Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Кафедры «Компьютерные технологии»

П. В. Федотов

Подход к безопасному внесению изменений в графы переходов автоматных систем

Бакалаврская работа

Научный руководитель
О. Г. Степанов

Санкт-Петербург
2009

Содержание

ВВЕДЕНИЕ

В последнее время для проектирования и разработки программных систем, в которых есть сущности со сложным поведением, все чаще используется *автоматное программирование*, иначе называемое *«программирование с явным выделением состояний»* [1], и поддерживающая его *SWITCH-технология* [2].

В *SWITCH*-технологии поведение системы описывается с помощью графов переходов. К настоящему времени создано немало графических сред, позволяющих создавать и редактировать графы переходов, инструментальных средств для автоматического преобразования графов переходов в код на различных языках программирования, а также разработано несколько языков автоматного программирования [3, 4].

Используя описанные средства, можно достаточно эффективно создавать и редактировать графы переходов, однако при внесении изменений приходится неизбежно сталкиваться с проблемой сохранения надежности автоматной системы.

Целью данной работы является разработка *подхода к безопасному внесению изменений в систему графов переходов автоматов*. Под безопасными изменениями мы подразумеваем такие изменения, которые сохраняют семантическую и синтаксическую корректность автоматной системы. В работе предлагается использовать рефакторинги автоматов, а модификации, изменяющие поведение системы, делать максимально простыми.

В главе 1 исследуется задача и даются определения базовых понятий. Глава 2 содержит классификацию изменений в графах переходов. Приводятся базовые изменения, а также методы рефакторинга автоматов. В главе 3 описывается метод безопасного внесения изменений в графах переходов,

использующий результаты главы 2. С помощью этого метода вносятся изменения в автоматную систему управления банкоматом.

ГЛАВА 1. ВНЕСЕНИЕ ИЗМЕНЕНИЙ В АВТОМАТНУЮ ПРОГРАММУ

1.1. Актуальность проблемы

Любая программа, используемая длительное время, подвергается модификации. Это связано с несколькими вполне естественными причинами: в ходе эксплуатации программной системы могут выявиться требования, которые не были очевидны изначально, а также могут обнаружиться ошибки в работе системы. Наконец, для проведения описанных изменений может возникнуть потребность изменить структуру программы с целью ее упрощения. Известно, что хорошую структуру удается создать не сразу – она должна развиваться по мере накопления опыта [5]. Поэтому почти любая программная система рано или поздно подвергается изменениям.

С появлением большого числа программ с явным выделением состояний возникает необходимость в поддержке таких программ. Из вышесказанного следует, что в существующую систему вносят изменения следующих типов:

1) изменения в системе в соответствии с модификацией требований к ней;
2) исправление ошибок;
3) изменения во внутренней структуре программы, имеющие целью облегчить понимание ее работы и упростить модификацию, не затрагивая наблюдаемого поведения.

По аналогии с существующим понятием *рефакторинг* [5] объектно-ориентированных программ, будем называть изменения последнего типа, применяемые к программам с явным выделением состояний, *рефакторинг автоматов*.

Внесение любых изменений в работающую систему сопряжено с риском появления ошибок и грозит потерей надежности системы. Вместе с тем,

надежность зачастую является важнейшим требованием, предъявляемым к системе, реализованной программой с явным выделением состояний. Поэтому исследования, направленные на разработку подхода к безопасному внесению изменений в такие программы, являются актуальными. Эти исследования позволят упростить процесс разработки и повысить качество создаваемых программ.

1.2. Автоматы и графы переходов

В настоящее время предложены, исследованы и применяются различные автоматные модели [1]. В данной работе будем рассматривать автоматы с графами переходов следующего вида [3, 6]:

Определим основные свойства графов переходов:

- граф изображает одно или несколько состояний системы и переходы между ними;
- состояния в графе могут быть объединены в группы; группы могут быть вложены друг в друга; состояния внутри группы равноправны;
- переходы могут начинаться в состоянии или в группе состояний, заканчиваться только в состоянии (переходы, начинающиеся в группе состояний, называются *групповыми переходами*); переходы могут начинаться и заканчиваться в одном и том же состоянии;
- каждое состояние помечено следующими атрибутами:
    - имя состояния;
    - действия по входу в состояние;
    - вложенные автоматы (возможно с номерами воздействий, с которыми они вызываются);
- переход может быть помечен следующими атрибутами:
    - условие перехода;
    - события;

- действия на переходе.

Обработка события происходит следующим образом: перебираются переходы, содержащие в качестве атрибута активное событие, выходящие из текущего состояния и содержащих его групп. Для каждого перехода вычисляется условие, которое является логической формулой. Эта формула может использовать переменные x_i.

Выполняется переход, для которого значение условия истинно. Выполнение перехода состоит из следующих шагов:

- выполняются действия на переходе (вызываются указанные выходные воздействия в порядке их следования);
- текущим выставляется состояние, в котором заканчивается переход;
- если произошла смена состояния (текущее состояние до начала обработки события отличается от состояния, в котором заканчивается переход) вызываются действия по входу в состояние;
- вызываются вложенные автоматы: если для автомата указан номер события, он вызывается с этим событием; иначе – с событием e_0.

Определим сказанное выше формально. Сначала введем A как систему автоматов. Каждый автомат $a \in A$ представляет собой пару $< d, c >$, где $d \in D$ – граф переходов автомата, а $c \in C_a$ – текущая конфигурация автомата. Граф переходов автомата имеет следующую структуру:

$$d_a =< S_a, SS_a, ST_a, G_a, E_a, Z_a, X_a >,$$

где S_a – множество состояний;

SS_a – множество начальных состояний;

ST_a – множество конечных состояний;

T_a – множество переходов;

G_a – множество групп состояний;

E_a – множество входных воздействий;

Z_a – множество выходных воздействий;

X_a – множество внутренних переменных.

Состояние представляется набором атрибутов:

$$S = \{< n, i, A_s, Z_s > | A_s \subset A_a, Z_s \subset Z_a\},$$

где n – номер состояния;

i – имя состояния;

A_s – множество вызываемых автоматов;

Z_s – множество выходных воздействий, выполняемых по входу в состояние;

$A_a = A/\{a\}$ – множество соседних автоматов (автомат не может вызывать себя рекурсивно).

Множество групп состояний представляет собой набор подмножеств состояний:

$$G_a \subset 2^{S_a}$$

Если в это множество включить дополнительные группы, состоящие из каждого состояния, то получим *полное множество групп состояний*: $F_a = G_a \cup S_a$. Тогда множество переходов определяется как

$$T = \{< s_t, e_t, c_t, Z_t > | s_t \in F_a, e_t \in S_a, c_t \subset C_a', Z_t \subset Z_a\},$$

где s_t – множество исходных состояний, e_t – конечное состояние, c_t – множество конфигураций, в которых выполняется переход, C_a' – *множество полных конфигураций автомата*, Z_t – множество выходных воздействий, выполняемых при осуществлении перехода.

Конфигурацией C_a автомата a с графом переходов d_a является тройка $< y_a, e_a, \chi_a >$, где $y \in S$ – текущее состояние, $e \in E \cup \{\varnothing\}$ – текущее входное воздействие или $\varnothing$, если автомат в данный момент не производит обработку никакого воздействия, χ – функция на множестве внутренних переменных X, определяющая значения этих переменных. *Внешней конфигурацией* системы

автоматов A является функция $\gamma : \gamma(a \in A) \in S_a, \gamma(a) = y_a$, определяющая текущее состояние каждого автомата.

Тогда полной конфигурацией C_a' является совокупность его собственной конфигурации и внешней конфигурации системы автоматов, в которой он исполняется.

Вызов автомата a с входным воздействием e в некоторой конфигурации системы автоматов, в первую очередь, изменяет конфигурацию автомата a так что $e_a = e$. Затем производится выбор перехода, который будет произведен. Для этого вычисляется множество выполнимых переходов $T' = \{t \mid t \in T_a, s_t \in y_a, C_a' \in c_t\}$. Для корректного автомата множество T' состоит из ровно одного перехода – этот переход и будет произведен.

Если был определен переход, то производится его выполнение: вызываются все входные воздействия Z_t, затем в качестве текущего состояния автомата y_a выставляется e_t. При этом если предыдущее состояние было отлично от e_t, также выполняются выходные воздействия Z_{e_t}. В конце вызываются вложенные автоматы A_{e_t}.

1.3. Корректность автоматов

Будем называть автомат *синтаксически корректным*, если он удовлетворяет всем свойствам разд. 1.2. Важно отметить, что синтаксически корректный автомат удовлетворяет требованиям полноты и непротиворечивости: множество исходящих переходов для любого состояния полно и непротиворечиво [3]. Это означает, что при обработке любого события выполняется ровно один переход.

Будем автомат называть *семантически корректным*, если его выполнение согласовано со спецификацией. Заметим, что спецификация может быть как неформальной – например, задавать требования словесно, так и формальной – к примеру, задаваться с помощью темпоральной логики. При этом выполнение

спецификации, заданной формально, в некоторых случаях можно проверить автоматически (*верифицировать*) [3].

Синтаксически и семантически корректный автомат будем называть просто *корректным*.

1.4. Безопасные изменения

Безопасными изменениями будем называть такие изменения автоматов, которые сохраняют их корректность.

К сожалению, далеко не все изменения являются безопасными. Часто разработчики, столкнувшиеся с необходимостью внесения изменений в программу с явным выделением состояний, оказываются вынуждены проектировать ее заново или бессистемно вносить изменения. Это является неэффективным и ненадежным, так как даже простейшие изменения влияют на корректность автоматов и могут привести к появлению трудно находимых ошибок. Например, добавление одного перехода между двумя состояниями автомата может нарушить непротиворечивость множества переходов автомата.

В заключение главы укажем еще один тип изменений. Иногда необходимо вносить изменения в систему, которая уже запущена и находится в эксплуатации. При проведении таких изменений могут возникать специфические трудности. Например, удаление состояния из работающего автомата может привести к ошибке, если это состояние активно в момент удаления.

Выводы по главе 1

1. Введена формальная модель автоматов и графов переходов.
2. Рассмотрены типы изменений, вносимые в автоматную систему.
3. Введено понятие «рефакторинг автоматов».

ГЛАВА 2. КЛАССИФИКАЦИЯ ИЗМЕНЕНИЙ ГРАФОВ ПЕРЕХОДОВ АВТОМАТОВ

Часто изменения, вносимые в программу с явным выделением состояний, достаточно сложны, и поэтому 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 изменения, которые будем называть *составными*, можно разложить в композицию базовых.

В отдельный класс выделим *рефакторинги автоматов*. Как было сказано выше, рефакторинги не изменяют поведение системы и применяются для улучшения ее структуры.

2.1. Базовые изменения

При описании каждого базового изменения графа переходов автомата будем придерживаться определенного формата, приведенного ниже:

- Сначала следует **название** изменения.
- За названием следует краткое **неформальное описание** приводимого изменения.
- Затем описываются **проблемы**, возникающие при проведении такого изменения.
- И наконец, приводится **формально описание**. Формальное описание базовых изменений основывается на данном выше формальном определении графа переходов и использует те же обозначения.

2.1.1. Добавление состояния

В граф переходов добавляется новое состояние.

Проблемы

- Достижимость состояния (необходимо добавить переходы, ведущие в это состояние и из него).

Формальное описание изменений

- $S' = S \cup \{s\}$, s – добавляемое состояние.

2.1.2. Удаление состояния

В графе переходов удаляется состояние и все связанные с этим состоянием переходы.

Проблемы

- При удалении состояния удаляются и все связанные с ним переходы. Получаем те же проблемы, что и при удалении перехода (разд. 2.1.10).
- Удаляемое состояние может быть стартовым (разд. 2.1.4).
- Удаляемое состояние может быть конечным (разд. 2.1.6).
- Удаляемое состояние может быть активным в момент удаления. Необходимо указать, в какое состояние перейдет автомат в этом случае.

Формальное описание изменений

$S' = S \setminus \{s\}$, s – удаляемое состояние,

$T' = T \setminus T_s$, где

$T_s = \{t \mid t =< s, e_t, c_t, Z_t >\in T\} \cup \{t \mid t =< s_t, s, c_t, Z_t >\in T\}$ –объединение множеств переходов, выходящих из s и входящих в s.

2.1.3. Установка стартового состояния

Состояние объявляется стартовым в автомате.

Проблемы

- Необходимо проверить, что нет других стартовых состояний. Если есть, то сделать их нормальными.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только начальная конфигурация автомата.

2.1.4. Объявление стартового состояния нормальным

Стартовое состояние объявляется нормальным.

Проблемы

- Необходимо стартовым объявить другое состояние.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только начальная конфигурация автомата.

2.1.5. Объявление нормального состояния конечным

Нормальное состояние объявляется конечным в автомате.

Проблемы

- Если из этого состояния выходили какие-то переходы, то их необходимо удалить.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только множество конфигураций автомата.

2.1.6. Объявление конечного состояния нормальным

Конечное состояние объявляется нормальным.

Проблемы

- Если в автомате не осталось ни одного конечного состояния, то он никогда не завершит свою работу.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только множество конфигураций автомата.

2.1.7. Добавление перехода

Добавление перехода между двумя состояниями. На переходе указывается событие, по которому данный переход осуществляется. Также может указываться условие и выходное воздействие.

Проблемы

- Полнота и непротиворечивость множества переходов.

Формальное описание изменений

$T' = T \cup \{t\}$, где t – добавляемый переход.

2.1.8. Изменение события на переходе

Изменяется событие, по которому активизируется переход.

Проблемы

- Полнота и непротиворечивость множества переходов.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только множество конфигураций c_t, при котором осуществляется переход t.

2.1.9. Изменение условия на переходе

Изменяется условие, при котором осуществляется переход.

Проблемы

- Полнота и непротиворечивость множества переходов.

Формальное описание изменений

Структура автомата при такой модификации автомата не изменяется. Изменяется только множество конфигураций c_t, при котором осуществляется переход t.

2.1.10. Удаление перехода

В автомате удаляется переход.

Проблемы

- Полнота и непротиворечивость множества переходов.

Формальное описание изменений

$T' = T \setminus \{t\}$, где t – удаляемый переход.

2.1.11. Изменение перехода

Переход имеет начальное и конечное состояния. При изменении перехода возможно:

- изменение начального состояния;
- изменение конечного состояния.

Данное изменение формально не является базовым, так как его можно разбить на более простые. Тем не менее, оно рассматривается нами, так как изменение перехода – это регулярно используемая модификация.

Проблемы

- Полнота и непротиворечивость множества переходов.

Техника

Изменения такого рода можно выполнять следующим образом:

1. Удаление перехода (разд. 2.1.10).
2. Добавление перехода (разд. 2.1.7).

2.2. Каталог рефакторингов

При описании составных изменений будем придерживаться следующего формата.

- Сначала следует **название** рефакторинга.
- За названием следует **неформальное описание** изменения.
- **Мотивация** описывает, почему следует пользоваться этим методом рефакторинга.
- Следом приводится **пример** применения рефакторинга.
- **Техника** содержит описание пошагового выполнения рефакторинга.

2.2.1. Группировка состояний

Несколько простых состояний объединяются в группу состояний. При этом добавляются групповые переходы, заменяющие одинаковые переходы, исходящие из всех группируемых состояний (рис. 1).

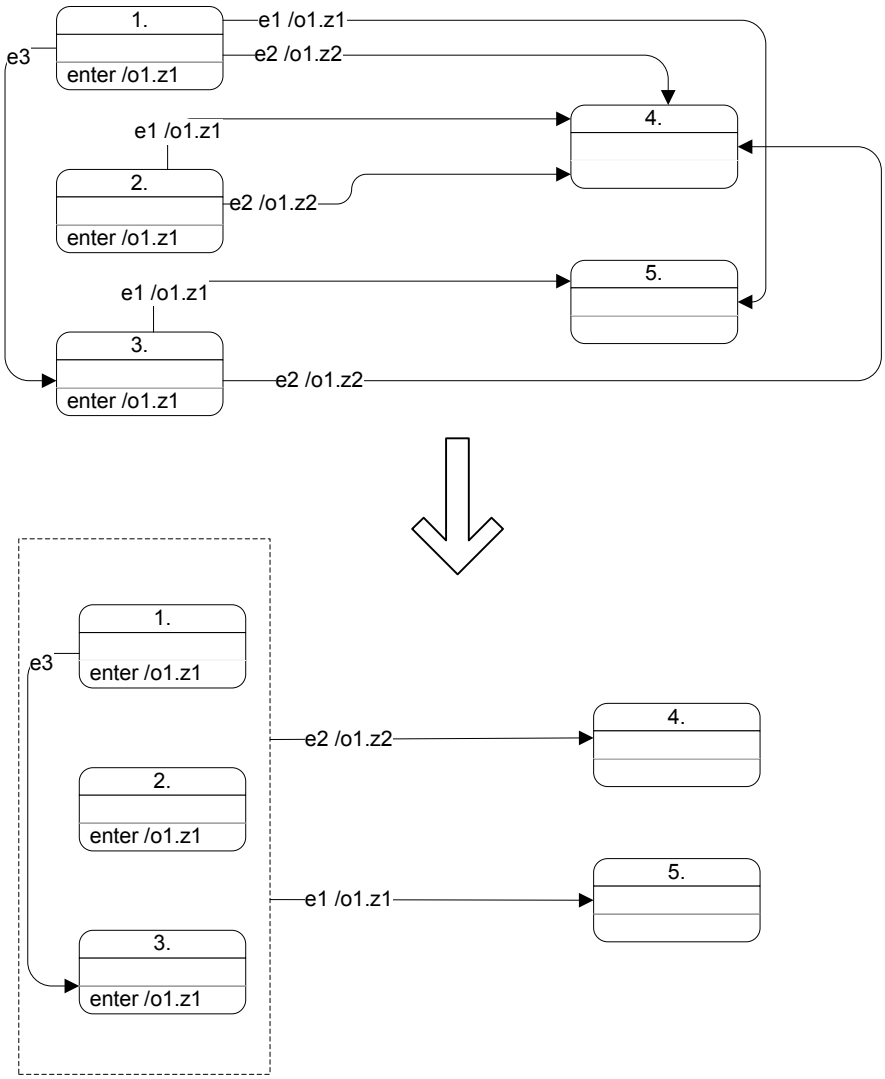

Рис. 1. Группировка состояний

Состояния, объединяемые в группу, могут иметь по несколько одинаковых переходов, в этом случае добавляется несколько групповых переходов.

Мотивация

Часто в графе переходов автомата имеется несколько состояний, которые имеют одинаковые связи с остальными элементами графа переходов. В таком случае разумно выделить группу состояний, упростив тем самым граф переходов.

Пример

Рассмотрим фрагмент графа переходов автомата «Панель в кабине лифта» [7], 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. Такую логику поведения можно реализовать следующим графом переходов (рис. 2).

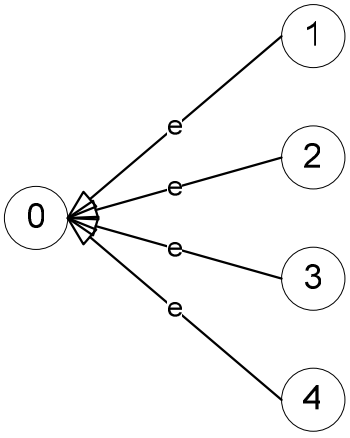

Рис. 2. Граф переходов без группировки состояний

Однако состояния 1 – 4 представляется разумным сгруппировать и соответствующим образом изменить конфигурацию автомата (рис. 3).

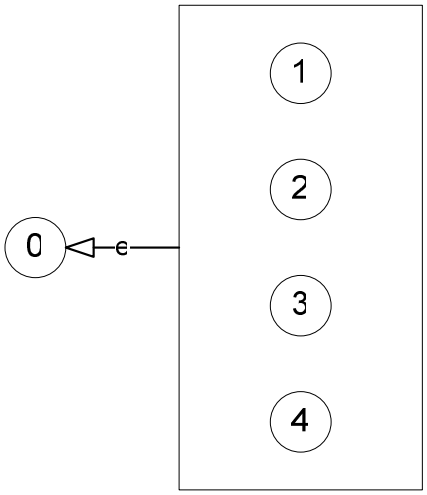

Рис. 3. Граф переходов с группой

Техника

1. Добавьте группу *g*, объединяющую состояния s_1, s_2, …, s_k.
2. Выберите один переход *t*, исходящий из s_1, который желательно заменить групповым.
3. Убедитесь, что каждое из состояний s_2, …, s_k имеет переход с такими же атрибутами, что и *t* (под атрибутами здесь понимаем конечное состояние, событие, условие и выходные воздействия).
4. Добавьте переход, исходящий из группы *g*, и имеющий те же атрибуты, что и *t*.
5. Удалите переходы, отмеченные в п. 2, 3.
6. Если еще остались переходы, которые желательно заменить групповыми, то повторите шаги 2 – 6.

2.2.2. Удаление группы состояний

При удалении сложного состояния все исходящие из него переходы добавляются в состояния, которые удаляемое состояние содержал (рис. 4).

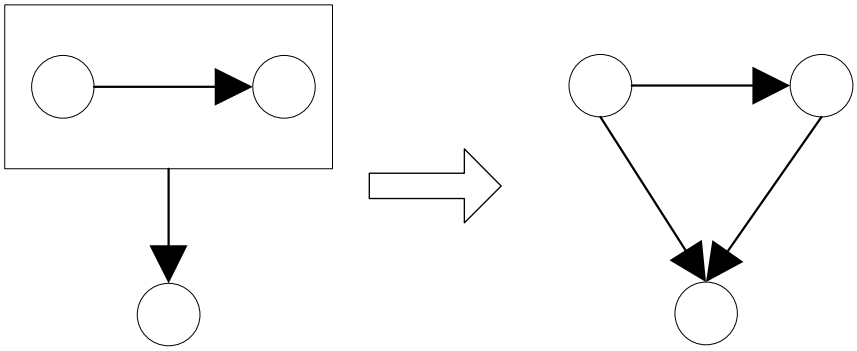

Рис. 4. Удаление группы состояний

Мотивация

Такое изменение полезно для последующей модификации автомата, если одно из состояний, входящее в удаляемое сложное состояние, меняет логику поведения.

Техника

Для удаления группы *g*, объединяющей состояния s_1, s_2, …, s_k, сделайте следующие шаги.

1. Выберите один переход *t*, исходящий из *g*.
2. Добавьте переходы, исходящие из состояний s_i ($i = 1 \ldots k$), с атрибутами перехода *t*.
3. Удалите переход *t*.
4. Если остались еще переходы, исходящие из *g*, повторите шаги 1 – 4.
5. Удалите группу *g*.

2.2.3. Слияние состояний

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

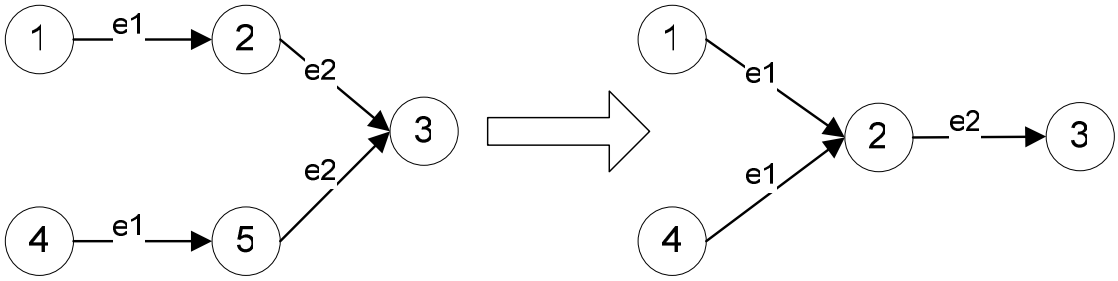

Рис. 5. Слияние состояний

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 системы может оказаться, что некоторые состояния дублируют логику поведения системы. В этом случае эти состояния могут быть заменены одним состоянием.

Пример

Рассмотрим пример, иллюстрирующий применение такого изменения (рефакторинга). Для этого добавим в автомат «Панель в кабине лифта» (разд. 2.2.1) следующие состояния:

5 – перегорела лампа «1»;

6 – перегорела лампа «2»;

7 – перегорела лампа «3»;

8 – перегорела лампа «S»;

9 – неисправность.

Измененный граф переходов представлен на рис. 6:

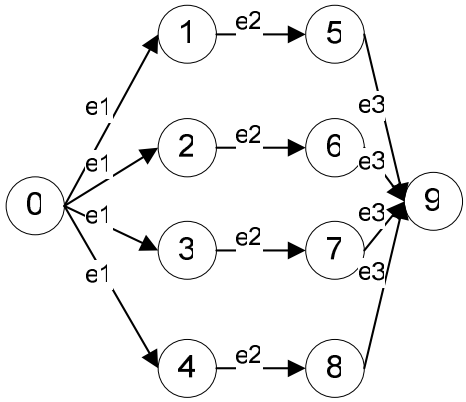

Рис. 6. Граф переходов до слияния состояний

Граф, получаемый после слияния состояний 5 – 8 в одно, приведен на рис. 7.

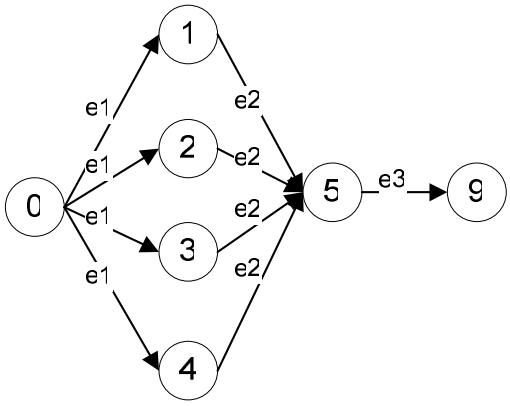

Рис. 7. Граф переходов после слияния состояний

Техника

Для слияния состояний $s_1, s_2, \ldots, s_k$ сделайте следующие шаги.

1. Убедитесь, что состояния $s_1, s_2, \ldots, s_k$ имеют одинаковые воздействия, вызываемые на входе в состояния.
2. Убедитесь, что состояния $s_1, s_2, \ldots, s_k$ имеют исходящие переходы с одинаковыми атрибутами.
3. Измените конечное состояние переходов, ведущих в $s_2, \ldots, s_k$, на состояние s_1.
4. Удалите состояния $s_2, \ldots, s_k$.

2.2.4. Выделение автомата

Часть логики системы переносится в отдельный вызываемый автомат (рис. 8).

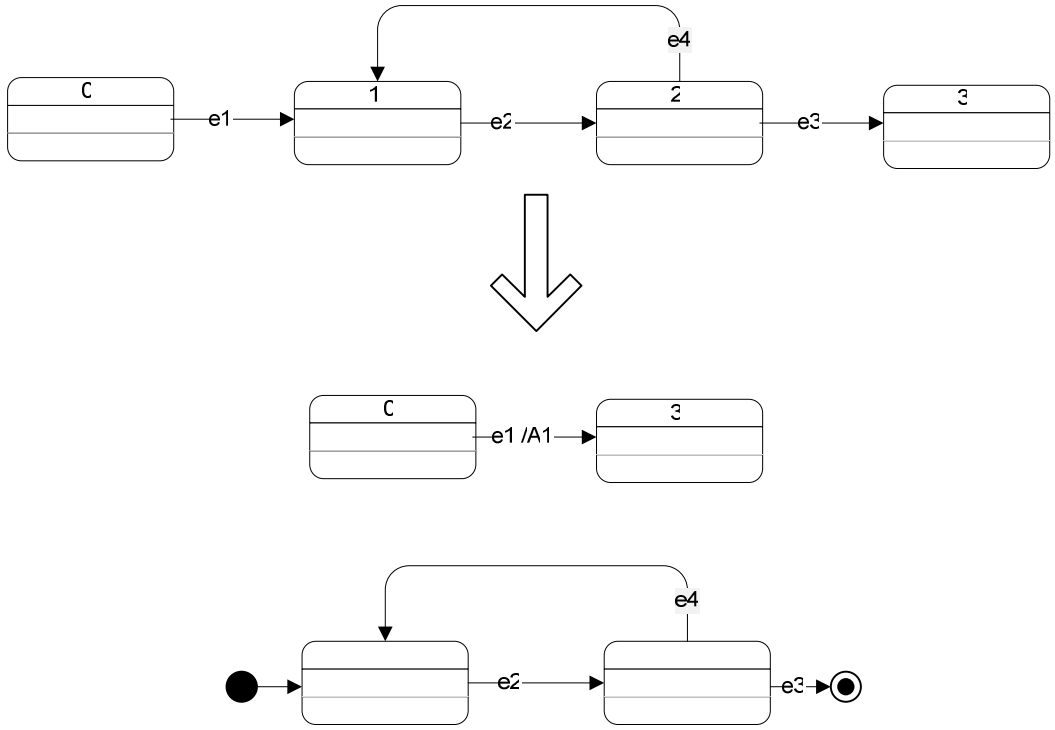

Рис. 8. Выделение автомата

Мотивация

Существует несколько критериев *автоматной декомпозиции* [1], следуя которым в большинстве случаев можно получить логичную архитектуру системы.

Декомпозиция *по режимам* уместна тогда, когда в поведении системы можно выделить несколько качественно различных режимов (каждый из которых, при необходимости, можно конкретизировать, выделив режимы более низкого уровня абстракции). В этом случае логично сопоставить автомат каждому из режимов, в которых поведение системы является сложным, или, иными словами, сопоставить отдельный автомат каждому абстрактному действию.

Декомпозиция *по объектам управления* применима в том случае, когда в системе присутствует несколько объектов управления. В этом случае логично поручить управление каждым из объектов отдельному автомату или поддереву в иерархии автоматов.

Следование такому критерию декомпозиции приводит к выделению в архитектуре системы пар автомат – объект управления (или группа автоматов – объект управления) и значительно приближает ее к «идеалу» парадигмы автоматного программирования – *множеству взаимодействующих автоматизированных объектов управления*.

Техника

Для выделения состояний $s_1, s_2, \ldots, s_k$ и исходящих из них переходов $t_1, t_2, \ldots, t_l$ в вызываемый автомат проделайте следующие шаги:

1. Убедитесь, что среди состояний $s_1, s_2, \ldots, s_k$ есть ровно одно такое состояние s_i, в которое ведет хотя бы один переход из состояния, отличного от $s_1, s_2, \ldots, s_k$. Обозначим отмеченный переход за t', а его начальное состояние – s'.
2. Убедитесь, что из отмеченного состояния s_i достижимы все остальные состояния выбранного подмножества.
3. Убедитесь, что переходы $t_1, t_2, \ldots, t_l$ имеют в качестве конечных состояний только состояния из множества $s_1, s_2, \ldots, s_k$ и ровно одно состояние s'', не входящее в это множество.
4. Создайте новый автомат.
5. Создайте в новом автомате состояния $s'_1, s'_2, \ldots, s'_k$ соответствующие состояниям $s_1, s_2, \ldots, s_k$.
6. Добавьте переходы между состояниями $s'_1, s'_2, \ldots, s'_k$ с такими же атрибутами, что и переходы между состояниями $s_1, s_2, \ldots, s_k$.
7. Сделайте состояние s'_i стартовым в созданном автомате.

8. Выберите среди состояний s_1, s_2, …, s_k такие, из которых есть переходы в s''. Пусть в созданном автомате им соответствуют состояния s'_{i_1}, s'_{i_2}, ..., s'_{i_m}. Добавьте из состояний s'_{i_1}, s'_{i_2}, ..., s'_{i_m} переходы, ведущие в конечное состояние созданного автомата. Присвойте этим переходам такие же атрибуты, как у соответствующих переходов изначального автомата.
9. Добавьте переход между состояниями s' и s'' с атрибутами перехода t' и вызовом созданного автомата.
10. Удалите состояния s_1, s_2, …, s_k.

Пример

Выделение автомата подробно рассматривается в разд. 3.2.

2.2.5. Встраивание вызываемого автомата

Вызываемый автомат встраивается в места своего вызова на переходах (рис. 9).

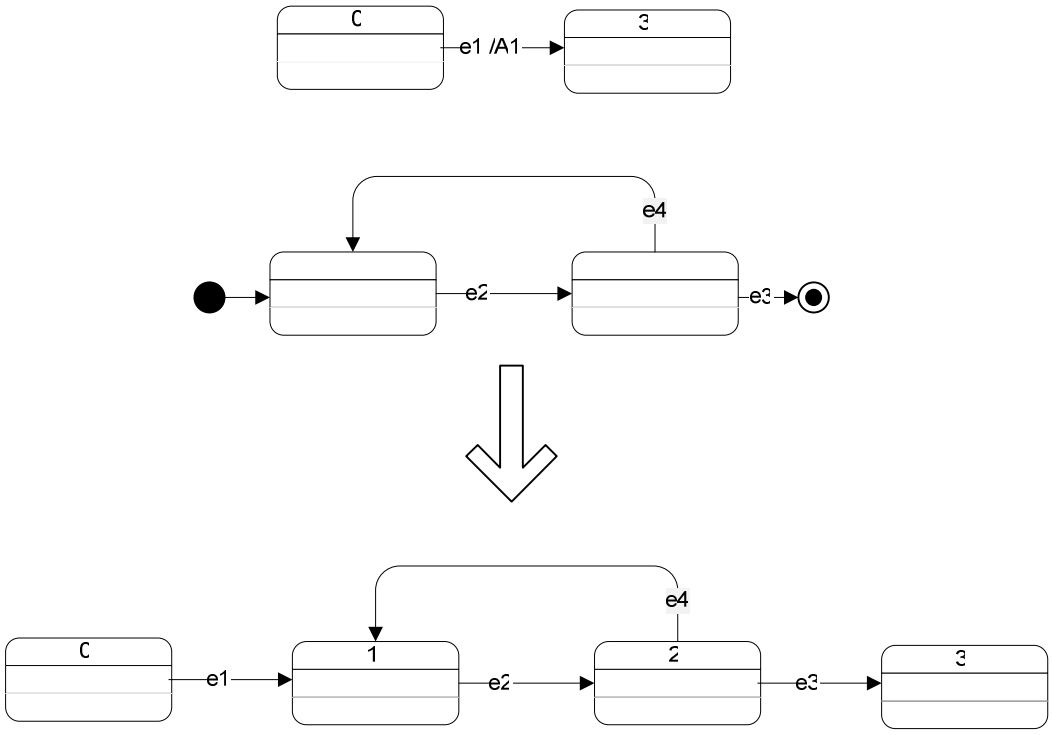

Рис. 9. Встраивание вызываемого автомата

Данный рефакторинг является противоположностью предыдущему.

Мотивация

Разместив всю логику поведения системы в одном графе переходов, можно добиться большей наглядности, так как такой граф можно охватить «одним взглядом».

Тем не менее, следует данный рефакторинг применять с осторожностью, потому как излишне «разросшийся» граф переходов не понятнее системы более простых. Более того, встраивание автомата нарушает автоматную декомпозицию, а следовательно, потенциально усложняет будущую модификацию системы.

Техника

Пусть автомат содержит состояния s_1, s_2, …, s_k (s_1 – стартовое), а его вызов совершается на переходе t, соединяющем состояния s' и s'' автомата A.

1. Добавьте в автомат A состояния s'_1, s'_2, …, s'_k, соответствующие состояниям s_1, s_2, …, s_k.
2. Для каждого перехода t_i, соединяющего состояния s_i и s_j встраиваемого автомата добавьте соответствующий переход с теми же атрибутами между парой состояний s'_i и s'_j.
3. Измените конечное состояние перехода t на состояние s'_1.
4. Для каждого перехода, ведущего из некоторого состояния s_i в конечное состояние встраиваемого автомата, добавьте переход с такими же атрибутами, ведущий из состояния s'_i в состояние s''.
5. Если вызовов автомата A больше нет, то удалите состояния s_1, s_2, …, s_k.

2.2.6. Переименование состояния

Изменение имени состояния.

Мотивация

Часто при анализе автоматов и их рефакторинге требуется изменить имена состояний так, чтобы они лучше отражали их семантику. Наименования являются важной частью информационной составляющей описания автомата и в значительной степени определяют скорость восприятия его семантики.

В некоторых случаях стоит выбрать более короткое имя для состояния, так как короткие имена делают граф переходов более компактным.

Переименование состояния является очень простым рефакторингом, но не стоит им пренебрегать: оно может существенно упростить восприятие логики поведения системы.

Техника

1. Убедитесь, что новое имя является уникальным для графа переходов.
2. Измените имя состояния.

2.2.7. Перемещение воздействия из состояния в переходы

Вызов выходного воздействия, совершаемого при входе в состояние, перемещается в переходы, входящие в рассматриваемое состояние (рис. 10).

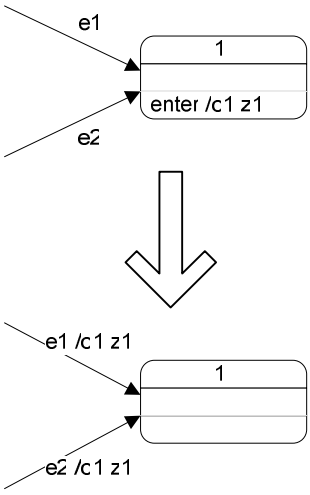

Рис. 10. Перемещение воздействия из состояния в переходы

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

Техника

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

2.2.8. Перемещение воздействия из переходов в состояние

Вызовы одинаковых выходных воздействий, совершаемых на переходах, входящих в одно состояние, заменяются одним воздействием, выполняемым при входе в это состояние (рис. 11).

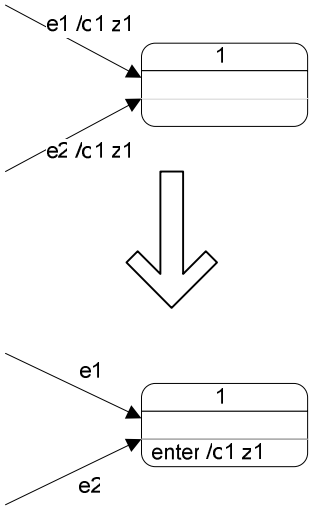

Рис. 11. Перемещение воздействия из переходов в состояние

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

Пример

Перемещение воздействия из переходов в состояние подробно рассматривается в разд. 3.2.

Техника

Для перемещения воздействия *o.z* внутрь состояния *s* проделайте следующие шаги.

1. Убедитесь, что воздействие *o.z* имеется на всех переходах, входящих в *s*; если на каком-то из переходов вызывается несколько воздействий, проверьте, что *o.z* вызывается последним.
2. Добавьте вызов *o.z* в конец списка воздействий, выполняемых при входе в состояние *s*.
3. Удалите вызовы воздействия *o.z* из всех переходов, входящих в состояние *s*.

Выводы по главе 2

1. Рассмотрены базовые изменения графов переходов.
2. Создан каталог рефакторингов автоматов.

ГЛАВА 3. БЕЗОПАСНОЕ ВНЕСЕНИЕ ИЗМЕНЕНИЙ

3.1. Основа подхода

Описанные в главе 2 классификация изменений и каталог рефакторингов, позволяют предложить следующий подход к внесению изменений в графы переходов автоматных систем. Основа подхода заключается в том, что любое сложное изменение графа переходов раскладывается в композицию рефакторингов автомата и базовых изменений.

Таким образом, автомат сначала «подготавливают» к изменениям, изменяя его структуру, но не затрагивая поведение (корректность этой фазы можно проверить автоматически, если спецификация исходного автомата была формализована), а модификации, изменяющие поведение, делают максимально простыми. Подготовленные списки потенциальных проблем, которые могут возникнуть при внесении базовых изменений, помогут выполнять дополнительные проверки после каждого такого изменения. При этом большая часть этих проверок, связанная с несемантическими требованиями к корректным автоматам, может быть проверена автоматически.

Такой подход позволяет избежать внесения в систему сложных изменений, которые наиболее плохо поддаются анализу. Заключительным шагом должна стать верификация получившегося автомата на соответствие измененной формальной спецификации. В случае обнаружения несоответствия спецификации, использование предложенного подхода значительно упрощает процедуру поиска ошибок, так как набор изменений, направленный на отражение новых требований, минимален.

Вместо них производится ряд более простых изменений, техника выполнения которых и проблемы при этом возникающие, описаны в

предыдущей главе. За счет этого существенно снижается риск появления ошибок в системе.

3.2. Применение подхода для безопасного внесения изменений на примере системы автоматов, отвечающей за работу банкомата

Рассмотрим возможное применение описанного подхода на примере внесения изменений в граф переходов автомата, отвечающего за работу банкомата. Моделирование работы банкомата с применением автоматного подхода осуществлено в работах [8, 9]. Приведем неформальное описание работы с банкоматом.

Модель общения банкомата с сервером банка построена на основе транзакций – в ходе взаимодействия с пользователем устройство банкомата накапливает вводимую информацию в специальном внутреннем списке, отправляя серверу по требованию совершения операции все описание транзакции. Так, в начале работы пользователь вводит номер карты. После этого банкомат запрашивает у него *PIN*-код. Как номер карты, так и введенный код запоминаются во внутреннем списке. Банкомат запрашивает у сервера авторизацию для карты. **В случае неверного ввода *PIN*-кода, карта возвращается.**

После успешной проверки *PIN*-кода пользователю становится доступно основное меню, в котором он может выбрать одно из следующих действий:

- снять деньги со счета;
- посмотреть остаток счета;
- забрать карту.

При выборе последнего пункта пользователю возвращается карта, и работа с банкоматом завершается. При выборе любого другого пункта пользователь должен еще дополнительно выбрать один из возможных вариантов или ввести число с клавиатуры. После этого автомат отправляет

серверу накопленные данные. При этом клиенту отображается информация о результате операции. После этого банкомат вновь отображает главное меню, или пользователь забирает карту.

Управление осуществляется системой взаимодействующих автоматов. Формальное описание, содержащее в себе объекты управления, источники событий и систему автоматов, приведено ниже.

Схема связей автоматов с поставщиками событий и объектами управления приведена на рис. 12.

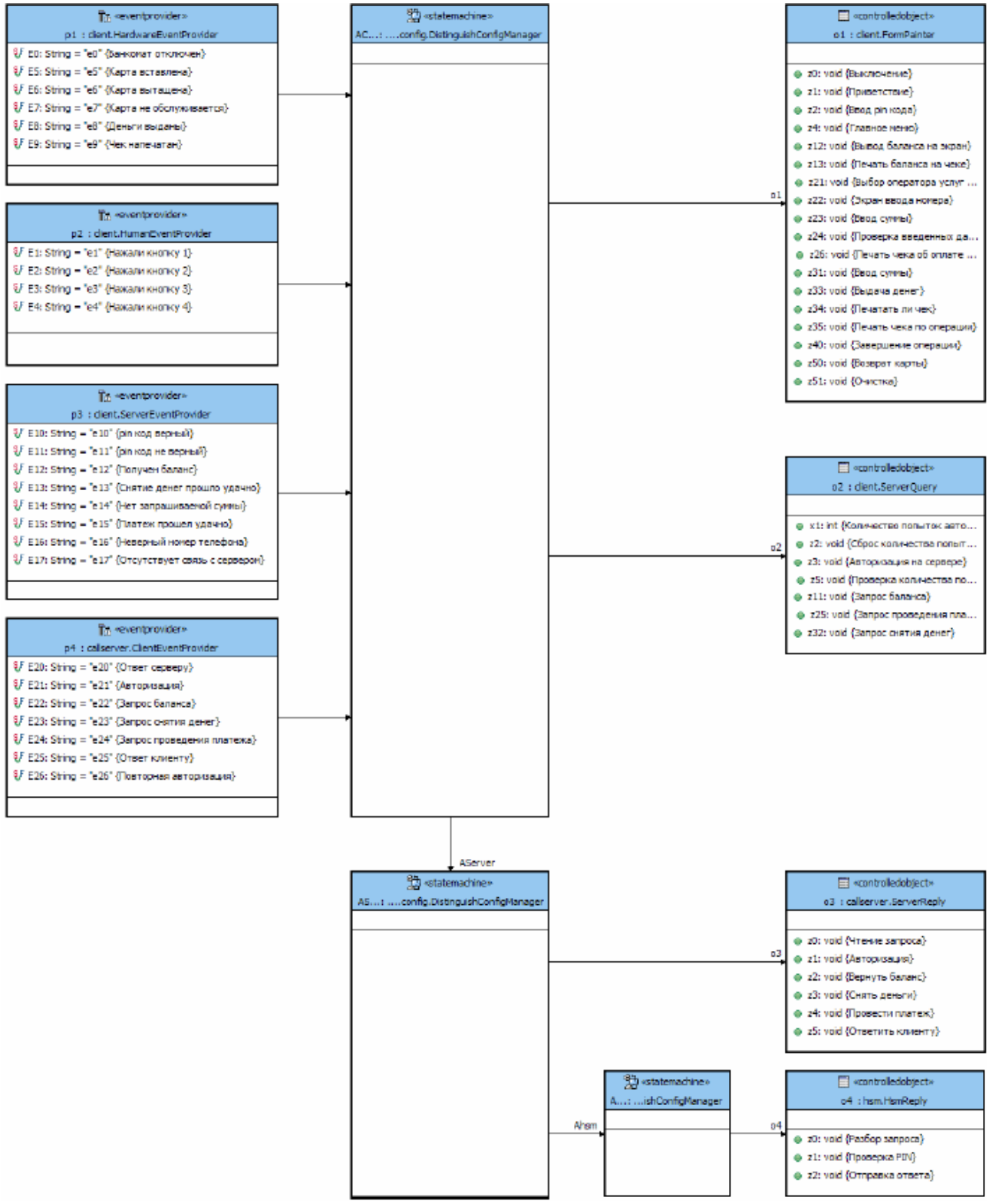

Рис. 12. Схема связей автоматов с поставщиками событий и объектами управления

Граф переходов автомата, отвечающего за работу банкомата, представлен на рис. 13.

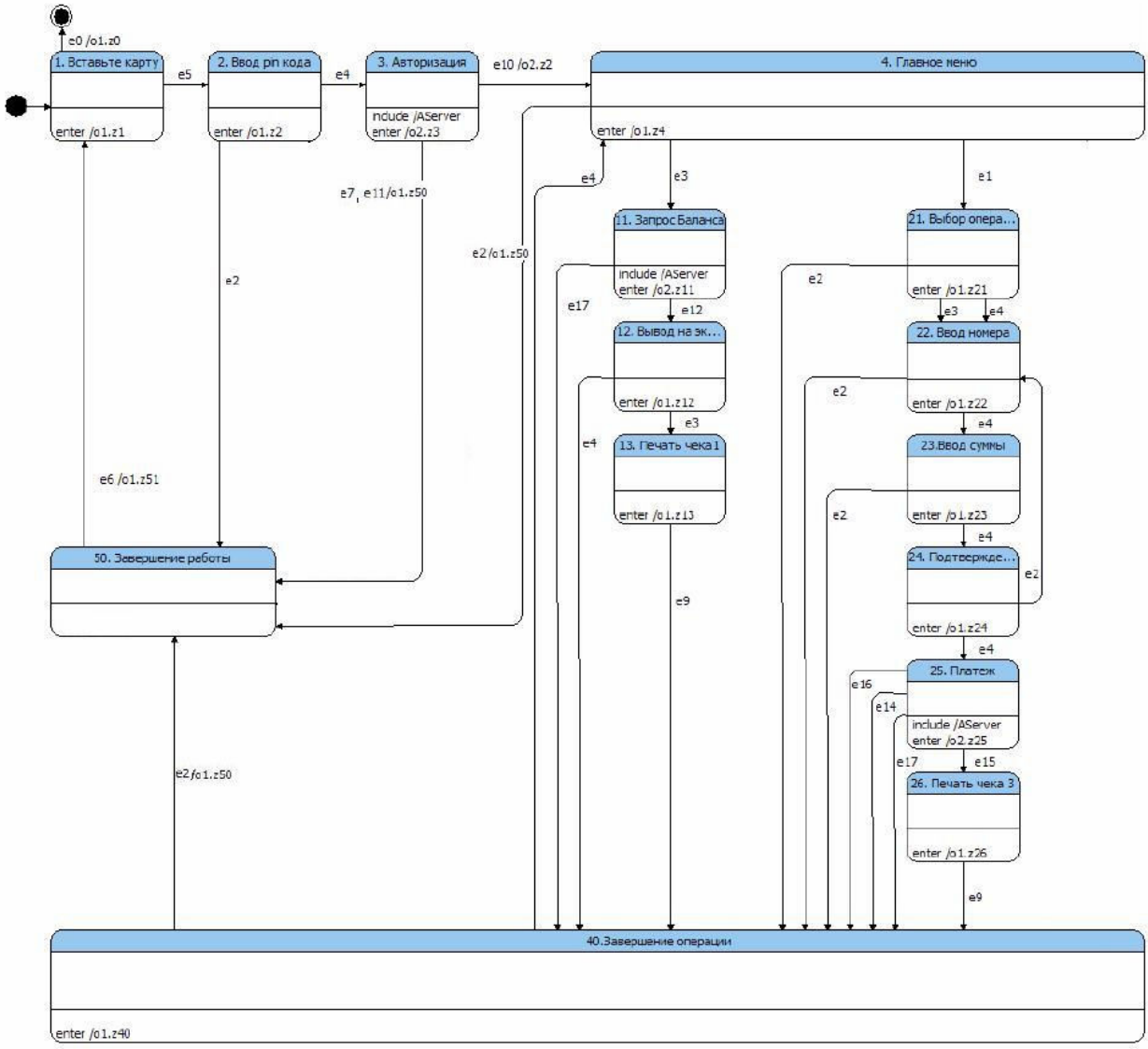

Рис. 13. Граф переходов автомата, отвечающего за работу банкомата

Рассмотрим ситуацию, когда к созданной системе изменяются требования. Пусть сценарий работы с банкоматом изменился следующим образом: «В случае неверного ввода *PIN*-кода он запрашивается повторно. В случае неверного ввода *PIN*-кода три раза подряд работа с банкоматом принудительно завершается».

Граф переходов автомата, отвечающего за работу банкомата, достаточно громоздок. Упростим его, выделив два вызываемых автомата: один – для режима просмотра остатка счета, и еще один – для режима снятия денег со

счета. Граф переходов автомата после применения таких рефакторингов представлен на рис. 14.

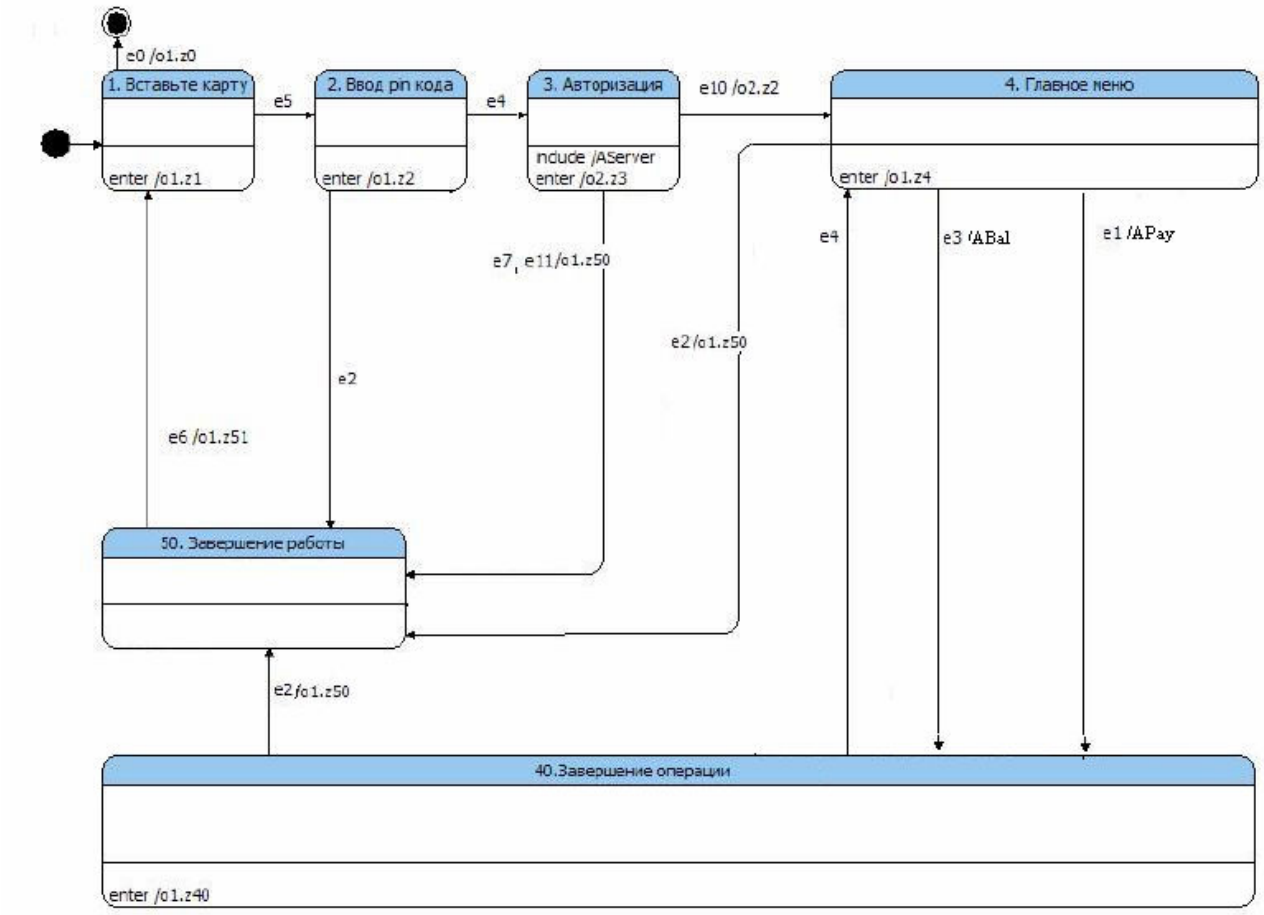

Рис. 14. Граф переходов автомата, отвечающего за работу банкомата, после выделения вызываемых автоматов

Граф переходов стал существенно легче. Теперь стало достаточно просто найти место, которое нуждается в изменении: из состояния *3* («Авторизация») по событию *e11* («pin код неверный») совершается действие *o1.z50* («Возврат карты») и переход в состояние *50* («Завершение работы»). Необходимо, чтобы по событию *e11* карта не возвращалась, а производился выбор действия в зависимости от числа неверных попыток ввода *PIN*-кода. Получаем, что требуется некоторым образом разделить событие *e11* и выходное воздействие *o1.z50*. Самый простой способ сделать это – осуществить перенос воздействий с переходов, ведущих в состояние *50*, внутрь этого состояния. Граф переходов автомата, получающийся после применения такого рефакторинга (рис. 15).

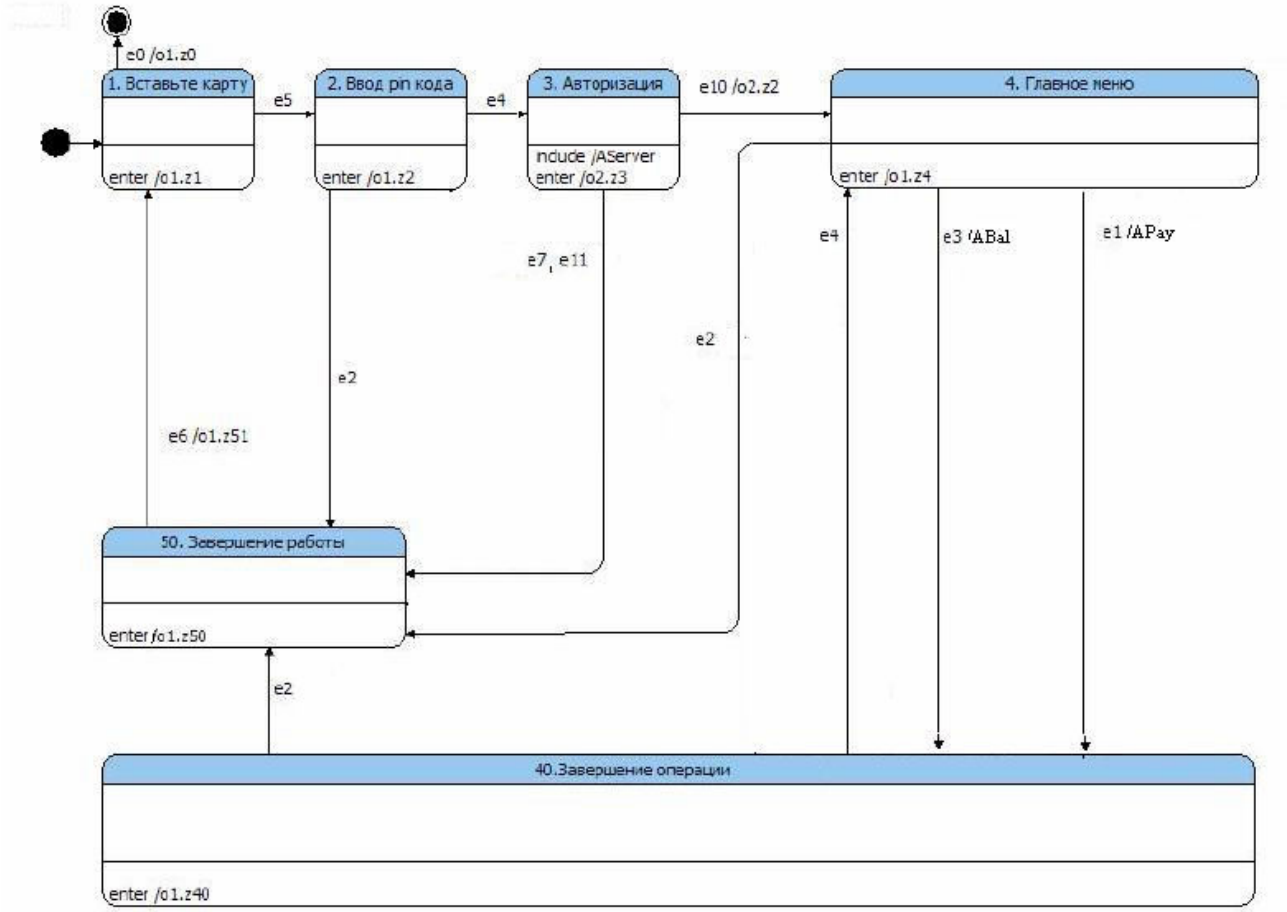

Рис. 15. Граф переходов автомата, отвечающего за работу банкомата, после переноса воздействия внутрь состояния

По событию *e11* должно осуществляться ветвление в зависимости от числа попыток. Поэтому добавим в граф переходов состояние *5* («Неверный pin»), в котором будет это ветвление осуществляться. Теперь установим конечным состоянием перехода по событию *e11* добавленное состояние. Граф переходов представлен на рис. 16.

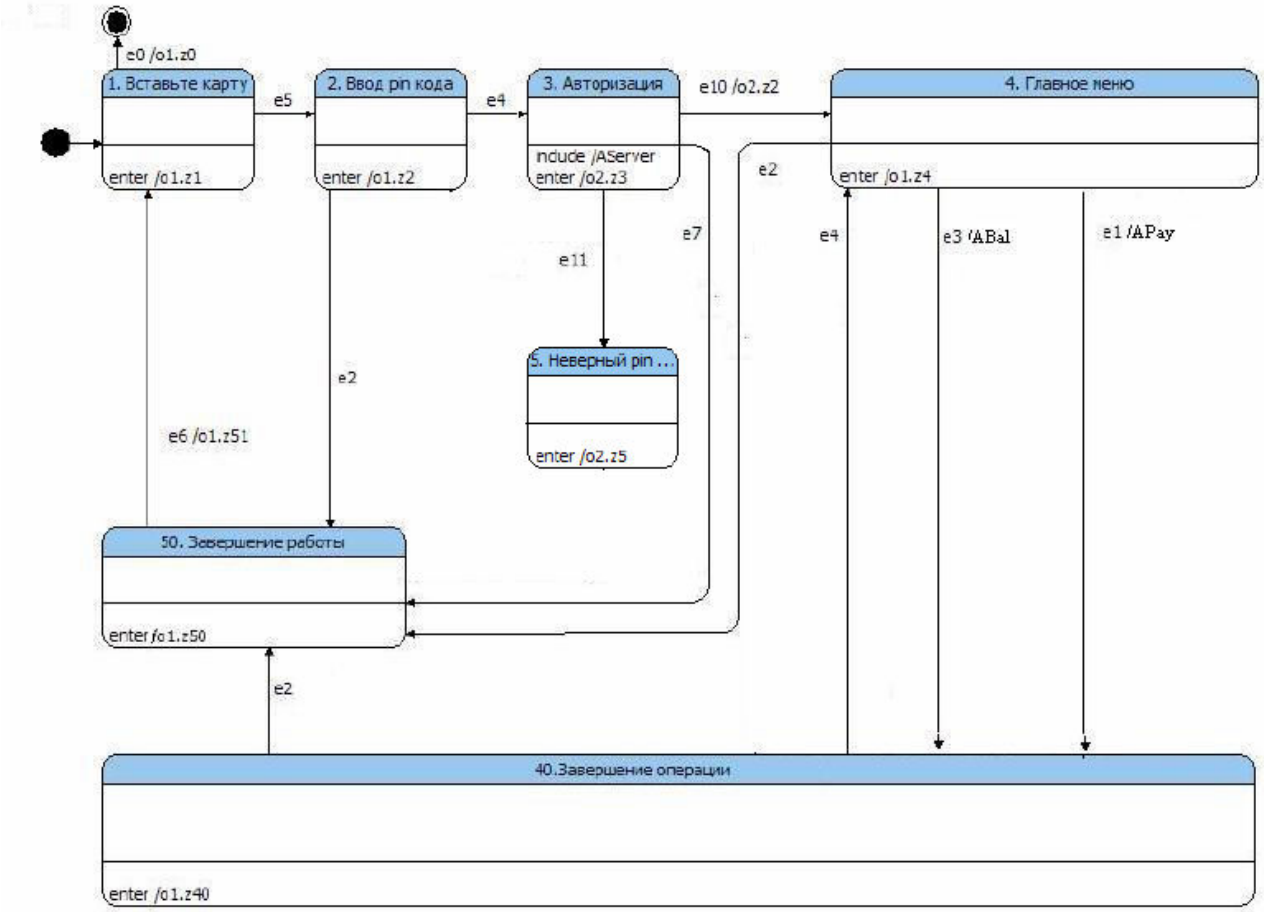

Рис. 16. Граф переходов автомата, отвечающего за работу банкомата, после добавления нового состояния

При входе в состояние *5* выполняется выходное воздействие *o2.z5* («Проверка количества попыток»). Воздействие *o2.z5* активизирует одно из двух событий: *e26* («Повторная авторизация») или *e7* («Карта заблокирована»). В первом случае, автомат должен перейти в состояние *2*, во втором – в состояние *50*. Соответствующие изменения графа переходов легко осуществить (рис. 17).

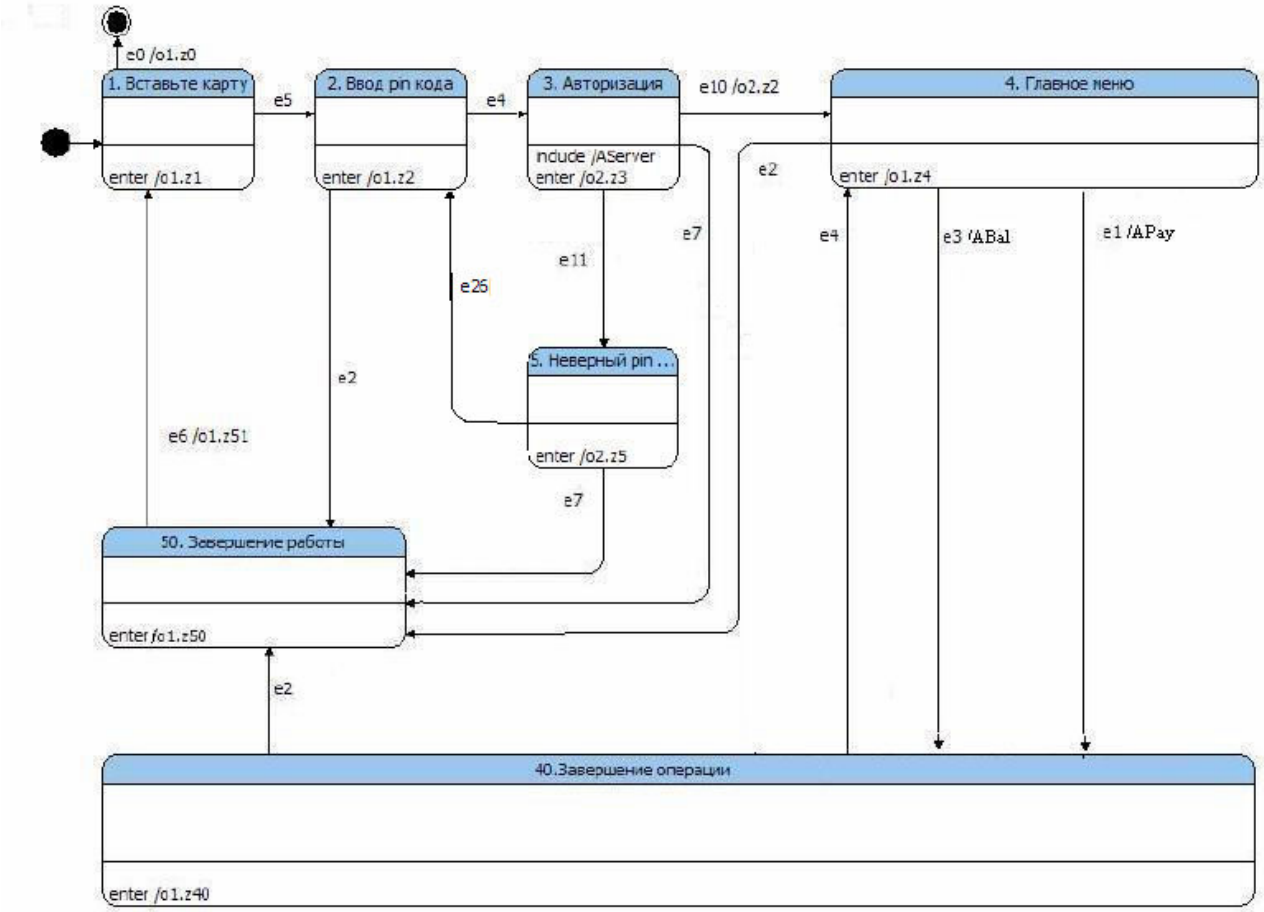

Рис. 17. Граф переходов автомата, отвечающего за работу банкомата

Полученный граф переходов удовлетворяет всем поставленным требованиям. Стоит отметить, что при модификации системы применялись только рефакторинги и базовые изменения, что позволило избежать анализа графа переходов после каждого проведенного изменения.

Выводы по главе 3

1. Предложен метод безопасного внесения изменений в графах переходов автоматных систем.
2. Предложенный метод применен для внесения изменений в систему автоматов, отвечающую за работу банкомата.

ЗАКЛЮЧЕНИЕ

1. Создана классификация изменений графов переходов автоматов. Рассмотрены проблемы, возникающие при проведении каждого конкретного изменения.

2. Введено понятие рефакторинга автоматов.

3. Создан каталог рефакторингов.

4. Разработан подход для безопасного внесения изменений в графы переходов автоматных систем. Предложено сложные изменения разбивать на композицию рефакторингов и базовых изменений.

5. Приведен пример использования описанного подхода.

СПИСОК ЛИТЕРАТУРЫ

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009. – 176 с. http://is.ifmo.ru/books/_book.pdf
2. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. – 628 с. http://is.ifmo.ru/books/switch/1
3. *Гуров В. С.* Технология проектирования и разработки объектно-ориентированных программ с явным выделением состояний (метод, инструментальное средство, верификация). Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО. 2008. http://is.ifmo.ru/disser/gurov_disser.pdf
4. *Лагунов И. А.* Разработка текстового языка автоматного программирования и его реализация для инструментального средства *UniMod* на основе автоматного подхода. Бакалаврская работа. СПбГУ ИТМО. 2008. http://is.ifmo.ru/papers/fsml/
5. *Фаулер М.* Рефакторинг. М.: Вильямс, 2003. – 435 с.
6. *Степанов О. Г.* Предметно-ориентированный язык автоматного программирования на базе динамического языка *Ruby*. Магистерская диссертация. СПбГУ ИТМО. 2006.
7. *Наумов А. С., Шалыто А. А.* Система управления лифтом. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/elevator/
8. *Козлов В. А., Комалева В. А.* Моделирование работы банкомата. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/bankomat/
9. *Балтийский И. А., Гиндин С. И.* Моделирование работы банкомата. СПбГУ ИТМО. 2008. http://is.ifmo.ru/unimod-projects/atm/