СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.4’2

РЕФАКТОРИНГ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРОГРАММ НА ОСНОВЕ ТЕОРИИ LP-СТРУКТУР

С. Д. Махортов, А. А. Ногих

Воронежский государственный университет

Поступила в редакцию 25.03.2019 г.

Аннотация. Рассматриваются вопросы автоматизации рефакторинга программ, составленных на основе парадигмы объектно-ориентированного программирования. В целях формализации рефакторинга используется теория LP-структур, предоставляющая эффективные алгебраические модели для различных систем в информатике. LP-структуры на решетке типов позволяют, по меньшей мере, формализовать проведение рефакторинга методом подъема общих атрибутов и проведение оптимизации иерархии типов. В статье обсуждаются границы применимости данного подхода, а также предлагается способ обобщения LP-структур на решетке типов с целью их использования для проведения рефакторинга более широкого класса программных систем.

Ключевые слова: рефакторинг, ООП, иерархия типов, LP-структуры, автоматизация, инструментальные средства разработки.

ВВЕДЕНИЕ

Разработка программной системы – лишь часть ее жизненного цикла. С течением времени система совершенствуется, адаптируется к новым требованиям, вынужденно уходит от изначально спроектированной структуры. Этот процесс неизбежно сопровождается ухудшением качества кода, что делает дальнейшие изменения системы все более сложными.

Остановить деградацию кода помогает рефакторинг. Это процесс такого изменения программной системы, при котором ее внешнее поведение сохраняется, а внутренняя структура улучшается [1].

В общем случае рефакторинг состоит из множества этапов, а именно – выбор частей системы и способов преобразования, оценка того, насколько при этом сохранится поведение модифицируемых частей, проведение изменений, оценка полученного результата и т. д. [2]. Часть этих этапов могут выполнять автоматизированные системы рефакторинга, тем самым позволяя существенно снизить затраты на разработку ПО.

В работе [3] показано, что математической основой решения задач рефакторинга может служить теория LP-структур, предоставляющая эффективные алгебраические модели для различного рода систем в информатике. Описанный подход позволяет проводить автоматическую оптимизацию иерархии типов программной системы, включая устранение дублирования кода путем «подъема» общих атрибутов по иерархии типов («Pull Up Field» в терминологии [1]). В силу того, что при использовании LP-структур на решетке типов атрибуты рассматриваются как часть иерархии типов, данный класс обладает более широкими возможностями адекватного моделирования иерархий типов по сравнению с теорией FCA (Formal Concept Analysis [4]).

Настоящая работа развивает подход, предложенный в [3]. Рассматривается ряд вопросов, возникающих при реализации подобной

инструментальной системы автоматизации рефакторинга, включая границы применимости данной методики, учет семантики атрибутов иерархии типов, особенности модификации кода программной системы.

Подходы, представленные в настоящей работе, не привязаны к конкретным языкам программирования и опираются лишь на базовые понятия методологии объектно-ориентированного программирования.

1. ОСНОВНЫЕ ПОНЯТИЯ

Ниже кратко представлены некоторые необходимые для дальнейшего изложения определения и результаты из статьи [3].

Под LP-структурой подразумевается алгебраическая система, представляющая собой математическую решетку $\mathbb{F}(\leq,\wedge,\vee)$ с дополнительно заданным на ней бинарным отношением $\leftarrow$, на которое накладываются следующие ограничения:

• транзитивность: из $a \leftarrow b$ и $b \leftarrow c$ следует $a \leftarrow c$ $(a,b,c \in \mathbb{F})$;

• дистрибутивность, в случае данной работы – $\vee$-дистрибутивность: из $b_1 \leftarrow a$ и $b_2 \leftarrow a$ следует $b_1 \vee b_2 \leftarrow a$;

• включение тавтологий: $\leq \subseteq \leftarrow$.

Между парами типов (классов) в объектно-ориентированном программировании существует как минимум два вида связей – наследование и агрегация.

В работе [3] множество типов и связи, соответствующие наследованию, моделирует решетка типов $\mathbb{F}$: если тип b является наследником a, то $b \leq a$. В результате рассмотрению подлежат лишь такие иерархии типов, для которых после введения указанного частичного порядка алгебраическая система $\mathbb{F}$ удовлетворяет определению решетки. Аналогичное ограничение действует и в настоящей статье.

Дополнительно на решетке $\mathbb{F}$ вводится бинарное отношение R, соответствующее агрегации: если тип b агрегирует объект типа a, то $(b,a) \in R$. Отметим, что $(\mathbb{F},R)$ еще не является LP-структурой, поскольку при таком построении нельзя гарантировать, что R будет удовлетворять требованиям определения LP-структуры.

Оба введенных отношения ($\leq$ и R) в предметной области имеют общую семантику – один тип получает возможности другого типа в виде доступа к его атрибутам. Семантически ясно, что такое отношение «обладания набором возможностей» (далее обозначено как $\xleftarrow{R}$) обязано быть рефлексивным и транзитивным.

Понятие $\vee$-дистрибутивности отношения $\xleftarrow{R}$ задает семантику монотонного логического вывода. При наличии $\vee$-дистрибутивности из $b_1 \xleftarrow{R} a$ и $b_2 \xleftarrow{R} a$ следует $b_1 \vee b_2 \xleftarrow{R} a$. Это означает, что если тип b_1 обладает возможностями a и b_2 обладает возможностями a, то и наименьший общий предок b_1 и b_2 также обладает возможностями типа a. Если поместить атрибут a в $b_1 \vee b_2$, то b_1 и b_2 получат его в порядке наследования, что будет соответствовать рефакторингу методом подъема общих атрибутов.

Однако в случае LP-структур на решетке типов выполнение $\vee$-дистрибутивности в ряде случаев нецелесообразно и должно зависеть от контекста. Иначе возможны ситуации, нежелательные с точки зрения качества кода [3]. Например, тип b будет обладать возможностями своего потомка a; тип b будет обладать возможностями a одновременно по нескольким линиям наследования и т. д. С учетом таких ситуаций логический вывод возможностей типов в LP-структуре теряет свою монотонность.

Для формализации и исследования немонотонного логического вывода на рассмотренных структурах были введены понятия $\vee$-совместимости пар $(b_1,a),(b_2,a) \in R$, $\vee$-дистрибутивности тройки (c_1,c_2,a), где $c_1,c_2,a \in \mathbb{F}$ и некоторые другие понятия, с которыми можно детально ознакомиться в [3].

Логическое замыкание отношения R на решетке $\mathbb{F}$ представляет собой отношение, содержащее все такие пары (a,b), что в типе b доступны возможности типа a (с учетом выполнения ограниченной $\vee$-дистрибутивности).

Логической редукцией отношения R на решетке $\mathbb{F}$ называется минимальное эквива-

лентное ему отношение R_0. Логическая редукция рассматриваемой LP-структуры соответствует иерархии типов с минимальным эквивалентным набором связей, что способствует минимизации дублирования кода.

В работе [3] доказаны теоремы о существовании логической редукции и логического замыкания для LP-структур на решетке типов, также обоснованы способы их построения.

2. ПОСТАНОВКА ЗАДАЧИ

Несмотря на то, что семантика элементов LP-структуры на решетке типов четко сформулирована, преобразование в нее структуры классов реальной программной системы не является тривиальной задачей.

Описанный в статье [3] подход опирается на следующие предположения.

• Экземпляры одного типа могут использоваться для «обладания возможностями» такого типа полностью взаимозаменяемо. То есть, предполагается, что все обращения к одному экземпляру можно заменить на обращения к другому экземпляру того же типа безо всяких последствий для работоспособности программы.

• Любым атрибутом можно всегда воспользоваться для «обладания возможностями» агрегируемого типа.

Очевидно, что далеко не каждая иерархия типов удовлетворяет таким ограничениям. Однако в случаях, когда не удается провести полноценную оптимизацию иерархии типов, можно, по крайней мере, попробовать выполнить подъем общих атрибутов.

В данной статье рассматриваются возможности распространения методики, описанной в [3], на более широкий класс иерархий типов. При этом для иерархий, удовлетворяющих указанным выше ограничениям, предлагается сохранение имеющегося подхода.

В работе [3] для отражения агрегации используется бинарное отношение на решетке типов. На практике это может повлечь сложности, обусловленные следующими обстоятельствами.

• Бинарное отношение на решетке типов не позволяет полноценно представлять иерархии типов, в которых одним типом агрегируются несколько экземпляров другого типа.

• Отсутствует возможность учета различной семантики конкретных экземпляров типов.

В качестве иллюстрации второго пункта можно привести следующую ситуацию, когда отсутствие возможности контроля над семантикой экземпляров типов в результате рефакторинга может привести к ухудшению качества кода.

Рассмотрим UML-диаграмму классов, показанную на рис. 1. Изображенные классы описывают передаваемые сообщения в некоторой многопользовательской системе. Сообщения могут поступать от одного пользователя другому («PersonalMessage») и быть публичными, имеющими только автора («PublicMessage»).

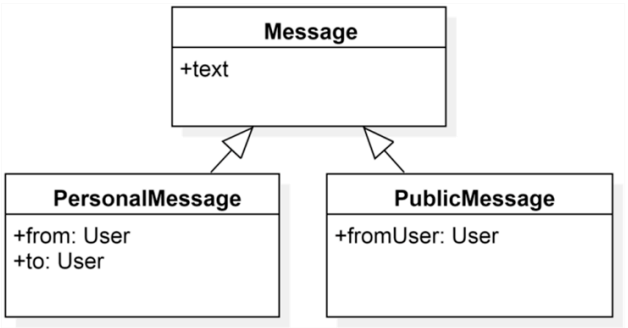

Рис. 1. UML-диаграмма классов, иллюстрирующая иерархию типов, для которой возможно осуществление подъема общих атрибутов

В случае, когда применением метода подъема общих атрибутов поля «to» из PersonalMessage и «fromUser» из PublicMessage будут перемещены в класс Message в виде одного общего поля, работоспособность программы сохранится. Однако такое изменение сделает код программы менее понятным для разработчика.

Для решения указанных проблем предлагается переключить фокус с абстрактного «обладания возможностями» типа на конкретные случаи агрегации типов данных. Для этого потребуется расширить понятие LP-структуры на решетке типов. Далее нужно предоставить способ преобразования иерар-

хии типов в такую LP-структуру и описать процесс проведения рефакторинга при помощи данной модели.

3. РАСШИРЕННАЯ LP-СТРУКТУРА НА РЕШЕТКЕ ТИПОВ

Пусть T – множество типов рассматриваемой системы.

Случаем агрегации назовем кортеж $(T_1, T_2, name, I)$, где $T_1 \in T$ – тип, содержащий в качестве атрибута экземпляр типа $T_2 \in T$, *name* соответствует идентификатору атрибута, а I представляет некоторую прочую информацию о данном атрибуте. Для краткости введем отображения $\text{agg\_source}(A)$ и $\text{agg\_target}(A)$, которые случаю агрегации A сопоставляют агрегирующий (T_1) и агрегируемый (T_2) типы соответственно. Будем обозначать множество всех случаев агрегации в рассматриваемой программной системе как T_{agg}.

Требуется дополнить определение LP-структуры на решетке типов для достижения следующих целей:

- возможность моделирования агрегации одним типом нескольких экземпляров другого типа;
- возможность задания ограничений на подъем в общий тип-предок конкретных случаев агрегации.

Для того чтобы в рамках LP-структуры допустить моделирование ситуаций, когда один тип агрегирует несколько экземпляров другого типа, предлагается внести в иерархию типов дополнительные «искусственные» типы. Такие типы будут представлять в модели случаи агрегации (точнее, их множества), что позволит сохранить возможность описания агрегации при помощи бинарного отношения на решетке типов.

Далее рассматривается формирование пары $(\mathbb{F}, R)$, используемой в [3] для построения LP-структуры на решетке типов. LP-структура, построенная с использованием таких пар, будет называться *расширенной*.

Для формирования $(\mathbb{F}, R)$ потребуются:

- определенные ранее множества T и T_{agg};
- множество искусственных типов $\tilde{T}$;
- отображение случаев агрегации на искусственные типы, которые будут «представлять» эти случаи агрегации в расширенной LP-структуре f_{agg}: $T_{agg} \to \tilde{T}$.

Предлагается следующий способ построения $(\mathbb{F}, R)$.

1. Формируется решетка типов $\mathbb{F}$, соответствующая отношению наследования аналогично [3].

2. В решетку добавляются элементы $\tilde{T}$. В силу особенностей постановки задачи в [3] (отсутствие всяких предположений о свойствах типов и атрибутов, а также проведение только подъема общих атрибутов), такие элементы не связываются отношением частичного порядка с элементами T.

3. Бинарное отношение агрегации R строится следующим образом: для каждого случая агрегации $t_{agg} \in T_{agg}$ в R добавляется пара $(x, \tilde{t})$, где $\tilde{t} = f_{agg}(t_{agg})$, а $x = \text{agg\_source}(t_{agg})$.

Отметим несколько свойств построенной алгебраической системы.

1. Появляется возможность моделирования агрегации одним типом нескольких экземпляров другого типа. Для этого достаточно обеспечить, чтобы для таких $t_{agg}^{(1)}, t_{agg}^{(2)} \in T_{agg}$, что $\text{agg\_target}\left(t_{agg}^{(1)}\right) = \text{agg\_target}\left(t_{agg}^{(2)}\right)$ и $\text{agg\_source}\left(t_{agg}^{(1)}\right) = \text{agg\_source}\left(t_{agg}^{(2)}\right)$, выполнялось $f_{agg}\left(t_{agg}^{(1)}\right) \neq f_{agg}\left(t_{agg}^{(2)}\right)$.

2. Появляется возможность контроля над подъемом атрибутов. Условия $f_{agg}\left(t_{agg}^{(1)}\right) \neq f_{agg}\left(t_{agg}^{(2)}\right)$ достаточно, чтобы избежать подъема случаев агрегации $t_{agg}^{(2)}$ и $t_{agg}^{(1)}$ в общий тип-предок при проведении логической редукции [3].

3. Зафиксируем некоторый $t \in \mathbb{F}$. Пусть $\tilde{T}_t = \{\tilde{t} \in \tilde{T} \mid (t', \tilde{t}) \in R, t \le t'\}$ – подмножество элементов $\tilde{T}$, связанных бинарных отношением с t либо с его предками. Пусть также R_0 – отношение, полученное в результате проведения логической редукции[4] LP-структуры, построенной на $(\mathbb{F}, R)$, а $\tilde{T}_t^0 = \{\tilde{t} \in \tilde{T} \mid (t', \tilde{t}) \in R_0, t \le t'\}$. В силу того, что элементы $\tilde{T}$ при данном построении не связаны отношением частичного порядка с элементами T, в ходе логической редукции может происходить только подъем атрибутов. Следовательно, справедливо $\tilde{T}_t = \tilde{T}_t^0$.

4. ПОСТРОЕНИЕ РАСШИРЕННОЙ LP-СТРУКТУРЫ НА РЕШЕТКЕ ТИПОВ

Задача расширения LP-структуры состоит в том, чтобы в результате проведения ее логической редукции получалась корректная иерархия типов, а также была возможность проведения преобразования кода в соответствии с полученными результатами. При построении такой алгебраической системы нужно учесть ряд требований и ограничений.

• Недопустимость подъема в общий тип-предок атрибутов, имеющих различную семантику использования.

• Отсутствие неоднозначности при построении расширенной LP-структуры на решетке типов и интерпретации полученных результатов. В частности, это означает, что все случаи агрегации должны быть однозначно различимы.

В настоящей работе предлагается учитывать такие ограничения путем формирования отношения эквивалентности на множестве случаев агрегации T_{agg}. Для $a,b \in T_{agg}$ возможность подъема в общий тип-предок будет рассматриваться только тогда, когда $a \sim b$. От такого отношения эквивалентности потребуем выполнения следующих условий.

1. Если $a,b \in T_{agg}$ и $a \sim b$, то $\text{agg\_target}(a) = \text{agg\_target}(b)$. Это очевидное требование, не допускающее совмещения атрибутов, относящихся к различным типам.

2. Если $a,b \in T_{agg}$ и $a \sim b$, то a и b имеют семантику, достаточно схожую для допущения их подъема в общий тип-предок.

3. Если $a,b \in T_{agg}$ и $a \sim b$, то $\text{agg\_source}(a) \neq \text{agg\_source}(b)$, а также типы $\text{agg\_source}(a)$ и $\text{agg\_source}(b)$ не должны иметь общих потомков. Данное требование гарантирует, что для каждого $t \in T$ все случаи агрегации t_{agg} такие, что $\text{agg\_source}(t_{agg}) = t$, будут относиться к различным классам эквивалентности.

Требования 1 и 3 имеют строгую математическую формулировку, в то время как требование 2 нуждается в дальнейшей формализации. Без учета свойств конкретных языков программирования предложить точный алгоритм и оценить, насколько он справляется с данной задачей, не представляется возможным. Следовательно, построение алгоритма для определения схожести семантики отдельных атрибутов находится за пределами настоящего исследования. Пока предположим, что эта задача уже решена, и обозначим ее результат как бинарное отношение $R_{eq} \subseteq T_{agg} \times T_{agg}$ такое, что если $(a,b) \in R_{eq}$, то семантика a и b допускает поднятие данных атрибутов в тип-предок.

Рассмотрим подробнее процесс построения искомого отношения эквивалентности ~. Требования 1–3 не описывают его полностью, а лишь задают ограничения, которым такое отношение должно удовлетворять. С целью допустить подъем как можно большего числа атрибутов будем преследовать цель минимизации числа классов эквивалентности. В такой постановке исходная задача приобретает вид задачи о раскраске графа.

Напомним, что правильной k-раскраской графа $G = (V,E)$ является отображение $c: V \to \{1,2,...,k\}$, для которого верно $\{u,v\} \in E \Rightarrow c(u) \neq c(v)$. Для заданного графа наименьшее k, при котором такая раскраска возможна, называется его хроматическим числом, и обозначается $\chi(G)$. Оптимальная раскраска графа – такая правильная раскраска, для которой требуется ровно $\chi(G)$ цветов [5].

Построим граф (V,E), для которого решение задачи об оптимальной раскраске позволит решить задачу поиска искомого отношения эквивалентности. Требование 1 изначально разделяет случаи агрегации на непересекающиеся подмножества. Поэтому построение графа (V,E) и проведение его оптимальной раскраски будем выполнять независимо для каждого $t \in T$ выполнением следующих шагов.

1. Выделим подмножество $\overline{T}_{agg} \subseteq T_{agg}$ такое, что $\forall t_{agg} \in \overline{T}_{agg} : \text{agg\_target}(t_{agg}) = t$.

2. Заполним V вершинами, изоморфными элементам $\overline{T}_{agg}$. Обозначим биекцию между элементами V и $\overline{T}_{agg}$ как $w: \overline{T}_{agg} \leftrightarrow V$.

3. Для каждых $a,b \in \overline{T}_{agg}$: если $(a,b) \notin R_{eq}$, то добавляем $\{w(a), w(b)\}$ в E.

4. Для каждых $a,b \in \overline{T}_{agg}$: если $\text{agg\_source}(a) = \text{agg\_source}(b)$ либо

$\text{agg\_source}(a)$ и $\text{agg\_source}(b)$ имеют по крайней мере один общий тип-потомок, то добавляем $\{w(a), w(b)\}$ в E.

Решением задачи об оптимальной раскраске графа является отображение $c: V \to \{1, 2, ..., k\}$. Если разбить $\overline{T}_{agg}$ на подмножества элементов t_{agg}, имеющих одинаковое значение $c(w(t_{agg}))$, то такое разбиение и будет искомым в силу того, что при построении графа (V, E) ребрами были соединены все пары вершин, которые не должны были оказаться в общем классе эквивалентности.

Как известно, задача оптимальной раскраски графа является NP-полной [5]. Однако в силу того, что после разбиения по $\text{agg\_target}(t_{agg})$ количество случаев агрегации (и, следовательно, количество изоморфных им вершин в графе) может быть невелико, экспоненциальная сложность алгоритма может не являться препятствием для его использования. В ином случае потребуются алгоритмы, предоставляющие приближенные решения задачи о раскраске графа за полиномиальное время.

После нахождения рассмотренного выше отношения эквивалентности ~ становится возможным построение расширенной LP-структуры на решетке типов. Описанный в разделе 3 подход в качестве входных данных требует множества T, T_{agg}, а также множество искусственных типов $\tilde{T}$ и отношение $f_{agg}: T_{agg} \to \tilde{T}$. При этом множества T и T_{agg} были определены ранее и остаются неизменными, а построение $\tilde{T}$ и f_{agg} описано ниже.

1. Множество $\tilde{T}$ заполняется элементами, изоморфными классам эквивалентности отношения ~. Обозначим биекцию между ними как $g: T_{agg} / \sim \leftrightarrow \tilde{T}$, где $T_{agg} / \sim$ – фактор-множество отношения эквивалентности ~.

2. Для каждого случая агрегации $t_{agg} \in T_{agg}$ пусть $t_{agg} / \sim$ – класс эквивалентности, в который входит t_{agg}. Тогда $f_{agg}(t_{agg}) = g(t_{agg} / \sim)$.

В силу требования 1 к отношению эквивалентности на T_{agg}, для такого построения справедливо $f_{agg}(t_{agg}^{(1)}) = f_{agg}(t_{agg}^{(2)}) \Rightarrow \text{agg\_target}(t_{agg}^{(1)}) = \text{agg\_target}(t_{agg}^{(2)})$. Это соотношение позволяет определить вспомогательное отображение $f_{orig}: \tilde{T} \to T$ следующим образом. Если $f_{orig}(\tilde{t}) = t$, то существует $f_{agg}(t_{agg}) = \tilde{t}$ и $t = \text{agg\_target}(t_{agg})$. То есть $t = f_{orig}(\tilde{t})$ – это тип, вместо которого в отношении агрегации на решетке типов участвует $\tilde{t}$.

При описанном в данном разделе построении множеств случаи агрегации, имеющие различную семантику, окажутся разнесенными в разные классы эквивалентности. Следовательно, для таких случаев $t_{agg}^{(1)}$ и $t_{agg}^{(2)}$ справедливо $f_{agg}(t_{agg}^{(1)}) \neq f_{agg}(t_{agg}^{(2)})$, что исключает формальный подъем таких атрибутов в общий тип-предок.

Кроме того, при данном построении все атрибуты, которыми тип владел напрямую или же которые получил в порядке наследования, оказываются в различных классах эквивалентности, и им ставятся в соответствие разные элементы $\tilde{T}$. Таким образом, для каждой пары $(t, \tilde{t}) \in R$ можно однозначно указать случай, соответствующий агрегации исходной иерархии типов. Такая однозначность (в плане соответствия случаев агрегации старой и новой иерархий) сохранится и после логической редукции построенной LP-структуры в силу свойства 3 из предыдущего раздела статьи.

5. ПРОВЕДЕНИЕ РЕФАКТОРИНГА

В общем виде предлагаемый процесс рефакторинга состоит из следующих шагов.

1. Обработка исходного кода программной системы, формирование множества типов T и множества случаев агрегации T_{agg}.

2. Построение расширенной LP-структуры на решетке типов. Проведение логической редукции полученной LP-структуры так, как описано в [3].

3. Применение изменений к исходному коду программной системы.

Для подмножеств типов и атрибутов исходной системы, соответствующих предположениям модели [3], предлагается проводить не только подъем общих атрибутов, но и оптимизацию иерархии типов. При этом LP-структура будет строиться не для всей иерархии типов, а только для подходящей

ее части, и весь процесс построения такой LP-структуры и проведения изменений будет происходить в точности как описано в [3].

Первый шаг существенно зависит от конкретного языка программирования. Построение расширенной LP-структуры было описано в разделе 4. Третий же шаг подразумевает следующие два вида изменений в программном коде.

• Модификация описания классов

• Замена обращений к полям классов. При подъеме атрибутов в тип-предок в общем случае не требуется, чтобы такие атрибуты имели одинаковые названия. В результате атрибут в типе-предке может получить название, отличающееся от тех, по которым он был доступен ранее.

Для осуществления таких изменений потребуется сформировать множество T'_{agg} случаев агрегации новой иерархии типов, а также построить отображение $f : T_{agg} \to T'_{agg}$ между исходной иерархией типов и результирующей. Далее описан способ получения T'_{agg} и f.

Пусть R_0 – отношение, полученное в результате проведения логической редукции LP-структуры. Для каждого $(t, \tilde{t}) \in R_0$ добавим в T'_{agg} случай агрегации $t'_{agg} = (T_1, T_2, name, I)$, где $T_1 = t$ и $T_2 = f_{orig}(\tilde{t})$. Способ определения *name* и I выходит за рамки настоящей работы и должен быть рассмотрен отдельно. Если же существует $t_{agg} \in T_{agg}$ такой, что $\text{agg\_source}\left(t_{agg}\right) = t$ и $f_{agg}\left(t_{agg}\right) = \tilde{t}$, это означает, что атрибут остался на прежнем месте, и можно добавить такой t_{agg} в T'_{agg} без изменений.

Теперь рассмотрим вопрос формирования отображения $f : T_{agg} \to T'_{agg}$. Для этой цели зафиксируем произвольный случай агрегации $t_{agg} \in T_{agg}$.

Пусть при построении расширенной LP-структуры на решетке типов этому случаю агрегации соответствует пара $(t, \tilde{t}) \in R$. Обозначим R_0 отношение, полученное в результате проведения логической редукции отношения R. В соответствии с предлагаемым подходом, в новой иерархии типов рассматриваемому случаю агрегации будет соответствовать любой из описанных моделью $R_{cand} = \{(t', \tilde{t}) \in R_0 \mid t \le t'\}$.

Необходимо учитывать следующие факты, порождаемые изложенным подходом.

• До проведения логической редукции элемент $(t, \tilde{t}) \in R$ однозначно представляет в модели случай агрегации t_{agg}.

• В ходе проведения логической редукции расширенной LP-структуры, описанной в настоящей работе, единственным изменением иерархии типов является подъем атрибутов и совмещение их в типе-предке.

• Логическая редукция LP-структуры на решетке типов гарантирует, что если $(a, b) \in R$ выполняется до ее проведения, то пара (a, b) нетранзитивна в $R_0 \bigcup \le$. Это имеет место в силу способа формирования множества неконфликтных $\vee$-дистрибутивных троек [3].

Из перечисленных свойств следует, что описанное ранее множество R_{cand} может состоять только из одного элемента. При построении T'_{agg} были добавлены случаи агрегации, соответствующие каждому элементу R_0. Пусть единственному элементу R_{cand} соответствует случай агрегации t'_{agg}, тогда положим $f\left(t_{agg}\right) = t'_{agg}$.

ЗАКЛЮЧЕНИЕ

Для иерархий типов, где в силу особенностей классов и атрибутов невозможно провести полную оптимизацию на основе модели [3], был предложен подход, позволяющий производить автоматизированный подъем общих атрибутов (рефакторинг «Pull Up Field»). Даже при таком вынужденном отказе от полной отпимизации иерархии типов сохраняется «интеллектуальность» исходной методики, проявляющаяся в избегании конфликтных ситуаций осуществления подъема атрибутов (см. подробнее в [3]). В результате преодолевается ограниченность исходной модели – открывается возможность проведения указанного рефакторинга всей иерархии типов. Затем можно осуществить оптимизацию той ее части, свойства которой позволяют это сделать.

Предложенный в настоящей работе подход может быть положен в основу автомати-

зированной инструментальной системы рефакторинга. Для реализации такой системы также потребуется решение следующих задач.

• Считывание иерархии типов из исходного кода программы.

• Выбор способа формирования имен для совмещенных в общем предке атрибутов.

• Способ определения того, достаточно ли схожа семантика атрибутов для осуществления их подъема в общий тип-предок.

• Модификация исходного кода программы: обновление определений типов и замена обращений к атрибутам.

Поскольку перечисленные задачи могут существенно зависеть от конкретного языка программирования, их решение в данном общем исследовании не рассматривалось.

Дальнейшие исследования в данном направлении подразумевают решение указанных выше задач, а также реализацию подобной инструментальной системы рефакторинга.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00037.

СПИСОК ЛИТЕРАТУРЫ

1. *Фаулер, М.* Рефакторинг: улучшение существующего кода / М. Фаулер. – Пер. с англ. – СПб. : Символ-Плюс, 2003. – 432 с.

2. *Mens, T.* A survey of software refactoring / T. Mens, T. Tourwe // Software Engineering IEEE Transactions. – 2005. – Vol. 30. – P. 126–139.

3. *Махортов, С. Д.* LP-структуры на решетках типов и некоторые задачи рефакторинга / С. Д. Махортов // Программирование. – 2009. – Т. 35, № 4. – С. 5–14.

4. *Godin, R.* Formal Concept Analysis-Based Class Hierarchy Design in Object-Oriented Software Development / R. Godin, P. Valtchev // Formal Concept Analysis / eds. B. Ganter, G. Stumme, R. Wille // Lecture Notes In Computer Science. – Springer Berlin/Heidelberg. – 2005. – Vol. 3626. – P. 304–323.

5. *Kosowski, A.* Classical coloring of graphs / Adrian Kosowski, Krzysztof Manuszewski // Contemporary Mathematics. – 2004. – Vol. 352. – P. 1–19.

Махортов Сергей Дмитриевич – д-р физ.-мат. наук, заведующий кафедрой программирования и информационных технологий, Воронежский государственный университет.
E-mail: msd_exp@outlook.com

Ногих Александр Алексеевич – магистрант 2 года обучения кафедры программирования и информационных технологий факультета компьютерных наук Воронежского государственного университета.
E-mail: a.nogikh@yandex.ru

APPLICATION OF LP STRUCTURES THEORY TO OBJECT-ORIENTED CODE REFACTORING

S. D. Makhortov, A. A. Nogikh

Voronezh State University

Annotation. The article explores problems that arise during the automatization of the object-oriented software refactoring. The described approach is based on the application of the Theory of LP Structures that provides lattice-based algebraic structures for describing logic production systems and systems that can be considered as such. Earlier it was shown that LP Structures on type lattices can be used to formalize the process of object-oriented software refactoring, including redundant attributes removal and the relocation of identical attributes into their common superclasses ("Pull Up Field" technique). The preconditions for employing LP Structures on type lattices are studied. An approach is suggested that extends the applicability of these algebraic structures to facilitate the refactoring process of a wider range of software systems.
Keywords: refactoring, object-oriented programming, type hierarchy, LP structures, refactoring automatization, software development tools.

Makhortov S. D. – Head of the Programming and Information Technologies Department, Doctor of Science, Voronezh State University, Voronezh, Russian Federation.
E-mail: msd_exp@outlook.com

Nogikh A. A. – 2nd year master's student of the Programming and Information Technologies Department, Computer Sciences Faculty, Voronezh State University.
E-mail: a.nogikh@yandex.ru