УДК 004.052

ВНЕСЕНИЕ ИЗМЕНЕНИЙ В АВТОМАТНЫЕ ПРОГРАММЫ

П.В. Федотов, О.Г. Степанов

Описан подход к внесению изменений в автоматные программы, позволяющий уменьшить число модификаций, которые могут привести к появлению ошибок. Подход основан на применении рефакторингов автоматных программ – последовательности небольших эквивалентных преобразований, сохраняющих поведение,. Предложен ряд рефакторингов автоматных программ, расширяющий набор классических рефакторингов на случай автоматного программирования.

Ключевые слова: автоматное программирование, автоматы, внесение изменений, рефакторинг.

Введение

Какой бы продуманной ни была архитектура программной системы, при изменении требований часто приходится модифицировать ее код. При произвольном изменении программы велик риск внесения ошибок. Одним из распространенных приемов, используемых при модификации кода, является рефакторинг – полное или частичное преобразование структуры программы с сохранением ее поведения [1]. Примерами рефакторингов в объектно-ориентированных программах являются переименование класса, выделение интерфейса, перемещение метода и т.д. Каждый вид рефакторинга реализует определенное изменение структуры программы, состоящее из набора небольших и технически простых шагов, позволяющих сделать процесс внесения изменений контролируемым.

Автоматные программы имеют свою специфику, и применение большей части стандартных рефакторингов в этих программах невозможно. Таким образом, при изменении требований автоматная программа должна быть спроектирована «с нуля» на основе новых требований. Этот процесс является неоправданно трудоемким, особенно в случае, когда изменения требований незначительны. Альтернативой является бессистемное внесение изменений в разработанную программу, что может привести к возникновению ошибок. Для решения описанной проблемы предлагается ряд рефакторингов автоматных программ и метод, позволяющий упростить процесс внесения изменений в автоматные программы и уменьшить число модификаций, которые могут привести к появлению ошибок.

Классификация изменений автоматных программ

С появлением большого числа программ с явным выделением состояний возникает необходимость в поддержке таких программ. В существующую программу вносят изменения следующих типов:

- изменения в программе в соответствии с изменившимися требованиями к ней;
- исправление ошибок;
- изменения в программе, имеющие целью облегчить понимание ее работы и упростить модификацию, не затрагивая наблюдаемого поведения.

По аналогии с существующим понятием рефакторинга [1] объектно-ориентированных программ будем называть изменения последнего типа, применяемые к программам с явным выделением состояний, рефакторингом автоматных программ.

Часто изменения, вносимые в программу с явным выделением состояний, достаточно сложны, а потому порождают массу проблем, плохо поддающихся анализу. С другой стороны, существует набор базовых изменений, которые являются примитивными изменениями какой-то одной составляющей графа переходов автомата. Такие изменения достаточно хорошо поддаются описанию. Остальные, более сложные, составные изменения можно представить в виде набора базовых изменений. В отдельный класс выделяются рефакторинги автоматов. Как было сказано выше, рефакторинги не меняют поведение программы и применяются для улучшения ее структуры.

Базовые изменения автоматов

На основе базовых изменений строятся более сложные сценарии реконфигурации автомата. Базовые изменения могут быть деструктивными и нарушать полноту и непротиворечивость графа переходов, а также семантические свойства. Конкретные проблемы, на которые стоит обратить внимание, при осуществлении базовых изменений рассмотрены в работе [2].

Базовые изменения автоматов:

- добавление состояния;
- удаление состояния;
- установка начального состояния;
- снятие начального состояния;
- добавление конечного состояния;
- удаление конечного состояния;
- добавление перехода;

- изменение события на переходе;
- изменение условия на переходе;
- удаление перехода;
- перемещение перехода:
  - изменение начального состояния;
  - изменение конечного состояния.

Рефакторинг автоматных программ

Описывается каталог рефакторингов автоматных программ. Выбор рефакторингов для каталога основан на наборе экспериментов. В качестве основы были взяты несколько автоматов, созданных студентами кафедры «Компьютерные технологии» СПбГУ ИТМО в ходе выполнения курсовых работ по курсу «Применение автоматов в программировании». Далее производились изменения требований к автомату, основанные на анализе предметной области. В соответствии с изменениями требовалось изменить структуру автомата. При этом часто даже простые изменения спецификации требовали значительной модификации графа переходов, так как его исходная структура оказывалась не приспособленной к внедряемому изменению. Для обеспечения совместимости структуры автомата и изменения спецификации требовалось произвести рефакторинг – модифицировать структуру автомата таким образом, чтобы изменения, вызванные изменившейся спецификацией, естественно вписались в новую структуру. На основе таких изменений и был разработан нижеприведенный каталог рефакторингов автоматных программ.

Группировка состояний

Описание. Несколько простых состояний объединяются в группу состояний. При этом добавляются групповые переходы, заменяющие одинаковые переходы, исходящие из всех группируемых состояний. Состояния, объединяемые в группу, могут иметь по несколько одинаковых переходов, в этом случае добавляется несколько групповых переходов.

Мотивация. Часто автомат имеет несколько состояний, которые имеют одинаковые переходы в другие состояния автомата. В этом случае можно выделить группу состояний, упростив структуру автомата.

Пример. Рассмотрим фрагмент графа переходов автомата «Панель в кабине лифта», отвечающий за выключение ламп в кнопках. Автомат имеет следующие пять состояний:

0. кнопки погашены;
1. светится «1»;
2. светится «2»;
3. светится «3»;
4. светится «S».

При наступлении события *e* («Выключение лампы в кнопке») в каждом из состояний 1–4 автомат должен перейти в состояние 0. Такое поведение реализуется графом переходов (рис. 1).

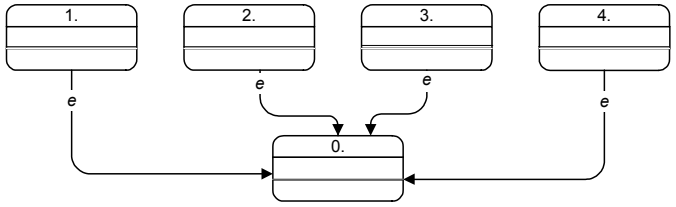

Рис. 1. Граф переходов без группировки состояний

Так как переходы из состояний 1–4 идентичны, их можно сгруппировать и соответствующим образом изменить конфигурацию автомата (рис. 2).

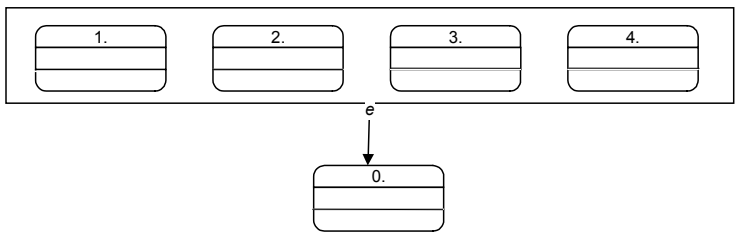

Рис. 2. Граф переходов с группировкой состояний

Техника. Для объединения состояний s_1, s_2,..., s_k в группу необходимо выполнить следующие действия.

1. Добавить группу *g*, объединяющую состояния s_1, s_2,..., s_k.
2. Выбрать один переход *t*, исходящий из s_1, который требуется заменить групповым.
3. Убедиться, что каждое из состояний s_2,..., s_k имеет переход с такими же атрибутами, что и *t* (под атрибутами понимаются конечное состояние, событие, условие и выходные воздействия).
4. Добавить переход, исходящий из группы *g* и имеющий те же атрибуты, что и *t*.
5. Удалить переходы, отмеченные в пунктах 2, 3.
6. Для оставшихся переходов, которые нужно заменить групповыми, повторить шаги 2–6.

Удаление группы состояний

Описание. При удалении группы состояний все исходящие из него переходы добавляются в состояния, находящиеся в группе, после чего сгруппированные состояния выносятся из группы.

Мотивация. Такое изменение полезно для последующей модификации автомата, если одно из состояний, входящее в удаляемую группу, изменяет логику поведения.

Техника. Для удаления группы *g*, объединяющей состояния s_1, s_2,..., s_k, необходимо выполнить следующие действия.

1. Выбрать переход *t*, исходящий из группы *g*.
2. Добавить переходы, исходящие из состояний s_1, s_2, ..., s_k, с атрибутами перехода *t*.
3. Удалить переход *t*.
4. Для оставшихся переходов, исходящих из группы *g*, повторить шаги 1–4.
5. Удалить группу *g*.

Слияние состояний

Описание. Несколько состояний с одинаковыми исходящими переходами сливаются в одно состояние. При этом сохраняются переходы, соединяющие эти состояния с другими состояниями автомата.

Производить слияние состояний можно только в том случае, если одинаковы атрибуты переходов, исходящих из этих состояний, и одинаковы воздействия, вызываемые при входе в сливаемые состояния.

Мотивация. В процессе изменения программы может оказаться, что некоторые состояния дублируют логику поведения программы. В этом случае эти состояния могут быть заменены одним состоянием.

Пример. Проиллюстрируем применение рефакторинга «слияние состояний» на примере «Панель в кабине лифта», предложенном в разделе «Группировка состояний», с добавлением следующих состояний:

5. перегорела лампа в кнопке «1»;
6. перегорела лампа в кнопке «2»;
7. перегорела лампа в кнопке «3»;
8. перегорела лампа в кнопке «S»;
9. неисправность.

Измененный граф переходов представлен на рис. 3.

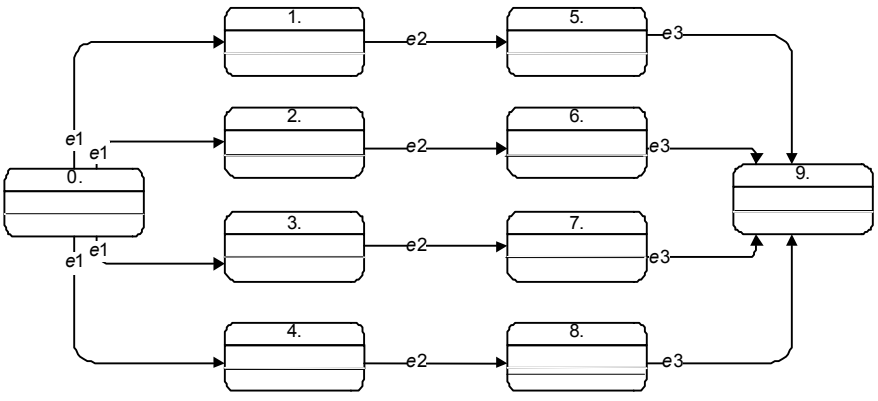

Рис. 3. Граф переходов автомата управления лифтом до слияния состояний 5–8

Граф переходов автомата после слияния состояний 5–8 в одно состояние представлен на рис. 4.

Техника. Для слияния состояний s_1, s_2,..., s_k необходимо выполнить следующие действия.

1. Убедиться, что состояния s_1, s_2,..., s_k имеют одинаковые воздействия, вызываемые при входе в состояние.

2. Убедиться, что состояния s_1, s_2,..., s_k имеют исходящие переходы с одинаковыми атрибутами и что эти переходы заканчиваются в одном и том же состоянии.
3. Изменить конечные состояния переходов, которые ведут в состояния s_2,..., s_k, на состояние s_1.
4. Удалить состояния s_2,..., s_k.

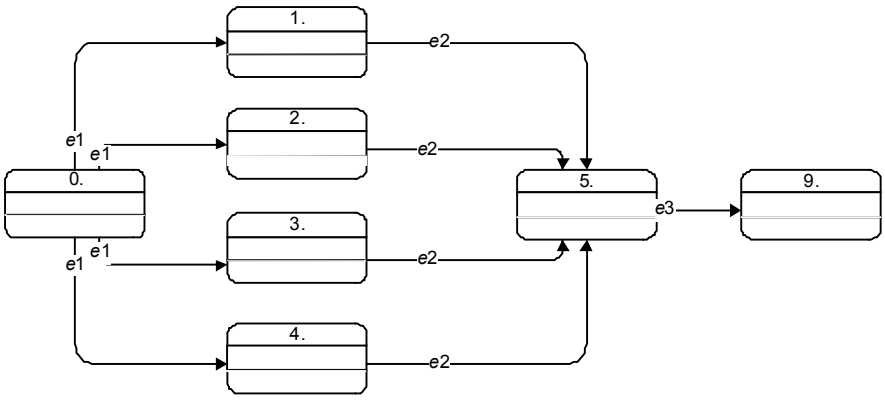

Рис. 4. Граф переходов после слияния состояний

Выделение автомата

Описание. Часть логики программы переносится в отдельный вызываемый автомат (рис. 5).

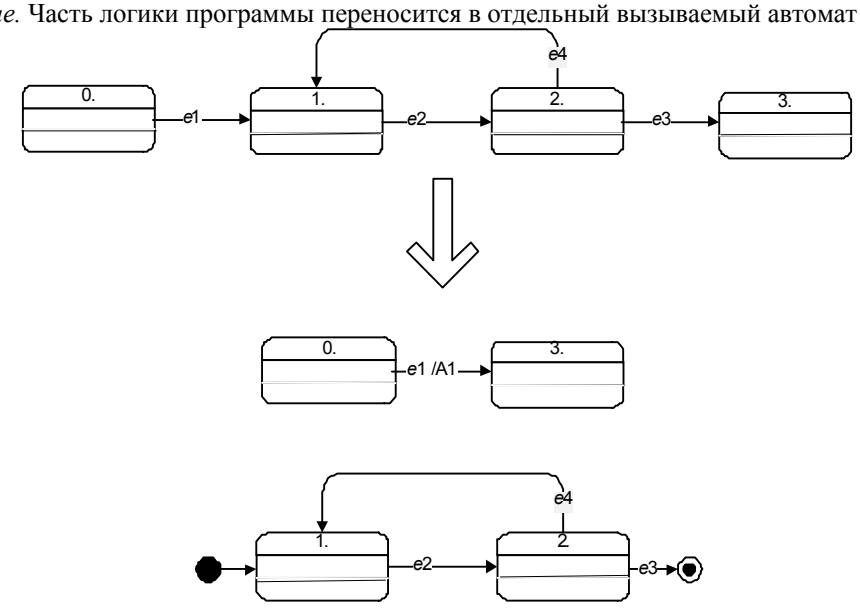

Рис. 5. Выделение автомата

Мотивация. Большие автоматы с множеством состояний, реализующие сразу несколько функций программы, сложны в поддержке. При внесении изменений в работу одной из функций необходимо учитывать реализацию других функций. Понимание работы такого автомата усложнено тем, что протоколы взаимодействия частей автомата, отвечающих за реализацию различных функций, не определены четко. Соответственно, поддержка сложных автоматов может привести к постоянному возникновению ошибок, обнаружить и исправить которые гораздо сложнее, чем в программах, в которых различные функции реализованы отдельными автоматами.

Для решения этой проблемы сложный автомат разделяют на несколько более простых, для каждого из которых четко определены обязанности и протоколы взаимодействия с другими автоматами и остальной программой. Такое разделение называют автоматной декомпозицией [3]. Автоматная декомпозиция обычно производится на этапе анализа требований, когда происходит построение изначального набора автоматов, которые требуется реализовать в программе. Однако часто в ходе эволюции програм-

мы один из автоматов накапливает реализацию чрезмерно большой функциональности. В этом случае требуется выполнить автоматную декомпозицию в уже существующей программе.

Техника. Для выделения состояний s_1, s_2,..., s_k и исходящих из них переходов t_1, t_2,..., t_l в вызываемый автомат необходимо выполнить следующие действия.

1. Убедиться, что среди состояний s_1, s_2,..., s_k есть ровно одно такое состояние s_i, в которое ведет хотя бы один переход из состояния, не входящего в множество s_1, s_2,..., s_k. Обозначим отмеченный переход как t', а его начальное состояние – s'.
2. Убедиться, что из отмеченного состояния s_i достижимы все остальные состояния выбранного подмножества.
3. Убедиться, что переходы t_1, t_2,..., t_l имеют в качестве конечных состояний только состояния из множества s_1, s_2,..., s_k и ровно одно состояние s'', не входящее в это множество.
4. Создать новый автомат.
5. Создать в новом автомате состояния s'_1, s'_2,..., s'_k, соответствующие состояниям s_1, s_2,..., s_k, с сохранением выходных воздействий.
6. Добавить переходы между состояниями s'_1, s'_2,..., s'_k с такими же атрибутами, что и переходы между состояниями s_1, s_2,..., s_k.
7. Объявить состояние s'_i начальным в созданном автомате.
8. Выбрать среди состояний s_1, s_2,..., s_k такие, из которых есть переходы в s''. Пусть в созданном автомате им соответствуют состояния s'_{p_1}, s'_{p_2},…, s'_{p_m}. Добавить из состояний s'_{p_1}, s'_{p_2},…, s'_{p_m} переходы, ведущие в конечное состояние созданного автомата. Присвоить этим переходам такие же атрибуты, как у соответствующих переходов изначального автомата.
9. Добавить переход между состояниями s' и s'' с атрибутами перехода t' и вызовом созданного автомата.
10. Удалить состояния s_1, s_2,..., s_k.

Встраивание вызываемого автомата

Описание. Вызываемый автомат встраивается в места своего вызова на переходах. Рефакторинг является обратным к предыдущему.

Мотивация. Размещением всей логики поведения программы в одном графе переходов можно добиться большей наглядности, так как такой граф можно охватить «одним взглядом».

Техника выполнения данного рефакторинга описана в работе [2].

Переименование состояния

Описание. Изменение имени состояния.

Мотивация. Описанные до сих пор рефакторинги изменяли структуру автомата, адаптировали ее для лучшей реализации текущей или изменившейся спецификации. Между тем, часто большей прозрачности и понятности структуры автомата можно добиться простой сменой имени одного или нескольких состояний. Часто при анализе автоматов и их рефакторинге требуется изменить имена состояний так, чтобы они лучше отражали их семантику. Четкое и полное выражение смысла состояния несколькими словами часто может оказаться нетривиальной задачей, которая не поддается решению с первого раза. Со временем, когда программист работает с задачей уже несколько дней или недель, в голову часто приходят более удачные метафоры.

Техника. Для изменения имени состояния необходимо выполнить следующие действия.

1. Убедиться, что новое имя является уникальным для графа переходов.
2. Изменить имя состояния.

Перемещение воздействия из состояния в переходы

Описание. Вызов выходного воздействия, совершаемого при входе в состояние, перемещается в переходы, входящие в рассматриваемое состояние (рис. 6).

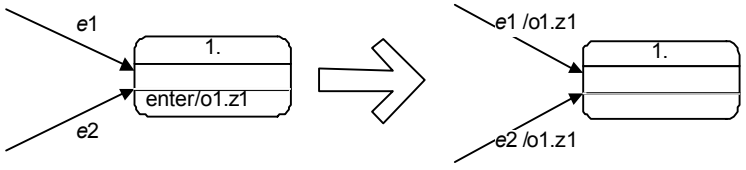

Рис. 6. Перемещение воздействия из состояния в переходы

Если при входе в состояние совершается несколько воздействий, то перемещается только первое. При необходимости рефакторинг можно повторить для остальных воздействий.

Техника. Для переноса воздействия из состояния в переходы необходимо выполнить следующие действия.

1. На каждом переходе, входящем в состояние, добавить вызов воздействия. Если на переходах уже были выходные воздействия, то добавляемое воздействие становится последним.
2. Вызов воздействия удалить из состояния.

Перемещение воздействия из переходов в состояние

Описание. Вызовы одинаковых выходных воздействий, совершаемых на переходах, входящих в одно состояние, заменяются одним воздействием, выполняемым при входе в это состояние.

Описываемое изменение будет являться рефакторингом, если одинаковое выходное воздействие имеют все переходы, входящие в состояние.

Техника выполнения данного рефакторинга описана в работе [2].

Метод внесения изменений в автоматные программы

Изменения, совершаемые исключительно в целях изменения структуры автомата без изменения его поведения, могут быть произведены совершенно безопасно, если они реализованы путем комбинирования нескольких рефакторингов. Остальные изменения призваны модифицировать логику работы автомата для исправления ошибок или адаптации автомата к изменившимся требованиям. Для проверки корректности таких изменений потребуется верифицировать получившийся автомат [4, 5] на соответствие оригинальной (в случае исправления ошибки) или новой (в случае адаптации автомата к изменившимся требованиям) спецификации. Результатом анализа такого сценария является коррекция либо автомата, либо спецификации. В данном случае предполагается, что спецификация верна, и в коррекции нуждается автомат. Точнее, в коррекции нуждается набор действий по изменению автомата, после которого он перестал (или не начал) удовлетворять спецификации.

Из рассмотрения заведомо можно исключить рефакторинги: их безопасность формально доказана [6]. Таким образом, анализу должен подвергнуться набор базовых изменений, не являющихся частью рефакторингов. К сожалению, полностью автоматизировать такой анализ не удается, и его потребуется, как минимум частично, выполнять вручную. Так как ручной анализ изменений – достаточно трудоемкий процесс, чем меньший набор изменений требуется анализировать, тем лучше.

Вышесказанное приводит к следующему методу внесения изменений в графы переходов автоматных программ. Основа метода заключается в разделении любого сложного изменения графа переходов на две фазы:

1. рефакторинг автомата;
2. набор модификаций, приводящих к изменению поведения автомата.

В ходе первой фазы автомат подготавливают к изменениям, модифицируя его структуру, не затрагивая при этом поведение (корректность этой фазы можно проверить автоматически, если спецификация исходного автомата была формализована). Целью рефакторинга является минимизация числа модификаций, выполняемых во второй фазе.

Предложенный метод позволяет избежать внесения в программу сложных изменений, с трудом поддающихся анализу. Наиболее сложные изменения доказуемо безопасны, потенциально опасные изменения просты.

Заключительным шагом должна стать верификация получившегося автомата на соответствие измененной формальной спецификации. При обнаружении несоответствия спецификации предложенный метод значительно упрощает процедуру поиска ошибок, так как набор изменений, направленный на отражение новых требований, минимален.

Заключение

В работе предложен метод модификации автоматных программ, основанный на рефакторинге, уменьшающий число изменений, которые могут привести к появлению ошибок. Метод может применяться на практике благодаря разработанному каталогу рефакторингов. Предложенный метод может быть усовершенствован за счет автоматизации рефакторингов автоматных программ.

Литература

1. Фаулер М. Рефакторинг: улучшение существующего кода. – СПб: Символ-Плюс, 2004. – 432 с.
2. Федотов П.В. Подход к безопасному внесению изменений в графы переходов автоматных систем. Кафедра «Технологии программирования» [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_fedotov_bak.pdf, свободный. – Яз. рус. (дата обращения 01.11.2010).

3. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2010. – 176 с.
4. Кузьмин Е.В., Соколов В.А. Моделирование, спецификация и верификация «автоматных» программ // Программирование. – 2008. – № 1. – С. 1–23.
5. Вельдер С.Э., Шалыто А.А. О верификации простых автоматных программ на основе метода Model Checking // Информационно-управляющие системы. – 2007. – № 3. – С. 285–288.
6. Степанов О.Г. Методы реализации автоматных объектно-ориентированных программ. Кафедра «Технологии программирования» [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/disser/stepanov_disser.pdf, свободный. – Яз. рус. (дата обращения 01.11.2010).

Федотов Павел Валерьевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, fedotov@rain.ifmo.ru

Степанов Олег Георгиевич – ООО «ИнтеллиДжей Лабс», кандидат технических наук, руководитель проекта, доцент, oleg.stepanov@jetbrains.com