфов, не подходит для ориентированных графов с множеством типов дуг. Нужен новый алгоритм. За основу можно взять метод поиска в глубину, условие эквивалентности вершин и процедуру «отсеивания» лишних эквивалентностей.

Общая идея алгоритма заключается в том, что нужно для каждой вершины целевого графа найти эквивалентные вершины в шаблоне, а затем переходя по дугам, без учета направлений, по одной вершине добавлять к возможному изоморфному подграфу. На каждом шаге нужно проверять корректность решения. Таким образом, у алгоритма есть состояние на каждом шаге — конфигурация. Каждая конфигурация может породить множество конфигураций. Конфигурация является конечной, если больше не осталось вершин шаблона, которым можно сопоставить вершину целевого графа. Результат алгоритма — все возможные варианты множеств пар изоморфных вершин, в каждом множестве представлены все вершины шаблона.

Для начала введем понятие эквивалентности вершин. Обозначим некоторые величины:

— $target$ — целевой граф;

— $pattern$ — граф шаблона;

— $incoming^l_G$ — множество входящих дуг с меткой l вершины графа G;

— $outgoing^l_G$ — множество исходящих дуг с меткой l вершины графа G;

Определение. $isEquivalent : Object \times Object \to \{0, 1\}$ — функция эквивалентности значений вершин.

$$object_{G_1} \cong object_{G_2} \Leftrightarrow isEquivalent(object_{G_1}, object_{G_2})$$

Определение. Вершины эквивалентны ($target \cong pattern$), если:

а) $(\forall l \in L_{pattern})(|incoming^l_{target}| \geq |incoming^l_{pattern}|)$;

б) $(\forall l \in L_{pattern})(|outgoing^l_{target}| \geq |outgoing^l_{pattern}|)$;

в) $object_{target} \cong object_{pattern}$.

Последовательность конфигураций от начальной до конечной приводит к получению очередного множества пар эквивалентных вершин, и имеет определенную структуру.

Определение. $C = (selected, current, checked)$ — конфигурация алгоритма поиска подграфов с множеством типов дуг, где

— $selected = \{e^n = (v^n_{target}, v^n_{pattern})\}^N_{n=1}$ — последовательность пар вершин целевого графа и шаблона, которые прошли проверку или требуют проверки;

— $current \in \mathbb{N}$ — номер элемента из $selected$;

— $checked = \{(v_{target}, v_{pattern})\}$ — множество проверенных пар вершин, гарантируется, что построенные из вершин этого множества графы изоморфны.

Определение. Конфигурация является начальной, если:

а) $selected = \{e^n\}^N_{n=1}, N \geq 1$;

б) $current = 1$;

в) $checked = \{e^1\}$.

Определение. Конфигурация является конечной, если $current > |selected|$.

Определим функции, использующиеся в алгоритме.

Определение. NEIGHBORS : $V \to 2^V$ — множество соседей вершины графа:

$$\text{NEIGHBORS}(v_G) = \{u : (u \neq v_G) \wedge ((\exists(u, v_G, l_G^i) \in E_G) | (\exists(v_G, v, l_G^j) \in E_G))\}$$

Определение. CHAINS : $2^V \times 2^V \to 2^{Chain}$ — множество последовательностей пар вершин, изоморфизм которых нужно поверить:

$$Chain = \{(v_t^n, v_p^n) : (v_t^n \cong v_p^n) \wedge ((\forall i, j)(v_t^i \neq v_t^j \wedge v_p^i \neq v_p^j))\}_{n=1}^N$$

$$\text{CHAINS}(V_t, V_p) = \{Chain = \{(v_t^n, v_p^n)\}_{n=1}^N : (v_t^n \in V_t) \wedge (v_p^n \in V_p)\}$$

Определение. CHECKED_TARGETS : $2^C \to 2^V$ — вершины целевого графа из множества проверенных пар конфигурации:

$$\text{CHECKED\_TARGETS}(C) = \{v_{target} : (v_{target}, v_{pattern}) \in checked_C\}$$

Определение. CHECKED_PATTERNS : $2^C \to 2^V$ — вершины шаблона из множества проверенных пар конфигурации:

$$\text{CHECKED\_PATTERNS}(C) = \{v_{pattern} : (v_{target}, v_{pattern}) \in checked_C\}$$

Определение. IS_VALID : $2^C \to \{0, 1\}$ — может ли конфигурация привести к изоморфизму:

$$\text{IS\_VALID}(C) = T_{outgoing} \subseteq P_{outgoing} \wedge T_{incoming} \subseteq P_{incoming}$$

— $(target, pattern) = selected_C^{current}$

— $(V_{target}, E_{target}) = G_{target}, target \in V_{target}$ — граф, в котором находится вершина $target$;

— $(V_{pattern}, E_{pattern}) = G_{pattern}, pattern \in V_{pattern}$ — граф, в котором находится вершина $pattern$;

— $T_{outgoing} = \{c : (target, v, c) \in E_{target} \wedge v \in \text{CHECKED\_TARGETS}(C)\}$;

— $T_{incoming} = \{c : (v, target, c) \in E_{target} \wedge v \in \text{CHECKED\_TARGETS}(C)\}$;

— $P_{outgoing} = \{c : (pattern, v, c) \in E_{pattern} \wedge v \in \text{CHECKED\_PATTERNS}(C)\}$;

— $P_{incoming} = \{c : (v, pattern, c) \in E_{pattern} \wedge v \in \text{CHECKED\_PATTERNS}(C)\}$.

Определение. ADD — процедура, добавляющая конфигурации в коллекцию конфигураций. Реализация зависит от типа коллекции.

Определение. TAKE — извлекает из коллекции конфигураций очередную конфигурацию. Реализация зависит от типа коллекции.

Коллекция конфигураций может быть стеком, очередью или другой структурой данных. Функции ADD и TAKE соответственно могут добавлять в конец стека или извлекать, добавлять в конец очереди, извлекать с начала очереди. Можно использовать асинхронные структуры данных и ленивые вычисления, что может быть более эффективно, если нужно получить первые N изоморфизмов. Также можно использовать приоритеты, например, мощность множества *checked*.

Определение. Алгоритм генерации начальных конфигураций:

Входные данные:

— V_t — множество вершин целевого графа;

— V_p — множество вершин шаблона.

Результат: $\{C\}$ — множество начальных конфигураций.

function INITIAL_CONFIGURATIONS(V_t, V_p)
  $C \leftarrow \varnothing$
  $p \leftarrow v : |\text{NEIGHBORS}(v)| = \min\limits_{w \in V_p}\{|\text{NEIGHBORS}(w)|\}$
  for all $(v_t, p) : (v_t \cong p) \wedge (v_t \in V_t)$ **do**
    for all $chain \in$ CHAINS(NEIGHBORS(v_t), NEIGHBORS(v_p)) **do**
      $C \leftarrow C \cup \{$ MAKE_CONFIGURATION(v_t, v_p, $chain$) $\}$
    end for
  end for
  return C
end function

Определение. Алгоритм «продвижения» конфигурации.

Входные данные: C — не конечная конфигурация.

Результат: C — конечная конфигурация или конфигурация с новым элементом в множестве *checked*.

function ADVANCE(C)
  $(selected, current, checked) \leftarrow C$
  $checked\_t \leftarrow$ CHECKED_TARGETS(C)
  $checked\_p \leftarrow$ CHECKED_PATTERNS(C)
  while $true$ **do**
    $current \leftarrow current + 1$
    $C_{next} \leftarrow (selected, current, checked)$

if $current > |selected|$ **then**

return C_{next}

end if

$(target, pattern) \leftarrow selected^{current}$

if $target \notin checked\_t \wedge pattern \notin checked\_p \wedge$ IS_VALID(C_{next}) **then**

return $(selected, current, checked \cup \{selected^{current}\})$

end if

end while

end function

Определение. Алгоритм построения производных конфигураций.

Входные данные: C — неконечная конфигурация.

Результат: $\{C\}$ — множество производных конфигураций.

function DERIVED_CONFIGURATIONS(C)

$result \leftarrow \varnothing$

$(selected, current, checked) \leftarrow C$

$(v_t, v_p) \leftarrow selected^{current}$

for all $chain \in$ CHAINS(NEIGHBORS(v_t), NEIGHBORS(v_p)) **do**

$addition \leftarrow chain \setminus selected$

if $addition \neq \varnothing$ **then**

$result \leftarrow result \cup \{(selected \cdot addition, current, checked)\}$

end if

end for

if $result = \varnothing$ **then**

$result \leftarrow \{C\}$

end if

return $result$

end function

Определение. Алгоритм поиска изоморфного подграфа в графе с одной компонентой связности.

Входные данные:

— G_{target} — целевой граф;

— $G_{pattern}$ — граф шаблона.

Результат: $\{\{(v_{target}, v_{pattern})\}\}$ — множество изоморфизмов.

function MATCH_ONE(G_{target}, $G_{pattern}$)

$isomorphisms \leftarrow \varnothing$

$Configurations \leftarrow$ INITIAL_CONFIGURATIONS(G_{target}, $G_{pattern}$)

while $Configurations \neq \varnothing$ **do**

$Configurations, C \leftarrow$ TAKE($Configurations$)

$C \leftarrow$ ADVANCE(C)

if C — конечная конфигурация **then**

if CHECKED_PATTERNS(C) $= V_{pattern}$ **then**

$isomorphisms \leftarrow isomorphisms \cup \{checked_C\}$

end if

else

$Configurations \leftarrow$ ADD($Configurations$, DERIVED_CONFIGURATIONS(C)

)

end if

end while

return $isomorphisms$

end function

Конфигурации не связаны, поэтому их проверка может выполняться параллельно. В этом алгоритме тело цикла **while** может выполняться параллельно, для этого нужно обеспечить синхронизацию коллекции $Configurations$.

В общем случае графы состоят из нескольких компонент связности. Чтобы выполнить поиск таких графов, можно выполнить поиск для отдельных компонент, а затем объединить результаты.

В этой работе будем считать, что шаблон должен содержать только одну компоненту связности.

Определение. COMPONENTS : $G \to 2^G$ — множество компонент связности графа:

COMPONENTS$(G) = \{(V, L, E) : V \subseteq V_G \wedge L \subseteq L_G \wedge E \subseteq E_G \wedge (\forall u, v \in V)(\exists\{w_1 = u, \ldots, w_n, \ldots, w_N = v : (\forall w_n, n = \overline{2,N})(w_{n-1} \in \text{NEIGHBORS}(w_n))\})$

Определение. Алгоритм поиска изоморфного подграфа в графе с множеством компонент связности.

Входные данные:

— G_{target} — целевой граф;

— $G_{pattern}$ — граф шаблона.

Результат: $\{\{(v_{target}, v_{pattern})\}\}$ — множество изоморфизмов.

function MATCH_GRAPH(G_{target}, $G_{pattern}$)

$isomorphisms \leftarrow \varnothing$

for all $component \in$ COMPONENTS(G_{target}) **do**

$isomorphisms \leftarrow isomorphisms \cup$ MATCH_GRAPH(component, $G_{pattern}$)

end for

return $isomorphisms$

end function

Описанный алгоритм не имеет доказательства корректности. Вместо этого его результат проверяется на корректность. Проверяются случаи, когда найденный изоморфизм ложный. Основная идея состоит в том, что нужно проверить, действительно ли вершины целевого графа соответствуют вершинам шаблона.

Нужно проверить, что:

а) связи вершины шаблона соответствуют связям цели;

б) вершины, связанные с вершиной шаблона, изоморфны вершинам цели.

Определение. Условие корректности изоморфизма I подграфа графа $G_t = (V_t, L_t, E_t)$ и графа $G_p = (V_p, L_p, E_p)$, где $I = \{(t, p) : t \in V_t \wedge p \in V_p\}$:

$$(\forall (t,p) \in I)$$

$$((\forall (p, p_x, l) \in E_p)(\exists (t, t_y, l) \in E_t : (t_y, p_x) \in I) \wedge (\forall (p_a, p, l) \in E_p)(\exists (t_b, t, l) \in E_t : (t_b, p_a) \in I))$$

2.4 Граф модели объектно-ориентированной программы

На основе модели нужно построить граф, где разные типы связей будут разными типами дуг. Структура графа представлена на рисунке 2.3.

Введём обозначения:

— $V_{Classifier} = \{v_{Classifier}\}$ — множество вершин, которые являются объектами класса **Classifier**;

— $V_{Type} = \{v_{Type}\}$ — множество вершин, которые являются объектами класса **Type**;

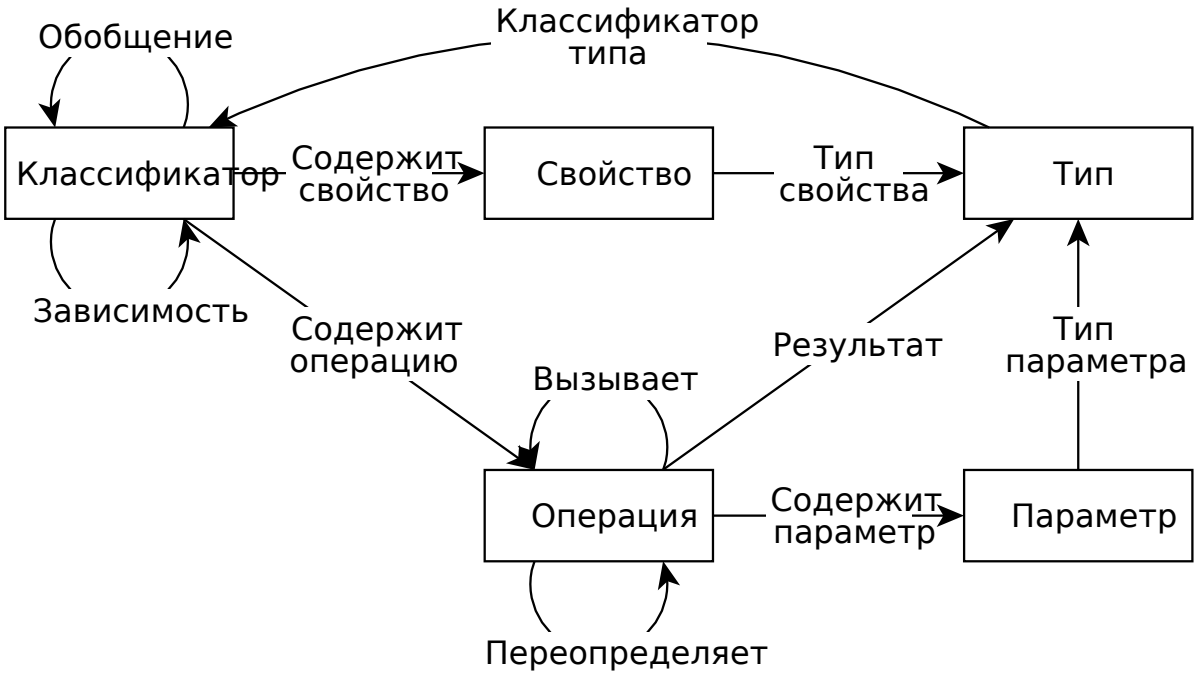

Рисунок 2.3 — Граф модели объектно-ориентированной программы

— $V_{Property} = \{v_{Property}\}$ — множество вершин, которые являются объектами класса **Property**;

— $V_{Operation} = \{v_{Operation}\}$ — множество вершин, которые являются объектами класса **Operation**;

— $V_{Parameter} = \{v_{Parameter}\}$ — множество вершин, которые являются объектами класса **Parameter**.

Определение. $Generalization$ — метка дуги для связи «обобщение»

$$e_{Generalization} = (v_{derived}, v_{general}, Generalization) : v_{derived}, v_{general} \in V_{Classifier}$$

— $v_{general}$ — базовый класс;

— $v_{derived}$ — производный класс от $v_{general}$ класс.

Определение. $Dependency$ — метка дуги для определения зависимости между классами

$$e_{Dependency} = (v_{client}, v_{supplier}, Dependency) : v_{client}, v_{supplier} \in V_{Classifier}$$

— $v_{supplier}$ — используемый класс;

— v_{client} — использующий $v_{supplier}$ класс.

Определение. $HasProperty$ — метка дуги для связи классов и их свойств

$$e_{HasProperty} = (v_{Classifier}, v_{Property}, HasProperty)$$

Определение. $HasOperation$ — метка дуги для связи классов и их операций

$$e_{HasOperation} = (v_{Classifier}, v_{Operation}, HasOperation)$$

Определение. $PropertyType$ — метка дуги для связи свойства и его типа

$$e_{PropertyType} = (v_{Property}, v_{Type}, PropertyType)$$

Определение. $OperationResult$ — метка дуги для связи операции и возвращаемого типа

$$e_{OperationResult} = (v_{Operation}, v_{Type}, OperationResult)$$

Определение. $TypeClassifier$ — метка дуги для связи типа и его класса

$$e_{TypeClassifier} = (v_{Type}, v_{Classifier}, TypeClassifier)$$

Определение. $Invocation$ — метка дуги для связи вызываемой и вызывающей операции

$$e_{Invocation} = (v_{invoker}, v_{invoked}, Invocation)$$

— $v_{invoked}$ — вызываемая операция;

— $v_{invoker}$ — вызывающая $v_{invoked}$ операция.

Определение. $Overriding$ — метка дуги для связи переопределяемой и переопределяющей операции в базовом и производном классах

$$e_{Overriding} = (v_{override}, v_{overridden}, Overriding)$$

— $v_{overridden}$ — переопределяемая операция;

— $v_{override}$ — преопределяющая $v_{overridden}$ операция;

Определение. $HasParameter$ — метка дуги для связи операции и ё параметров

$$e_{HasParameter} = (v_{Operation}, v_{Parameter}, HasParameter)$$

Определение. $ParmeterType$ — метка дуги для связи параметра и его типа

$$e_{ParmeterType} = (v_{Parameter}, v_{Type}, ParmeterType)$$

Определение. $isEquivalent : Classifier \times Classifier \to \{0, 1\}$ — функция эквивалентности классов

$$isEquivalent(target, pattern) = 1$$

Определение. $isEquivalent : Type \times Type \to \{0, 1\}$ — функция эквивалентности типов

$$isEquivalent(target, pattern) = (lower_{target}, upper_{target}) \cong (lower_{pattern}, upper_{pattern}) \\ \wedge isUnique_{target} \cong isUnique_{pattern} \wedge isOrdered_{target} \cong isOrdered_{pattern}$$

Определение. $isEquivalent : Property \times Property \to \{0, 1\}$ — функция эквивалентности свойств

$$isEquivalent(target, pattern) = visibility_{target} \cong visibility_{pattern} \\ \wedge aggregation_{target} \cong aggregation_{pattern} \wedge isStatic_{target} \cong isStatic_{pattern}$$

Определение. $isEquivalent : Operation \times Operation \to \{0, 1\}$ — функция эквивалентности операций

$$isEquivalent(target, pattern) = visibility_{target} \cong visibility_{pattern} \\ \wedge isStatic_{target} \cong isStatic_{pattern}$$

Определение. $isEquivalent : Parameter \times Parameter \to \{0, 1\}$ — функция эквивалентности параметров

$$isEquivalent(target, pattern) = direction_{target} \cong direction_{pattern} \\ \wedge position_{target} \cong position_{pattern}$$

Для любых других пар объектов $isEquivalent(target, pattern) = 0$.

Значения атрибутов могут принимать значение $\varnothing$, если их значение не нужно определять или оно неизвестно.

Определение. Значения атрибутов эквивалентны ($x_{target} \cong x_{pattern}$), если выполняется одно из условий:

- $x_{pattern} = \varnothing$
- $x_{target} = x_{pattern}$

На основе всех определений вершин и дуг, определим граф.

Определение. $G = (V, L, E)$ — граф модели объектно-ориентированной программы, где:

- $V = V_{Classifier} \cup V_{Type} \cup V_{Property} \cup V_{Operation} \cup V_{Parameter}$ — множество вершин

— $L = \{Generalization,\ Dependency,\ HasProperty,\ HasOperation,\ PropertyType,\ OperationResult,\ TypeClassifier,\ Invocation,\ Overriding,\ HasParameter,\ ParameterType\}$ — множество вершин

— $E = \{(u, v, l) : u, v \in V \wedge l \in L\}$ — множество дуг

2.5 Алгоритм поиска шаблона в модели объектно-ориентированной программы

Основная идея алгоритма в том, что нужно построить графы для модели программы и шаблона и выполнить поиск изоморфных шаблону подграфов модели. Определим необходимые функции и используемые алгоритмы.

Определение. INDIR_GENERALS : $Classifier \to 2^{Classifier}$ — множество всех не напрямую базовых классов.

$$\text{INDIR\_GENERALS}(c) = \begin{cases} \varnothing \text{ , если } c.generals = \varnothing \\ \{x : (\exists g \in c.generals)(x \in \text{INDIR\_GENERALS}(g))\} \end{cases}$$

Определение. DOVERRID : $Operation \times 2^{Classifier} \to 2^{Operation}$ — множество переопределяемых операций в заданных классах

$$\text{DOVERRID}(op, classifiers) = \{x : x = op \wedge (\exists g \in classifiers)(x \in g.operations)\}$$

Определение. IOVERRID : $Operation \times 2^{Classifier} \to 2^{Operation}$ — множество переопределяемых операций в заданных классах и их базовых классах

$$\text{IOVERRID}(op, C) = \begin{cases} \varnothing \text{ , если } C = \varnothing \\ \text{DOVERRID}(op, C) \text{ , если DOVERRID}(op, C) \neq \varnothing \\ \text{IOVERRID}(op, \{s : (\exists c \in C)(s \in c.generals)\}) \end{cases}$$

Определение. OVERRIDDEN : $Operation \to 2^{Operation}$ — множество переопределяемых операций

$$\text{OVERRIDDEN}(operation) = \text{IOVERRID}(operation, operation.owner.generals)$$

Определение. Алгоритм построения графа модели объектно-ориентированной программы.

Входные данные: $model$ — модель программы;

Результат: G — граф модели программы.

function MODEL_GRAPH($model$)

$E \leftarrow \varnothing$

for all $classifier \in model.classifiers$ **do**

for all $general \in \text{INDIR\_GENERALS}(classifier)$ **do**

$E \leftarrow E \cup \{(classifier, general, Generalization)\}$

end for

for all $supplier \in classifier.suppliers$ **do**

$E \leftarrow E \cup \{(classifier, supplier, Dependency)\}$

end for

for all $property \in classifier.properties$ **do**

$E \leftarrow E \cup \{(classifier, property, HasProperty)\}$

$E \leftarrow E \cup \{(property, property.type, PropertyType)\}$

$E \leftarrow E \cup \{(property.type, property.type.classifier, TypeClassifier)\}$

end for

for all $operation \in classifier.operations$ **do**

$E \leftarrow E \cup \{(classifier, operation, HasOperation)\}$

if $operation.result \neq \varnothing$ **then**

$E \leftarrow E \cup \{(classifier, operation.result, Dependency)\}$

$E \leftarrow E \cup \{(operation, operation.result, OperationResult)\}$

$E \leftarrow E \cup \{(operation.result, operation.result.classifier, TypeClassifier)\}$

end if

for all $invoked \in operation.invocations$ **do**

$E \leftarrow E \cup \{(operation, invoked, Invocation)\}$

end for

$overidden \leftarrow \text{OVERRIDDEN}(operation)$

if $overidden \neq \varnothing$ **then**

$E \leftarrow E \cup \{(operation, x, Overriding) : \{x\} \subseteq overridden\}$

end if

for all $parameter \in operation.parameters$ **do**

$E \leftarrow E \cup \{(classifier, parameter.type.classifier, Dependency)\}$

$E \leftarrow E \cup \{(operation, parameter, HasParameters)\}$

$E \leftarrow E \cup \{(parameter, parameter.type, ParameterType)\}$

$E \leftarrow E \cup \{(parameter.type, parameter.type.classifier, TypeClassifier)\}$

end for

end for

end for

$V \leftarrow \varnothing$

$L \leftarrow \varnothing$

for all $(u, v, l) \in E$ **do**

$V \leftarrow V \cup \{u, v\}$

$L \leftarrow L \cup \{l\}$

end for

return (V, L, E)

end function

Определение. Алгоритм поиска шаблона в модели объектно-ориентированной программы.

Входные данные:

— $model_{target}$ — модель программы;

— $model_{pattern}$ — модель шаблона.

Результат: $\{\{(object_{target}, object_{pattern})\}\}$ — множество вариантов соответствий объектов модели программы шаблону.

function MATCH_MODEL($model_{target}$, $model_{pattern}$)

$G_{target} \leftarrow$ MODEL_GRAPH($model_{target}$)

$G_{pattern} \leftarrow$ MODEL_GRAPH($model_{pattern}$)

return MATCH_GRAPH(component, $G_{pattern}$)

end function

3 Технологический раздел

3.1 Формат описания модели объектно-ориентированной программы

Основой является формат YAML [7]. Формат был выбран благодаря следующим достоинствам:

— текстовое представление;

— поддержка рекурсивных структур и ссылок;

— поддержка пользовательских типов данных.

Корневой элемент модели помечается тегом `!Model` и является списком элементов модели:

```
!Model:
- <Classifier>
```

`Classifier` — любой из элементов, помеченный тегом `!Class`, `!Classifier`, `!Interface`, `!PrimitiveType`, который имеет структуру:

```
!Classifier
name: <str>
properties:
  - <Property>
operations:
  - <Operation>
generals:
  - <Classifier>
suppliers:
  - <Classifier>
```

Каждый из атрибутов является опциональным. Если не задано имя (`name`), то элементу будет задано уникальное имя вида `anonymous_<число>`. Выполняется для такого же атрибута других типов элементов.

`Property` — элемент, помеченный тегом `!Property` со следующей структурой:

```
!Property
name: <str>
```

```
type: <Type>
visibility: <Visibility>
aggregation: <Aggregation>
is_static: <bool>
owner: <Classifier>
```

`Operation` — элемент, помеченный тегом `!Operation` со следующей структурой:

```
!Operation
name: <str>
result: <Type>
visibility: <Visibility>
parameters:
  - <Parameter>
is_static: <bool>
invocations:
  - <Operation>
owner: <Classifier>
```

`Type` — элемент, помеченный тегом `!Type` со следующей структурой:

```
!Operation
classifier: <Classifier>
lower: <int>
upper: <int>
is_ordered: <bool>
is_unique: <bool>
```

`Parameter` — элемент, помеченный тегом `!Parameter` со следующей структурой:

```
!Parameter
type: <Type>
direction: <Direction>
position: <int>
owner: <Operation>
```

`Visibility` — элемент, помеченный тегом `!Visibility`, имеющий одно из допустимых значений:

— `'public'`

— `'protected'`

— `'private'`

`Aggregation` — элемент, помеченный тегом `!Aggregation`, имеющий одно из допустимых значений:

— `'none'`

— `'shared'`

— `'composite'`

`Direction` — элемент, помеченный тегом `!Direction`, имеющий одно из допустимых значений:

— `'in'`

— `'out'`

— `'inout'`

YAML предоставляет возможность ссылаться на другие элементы по адресу. Каждый элемент должен быть описан один раз. Например, если нужно описать связь наследования между двумя классами, можно сделать следующим образом:

```
!Model
- &id001 !Class
  name: Base
- !Class
  name: Derived
  generals:
  - *id001
```

`&id001` означает, что классу с названием `Base` присваивается такой адрес. В перечислении базовых классов в классе с названием `Derived` указывается ссылка `*id001` на `Base`.

3.2 Структура программного комплекса

Для реализации разработанного метода разработан программный комплекс, который позволяет выполнять поиск шаблонов проектирования в программах, написанных на языке программирования **Java**.

В комплекс входят следующие программы:

— **java_source_model** — программа для построения модели программы на языке Java;

— **java_bytecode_model** — программа для построения модели программы на основе байт-кода для виртуальной машины **Java**;

— **pattern_model** — программа для построения модели шаблона;

— **match_pattern** — программа для поиска шаблонов проектирования.

Основной сценарий использования программного комплекса:

а) построение модели программы с помощью **java_source_model** или **java_bytecode_model**;

б) построение модели шаблона из заранее заготовленных с помощью **pattern_model**;

в) запуск **match_pattern** с результатами первых двух шагов в качестве входных данных.

Модель программы или шаблона можно описать вручную.

Программы **java_source_model**, **pattern_model**, **match_pattern** написаны на языке **Python** версии 2.7. **java_bytecode_model** написана на **Java** версии 1.8.

Структура компонентов и зависимостей программного комплекса представлена на рисунке 3.1.

3.2.1 Программа для построения модели программы на языке Java

Выполняет синтаксический анализ исходного кода на языке **Java** с выводом полных имен типов. Программа может работать с одним или множество файлов формата **.java**. Для синтаксического анализа используется библиотека **plyj** [8]. Определение полных имен внешних типов выполняется с помошью **.class** или **.jar**-файлов. Для их анализа используется библиотека **javatools** [9]. Вывод модели в формате **.yaml** выполняется с помощью библиотеки **PyYAML** [10].

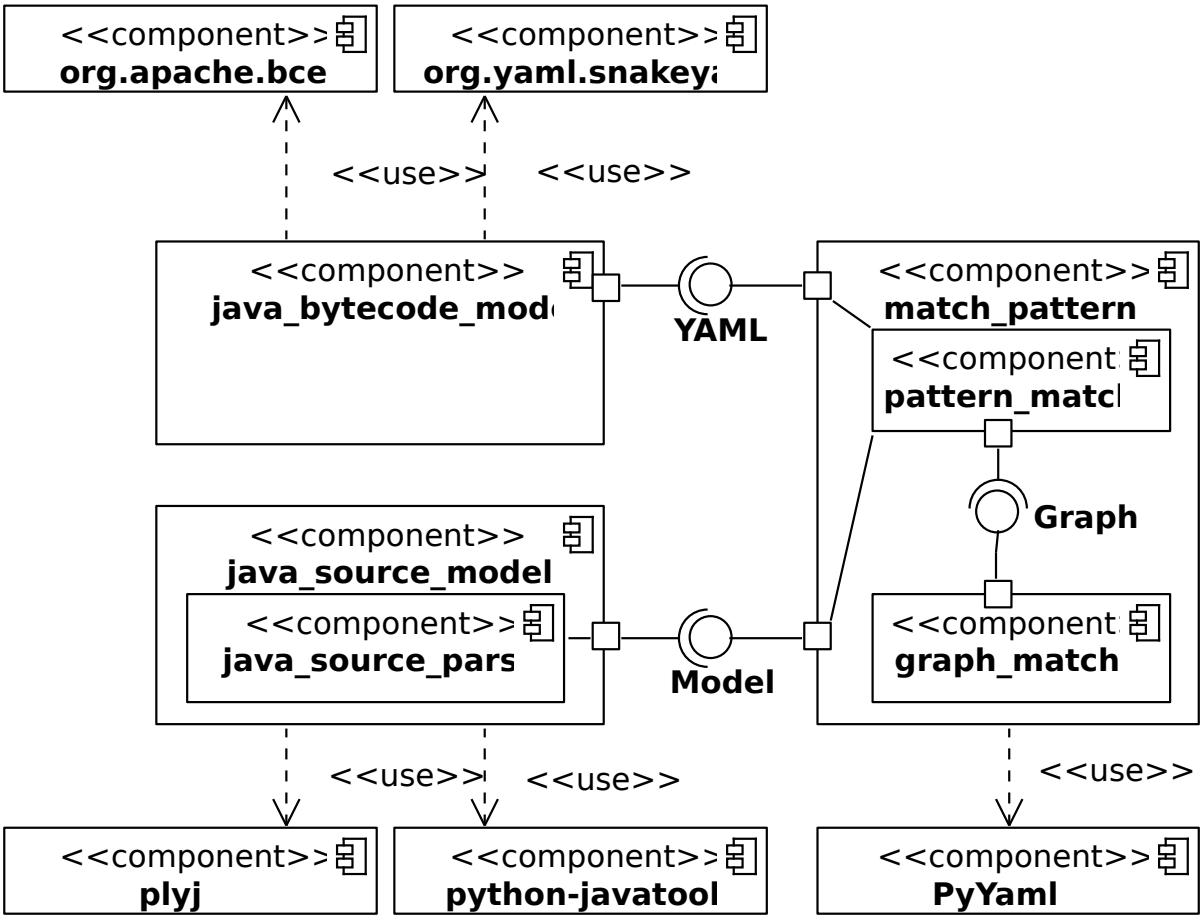

Рисунок 3.1 — Структура компонентов и зависимостей программного комплекса

Интерфейс запуска:

```
python java_source_model.py [-h] [-p <external_path>] <path> [<path> ...]
```

— `path` — путь к .java файлу или директории;

— `-h` или `--help` — выводит сообщение с описанием параметров запуска;

— `-p <external_path>` или `--path <external_path>` — задает путь `external_path` к **.class**, **.jar** файлам или директории;

Выводит модель в формате **.yaml** в **stdout**. Сообщения об ошибках выводятся в **stderr**.

Алгоритм работы программы:

а) рекурсивно найти все **.java**-файлы по всем заданным путям;

б) выполнить синтаксический анализ всех **.java**-файлов и построить деревья вывода;

в) во всех деревьях вывода установить полные имена классов в их определении.

г) создать пустые элементы модели: классы, интерфейсы и перечисления;

д) создать пустые элементы модели для внешних зависимостей;

е) во всех деревьях вывода установить полные имена типов в любых местах их использования;

ж) определить связи обобщения;

з) заполнить элементы модели свойствами, операциями и создать типы данных;

и) определить связи зависимости;

к) вывести модель в формате **.yaml**.

При анализе программ с множеством пакетов и вложенных классов может возникнуть ситуация конфликта имен. Допустим `class A` есть в `package x` и в `package y`. Это два разных класса. В исходном коде и дереве вывода скорее всего будет представлено короткое название для каждого класса. При построении модели недопустимо использовать эти короткие названия. Нужно вывести полное названия типа. Таким образом, в модель попадут классы `x.A` и `y.A`. В этом случае конфликта имен не будет.

Проблема имеет место не только в определениях типов, но и в их использовании. Например, есть `class x.A`, `class y.A` и в `class z.B` определен метод `A f()`. Нужно определить полное имя возвращаемого типа. Сюда же можно отнести определение типа поля класса, локальной переменной, параметра метода, базового класса. В данном случае нужно учитывать, импорт какого пакета был выполнен. Аналогичная ситуация с вложенными классами.

Другая проблема, которая не решалась в этой программе, заключается в определении зависимостей между вызываемым и вызывающим методов.

Программа не во всех случая определяет полные имена типов. В таких случая классы, интерфейсы и перечисления их представляющие игнорируются при построении модели. Здесь не будут описаны все ситуации, когда тип определяется, а когда нет. Разработка программы была приостановлена. Для дальнейшей разработки требовалось написание части компилятора **Java**. Реальной необходимости в этом не было поэтому взамен была написана программа на **Java**, которая содержит всю нужную функциональность. Она описана в следующем разделе.

3.2.2 Программа для построения модели программы на основе байт-кода для виртуальной машины Java

Анализирует байткод виртуальной машины **Java**. Для этого используется библиотека **Apache Commons BCEL** (The Byte Code Engineering Library) версии 6.0 [11]. Для сборки программы используется **Apache Maven** версии 3 [12].

Интерфейс запуска:

```
java -jar java_bytecode_model.jar <path> [<path> ...]
```

— `path` — путь к **.class**, **.jar**-файлу или директории;

Выводит модель в формате **.yaml** в **stdout**. Сообщения об ошибках выводятся в **stderr**.

Алгоритм работы программы:

а) рекурсивно и разобрать найти все **.jar** и **.class**-файлы по всем заданным путям, **.class**-файлы в **.jar**-файлах;

б) создать пустые элементы модели для классов, интерфейсов и перечислений;

в) определить связи — обобщения;

г) создать типы данных;

д) заполнить элементы модели свойствами и операциями;

е) определить связи — зависимости;

ж) определить связи — вызовы методов;

з) вывести модель в формате **.yaml**.

3.2.3 Программа для построения модели шаблона

Строит модель для одного из заготовленных шаблонов.

Интерфейс запуска:

```
python pattern_model.py [-h] <name>
```

— `name` — название шаблона проектирования;

— `-h` или `--help` — выводит сообщение с описанием параметров запуска.

Выводит модель в формате **.yaml** в **stdout**. Сообщения об ошибках выводятся в **stderr**.

Реализованы следующие шаблоны:

— **AbstractFactory** — абстрактная фабрика;

— **BaseDerived** — связка базового и производного классов;

— **Bridge** — мост;

— **ChainOfResponsibility** — цепочка ответственности;

— **Decorator** — декоратор;

— **Empty** — пустой, используется для тестирования;

— **Memento** — хранитель;

— **OverriddenMethodCall** — вызов метода базового класса, переопределеного в производном;

— **Visitor** — посетитель.

3.2.4 Программа для поиска шаблонов проектирования

Выполняет поиск изоморфизмов между моделями объектно-ориетированных систем. Для чтения модели используется библиотека **PyYAML**.

Интерфейс запуска:

```
python match_pattern.py [-h] [-l <limit>] [-v <level>] [-f <format>] \
                        [-d <dir>] <pattern> <target>
```

— `pattern` — путь к **.yaml**-файлу модели шаблона;

— `target` — путь к **.yaml**-файлу целевой модели;

— `-h` или `--help` — выводит сообщение с описанием параметров запуска;

— `-l <limit>` или `--limit <limit>` — задает максимальное количество изоморфизмов `limit`, которое нужно найти; в наибольшей;

— `-v <level>` или `--verbosity <level>` — включает вывод журнала в **stderr** с уровнем подробности `level`:

`DEBUG` — подробный отладочный вывод;

`INFO` — выводится основная информация;

— `-f <format>` или `--format <format>` — определяет формат вывода `format`:

`txt` — текст;

`dot` — граф в формате **DOT**, каждая вершина содержит название элемента шаблона и цели;

`dot.svg` — отображение в **SVG**-формате графа в формате **DOT**;

`yaml` — в формате **YAML** в виде списка словарей `{pattern: <pattern>, target: <target>}`

— `-d <dir>` или `--dir <dir>` — определяет путь к директории, куда будут записываться файлы с вариантами соответствий вместо вывода в **stdout**.

Выводит варианты соответствий элементов в **stdout**. Сообщения об ошибках выводятся в **stderr**.

3.2.4.1 Модуль graph_matcher

В этом модуле реализован алгоритм поиска изоморфного подграфа. Структура классов представлена на рисунке 3.2.

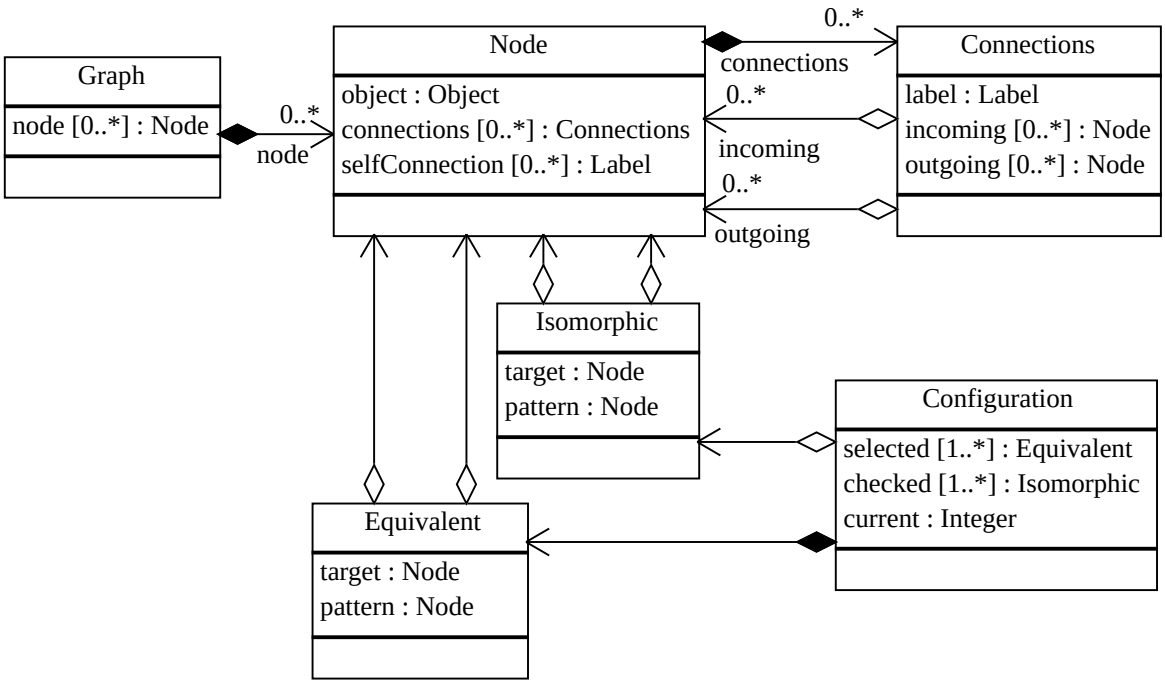

Рисунок 3.2 — Структура классов модуля graph_matcher

Graph — граф.

Атрибуты:

— **node** — множество вершин;

Node — вершина графа.

Атрибуты:

— **object** — значение вершины;

— **connections** — множество множеств дуг одного типа;

— **selfConnection** — множество типов дуг, которые соединяют вершину саму с собой;

Connections — множество дуг одного типа.

Атрибуты:

— **label** — метка дуг;

— **incoming** — множество вершин, в которые входят дуги, исходящие из текущей вершины;

— **selfConnection** — множество вершин, из которых исходят дуги, входящие в текущую вершину.

Equivalent — пара эквивалентных вершин.

Атрибуты:

— **target** — вершина целевого графа;

— **pattern** — вершина шаблона.

Isomoprhic — пара изоморфных вершин.

Атрибуты:

— **target** — вершина целевого графа;

— **pattern** — вершина шаблона.

Configuration — конфигурация алгоритма поиска изоморфного подграфа.

Атрибуты:

— **selected** — последовательность пар вершин, которые нужно проверить;

— **checked** — множество изоморфных вершин;

— **current** — индекс текущей вершины в последовательности **selected**.

Реализация алгоритма поиска изоморфных подграфов — функция **match**:

```
def match(target_graph, pattern_graph, match_largest_target_component=False)
```

Параметры:

— **target_graph** — целевой граф, объект типа **Graph**;

— **pattern_graph** — граф шаблона, объект типа **Graph**;

Возвращает генератор списков объектов типа **Isomoprhic**.

Здесь же реализована проверка результата алгоритма — функция **check**:

```
def check(isomorphism)
```

isomorphism — коллекция или генератор объектов типа **Isomoprhic**.

3.2.4.2 Модуль pattern_matcher

В этом модуле реализован алгоритм поиска шаблона проектирования в модели объектно-ориентированной системы. Структура связей обобщения классов представлена на рисунке 3.3. Структура ассоциаций классов представлена на рисунке 3.4.

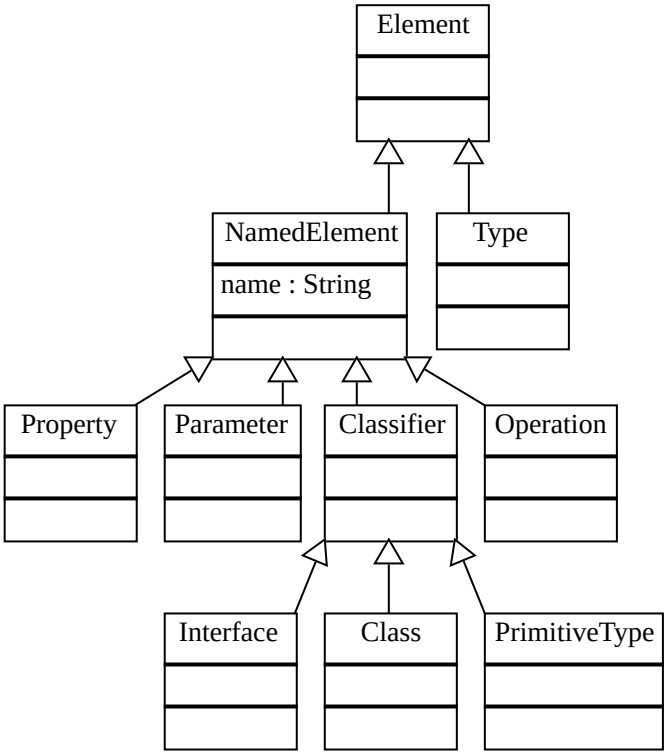

Рисунок 3.3 — Структура классов модуля pattern_matcher. Обобщения

Element — базовый элемент модели.

NamedElement — именованный элемент модели.

Атрибуты:

— **name** — название, уникальное в зависимости от контекста.

Aggregation — тип агрегации для свойства.

Допустимые значения:

— **none** — отсутствует;

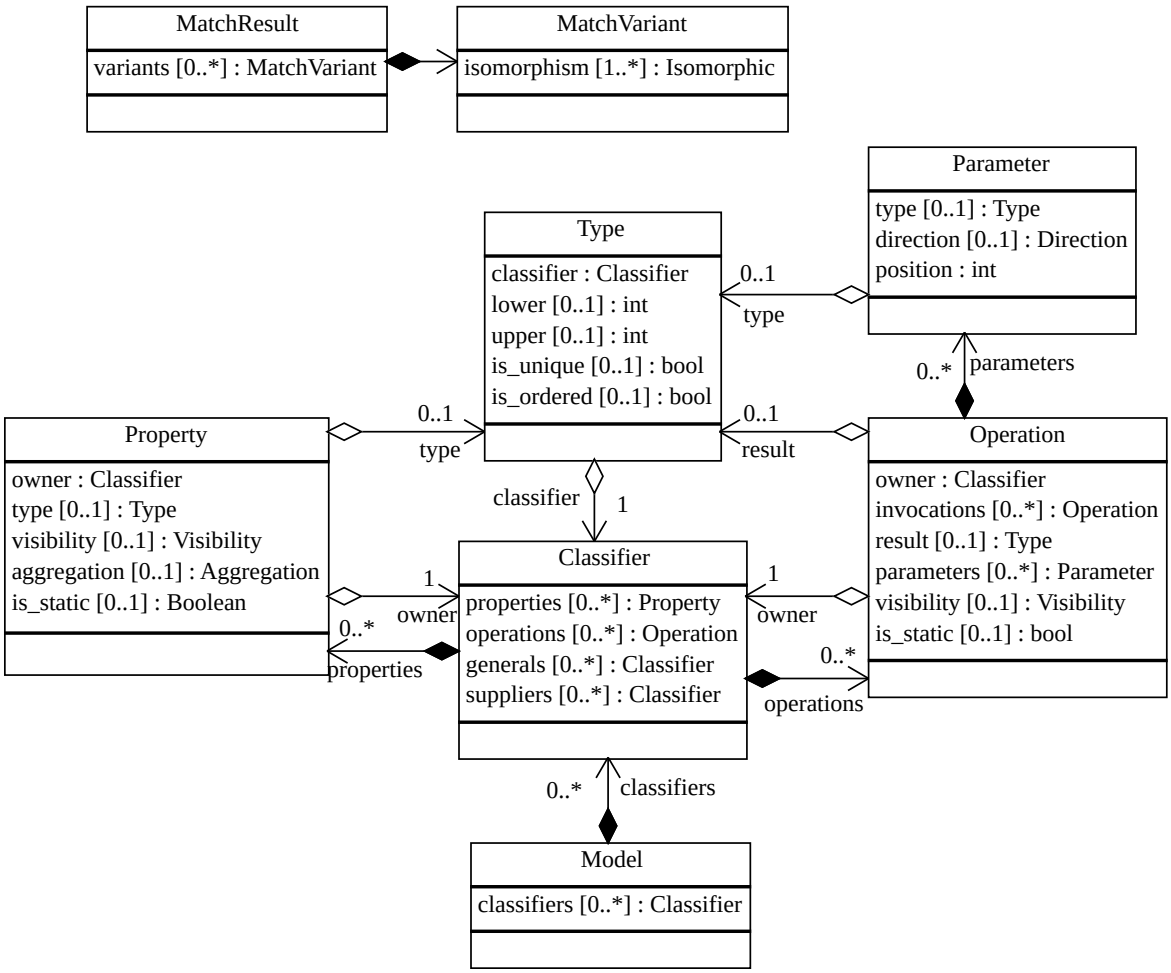

Рисунок 3.4 — Структура классов модуля pattern_matcher. Ассоциации

— **shared** — классификатор ссылается на объект;

— **composite** — классификатор владеет объектом;

Direction — тип параметра метода.

Допустимые значения:

— **in** — метод только читает значение параметра и может изменять его состояние;

— **out** — метод только изменяет значение параметра;

— **inout** — метод читает и изменяет значение параметра;

Visibility — доступность свойства или метода, допустимые значения.

Атрибуты:

— **private** — только из этого же классификатора;

— **protected** — из этого класса и всех производных классификаторов;

— **public** — из любого места;

Classifier — классификатор.

Атрибуты:

— **name** — имя, уникальное для модели;

— **poperty** — свойства классификатора;

— **operation** — операции классификатора;

— **general** — базовые классификаторы;

— **suppliers** — используемые классификаторы;

Property — свойство классификатора.

Атрибуты:

— **name** — имя, уникальное для классификатора;

— **owner** — класс, которому принадлежит свойство;

— **type** — тип свойства;

— **visibility** — доступность;

— **is_static** — имеет ли свойство одно значение для всех объектов классификатора;

Operation — операция классификатора. Атрибуты:

— **name** — имя, уникальное для классификатора, типа результата и списка типов параметров;

— **owner** — классификатора, которому принадлежит операция;

— **invoke** — вызываемые операции;

— **result** — тип результата;

— **parameter** — список параметров;

— **visibility** — доступность;

— **is_static** — запрещено ли методу оперировать состоянием объекта;

Parameter — параметр метода класса.

Атрибуты:

— **name** — имя, уникальное для операции;

— **type** — тип значения;

— **direction** — тип параметра;

— **position** — порядковый номер;

Type — любой тип данных.

Атрибуты:

— **classifier** — классификатор типа;

— **lower** — нижняя граница множественности $[0, upper]$;

— **upper** — верхняя граница множественности $[lower, \infty]$;

— **is_unique** — должны ли значения быть уникальными, если есть множественность;

— **is_ordered** — упорядочены ли значения, если есть множественность.

Class — класс.

Interface — интерфейс.

PrimitiveType — примитивный тип данных, например **int** в **Java**.

Model — модель объектно-ориентированной системы.

MatchResult — результат поиска шаблонов проектирования.

MatchVariant — вариант соответствия элементов цели элементам шаблона.

Атрибуты:

— **isomorphic** — кортеж (**tuple**) пар объектов типа **Element**.

Реализация алгоритма поиска шаблонов проектирования — функция **match**:

```
def match(target, pattern, limit=None)
```

Параметры:

— **target** — целевая модель, объект типа **Model**;

— **pattern** — модель шаблона, объект типа **Model**;

— **limit** — **int**, максимальное количество вариантов, которое нужно найти. Если **None**, то выполняется поиск всех;

3.3 Тестирование

Все тесты написаны на языке **Python**. Для запуска используется фреймворк **pytest** [13].

3.3.1 Модульное тестирование

Для написания модульных тестов использовались библиотеки:

— **unittest** — для описания структуры тестов;

— **PyHamcrest** [14] — для описания проверок в тестах.

3.3.1.1 Тестирование модуля graph_matcher

Протестировано создание объектов классов и их состояния:

— **Node**;

— **Configuration**;

— **Equivalent**;

— **Isomorphic**;

— **Graph**.

Функций:

— **Graph.get_connected_cmponents** — реализация алгоритма поиска компонент связности графа;

— **match** — реализация алгоритма поиска изоморфных подграфов.

Цель — проверить простейшие ситуации на графах из нескольких вершин. Для этого написаны тесты, представленные в таблице 3.1.

Также протестированы случаи:

— у вершин заданы функции эквивалентности;

— полные графы из четырех вершин;

— все возможные графы из 2-4 вершин;

В случае со всеми возможными графами из 2, 3, 4 вершин применяется следующий алгоритм:

for all $n \in \{2, 3, 4\}$ **do**
  for all $G_t \in$ множество всех возможных графов из одной компоненты **do**
    for all $G_p = (V_p, L_p, E_p) \in$ множество всех возможных подграфов G_t **do**
      for all $I \in \text{MATCH}(G_t, G_p)$ **do**
        $\text{ASSERT}(\text{CHECK}(I))$
        $V_p^I \leftarrow \{v_p : (v_t, v_p) \in I\}$

№	Целевой граф	Граф шаблона	Результат
1	$(\varnothing, \varnothing, \varnothing)$	$(\varnothing, \varnothing, \varnothing)$	$\varnothing$
2	$(\{1\}, \varnothing, \varnothing)$	$(\{a\}, \varnothing, \varnothing)$	$\{(1, a)\}$
3	$(\{1, 2\}, \varnothing, \varnothing)$	$(\{a\}, \varnothing, \varnothing)$	$\{(1, a)\}$, $\{(2, a)\}$
4	$(\{1\}, \varnothing, \varnothing)$	$(\{a, b\}, \varnothing, \varnothing)$	$\{(1, a)\}$, $\{(1, b)\}$
5	$(\{1, 2\}, \{l\}, \{(1, 2, l)\})$	$(\{a, b\}, \{l\}, \{(a, b, l)\})$	$\{(1, a), (2, b)\}$
6	$(\{1, 2\}, \{x\}, \{(1, 2, x)\})$	$(\{a, b\}, \{y\}, \{(a, b, y)\})$	$\varnothing$
7	$(\{1\}, \{l\}, \{(1, 1, l)\})$	$(\{a\}, \{l\}, \{(a, a, l)\})$	$\{(1, a)\}$
8	$(\{1, 2, 3\}, \{l\}, \{(1, 2, l), (2, 3, l)\})$	$(\{a, b\}, \{l\}, \{(a, b, l), (b, a, l)\})$	$\varnothing$
9	$(\{1, 2, 3\}, \{l\}, \{(1, 2, l), (2, 3, l)\})$	$(\{a, b\}, \{l\}, \{(a, b, l)\})$	$\{(1, a), (2, b)\}$, $\{(2, a), (3, b)\}$
10	$(\{1, 2, 3, 4\}, \{l\}, \{(1, 2, l), (3, 4, l)\})$	$(\{a, b\}, \{l\}\ , \{(a, b, l)\})$	$\{(1, a), (2, b)\}$, $\{(3, a), (4, b)\}$

Таблица 3.1 — Простейшие тесты алгоритма поиска изоморфных подграфов

$\textsc{assert}(V_p = V_p^I)$
end for
end for
end for
end for
return C

Тестирование для большего количества вершин требует неприемлемо большого количества времени.

В процессе разработки были найдены ошибки, часть из которых была исправлена, и для которых были написаны тесты:

— выполнение проверки на изоморфизм графов, состоящих из вершин, которые находятся в множестве *checked* конфигурации алгоритма;

— поиск изоморфного подграфа из двух компонент, имеющих по одной дуге, в графе из одной компоненты, имеющей несколько дуг.

В первом случае ошибка проявлялась на графах, представленных на рисунках 3.5 3.6.

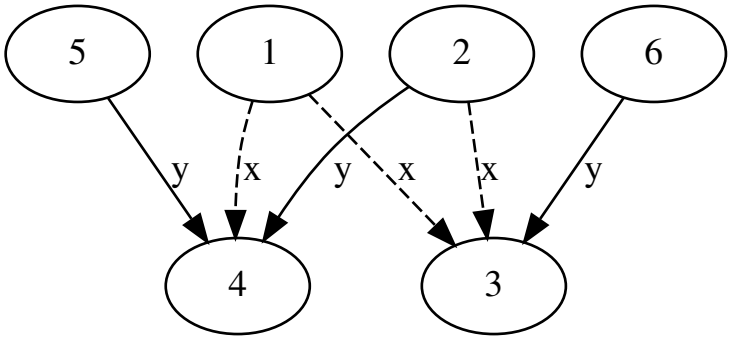

Рисунок 3.5 — Целевой граф

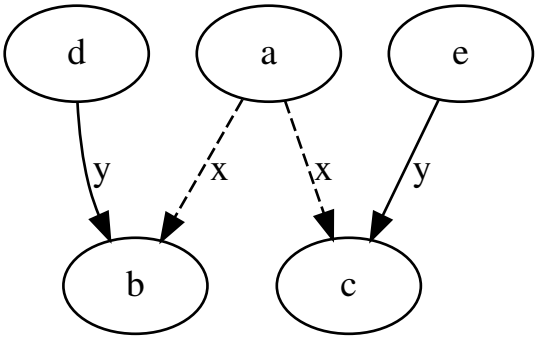

Рисунок 3.6 — Граф шаблона

Ожидался следующий результат:

— $\{(1, a), (2, e), (3, b), (4, c), (6, d)\}$
— $\{(1, a), (3, b), (4, c), (5, e), (6, d)\}$
— $\{(1, a), (2, d), (3, c), (4, b), (6, e)\}$
— $\{(1, a), (3, c), (4, b), (5, d), (6, e)\}$

Первый вариант результат представлен на рисунке 3.7.

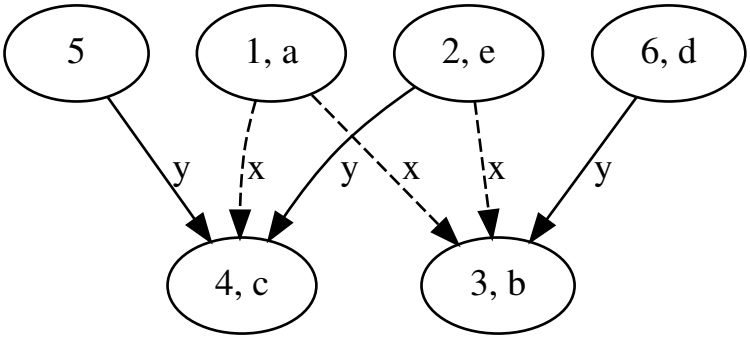

Рисунок 3.7 — Вариант изоморфизма графов

Отсутствие проверки приводило к дополнительным вариантам:

— $\{(1, a), (2, e), (3, c), (4, b), (5, d)\}$

— $\{(1, a), (2, d), (3, b), (4, c), (5, e)\}$

Первый вариант представлен на рисунке 3.8. Здесь вершина $4 \cong b$ соединена с тремя вершинами шаблона, что невозможно, так как вершина b имеет только две связи.

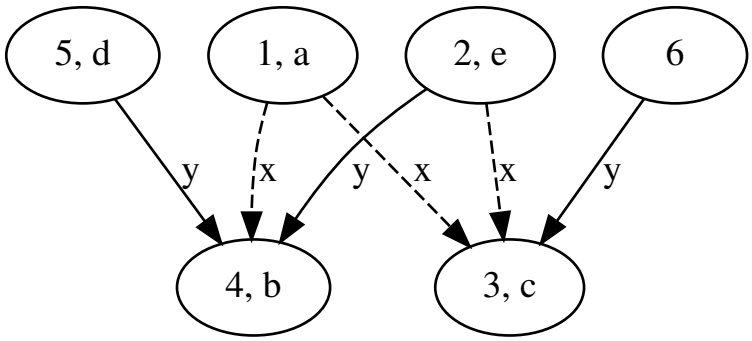

Рисунок 3.8 — Варианта ошибочного изоморфизма графов

Во втором случае ошибка заключается в отсутствии объединения результатов. Для графов $G_t = (\{1, 2, 3, 4\}, \{l\}, \{(1, 2, l), (2, 3, l), (3, 4, l)\})$ и $G_p = (\{a, b, c, d\}, \{l\}, \{(a, b, l), (c, d, l)\})$ результат должен быть таким: $\{(1, a), (2, b), (3, c), (4, d)\}$, $\{(2, a), (3, b)\}$, $\{(1, c), (2, d), (3, a), (4, b)\}$, $\{(2, c), (3, d)\}$; но результат следующий: $\{(1, a), (2, b)\}$, $\{(2, a), (3, b)\}$, $\{(3, a), (4, b)\}$, $\{(1, c), (2, d)\}$, $\{(2, c), (3, d)\}$, $\{(3, c), (4, d)\}$. Учитывая, что в данной работе не используются шаблоны из нескольких компонент связности, можно исправить ошибку позднее, когда это потребуется.

Подробные результаты приведены в приложении А.

3.3.1.2 Тестирование модуля pattern_matcher

Протестировано создание объектов классов и их состояния:

— **Aggregation**;

— **Class**;

— **Classifier**;

— **Direction**;

— **Interface**;

— **Model**;

— **MatchVariant**;

— **MatchResult**;

— **Operation**;

— **Parameter**;

— **PrimitiveType**;

— **Property**;

— **Type**;

— **Visibility**.

Функций:

— **eq_ignore_order** — сравнение коллекций без учета порядка;

— **match** — реализация алгоритма поиска шаблона проектирования в модели;

— **check** — проверка результата **match**;

— чтения и записи модели и отдельных ее элементов в формате **YAML**.

Подробные результаты приведены в приложении В.

3.3.2 Функциональное тестирование

Здесь проверялась работа программ, использовались заранее подготовленные данные и ожидаемые результаты в виде файлов.

3.3.2.1 Тестирование программы java_bytecode_model

В качестве входных данных используются **.java**-файлы. Выполняется их компиляция и запуск **java_bytecode_model** с получившимися **.class**-файлами. Результат выполнения программы сравнивается с заранее заготовленными **.yaml**-файлами.

Проводятся следующие тесты создания модели для:

— пустого файла;

— одного класса (**class**);

— одного перечисления (**enum**);

— одного интерфейса (**interface**);

— двух классов, связанных обобщением (**extends**);

— двух интерфейсов и класса, связанных обобщением (**implements**);

— иерархии обобщения из шести классов, на трех уровнях;

— класса со свойствами;

— класса с методами;

— зависимостей между классами;

— вызовов методов внутри класса и между классами;

— вызова переопределенного метода.

Подробные результаты приведены в приложении Г.

3.3.2.2 Тестирование программы match_pattern

Для моделей, описанных в **.yaml**-файлах, выполняется поиск шаблонов, реализованных в программе **pattern_model**. Результат сравнивается с записанным в файл выводом программы.

Проводятся тесты поиска:

— пустого шаблона в пустой модели;

— шаблона **BaseDerived** в модели с одной связью обобщения;

— шаблона **BaseDerived** в модели с множеством связей обобщения без ограничения количества вариантов;

— шаблона **BaseDerived** в модели с множеством связей обобщения с ограничением количества вариантов;

— шаблона **OverriddenMethodCall** в соответствующей модели;

Подробные результаты приведены в приложении Д.

4 Исследовательский раздел

4.1 Описание используемых шаблонов проектирования

Для проведения исследования были выбраны следующие шаблоны проектирования:

— «абстрактная фабрика» — предоставляет интерфейс для создания групп связанных или зависимых объектов, не указывая их конкретный класс;

— «адаптер» — конвертирует интерфейс класса в другой интерфейс, ожидаемый клиентом. Позволяет классам с разными интерфейсами работать вместе;

— «вызов переопределённого метода» — представляет ситуацию, когда вызывается метод интерфейса, для которого имеется реализация в производном классе. Не является полноценным шаблоном проектирования, а скорее некоторым элементом. Рассматривается, чтобы убедиться в возможности находить самые простые конструкции;

— «декоратор» — динамически предоставляет объекту дополнительные возможности. Представляет собой гибкую альтернативу наследованию для расширения функциональности.

— «мост» — разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.

— «посетитель» — представляет операцию, которая будет выполнена над объектами группы классов. Даёт возможность определить новую операцию без изменения кода классов, над которыми эта операция производится.

— «хранитель» — не нарушая инкапсуляцию, определяет и сохраняет состояние объекта и позволяет восстановить объект в этом состоянии.

— «цепочка ответственности» — избегает связывания отправителя запроса с его получателем, давая возможность обработать запрос более чем одному объекту. Связывает объекты-получатели и передает запрос по цепочке, пока объект не обработает его.

Графы моделей шаблонов представлены в приложении Е.

4.2 Поиск шаблонов проектирования в существующих программах

Проект «java-design-patterns»

Проект находится в открытом доступе, размещен на ресурсе github.com [15]. Интересен тем, что содержит сборник из 48 примеров реализации различных шаблонов проектирования на языке **Java**, что очень хорошо подходит для проверки работы программного комплекса. Здесь можно увидеть некоторые из особенностей реализации шаблонов, поэкспериментировать с разными представлениями шаблона.

Поиск шаблона «абстрактная фабрика»

Первый реализованный пример шаблона в проекте. Реализация совпадает с разными описаниями. Шаблон находится, результат представлен на рисунке 4.1.

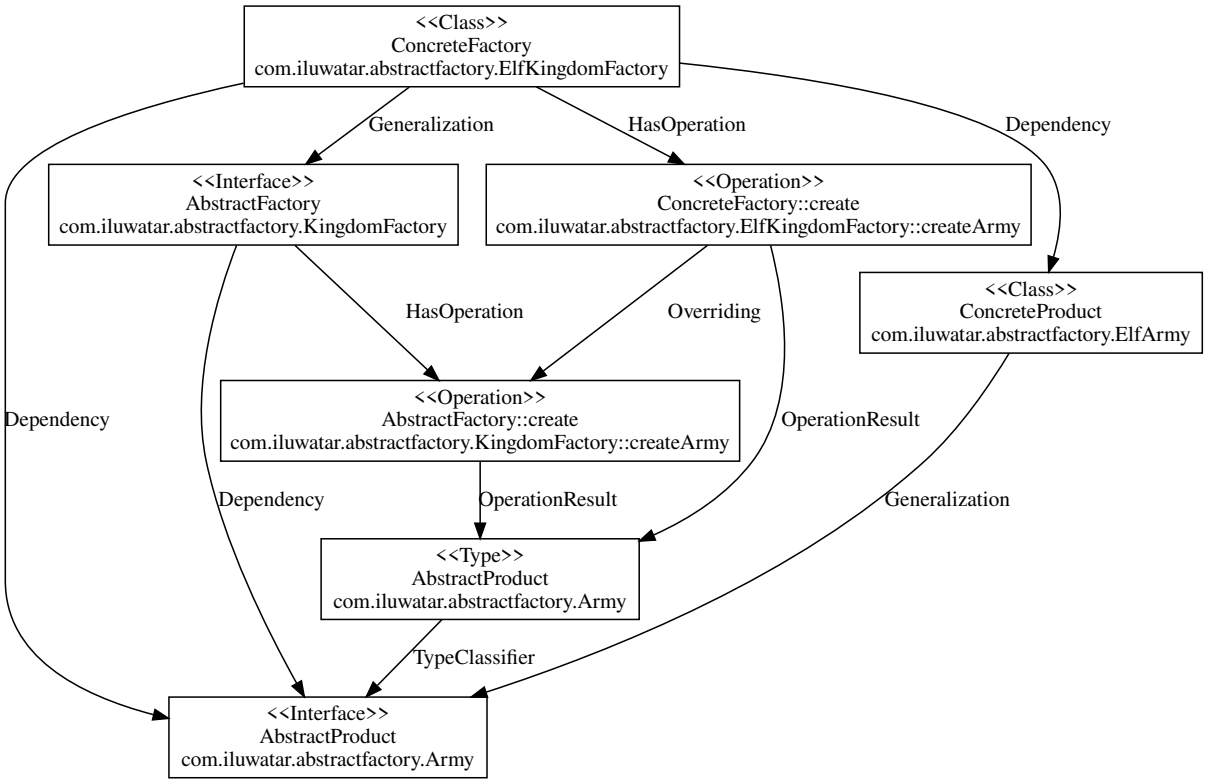

Рисунок 4.1 — Результат поиска шаблона проектирования «абстрактная фабрика» в примере его реализации

Поиск шаблона «адаптер»

Для шаблона есть пример, и шаблон в нем находится. Результат представлен на рисунке 4.2

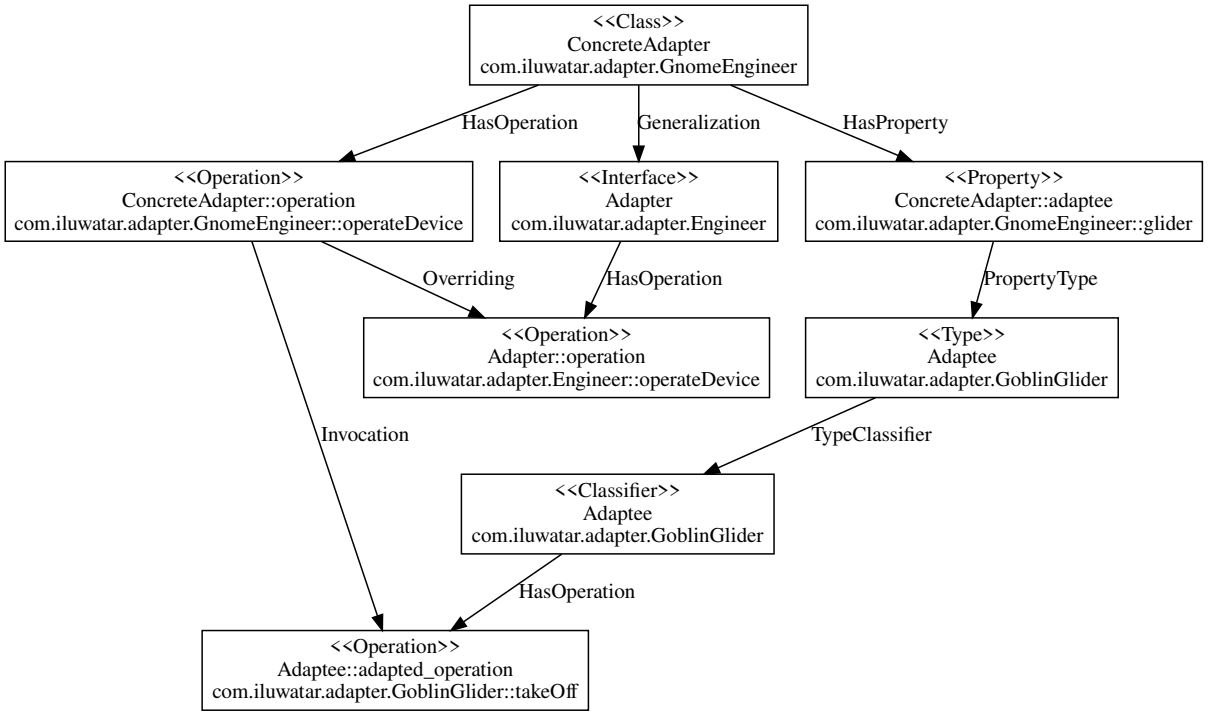

Рисунок 4.2 — Результат поиска шаблона проектирования «адаптер» в примере его реализации

Поиск шаблона «мост»

В примере реализации не найден, но найден в других примерах. Проблема заключается в том, что используется дополнительное обобщение, чтобы не было дублирования кода. Класс **Abstraction** разделен на две части: базовый абстрактный класс **MagicWeapon**, который совязан ассоциацией с **MagicWeaponImp**, являющимся интерфейсом **Implementor**; и конкретными реализациями: **BlindingMagicWeapon**, **FlyingMagicWeapon**, **SoulEatingMagicWeapon**. Здесь требуется механизм, который позволит описать класс так, все ассоциации базового класса также являются и ассоциациями производного. Нужна сущность, которая может объединять иерархии наследования классов в некоторый надкласс. UML-диаграммы классов не предоставляют такого механизма. В данной работе эта проблема осталась не решенной.

Шаблон был найден в следующих примерах:

— adapter;

— decorator;

— intercepting-filter;

— mediator;

— model-view-presenter;

— null-object;

— poison-pill;

— property;

— service-layer;

— state;

— strategy.

Поиск шаблона «посетитель»

Шаблон реализован в специальном примере и находится. Результат представлен на рисунке 4.3

Библиотека «Apache BCEL»

Исследовалась в качестве примера реального проекта. Модель проекта в **YAML**-формате занимает 4,1 МБ. Граф модели состоит из 7700 вершин и 52000 дуг. Здесь были найдены шаблоны «адаптер» (см. рисунок 4.4), «мост» (см. рисунок 4.5). Другие шаблоны не найдены.

Поиск шаблонов проектирования в других проектах

Исследовались другие проекты, написанные на языках **Java** и один на **Scala**. Сводные результаты по всем проектам приведены в таблице 4.1.

4.3 Исследование производительности

Реализация алгоритма поиска изоморфных подграфов — самая уязвимой с точки зрения производительности частью программного комплекса. Использовался язык **Python**, для которого существует ряд виртуальных машин. Рассмотрим некоторые из них и сравним производительность программы **match_pattern**.

— **CPython** — написана на языке **C**. Код программы компилируется в байт-код и интерпретируется.

— **PyPy** — написана на языке **Python**. Использует технологию **JIT**-компиляции, компилируя в процессе выполнения программы код на **Python** в машинный код.

Работу программы **match_pattern** можно разделить на две части: загрузка модели и поиск шаблона проектирования. Для моделей больших проектов загрузка

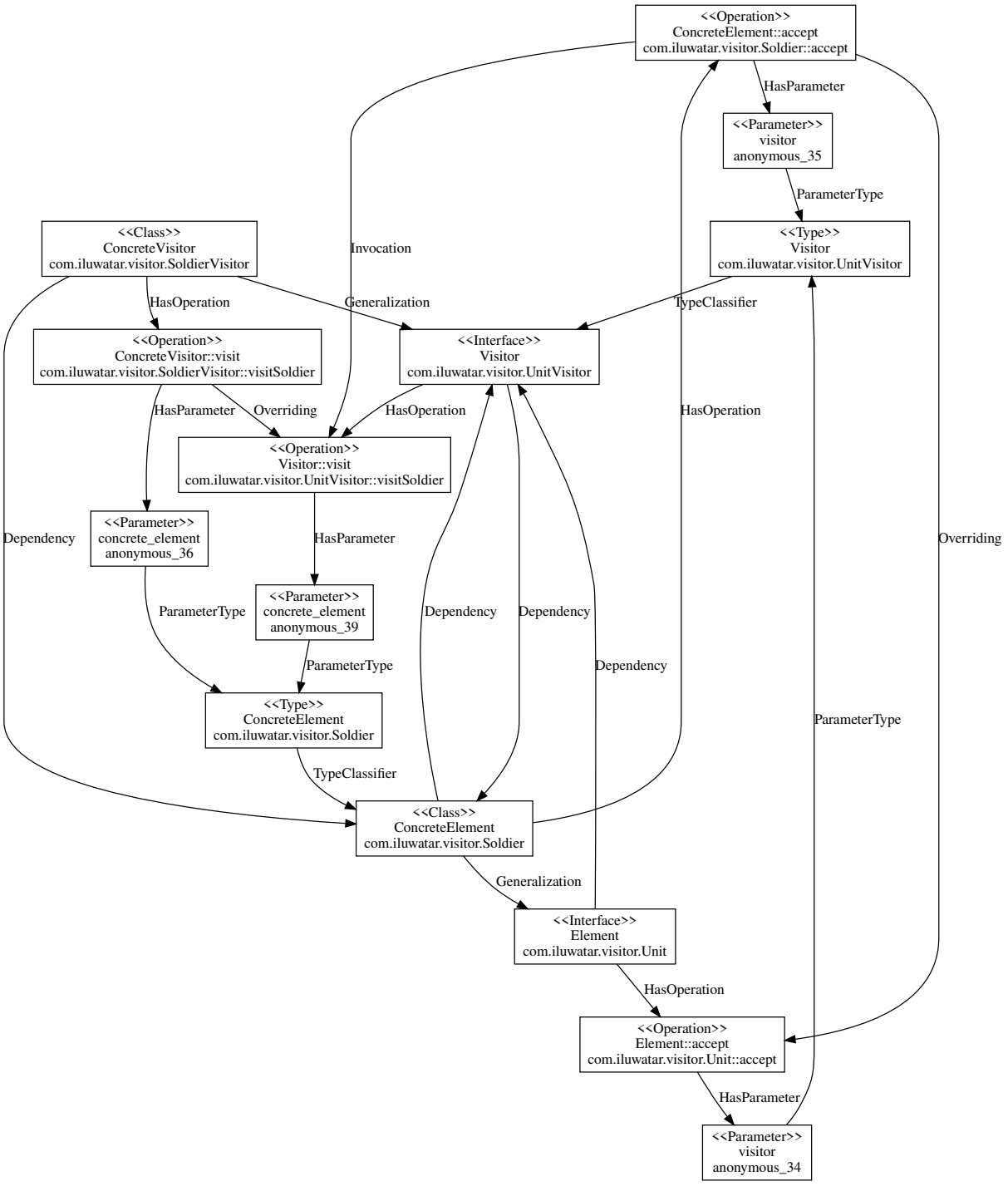

Рисунок 4.3 — Результат поиска шаблона проектирования «посетитель» в примере его реализации

может занимать достаточно большое время. Рассмотрим проект **Apache Zookeeper**. Модель состоит из 101413 элементов. Размер **.yaml**-файла — 33 МБ. Результаты представлены в таблице 4.2.

В следующем тесте в проекте «Apache BCEL» выполнялся поиск шаблона проектирования «вызов переопределённого метода». Программа выполняется до нахождения первых пяти результатов. Отладочный вывод и результат программ совпа-

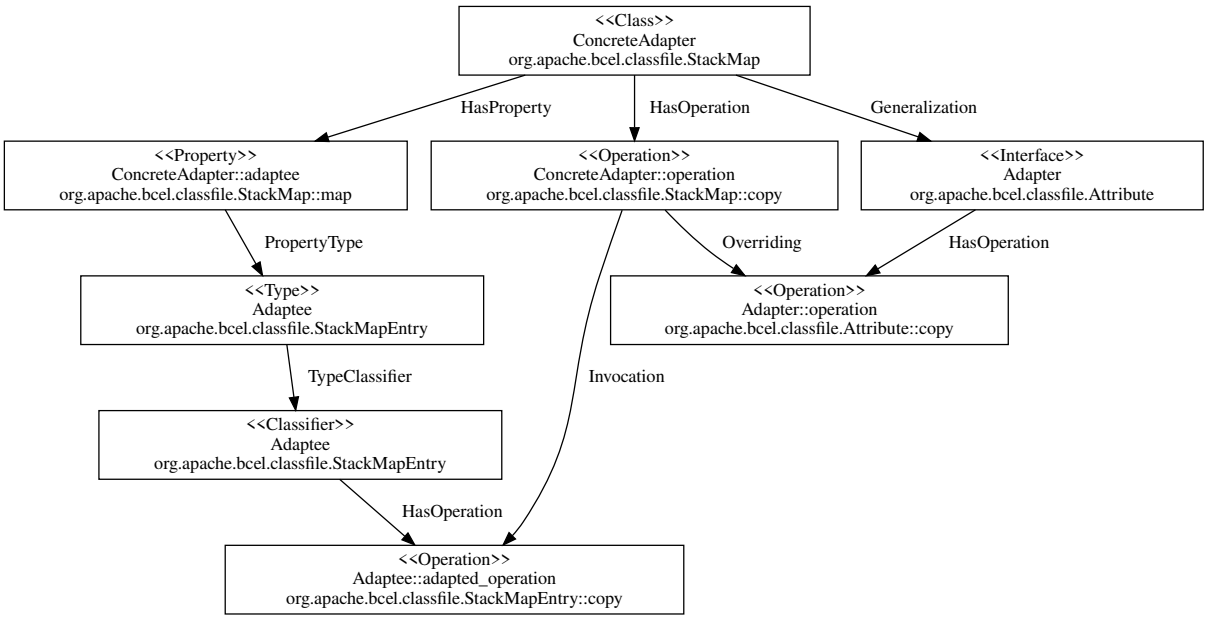

Рисунок 4.4 — Результат поиска шаблона проектирования «адаптер» в «Apache BCEL»

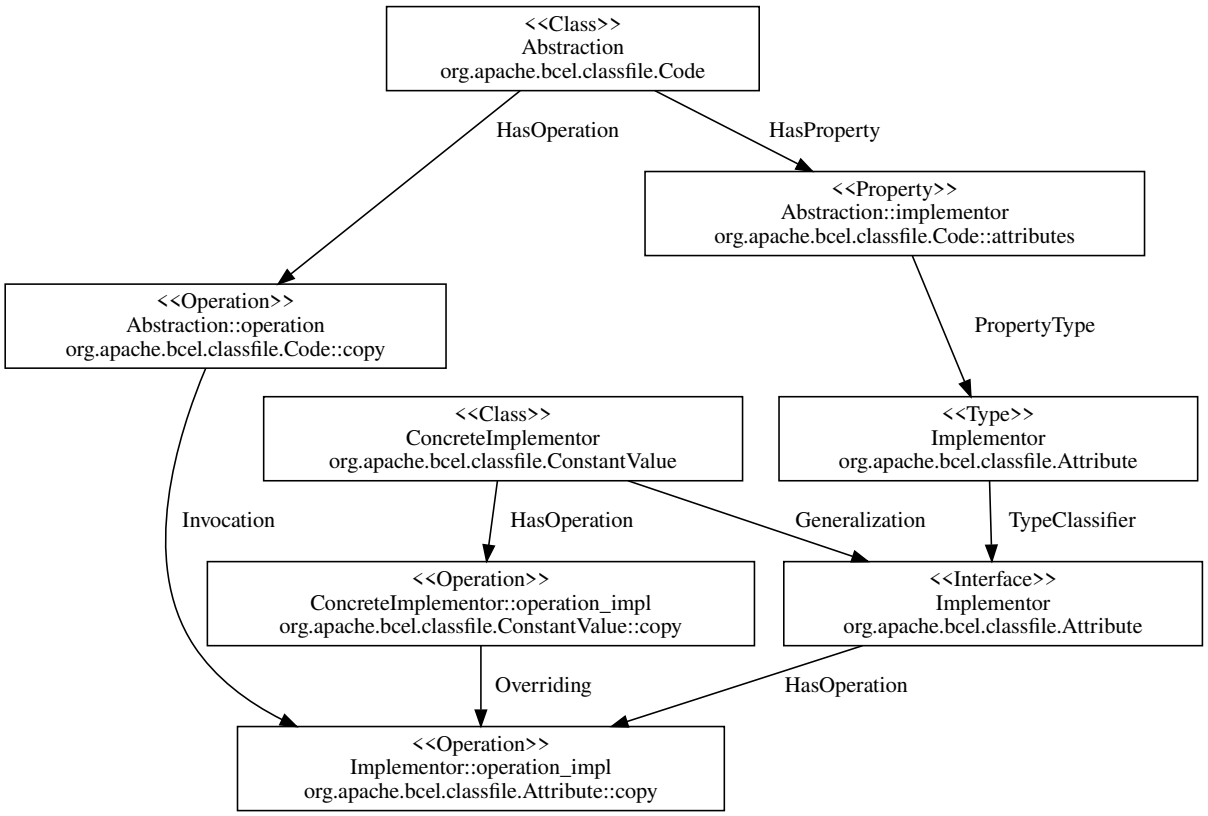

Рисунок 4.5 — Результат поиска шаблона проектирования «мост» в «Apache BCEL»

дает. Здесь учитывалось только время поиска шаблона без загрузки модели. Результаты представлены в таблице 4.3.

Название проекта	Язык	Найденные шаблоны проектирования
Netflix EVCacahe	Java	«вызов переопределённого метода», «мост»
airbnb aerosolve	Java	«вызов переопределённого метода», «мост»
java-design-patterns	Java	«абстрактная фабрика», «адаптер», «мост», «посетитель», «вызов переопределённого метода»
Apache BCEL	Java	«адаптер», «вызов переопределённого метода», «мост»
Apache Zookeeper	Java	«вызов переопределённого метода», «мост»
scodec	Scala	«вызов переопределённого метода»

Таблица 4.1 — Результаты поиска шаблонов проектирования в различных проектах

Виртуальная машина	Время чтения файла модели, с	Относительное время чтения файла модели	Используемый объем памяти, КБ	Относительный используемый объем памяти
PyPy	26	1,0	746848	1,0
CPython	180	6,9	3078436	4,1

Таблица 4.2 — Тестирование производительности чтения файла модели на различных виртуальных машинах

Виртуальная машина	Время поиска, с	Относительное время поиска	Используемый объем памяти, КБ	Относительный используемый объем памяти
CPython	8,6	1,0	223004	1,64
PyPy	10,3	1,2	136436	1,0

Таблица 4.3 — Тестирование производительности поиска шаблона проектирования «вызов переопределённого метода» в «Apache BCEL»

4.4 Выводы по результатам исследования

Метод позволяет находить шаблоны проектирования. «Вызов переопределенного метода» встречается во всех проектах. Это одна из самых распространённых конструкций. Именно для этого существует разделение между интерфейсом и реализацией. Не удивительно, что такую базовую возможность ООП используют везде. Наиболее часто встречаются шаблоны проектирования «адаптер» и «мост». Можно утверждать, что модели достаточно хорошо отражают эти шаблоны, и что они действительно используются. Для других шаблонов можно предположить, что они предназначены для специфических задач, которые не решались в рассматриваемых проектах.

Для программы поиска шаблонов проектирования лучше использовать виртуальную машину **PyPy**. Она незначительно уступает **CPython** по времени поиска, но значительно превосходит по времени загрузки модели и более эффективно использует память.

Заключение

Разработан метод поиска шаблонов проектирования в объектно-ориентированных программах на основе изоморфных подграфов. Описана модель представления программ и структура граф модели. Разработаны алгоритмы построения графа и поиска изоморфных подграфов. Разработан программный комплекс, который позволяет построить модель для программ на языке **Java** и выполнять поиск шаблонов проектирования. Проведено исследование, которое показывает работоспособность метода — шаблоны находятся.

Метод является основой для задач верификации и оценки качества структур объектно-ориентированных программ. Может применяться для оценки похожести структур программ.

Список использованных источников

1. OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1, 2011, Режим доступа: http://www.omg.org/spec/UML/2.4.1/ (дата обращения 13.06.2015).

2. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software Gamma. New Jersey: Pearson Education, 1994. 395 p.

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. 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.

5. 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.

6. Ullmann J.R., An Algorithm for Subgraph Isomorphism. Association for Computing Machinery. 1976, vol. 23, issue 1 P. 31 – 42.

7. The Official YAML Web Site, Режим доступа: http://yaml.org/ (дата обращения 13.06.2015).

8. musiKk/plyj · GitHub — A Java parser written in Python using PLY, Режим доступа: https://github.com/musiKk/plyj (дата обращения 13.06.2015).

9. briencj/python-javatools · GitHub — Tools for examining Java bytecode in Python, Режим доступа: https://github.com/obriencj/python-javatools (дата обращения 13.06.2015).

10. PyYAML, Режим доступа: http://pyyaml.org/ (дата обращения 13.06.2015).

11. Apache Commons BCEL, Режим доступа: https://commons.apache.org/proper/commons-bcel/ (дата обращения 13.06.2015).

12. Apache Maven Project, Режим доступа: https://maven.apache.org/ (дата обращения 13.06.2015).

13. pytest, Режим доступа: http://pytest.org/ (дата обращения 13.06.2015).

14. hamcrest/PyHamcrest · GitHub — Hamcrest matchers for Python, Режим доступа: https://github.com/hamcrest/PyHamcrest (дата обращения 13.06.2015).

15. iluwatar/java-design-patterns · GitHub — Design pattern samples implemented in Java, Режим доступа: https://github.com/iluwatar/java-design-patterns (дата обращения 13.06.2015).

Приложение А Тестирование модуля graph_matcher

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.4.1, py-1.5.2, pluggy-0.6.0 -- /usr/bin/
cachedir: .pytest_cache
rootdir: /home/elsid/dev/master/src/match_pattern, inifile:
collecting ... collected 51 items

Check.test_check_empty_should_succeed PASSED [  1%]
MakeEquivalent.test_make_should_succeed PASSED [  3%]
MakeIsomorphic.test_make_should_succeed PASSED [  5%]
MakeConfiguration.test_make_should_succeed PASSED [  7%]
MakeGraph.test_make_complex_should_succeed PASSED [  9%]
MakeGraph.test_make_empty_should_succeed PASSED [ 11%]
MakeGraph.test_make_one_node_and_one_arc_should_succeed PASSED [ 13%]
MakeGraph.test_make_one_node_and_one_arc_with_duplication_should_succeed PASSED
 [ 15%]
MakeGraph.test_make_one_node_and_one_labeled_arc_should_succeed PASSED [ 17%]
MakeGraph.test_make_self_connected_by_labeled_arc_node_should_succeed PASSED [
19%]
MakeGraph.test_make_self_connected_node_should_succeed PASSED [ 21%]
MakeGraph.test_make_with_one_arc_should_succeed PASSED [ 23%]
MakeGraph.test_make_with_one_labeled_arc_should_succeed PASSED [ 25%]
MakeGraph.test_make_with_one_node_should_succeed PASSED [ 27%]
GraphGetConnectedComponents.test_empty_should_succeed PASSED [ 29%]
GraphGetConnectedComponents.test_three_connected_should_succeed PASSED [ 31%]
GraphGetConnectedComponents.test_two_components_should_succeed PASSED [ 33%]
GraphGetConnectedComponents.test_two_connected_should_succeed PASSED [ 35%]
GraphGetConnectedComponents.test_two_double_connected_should_succeed PASSED [ 3
7%]
GraphGetConnectedComponents.test_two_labeled_connected_should_succeed PASSED [
39%]
GraphGetConnectedComponents.test_two_not_connected_should_succeed PASSED [ 41%]
GraphGetConnectedComponents.test_two_twice_labeled_connected_should_succeed PAS
```

```
SED [ 43%]
AsDot.test_empty PASSED                    [ 45%]
AsDot.test_with_one_arc PASSED             [ 47%]
AsDot.test_with_one_labeled_arc PASSED     [ 49%]
AsDot.test_with_one_node PASSED            [ 50%]
AsDot.test_with_two_arcs_between_two_nodes PASSED [ 52%]
Match.test_check_current_isomorphism_before_add_to_visited PASSED [ 54%]
Match.test_match_chain_of_two_in_chain_of_three_should_succeed PASSED [ 56%]
Match.test_match_complete_graphs_and_should_succeed PASSED [ 58%]
Match.test_match_different_graphs_should_be_empty_result PASSED [ 60%]
Match.test_match_empty_should_succeed PASSED [ 62%]
Match.test_match_generated_graphs_should_succeed PASSED [ 64%]
Match.test_match_one_component_in_two_should_succeed PASSED [ 66%]
Match.test_match_two_components_in_one_should_succeed PASSED [ 68%]
Match.test_match_with_many_components_and_special_equiv_should_succeed PASSED [
 70%]
Match.test_match_with_one_and_with_two_components_should_succeed PASSED [ 72%]
Match.test_match_with_one_arc_should_succeed PASSED [ 74%]
Match.test_match_with_one_different_labeled_arcs_should_be_empty_result PASSED
[ 76%]
Match.test_match_with_one_explicit_labeled_arc_should_succeed PASSED [ 78%]
Match.test_match_with_one_node_should_succeed PASSED [ 80%]
Match.test_match_with_self_connected_nodes_should_succeed PASSED [ 82%]
Match.test_match_with_two_and_three_components_should_succeed PASSED [ 84%]
Match.test_match_with_two_components_should_succeed PASSED [ 86%]
MakeNode.test_make_single_should_succeed PASSED [ 88%]
NodeGetConnectedComponent.test_empty_should_succeed PASSED [ 90%]
NodeGetConnectedComponent.test_three_connected_should_succeed PASSED [ 92%]
NodeGetConnectedComponent.test_two_components_should_succeed PASSED [ 94%]
NodeGetConnectedComponent.test_two_connected_should_succeed PASSED [ 96%]
NodeGetConnectedComponent.test_two_double_connected_should_succeed PASSED [ 98%]
NodeGetConnectedComponent.test_two_not_connected_should_succeed PASSED [100%]

========================== 51 passed in 1.31 seconds ===========================
```

Приложение Б Тестирование модуля java_source_parser

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.4.1, py-1.5.2, pluggy-0.6.0 -- /usr/bin/
cachedir: .pytest_cache
rootdir: /home/elsid/dev/master/src/match_pattern, inifile:
collecting ... collected 65 items

GetVisibility.test_get_from_field_without_visibility_should_be_private PASSED [
  1%]
GetVisibility.test_get_from_private_field_should_be_private PASSED [  3%]
GetVisibility.test_get_from_protected_field_should_be_protected PASSED [  4%]
GetVisibility.test_get_from_public_field_should_be_public PASSED [  6%]
GetVisibility.test_get_from_public_method_should_be_public PASSED [  7%]
HasDuplications.test_empty_should_return_false PASSED [  9%]
HasDuplications.test_many_different_should_return_false PASSED [ 10%]
HasDuplications.test_many_with_two_same_should_return_true PASSED [ 12%]
HasDuplications.test_two_same_should_return_true PASSED [ 13%]
HasDuplications.test_with_one_should_return_false PASSED [ 15%]
GetNameValue.test_get_from_int_should_return_error PASSED [ 16%]
GetNameValue.test_get_from_plyj_name_should_succeed PASSED [ 18%]
GetNameValue.test_get_from_str_should_succeed PASSED [ 20%]
FormatTypeArguments.test_format_empty_should_succeed PASSED [ 21%]
FormatTypeArguments.test_format_one_should_succeed PASSED [ 23%]
FormatTypeArguments.test_format_two_should_succeed PASSED [ 24%]
GetTypeName.test_get_from_plyj_array_type_should_succeed PASSED [ 26%]
GetTypeName.test_get_from_plyj_array_type_with_add_dimensions_should_succeed PA
SSED [ 27%]
GetTypeName.test_get_from_plyj_type_should_succeed PASSED [ 29%]
GetTypeName.test_get_from_plyj_type_with_add_dimensions_should_succeed PASSED [
 30%]
GetTypeName.test_get_from_str_should_succeed PASSED [ 32%]
GetTypeName.test_get_from_template_plyj_type_should_succeed PASSED [ 33%]
GetTypeName.test_get_from_template_with_inserted_template_should_succeed PASSED
```

```
 [ 35%]
GetTypeName.test_get_from_template_with_two_args_should_succeed PASSED [ 36%]
GetClassifierName.test_get_from_plyj_array_type_should_succeed PASSED [ 38%]
GetClassifierName.test_get_from_plyj_type_should_succeed PASSED [ 40%]
GetClassifierName.test_get_from_str_should_succeed PASSED [ 41%]
GetClassifierName.test_get_from_template_with_inserted_template_should_succeed
PASSED [ 43%]
MakeVariableType.test_make_array_should_succeed PASSED [ 44%]
MakeVariableType.test_make_primitive_should_succeed PASSED [ 46%]
FillClassifiers.test_fill_class_with_field_instance_creation_item_should_succee
d PASSED [ 47%]
FillClassifiers.test_fill_class_with_overriding_method_in_object_should_succeed
 PASSED [ 49%]
FillClassifiers.test_fill_empty_should_succeed PASSED [ 50%]
FillClassifiers.test_fill_enum_with_overriding_in_item_should_succeed PASSED [
52%]
FillClassifiers.test_fill_one_class_with_local_class_should_succeed PASSED [ 53
%]
FillClassifiers.test_fill_one_class_with_one_operation_should_succeed PASSED [
55%]
FillClassifiers.test_fill_one_class_with_one_property_should_succeed PASSED [ 5
6%]
FillClassifiers.test_fill_one_class_with_two_overloaded_operations_should_succe
ed PASSED [ 58%]
FillClassifiers.test_fill_one_class_with_two_same_operations_should_return_erro
r PASSED [ 60%]
FillClassifiers.test_fill_one_class_with_two_same_property_should_return_error
PASSED [ 61%]
FillClassifiers.test_fill_recursive_class_should_succeed PASSED [ 63%]
FillClassifiers.test_fill_two_same_sub_classes_should_return_error PASSED [ 64%]
MakeClass.test_make_should_succeed PASSED [ 66%]
MakeInterface.test_make_should_succeed PASSED [ 67%]
MakeClassifiers.test_declaration_of_two_same_classes_should_return_error PASSED
 [ 69%]
```

```
MakeClassifiers.test_declaration_of_two_same_interfaces_should_return_error PAS
SED [ 70%]
MakeClassifiers.test_make_from_empty_should_succeed PASSED [ 72%]
MakeClassifiers.test_make_one_class_should_succeed PASSED [ 73%]
MakeClassifiers.test_make_one_interface_should_succeed PASSED [ 75%]
MakeClassifiers.test_make_several_class_should_succeed PASSED [ 76%]
MakeDependencies.test_make_from_empty_should_succeed PASSED [ 78%]
MakeDependencies.test_make_from_type_method_should_succeed PASSED [ 80%]
GenerateSubpaths.test_empty_should_succeed PASSED [ 81%]
GenerateSubpaths.test_multiple_import_should_succeed PASSED [ 83%]
GenerateSubpaths.test_single_import_should_succeed PASSED [ 84%]
SetFullTypeNames.test_set_empty_should_succeed PASSED [ 86%]
MakeGeneralizations.test_make_class_extends_and_implements_should_succeed PASSE
D [ 87%]
MakeGeneralizations.test_make_class_extends_class_should_succeed PASSED [ 89%]
MakeGeneralizations.test_make_class_implements_two_interfaces_should_succeed PA
SSED [ 90%]
MakeGeneralizations.test_make_from_empty_should_succeed PASSED [ 92%]
MakeGeneralizations.test_make_interface_extends_two_interfaces_should_succeed P
ASSED [ 93%]
MakeModel.test_decorator_from_text_should_succeed PASSED [ 95%]
MakeModel.test_from_empty_should_succeed PASSED [ 96%]
MakeModel.test_from_text_should_succeed PASSED [ 98%]
MakeModel.test_parse_with_syntax_errors_should_return_errors PASSED [100%]

========================== 65 passed in 0.30 seconds ==========================
```

Приложение В Тестирование модуля pattern_matcher

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.4.1, py-1.5.2, pluggy-0.6.0 -- /usr/bin/
cachedir: .pytest_cache
rootdir: /home/elsid/dev/master/src/match_pattern, inifile:
collecting ... collected 92 items

MakeAggregation.test_make PASSED   [  1%]
MakeAggregation.test_yaml_dump PASSED [  2%]
MakeAggregation.test_yaml_load PASSED [  3%]
MakeClassifier.test_dump_and_load_yaml_classifier_with_name_should_succeed PASS
ED [  4%]
MakeClassifier.test_dump_and_load_yaml_classifier_with_operation_should_succeed
 PASSED [  5%]
MakeClassifier.test_dump_and_load_yaml_classifier_with_property_should_succeed
PASSED [  6%]
MakeClassifier.test_dump_and_load_yaml_recursive_classifier_should_succeed PASS
ED [  7%]
MakeClassifier.test_eq_empty_should_succeed PASSED [  8%]
MakeClassifier.test_eq_two_recursive_should_succeed PASSED [  9%]
MakeClassifier.test_eq_with_property_should_succeed PASSED [ 10%]
MakeClassifier.test_equivalent_pattern_should_succeed PASSED [ 11%]
MakeClass.test_dump_and_load_yaml_clazz_with_name_should_succeed PASSED [ 13%]
MakeClass.test_dump_and_load_yaml_clazz_with_property_should_succeed PASSED [ 1
4%]
MakeClass.test_dump_and_load_yaml_recursive_clazz_should_succeed PASSED [ 15%]
MakeClass.test_repr_should_succeed PASSED [ 16%]
MakeClass.test_str_should_succeed PASSED [ 17%]
MakeDirection.test_make PASSED       [ 18%]
MakeDirection.test_yaml_dump PASSED  [ 19%]
MakeDirection.test_yaml_load PASSED  [ 20%]
MakeInterface.test_dump_and_load_yaml_recursive_interface_should_succeed PASSED
 [ 21%]
```

```
MakeInterface.test_repr_should_succeed PASSED [ 22%]
MakeInterface.test_str_should_succeed PASSED [ 23%]
EqIgnoreOrder.test_compare_different_len_should_be_not_equal PASSED [ 25%]
EqIgnoreOrder.test_compare_empty_should_be_equal PASSED [ 26%]
EqIgnoreOrder.test_compare_empty_to_not_empty_should_be_not_equal PASSED [ 27%]
EqIgnoreOrder.test_compare_equal_should_be_equal PASSED [ 28%]
EqIgnoreOrder.test_compare_nested_tuples_equal_should_be_equal PASSED [ 29%]
EqIgnoreOrder.test_compare_twice_nested_tuples_equal_should_be_equal PASSED [ 3
0%]
EqIgnoreOrder.test_compare_values_with_all_permutations_should_be_equal PASSED
[ 31%]
Check.test_check_empty_should_succeed PASSED [ 32%]
Check.test_check_model_with_one_generalization_should_return_error PASSED [ 33%]
Check.test_check_model_with_one_generalization_should_succeed PASSED [ 34%]
MakeMatchVariant.test_make_empty_should_succeed PASSED [ 35%]
MakeMatchVariant.test_make_not_empty_should_succeed PASSED [ 36%]
MakeMatchResult.test_make_empty_should_succeed PASSED [ 38%]
MakeMatchResult.test_make_not_empty_should_succeed PASSED [ 39%]
AllIndirectGenerals.test_for_classifier_in_diamond_generalization PASSED [ 40%]
AllIndirectGenerals.test_for_classifier_in_generalization_chain PASSED [ 41%]
AllIndirectGenerals.test_for_classifier_in_generalization_tree PASSED [ 42%]
AllIndirectGenerals.test_for_classifier_with_one_general PASSED [ 43%]
AllIndirectGenerals.test_for_classifier_with_three_generals PASSED [ 44%]
AllIndirectGenerals.test_for_classifier_without_generals PASSED [ 45%]
FindOverridden.test_find_direct_in_generalization_chain PASSED [ 46%]
FindOverridden.test_find_for_classifier_with_one_general PASSED [ 47%]
FindOverridden.test_find_for_classifier_without_generals PASSED [ 48%]
FindOverridden.test_find_indirect_in_generalization_chain PASSED [ 50%]
MatchModel.test_match_abstract_factory_pattern_in_bukkit_example PASSED [ 51%]
MatchModel.test_match_abstract_factory_patterns PASSED [ 52%]
MatchModel.test_match_decorator_pattern_in_burgers PASSED [ 53%]
MatchModel.test_match_decorator_pattern_in_burgers_limit_one PASSED [ 54%]
MatchModel.test_match_decorator_patterns PASSED [ 55%]
MatchModel.test_match_empty_should_has_empty_match_result PASSED [ 56%]
```

```
MatchModel.test_match_visitor_patterns PASSED [ 57%]
ReprModel.test_repr_abstract_factory_empty_should_succeed PASSED [ 58%]
ReprModel.test_repr_decorator_empty_should_succeed PASSED [ 59%]
ReprModel.test_repr_empty_should_succeed PASSED [ 60%]
YamlModel.test_yaml_dump PASSED          [ 61%]
YamlModel.test_yaml_dump_abstract_factory_pattern PASSED [ 63%]
YamlModel.test_yaml_dump_burgers PASSED  [ 64%]
YamlModel.test_yaml_dump_decorator_pattern PASSED [ 65%]
YamlModel.test_yaml_dump_memento_pattern PASSED [ 66%]
YamlModel.test_yaml_load PASSED          [ 67%]
YamlModel.test_yaml_load_abstract_factory_pattern PASSED [ 68%]
YamlModel.test_yaml_load_burgers PASSED  [ 69%]
YamlModel.test_yaml_load_decorator_pattern PASSED [ 70%]
YamlModel.test_yaml_load_memento_pattern PASSED [ 71%]
MakeOperation.test_dump_and_load_yaml_with_attrs_and_parameters_should_succeed
PASSED [ 72%]
MakeOperation.test_dump_and_load_yaml_with_name_should_succeed PASSED [ 73%]
MakeOperation.test_equivalent_pattern_should_succeed PASSED [ 75%]
MakeOperation.test_str_should_succeed PASSED [ 76%]
MakeParameter.test_dump_and_load_yaml_with_name_and_direction_should_succeed PA
SSED [ 77%]
MakeParameter.test_dump_and_load_yaml_with_name_should_succeed PASSED [ 78%]
MakeParameter.test_dump_and_load_yaml_with_unicode_name_should_succeed PASSED [
 79%]
MakeParameter.test_str_should_succeed PASSED [ 80%]
MakePrimitiveType.test_dump_and_load_yaml_recursive_primitive_type_should_succe
ed PASSED [ 81%]
MakePrimitiveType.test_repr_should_succeed PASSED [ 82%]
MakePrimitiveType.test_str_should_succeed PASSED [ 83%]
MakeProperty.test_dump_and_load_yaml_with_attributes_should_succeed PASSED [ 84
%]
MakeProperty.test_dump_and_load_yaml_with_name_should_succeed PASSED [ 85%]
MakeProperty.test_str_should_succeed PASSED [ 86%]
ReprMultiplicity.test_different_combinations_should_succeed PASSED [ 88%]
```

```
MakeType.test_dump_and_load_yaml_should_succeed PASSED [ 89%]
MakeType.test_eq_should_succeed PASSED    [ 90%]
MakeType.test_equivalent_pattern_range_should_succeed PASSED [ 91%]
MakeType.test_make_with_wrong_lower_should_throw_exception PASSED [ 92%]
MakeType.test_make_with_wrong_range_should_throw_exception PASSED [ 93%]
MakeType.test_make_with_wrong_upper_should_throw_exception PASSED [ 94%]
MakeType.test_str_should_succeed PASSED   [ 95%]
MakeType.test_sub_equivalent_pattern_should_succeed PASSED [ 96%]
MakeVisibility.test_make PASSED     [ 97%]
MakeVisibility.test_yaml_dump PASSED [ 98%]
MakeVisibility.test_yaml_load PASSED [100%]

========================== 92 passed in 0.44 seconds ==========================
```

Приложение Г Тестирование программы java_bytecode_model

Описание тестов

Пустой файл

Название теста: `test_one_file[empty.java]`

Ожидаемый результат:

```
!Model []
```

Построение модели класса

Входные данные:

```
class Class {}
```

Название теста: `test_one_file[class.java]`

Ожидаемый результат (файл `empty.yaml`):

```
!Model
- &id001 !PrimitiveType {name: void}
- !Class
  suppliers:
  - *id001
  - &id002 !Classifier {name: java.lang.Object}
  name: Class
- *id002
```

Построение модели перечисления

Название теста: `test_one_file[enumeration.java]`

Входные данные:

```
enum Enumeration {}
```

Ожидаемый результат:

```
!Model
- &id001 !Class
  generals:
```

```
- &id003 !Classifier {name: java.lang.Enum}
operations:
- !Operation
  owner: *id001
  result: &id005 !Type
    classifier: *id001
  visibility: !Visibility 'PUBLIC'
  is_static: true
  name: values
- &id002 !Operation
  owner: *id001
  result: !Type
    classifier: *id001
  visibility: !Visibility 'PUBLIC'
  is_static: true
  name: valueOf
  parameters:
  - !Parameter
    owner: *id002
    position: 1
    type: !Type
      classifier: &id008 !Classifier {name: java.lang.String}
    direction: !Direction 'IN'
- !Operation
  owner: *id001
  result: !Type
    classifier: &id004 !PrimitiveType {name: void}
  visibility: !Visibility 'PRIVATE'
  is_static: true
  name: <clinit>
suppliers:
- *id001
- *id003
- &id006 !Classifier {name: java.lang.Object}
```

```
  - &id007 !Classifier {name: java.lang.Class}
  - *id004
  - &id009 !PrimitiveType {name: int}
  name: Enumeration
  properties:
  - !Property
    owner: *id001
    visibility: !Visibility 'PRIVATE'
    is_static: true
    name: $VALUES
    type: *id005
- *id004
- *id003
- *id006
- *id007
- *id008
- *id009
```

Построение модели интерфейса

Название теста: `test_one_file[interface.java]`

Входные данные:

```
interface Interface {}
```

Ожидаемый результат:

```
!Model
- !Interface {name: Interface}
```

Построение модели абстрактного класса

Название теста: `test_one_file[abstract_class.java]`

Входные данные:

```
abstract class AbstractClass {}
```

Ожидаемый результат:

```
!Model
- &id001 !PrimitiveType {name: void}
- !Interface
  suppliers:
  - *id001
  - &id002 !Classifier {name: java.lang.Object}
  name: AbstractClass
- *id002
```

Построение модели двух классов, связанных обобщением (extends)

Название теста: `test_one_file[extends.java]`

Входные данные:

```
class Base {}
class Derived extends Base {}
```

Ожидаемый результат (файл `extends.yaml`):

```
!Model
- &id001 !PrimitiveType {name: void}
- !Class
  generals:
  - &id003 !Class
    suppliers:
    - *id001
    - &id002 !Classifier {name: java.lang.Object}
    name: Base
  suppliers:
  - *id001
  - *id002
  name: Derived
- *id002
- *id003
```

Построение модели двух интерфейсов и класса, связанных обобщением (implements)

Название теста: `test_one_file[implements.java]`

Входные данные:

```
interface InterfaceA {}
interface InterfaceB {}
class Implementation implements InterfaceA, InterfaceB {}
```

Ожидаемый результат (файл `implements.yaml`):

```
!Model
- &id001 !Interface {name: InterfaceA}
- &id002 !Interface {name: InterfaceB}
- &id003 !PrimitiveType {name: void}
- &id004 !Classifier {name: java.lang.Object}
- !Class
  generals:
  - *id001
  - *id002
  suppliers:
  - *id003
  - *id004
  name: Implementation
```

Построение модели иерархии обобщения из шести классов, на трех уровнях

Название теста: `test_one_file[hierarchy.java]`

Входные данные:

```
class Vehicle {}
class Automobile extends Vehicle {}
class Truck extends Vehicle {}
class Motorcycle extends Vehicle {}
class Convertible extends Automobile {}
class Sedan extends Automobile {}
```

Ожидаемый результат (файл `hierarchy.yaml`):

```
!Model
- &id001 !Class
  suppliers:
  - &id002 !PrimitiveType {name: void}
  - &id003 !Classifier {name: java.lang.Object}
  name: Vehicle
- !Class
  generals:
  - *id001
  suppliers:
  - *id002
  - *id003
  name: Motorcycle
- &id004 !Class
  generals:
  - *id001
  suppliers:
  - *id002
  - *id003
  name: Automobile
- !Class
  generals:
  - *id004
  suppliers:
  - *id002
  - *id003
  name: Convertible
- *id002
- !Class
  generals:
  - *id001
  suppliers:
```

```
  - *id002
  - *id003
  name: Truck
- *id003
- !Class
  generals:
  - *id004
  suppliers:
  - *id002
  - *id003
  name: Sedan
```

Построение модели класса со свойствами

Название теста: `test_one_file[class_with_properties.java]`

Входные данные:

```
class Class {
    public int publicProperty;
    protected int protectedProperty;
    private int privateProperty;
    public static int staticProperty;
}
```

Ожидаемый результат:

```
!Model
- &id001 !PrimitiveType {name: void}
- &id002 !Class
  suppliers:
  - &id004 !Classifier {name: java.lang.Object}
  - *id001
  name: Class
  properties:
  - !Property
    owner: *id002
```

```
    visibility: !Visibility 'PUBLIC'
    is_static: false
    name: publicProperty
    type: &id003 !Type
      classifier: &id005 !PrimitiveType {name: int}
  - !Property
    owner: *id002
    visibility: !Visibility 'PROTECTED'
    is_static: false
    name: protectedProperty
    type: *id003
  - !Property
    owner: *id002
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: privateProperty
    type: *id003
  - !Property
    owner: *id002
    visibility: !Visibility 'PUBLIC'
    is_static: true
    name: staticProperty
    type: *id003
- *id004
- *id005
```

Построение модели класса с методами

Название теста: `test_one_file[class_with_operations.java]`

Входные данные:

```
class Class {
    public void publicOperation() {}
    protected void protectedOperation() {}
    private void privateOperation() {}
```

```
    public void operationWithParameters(int x, float y) {}
    public static void staticOperation() {}
    public void overloadedOperation(int x) {}
    public void overloadedOperation(float x) {}
}
```

Ожидаемый результат:

```
!Model
- &id002 !PrimitiveType {name: void}
- &id001 !Class
  operations:
  - !Operation
    owner: *id001
    result: &id003 !Type
      classifier: *id002
    visibility: !Visibility 'PUBLIC'
    is_static: false
    name: publicOperation
  - !Operation
    owner: *id001
    result: *id003
    visibility: !Visibility 'PROTECTED'
    is_static: false
    name: protectedOperation
  - !Operation
    owner: *id001
    result: *id003
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: privateOperation
  - &id004 !Operation
    owner: *id001
    result: *id003
    visibility: !Visibility 'PUBLIC'
```

```
  is_static: false
  name: operationWithParameters
  parameters:
  - !Parameter
    owner: *id004
    position: 1
    type: &id006 !Type
      classifier: &id011 !PrimitiveType {name: int}
    direction: !Direction 'IN'
  - !Parameter
    owner: *id004
    position: 2
    type: &id008 !Type
      classifier: &id010 !PrimitiveType {name: float}
    direction: !Direction 'IN'
- !Operation
  owner: *id001
  result: *id003
  visibility: !Visibility 'PUBLIC'
  is_static: true
  name: staticOperation
- &id005 !Operation
  owner: *id001
  result: *id003
  visibility: !Visibility 'PUBLIC'
  is_static: false
  name: overloadedOperation
  parameters:
  - !Parameter
    owner: *id005
    position: 1
    type: *id006
    direction: !Direction 'IN'
- &id007 !Operation
```

```
    owner: *id001
    result: *id003
    visibility: !Visibility 'PUBLIC'
    is_static: false
    name: overloadedOperation
    parameters:
    - !Parameter
      owner: *id007
      position: 1
      type: *id008
      direction: !Direction 'IN'
  suppliers:
  - &id009 !Classifier {name: java.lang.Object}
  - *id002
  name: Class
- *id009
- *id010
- *id011
```

Построение модели зависимостей между классами

Название теста: `test_one_file[dependencies.java]`

Входные данные:

```
class SupplierLocalVariableType {}
class SupplierObjectConstructor {}

class Client {
    void operation() {
        SupplierLocalVariableType local;
        new SupplierObjectConstructor();
    }
}
```

Ожидаемый результат:

```
!Model
- !Class
  suppliers:
  - &id001 !Classifier {name: java.lang.Object}
  - &id002 !PrimitiveType {name: void}
  name: SupplierLocalVariableType
- &id004 !Class
  suppliers:
  - *id001
  - *id002
  name: SupplierObjectConstructor
- *id002
- *id001
- &id003 !Class
  operations:
  - !Operation
    owner: *id003
    result: !Type
      classifier: *id002
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: operation
  suppliers:
  - *id004
  - *id001
  - *id002
  name: Client
```

Построение модели вызовов методов внутри класса и между классами

Название теста: `test_one_file[invocations.java]`

Входные данные:

```
class A {
    void invoked() {}
    void invoke() {
        invoked();
    }
    void invokeB() {
        new B().invoked();
    }
}
class B {
    void invoked() {}
}
```

Ожидаемый результат:

```
!Model
- &id001 !Class
  operations:
  - &id003 !Operation
    owner: *id001
    result: &id002 !Type
      classifier: &id005 !PrimitiveType {name: void}
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: invoked
  - !Operation
    owner: *id001
    result: *id002
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: invoke
    invocations:
    - *id003
  - !Operation
    owner: *id001
```

```
    result: *id002
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: invokeB
    invocations:
    - &id004 !Operation
      owner: &id006 !Class
        operations:
        - *id004
        suppliers:
        - &id007 !Classifier {name: java.lang.Object}
        - *id005
        name: B
      result: *id002
      visibility: !Visibility 'PRIVATE'
      is_static: false
      name: invoked
  suppliers:
  - *id006
  - *id007
  - *id005
  name: A
- *id006
- *id005
- *id007
```

Построение модели вызова переопределенного метода

Название теста: `test_one_file[overridden_method_call.java]`

Входные данные:

```
interface Interface {
    abstract void operation();
}
class Implementation implements Interface {
```

```
    public void operation() {}
}
class Client {
    Interface x = new Implementation();
    void invokeOperation() {
        x.operation();
    }
}
```

Ожидаемый результат (файл `overridden_method_call.yaml`):

```
!Model
- &id002 !PrimitiveType {name: void}
- &id005 !Classifier {name: java.lang.Object}
- &id001 !Class
  operations:
  - !Operation
    owner: *id001
    result: &id004 !Type
      classifier: *id002
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: invokeOperation
    invocations:
    - &id003 !Operation
      owner: &id006 !Interface
        operations:
        - *id003
        name: Interface
      result: *id004
      visibility: !Visibility 'PUBLIC'
      is_static: false
      name: operation
  suppliers:
  - *id005
```

```
  - *id006
  - *id002
  - &id007 !Class
    generals:
    - *id006
    operations:
    - !Operation
      owner: *id007
      result: *id004
      visibility: !Visibility 'PUBLIC'
      is_static: false
      name: operation
    suppliers:
    - *id005
    - *id002
    name: Implementation
  name: Client
  properties:
  - !Property
    owner: *id001
    visibility: !Visibility 'PRIVATE'
    is_static: false
    name: x
    type: !Type
      classifier: *id006
- *id007
- *id006
```

Результат выполнения тестов

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.4.1, py-1.5.2, pluggy-0.6.0 -- /usr/bin/
cachedir: .pytest_cache
rootdir: /home/elsid/dev/master/src/test, inifile:
collecting ... collected 13 items
```

```
test_one_file[enumeration.java] PASSED           [  7%]
test_one_file[class_with_properties.java] PASSED [ 15%]
test_one_file[abstract_class.java] PASSED        [ 23%]
test_one_file[interface.java] PASSED             [ 30%]
test_one_file[class.java] PASSED                 [ 38%]
test_one_file[extends.java] PASSED               [ 46%]
test_one_file[hierarchy.java] PASSED             [ 53%]
test_one_file[overridden_method_call.java] PASSED [ 61%]
test_one_file[invocations.java] PASSED           [ 69%]
test_one_file[class_with_operations.java] PASSED [ 76%]
test_one_file[empty.java] PASSED                 [ 84%]
test_one_file[implements.java] PASSED            [ 92%]
test_one_file[dependencies.java] PASSED          [100%]

========================== 13 passed in 7.69 seconds ==========================
```

Приложение Д Тестирование программы match_pattern

Описание тестов

Поиск пустого шаблона в пустой модели

Название теста: `test_match_empty_in_empty`

Входные данные — файл `empty.yaml`.

Ожидаемый результат — пустой вывод.

Поиск шаблона BaseDerived в модели с одной связью обобщения

Название теста: `test_match_base_derived`

Входные данные — файл `extends.yaml`.

Ожидаемый результат:

```
class Base === classifier Base
class Derived === classifier Derived
```

Поиск шаблона BaseDerived в модели с множеством связей обобщения

Название теста: `test_match_all_base_derived_in_hierarchy`

Входные данные — файл `hierarchy.yaml`.

Ожидаемый результат:

```
class Automobile === classifier Base
class Convertible === classifier Derived

class Automobile === classifier Base
class Sedan === classifier Derived

class Automobile === classifier Derived
class Vehicle === classifier Base

class Convertible === classifier Derived
class Vehicle === classifier Base

class Motorcycle === classifier Derived
```

```
class Vehicle === classifier Base

class Sedan === classifier Derived
class Vehicle === classifier Base

class Truck === classifier Derived
class Vehicle === classifier Base
```

Поиск шаблона OverriddenMethodCall в соответствующей модели

Название теста: `test_match_overridden_method_call`

Входные данные — файл `overridden_method_call.yaml`.

Ожидаемый результат:

```
+Implementation::operation(): void === Implementation::operation()
+Interface::operation(): void === Interface::operation()
-Client::invokeOperation(): void === Client::invoke_operation()
class Client === classifier Client
class Implementation === class Implementation
interface Interface === interface Interface
```

Результат выполнения тестов

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.14, pytest-3.4.1, py-1.5.2, pluggy-0.6.0 -- /usr/bin/
cachedir: .pytest_cache
rootdir: /home/elsid/dev/master/src/test, inifile:
collecting ... collected 6 items

test_match_empty_in_empty PASSED                      [ 16%]
test_match_base_derived PASSED                        [ 33%]
test_match_overridden_method_call PASSED              [ 50%]
test_match_all_base_derived_in_hierarchy PASSED       [ 66%]
test_match_one_base_derived_in_hierarchy PASSED       [ 83%]
test_match_three_base_derived_in_hierarchy PASSED     [100%]
```

```
============================ 6 passed in 0.83 seconds ============================
```

Приложение Е Графы моделей шаблонов проектирования

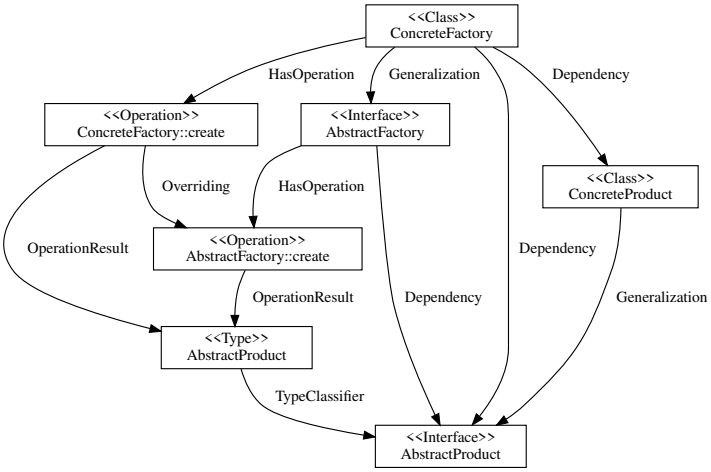

Рисунок Е.1 — Граф модели шаблона проектирования «абстрактная фабрика»

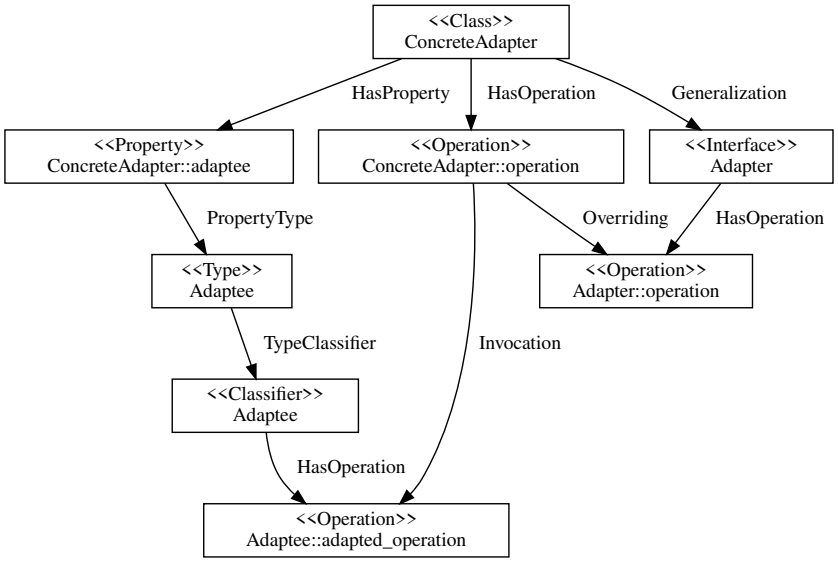

Рисунок Е.2 — Граф модели шаблона проектирования «абстрактная фабрика»

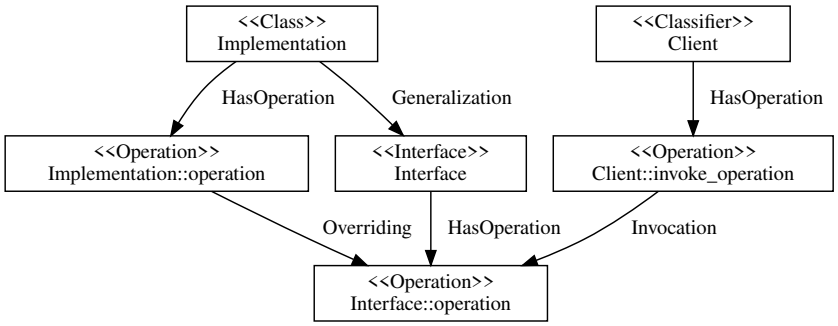

Рисунок Е.3 — Граф модели шаблона проектирования «вызов переопределённого метода»

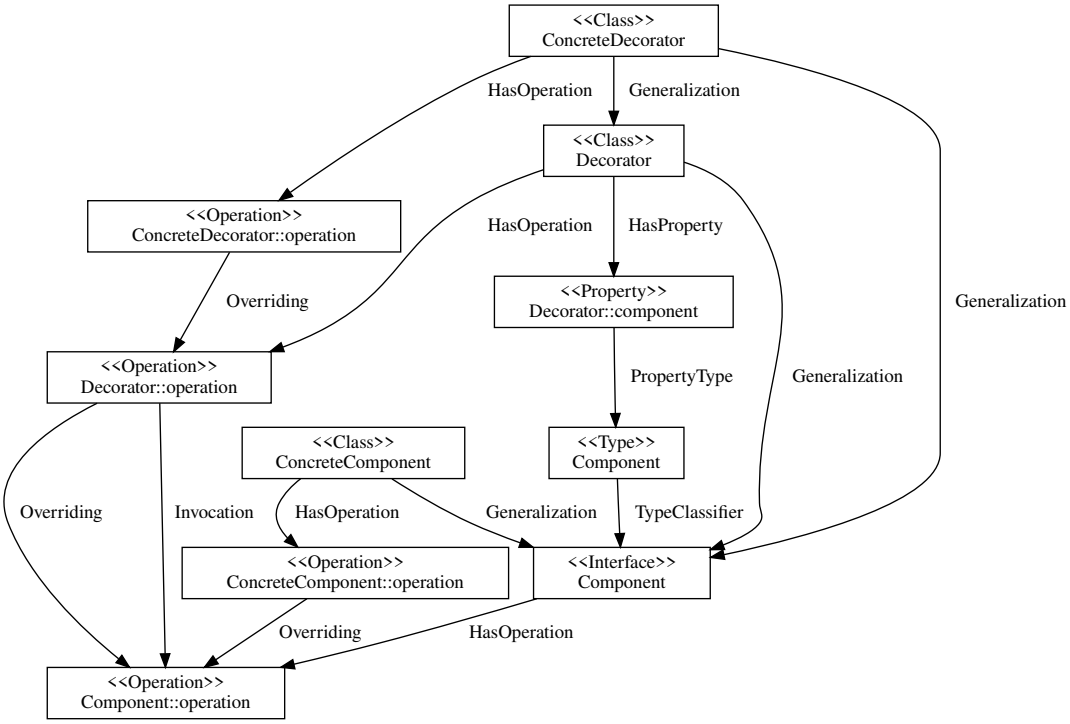

Рисунок Е.4 — Граф модели шаблона проектирования «декоратор»

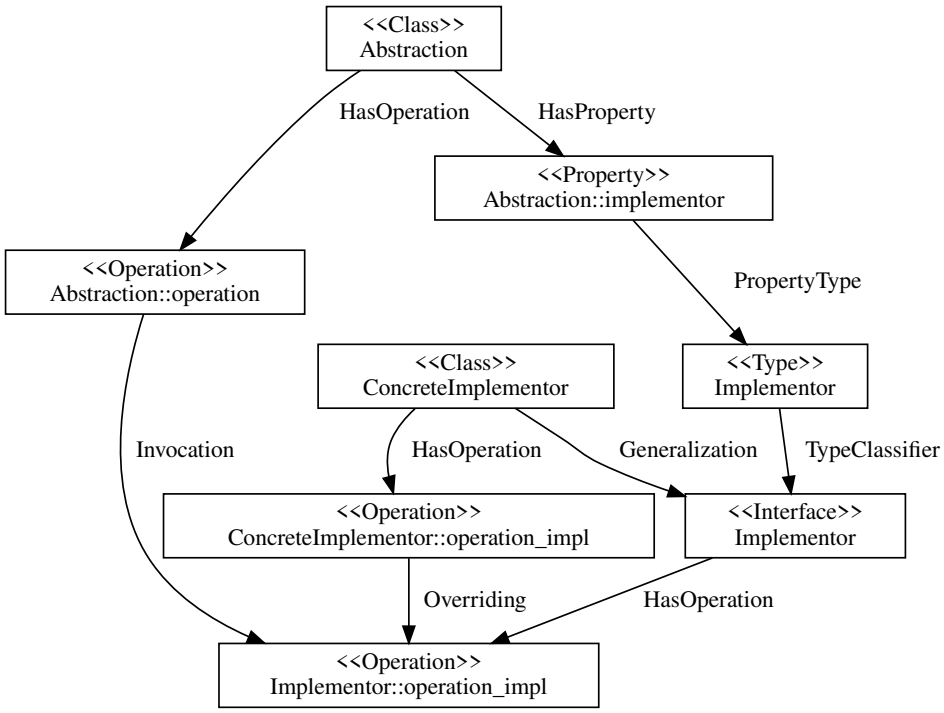

Рисунок Е.5 — Граф модели шаблона проектирования «мост»

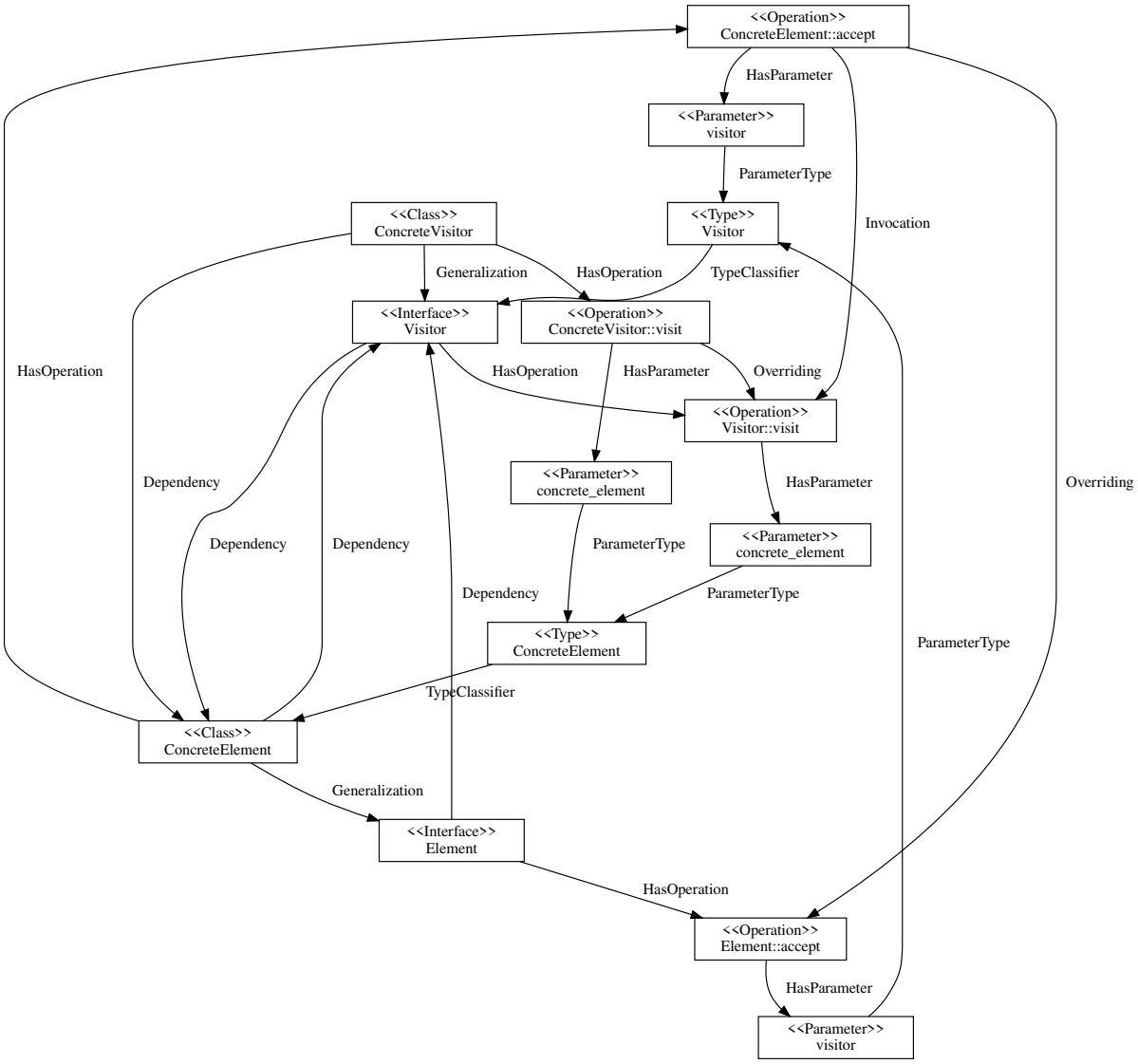

Рисунок Е.6 — Граф модели шаблона проектирования «посетитель»

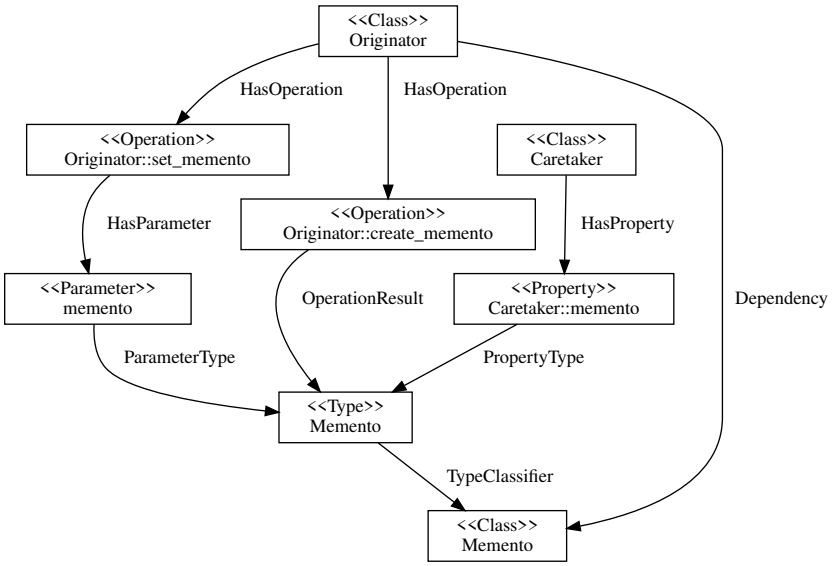

Рисунок Е.7 — Граф модели шаблона проектирования «хранитель»

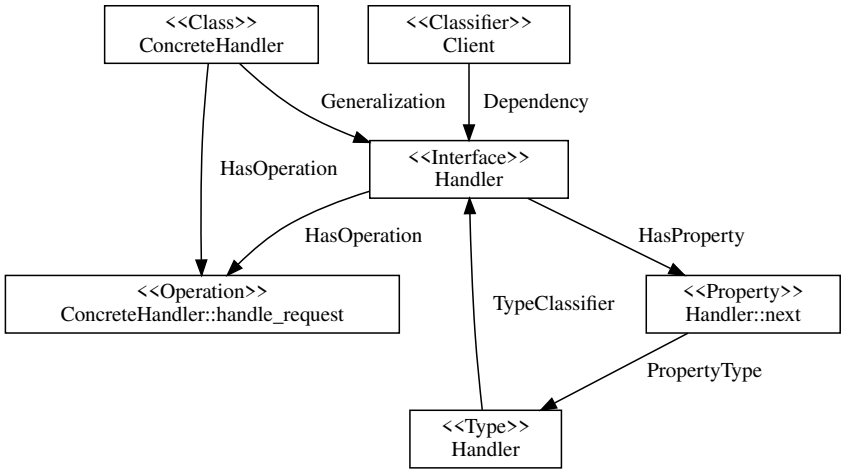

Рисунок Е.8 — Граф модели шаблона проектирования «цепочка ответственности»