УДК 004.05

С. А. Власов, В. В. Бураков

ПОДХОД К СПЕЦИФИКАЦИИ СТРУКТУРНОЙ МОДИФИКАЦИИ КОДА БОРТОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Представлен подход к спецификации структурных модификаций бортового программного обеспечения, направленных на повышение показателей надежности и безопасности. Подход основан на применении модификаций, описанных графовыми продукциями над графовым представлением исходного кода.

Ключевые слова: *рефакторинг, модификация, граф, продукция.*

К программному обеспечению, входящему в состав бортовых комплексов, традиционно предъявляются повышенные требования по надежности и безопасности. Помимо этого, к важным свойствам таких программ относят высокое качество, поддающееся проверке, непротиворечивость, возможность повторного использования, низкую стоимость технического обслуживания, быструю интеграцию с аппаратными средствами, возможность переноса на другие платформы [1]. Вместе с тем не менее важным представляется

обеспечение такого качества бортового программного обеспечения, как сопровождаемость. Хороший уровень сопровождаемости обеспечивается оптимальной структурой программы, для достижения которой используется рефакторинг. К процессу рефакторинга программного обеспечения бортовых систем, как и к самим программам, предъявляются повышенные требования по надежности и безопасности. Обеспечить эти требования в отсутствие непротиворечивого математического базиса представляется практически невозможным [2].

Рефакторинг исходного кода объектно-ориентированных программ заключается в изменении структуры кода в соответствии с заданными в системе шаблонами. Большую работу по созданию таких шаблонов провел М. Фаулер [3], составив один из наиболее полных каталогов шаблонов рефакторинга. Каталог состоит более чем из 120 преобразований, применение которых упрощает работу программиста.

Наиболее близкими, с функциональной точки зрения, аналогами разрабатываемой системы являются Eclipce IDE и плагин, разработанный Jet Brains к среде разработки Microsoft Visual Studio — ReSharper. Функциональность обоих программных решений включает поиск дефектов, метрический анализ исходного кода и инструмент рефакторинга. Эти системы умеют определять явные стилистические, структурные и поведенческие дефекты исходного кода, предлагая несколько вариантов их решения, а инструмент рефакторинга помогает быстро изменить структуру исходного кода, сохраняя ее работоспособность. Основным недостатком указанных систем является ограниченность набора шаблонов и невозможность их пополнения [4]. Существует несколько подходов к реализации систем с пополняемым набором шаблонов, их различие заключается в способе представления исходного кода, и как следствие — способе представления шаблонов. Наиболее подходящей (ввиду своей математической природы) является теория графов, а наглядность представления программных сущностей и их поведения положительно сказывается на простоте задания самих шаблонов рефакторига.

Модель представления исходного кода. Для описания программных сущностей и их взаимоотношений были выбраны графы, а для описания преобразований — графовые морфизмы. В рассматриваемом подходе это будет помеченный типизированный граф. Дадим несколько определений.

Ориентированный граф $G = (V,E,s,t)$ состоит из двух множеств, конечного множества V, элементы которого называются вершинами, и конечного множества E, элементы которого называются ребрами. Каждое ребро связано с упорядоченной парой вершин. Для обозначения вершин используются символы $v_1,v_2,v_3\ldots$, а для обозначения ребер — символы $e_1,e_2,e_3\ldots$ Если $e_1 = (v_i,v_j)$, то v_i, и v_j называются оконечными вершинами e_1, при этом v_i — начальная вершина, а v_j — конечная вершина e_1. Функции s: $E\rightarrow V$ и t: $E\rightarrow V$ связывают с каждым ребром в точности одну начальную и одну конечную вершины [1].

Помеченный граф. Пусть $L=(VL,EL)$, L-помеченный граф G представляет собой двойку (g,l) такую, что имеет место:

1) $g = (V,E,s,t)$ — граф;

2) $l=(vl:V\rightarrow VL,\ el:E\rightarrow EL)$ — пара функций пометки соответственно вершин и ребер, при этом vl является инъективной функцией.

Помеченный типизированный граф. Пусть $T=(VT,ET)$ — пара непересекающихся конечных множеств предопределенных типов вершин и ребер. L-помеченный T-типизированный граф G является двойкой $(g,type)$, такой что g является L-помеченным графом, а $type=(vt:V\rightarrow VT,\ et:E\rightarrow ET)$ — пара функций, vt связывает с каждой вершиной из V ее тип из VT, а et — с каждом ребром из E его тип из ET.

Подграф. H является подграфом G (обозначается $H \subseteq G$) если существует инъективный графовый морфизм $m:H\rightarrow G$, называемый соответствием H в G.

Иными словами, каждая вершина и каждая дуга имеют соответственно тип как обязательный атрибут и, по необходимости, имя; каждая вершина связана с дугой, заканчивающейся другой вершиной. В графе не допускаются петли и связь двух вершин более чем одной дугой (параллельные дуги).

В качестве примера рассмотрим следующий участок исходного кода.

```
class Phone
{
   public:
 void dialNumber(int number);
 bool busy;
 Device dial;
};

public: void Phone::dialNumber(int number)
{
   dial->getDevice();
   busy = dial->line(number);
}
```

Вершины характеризуют такие программные сущности, как классы, методы, параметры. Примеры типов вершин приведены в табл. 1.

Таблица 1

Тип	Описание	Пример
Cl	Класс	class Phone
Ty	Тип	void
Va	Переменная	busy
Si	Сигнатура метода	dialNumber(int number)
Pa	Параметр	number
Bo	Тело метода	dial->getDevice(); …
Ex	Выражение	busy = dial->line(number);
Vi	Область видимости	public

Дуги графа характеризуют отношения между двумя вершинами, такие как наследование, связь переменной с типом, область видимости и т.д. На рис. 1 изображен граф, соответствующий представленному выше исходному коду. Таким образом, ввиду отсутствия ограничений на типы дуг и вершин и их конечное число можно говорить о полноте, или адекватности, данной графовой модели исходному коду. Пример дуг представлен в табл. 2.

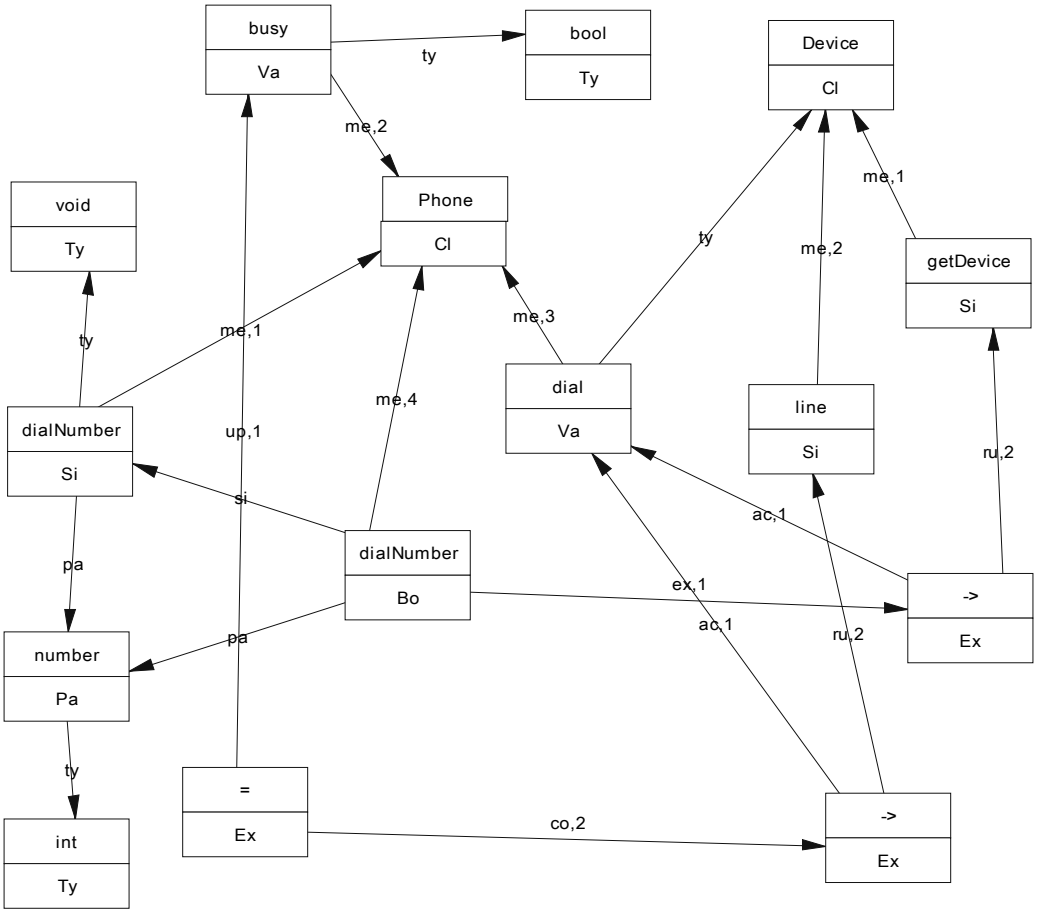

Рис. 1

Таблица 2

Тип	Описание
si:Si→Bo	Связь сигнатуры и тела метода
ge:Cl→Cl	Наследование для классов
me:Bo→Cl	Принадлежность метода классу
me:Va→Cl	Принадлежность поля классу
ty:Pa→Cl	Классовый тип формального параметра
ty:Pa→Ty	Стандартный тип формального параметра
ty:Va→Ty	Стандартный тип переменной
pa:Si→Pa	Формальный параметр
pa:Ex→Ex	Фактический параметр
ex:Bo→Ex	Выражение в теле метода
co:Ex→Ex	Подвыражение
ru:Ex→Si	Вызов метода
ac:Ex→Va	Доступ к переменной
up:Ex→Va	Модификация переменной

Программная реализация описанного выше графа была создана в рамках разрабатываемой системы модификации исходного кода для среды разработки Microsoft Visual Studio IDE. Как и большинство программных решений этого типа, система выполнена в виде плагина к среде разработки.

Основные функции системы:

— построение графовой модели исходного кода проекта, открытого в среде разработки;

— исполнение рефакторингов на графовой модели исходного кода;

— преобразование измененной графовой модели исходного кода в текст программы;

— ведение пополняемого каталога шаблонов рефакторинга;

— средство визуального моделирования шаблонов рефакторинга.

Спецификация преобразований исходного кода. Для описания преобразований (рефакторингов) используется теория перезаписи (перезаписывания) графов. Рефакторинг состоит из набора графовых продукций, представляющих собой логически законченные этапы преобразования исходного кода в соответствии с сутью самого преобразования.

Продукция p:($L \rightarrow R$) состоит из имени продукции p и инъективного частичного графового морфизма r, называемого морфизмом продукции. Графы L и R — соответственно левая и правая части продукции.

При перезаписи графа в левой части продукции описываются те вершины и ребра, которые должны присутствовать в исходном графе, а в правой — вид этой части графа после применения продукции. Частичный морфизм описывает отношения объектов левой части продукции L к правой R. Соблюдается ряд правил:

— объекты, не описанные морфизмом, удаляются;

— объекты правой части продукции, не имеющие прообраза в левой части, создаются заново;

— объекты, не изменяемые морфизмом, формируют контекст продукции.

Любой рефакторинг имеет ряд ограничений, или условий применимости к конкретному исходному коду. Например, переименование переменной не должно создать переменную с одинаковым именем в одной и той же области видимости имен, приводя тем самым к ошибке. Таким образом, к набору продукций добавляются условия, описанные одиночными, недопустимыми графами. Присутствие вершин и ребер недопустимого графа в графе исходного кода показывает невозможность выполнения данного рефакторинга, соответственно поиск недопустимых графов в графе исходного кода производится до выполнения преобразований.

Для практического применения графовое описание рефакторинга должно иметь обобщенный характер, т.е. отвечать двум критериям:

— не зависеть от таких деталей кода, как имена программных сущностей, их тип, и порядка следования операторов исходного кода;

— поддерживать множественную параметризацию.

Для обеспечения множественности входных параметров в качестве входных данных также выбран граф. Пользователь выделяет участок кода, который в преобразованном виде поступает на вход и служит шаблоном для заполнения графов рефакторинга. Первый критерий обеспечивается путем выделения трех категорий имен вершин и дуг:

— вершины, заполняемые из графа входных параметров;

— вершины, требующие дополнительного ввода (например, имени новой переменной при переименовании);

— вершины, логическим значением которых является „все“.

Для простоты задания в имя вводится расширение „%“ или „@“, или „*“. В свою очередь, сами имена, дополненные данным расширением, являются абстрактными обозначениями, определяющими соответствие между графами продукций, самими продукциями и графами условий. Вершины, имеющие подобные обозначения, заполняются в соответствии с графом входных параметров.

Пример спецификации преобразования кода. В качестве примера рассмотрим распространенный рефакторинг „выделение метода“. Суть данного рефакторинга заключается в выделении части кода, в основном самодостаточной, вынесении ее в новый метод, а также формировании списка формальных и фактических параметров.

Создание нового метода. Первым преобразованием необходимо создать заготовку для нового метода, т.е. описать прототип и тело метода, пока пустое. Левая и правая части этого преобразования представлены на рис. 2.

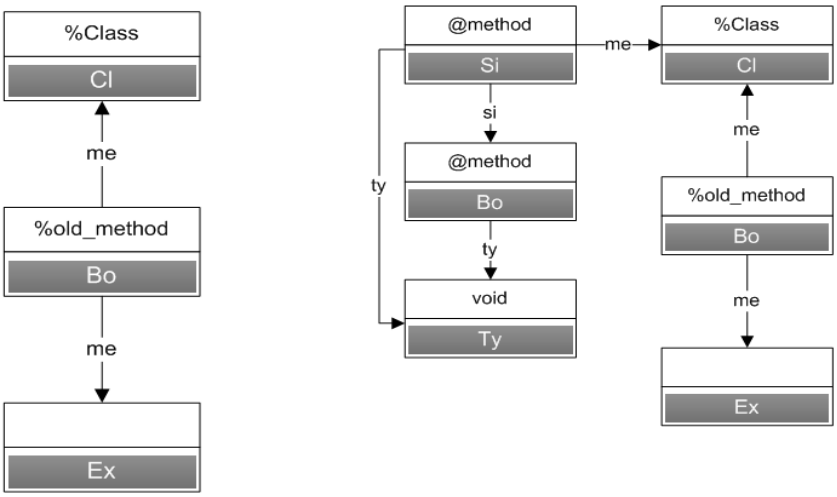

Рис. 2

Морфизм, определенный данной продукцией, создает в графе исходного кода вершины сигнатуры и реализации метода с именем @method и типом void. Связь вершин %old_method с вершиной %Class в обеих частях продукции определяет класс местоположения нового метода (в том же классе, что и оригинальный метод).

Перемещение кода в новый метод. На следующем шаге необходимо переместить выделенный пользователем исходный код в новый метод. Данное преобразование достаточно просто реализовать, поскольку в принятом описании вершин и дуг для обозначения строчки кода используется последовательность вершин *Bo*:*Ex*→*Ex*. Таким образом, каждая строка кода представляется набором выражений, соединенных одной дугой с вершиной „тело метода“ (*Bo*). Следовательно, для перемещения кода в графе исходного кода необходимо для каждой вершины (подходящей для преобразования) произвести преобразование (рис. 3).

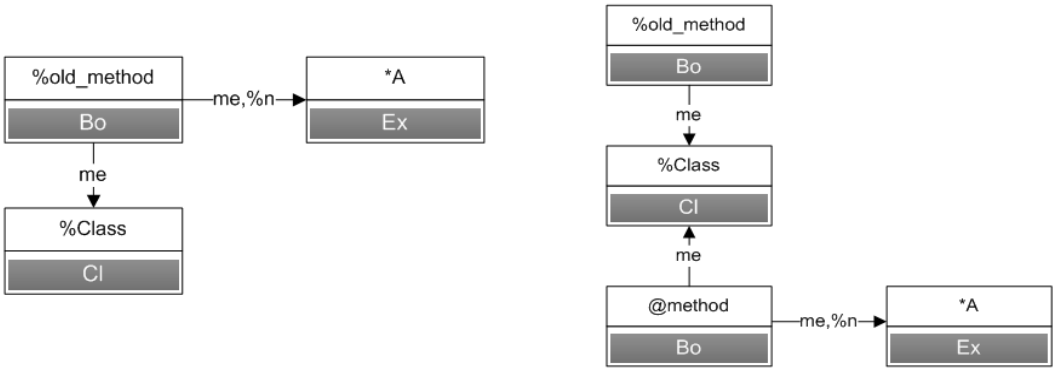

Рис. 3

В левой части продукции есть отношение *me*:*Bo*→*EX*, обозначающее одно выражение в теле метода (одну строку кода). Имя вершины %old_method заполняется на этапе выполнения и принимает значение имени метода оригинального расположения кода. Вершина с именем **A* обозначает необходимость выполнить данное преобразование для всех вершин такого типа из графа входных параметров. В правой части дуга перемещена из вершины старого метода в вершину нового.

Добавление параметров функции. Последним преобразованием необходимо создать формальные и фактические параметры нового метода и подставить его вызов в нужное место. Для этого выполним преобразование, показанное на рис. 4.

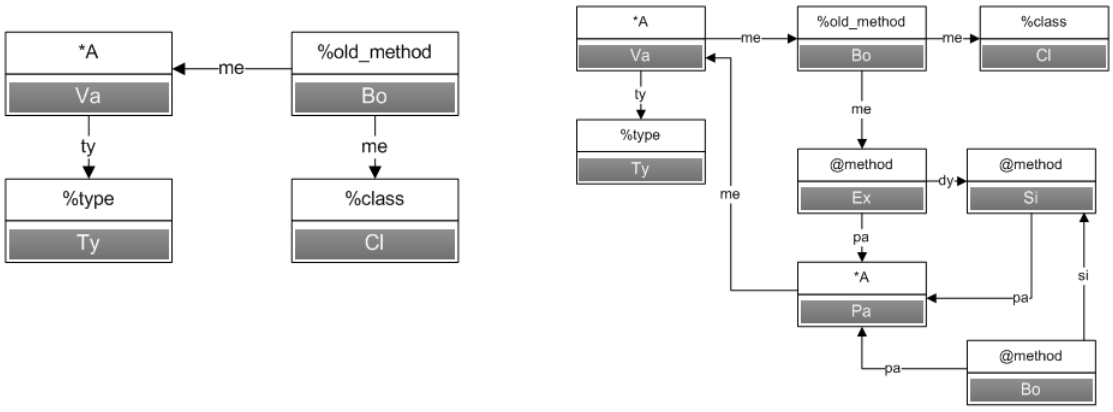

Рис. 4

Для сохранения работоспособности программы необходимо все переменные из перемещаемого кода проверить на локальную принадлежность к старому методу и вынести их в список параметров нового метода. В левой части фактически определяются локальные переменные (отношение *me*:*Bo*→*Va*). В правой — для найденных вершин создаются новые отношения: *pa*:*Si*→*Pa* (формальный параметр метода), *me*:*Pa*→*Va* (связь вершины формального параметра и вершины переменной), *pa*:*Bo*→*Pa* (фактический параметр метода).

В заключение нужно отметить, что представленный в этой статье пример рефакторинга можно отнести к сложно описываемым, но даже для его воплощения потребовалось всего три набора продукций. Данный подход в полной мере пригоден для описания любых структурных преобразований, он позволяет создавать группировки преобразований и новые (например, исправляющие стилистику кода) преобразования. Описанный подход является важным элементом системы контроля и улучшения качества программного обеспечения, способен обеспечить необходимую математическую основу для создания системы автоматизации преобразований, гарантировать требуемые уровни безопасности и надежности бортового программного обеспечения.

СПИСОК ЛИТЕРАТУРЫ

1. *Титов А.* Все, что вы хотели спросить о сертификации бортового программного обеспечения, но боялись узнать // Компьютерра. 2007. № 45.

2. *Бураков В. В.* Управление качеством программных средств. СПб: СПбГУАП, 2009. 206 с.

3. *Фаулер М.* Каталог паттернов рефакторинга 2010 [Электронный ресурс]: <refactoring.com>.

4. Рефакторинг с использованием шаблонов. М.: Вильямс, 2008. 510 с.

Сведения об авторах

Станислав Александрович Власов — аспирант; Санкт-Петербургский государственный университет аэрокосмического приборостроения, кафедра компьютерной математики и программирования; E-mail: zeronetlog@gmail.com

Вадим Витальевич Бураков — канд. техн. наук, доцент; Санкт-Петербургский государственный университет аэрокосмического приборостроения, кафедра компьютерной математики и программирования; E-mail: burakov@aanet.ru

Рекомендована ГУАП

Поступила в редакцию
04.04.11 г.