С авторами можно связаться по адресу: mail@avrorasystems.com, "для Шалыто".

Статья опубликована в журнале "Программист", 2002. №4. С.74-80.

Реализация автоматов при программировании событийных систем

© 2002 г. А.А. Шалыто, Н.И. Туккель

Федеральный научно-производственный центр – ФГУП "НПО "Аврора"",
Санкт-Петербургский государственный институт точной механики и оптики (технический университет)

В работе [1] был предложен подход к алгоритмизации и программированию задач логического управления, использующий конечные автоматы, который был назван "автоматным программированием", "программированием с явным выделением состояний" или "SWITCH-технологией" (www.softcraft.ru).

Изложим основные особенности этого подхода. Если процедурное программирование базируется на терминах "данные" и "процедуры", объектно-ориентированное – на "атрибутах" и "методах", а событийное – на "событиях" и "обработчиках событий", то в автоматном программировании, как и в теории управления, используются три термина: "входные воздействия", "состояния" и "выходные воздействия", причем состояния рассматриваются как абстракции и выделяются на этапе проектирования явно. По аналогии с Н. Виртом, который назвал свою книгу "Алгоритмы + Структуры данных = Программы", может быть введено соотношение: "Состояния + Входные воздействия + Выходные воздействия = Автомат". При этом автомат обычно рассматривается как конечный и детерминированный и предназначается для формального описания логики программ. Универсальность применения автоматов в программировании обоснована в работе [2].

Отметим, что автоматное программирование не противопоставляется процедурному, объектно-ориентированному и событийному, а может использоваться совместно с каждым их них в части описания логики, что продемонстрировано в проектах, приведенных на сайте www.softcraft.ru.

Одна из основных особенностей автоматного программирования состоит в том, что для обеспечения связи состояний с переменными введен этап, названный в работе [1] кодированием состояний. Такое кодирование применяется в теории автоматов, однако в традиционном программировании оно выполняется неявно путем эвристического введения флагов, что затрудняет понимание программ. В предлагаемом подходе для кодирования любого числа состояний автомата применяется одна многозначная переменная, что предопределило реализацию автоматов с помощью конструкции switch языка С. Указанная реализация выполняется формально по графу переходов, визуально описывающему поведение автомата. Под поведением при этом, в отличие от объектно-ориентированного программирования, понимаются не только его внешние проявления, но и внутренняя сущность, связанная с переходами между состояниями, за которыми можно наблюдать по значениям указанной переменной. Понятие "наблюдаемость" введено в программирование в работе [1].

При описании поведения сложных систем обычно используется не один автомат, а система взаимосвязанных автоматов, что позволяет, в частности, описывать параллельные процессы.

Предлагаемый подход, в отличие от других подходов, рассмотренных в работе [1], обеспечивает применение автоматов при проектировании, программировании, документировании и протоколировании.

Такой подход может эффективно применяться при разработке программного обеспечения программируемых логических контроллеров, предназначенных для управления технологическими процессами.

При переходе от программируемых контроллеров к промышленным компьютерам, функционирующим под управлением многозадачных операционных систем реального времени, технология была развита. При этом в качестве входных воздействий, кроме входных переменных, используются также и события, что позволило применять предложенную технологию при программировании событийных систем.

К этому классу систем относятся разнообразные приложения - от графических пользовательских интерфейсов [3] до компьютерных игр [4]. К ним также могут быть отнесены системы управления различными техническими объектами (бытовая техника, лифты, дизель-генераторы и т.д.)

Целесообразность применения автоматного подхода для указанного класса систем подтверждается тем, что создатель операционной системы UNIX К.Томпсон на вопрос о текущей работе ответил: "Мы создали язык генерации машин с конечным числом состояний, так как реальный селекторный телефонный разговор – это группа взаимодействующих машин с конечным числом состояний. Этот язык применяется в "Bell Labs" по прямому назначению – для создания указанных машин, а вдобавок с его помощью стали разрабатывать драйверы" [5].

Более того, в работе [6] приведено высказывание одного из участников семинара "Software 2000: a View of the Future": "Я знаю людей из "Боинга", занимающихся системами стабилизации самолетов с использованием чистой теории автоматов. Даже трудно себе представить, что им удалось сделать при помощи этой теории".

Автоматный подход может применяться и при разработке объектно-ориентированных программ с целью наглядного описания их поведения [7]. Пример применения такого подхода, названного авторами настоящей работы "объектно-ориентированным программированием с явным выделением состояний", приведен на сайте www.softcraft.ru.

Еще одним направлением использования автоматного подхода является визуализация вычислительных алгоритмов [8].

Таким образом, область применения SWITCH-технологии заключается в решении задач управления, понимаемого весьма широко.

Наибольшие трудности при использовании автоматного подхода связаны с пониманием особенностей функционирования автоматов в событийных системах. Настоящая работа призвана на примере программной реализации пояснить эти особенности. Выполнено сравнение предлагаемого подхода с аналогичным подходом, применяемым в языке объектно-ориентированного моделирования UML [7].

Классификация событий

События, под которыми в работе [7] понимаются все виды изменений, воспринимаемых системой, можно разделить на внешние (по отношению к программе) и внутренние (порождаемые внутри программы). К внешним можно отнести события, связанные с нажатием кнопок, включением/отключением переключателей и сигнализаторов, а в общем случае – сообщения от других процессов.

Внутренние события делятся на специальные, которые обеспечивают работу программы, и события, предназначенные для взаимодействия автоматов между собой (взаимодействие по вызываемости) или с вспомогательными модулями программы. При этом под вызываемостью понимается формирование внутреннего события в выходном воздействии одного из автоматов и обработка этого события другим автоматом. К специальным событиям относятся, например, событие запуска программы или событие, переводящее вложенный автомат в начальное состояние.

Отметим, что события срабатывания таймеров можно отнести как к внешним, так и к внутренним событиям в зависимости от реализации таймеров. Если они реализуются

операционной системой, то события их срабатывания являются внешними. Если же они реализованы внутри рассматриваемой программы, то события будут внутренними.

Обработка событий

В работе [3] приведена структурная схема построения событийных систем, в которой наряду с внешними событиями присутствуют их обработчики. Однако, в отличие от традиционного подхода, при котором обработчики содержат логику и взаимодействуют между собой через флаги, в предложенной схеме обработчики минимально "нагружены" логикой и зачастую содержат только обращение к автомату, обрабатывающему указанные события. При этом обработчики могут содержать "фильтры", выделяющие из потока событий требуемое, что демонстрируется в примере, приведенном в работе [3]. Предлагаемый подход по сравнению с традиционным позволяет упростить понимание логики программ.

Для обработки события автомат должен быть запущен после возникновения события. Под запуском автомата будем понимать вызов функции, которая его реализует. Объявим эту функцию как void A(int e). При этом фраза "запуск автомата с событием e10" означает вызов этой функции из обработчика события с передачей ей номера события в качестве параметра: A(10). Запуск автомата с некоторым событием e также можно назвать "обработка автоматом события e". Завершение обработки события соответствует завершению выполнения функции, реализующей автомат.

В ряде случаев для обработки события одного автомата недостаточно, и сложное поведение программы обеспечивается системой взаимосвязанных автоматов. Эта система обычно имеет иерархическую структуру, отражающую вложенность автоматов. При этом обработка события выполняется следующим образом: из обработчика события запускается головной автомат в иерархии, который, в свою очередь, последовательно запускает с тем же событием автоматы, вложенные в его текущее состояние. Те, в свою очередь, запускают автоматы, вложенные в них.

В отличие от внешних событий, при обработке внутренних событий автоматы обычно запускаются не из обработчиков событий.

Для корректной реализации вложенных автоматов введено специальное внутреннее событие e0, порождаемое в головном автомате и сообщающее вложенному автомату, что головной автомат перешел в состояние, которое содержит этот вложенный автомат. Реакция на событие e0 определена графом переходов вложенного автомата и в большинстве случаев заключается в обеспечении его перехода в начальное состояние.

Входные воздействия автоматов

Рассмотрим разновидности входных воздействий автоматов в событийных системах: события; входные переменные; предикаты, проверяющие номера состояний других автоматов, которые могут быть отнесены к входным переменным.

При программной реализации события, как отмечалось выше, запускают автоматы, а входные переменные опрашиваются после запуска автоматов при проверке условий переходов. Все разновидности входных воздействий могут участвовать в условиях на дугах и петлях графов переходов.

Различия между событиями и входными переменными поясним на примере кнопки (рис. 1). Она может являться источником следующих входных воздействий:

– событие "Нажатие кнопки";

– событие "Отпускание кнопки";

– входная переменная "Кнопка нажата" (ее инверсия – "Кнопка отпущена").

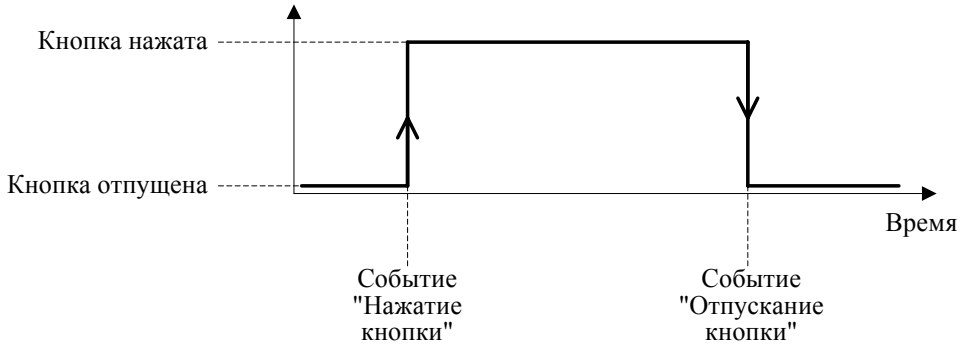

Рис. 1. Временная диаграмма нажатия кнопки

Рассмотренный пример показывает, что один и тот же источник информации может одновременно использоваться и как входная переменная, и как событие: входные переменные определяются значениями входных воздействий, а события соответствуют изменениям этих значений. Выскажем рекомендации по применению тех или иных входных воздействий при проектировании автоматных программ.

1. Входные воздействия лучше использовать как входные переменные. Кроме того, в качестве входных переменных могут применяться "запомненные" события.

2. Использование событий целесообразно для входных воздействий, имеющих кратковременный характер (например, кнопки без памяти и таймеры) и для быстрой реакции на их изменения.

3. В некоторых случаях входное воздействие может одновременно применяться и как переменная, и как событие. В качестве примера приведем сигнализатор открытого положения двери микроволновой печи. Входное воздействие от этого сигнализатора должно рассматриваться как входная переменная, так как его необходимо опрашивать, проверяя допустимость включения печи. Одновременно это воздействие должно рассматриваться как событие, так как при открытии двери печи во время ее работы, печь должна быть отключена немедленно, а не после опроса входной переменной "Дверь открыта" в результате прихода какого-либо другого события. Событие "Открытие двери" может не использоваться в условиях переходов на графе, а применяться только для запуска автомата, соответствующий переход в котором будет осуществляться по входной переменной "Дверь открыта".

Особенности функционирования автоматов

В работе [3] была приведена нотация для визуализации семантики графов переходов, используемых в событийных системах (рис. 2).

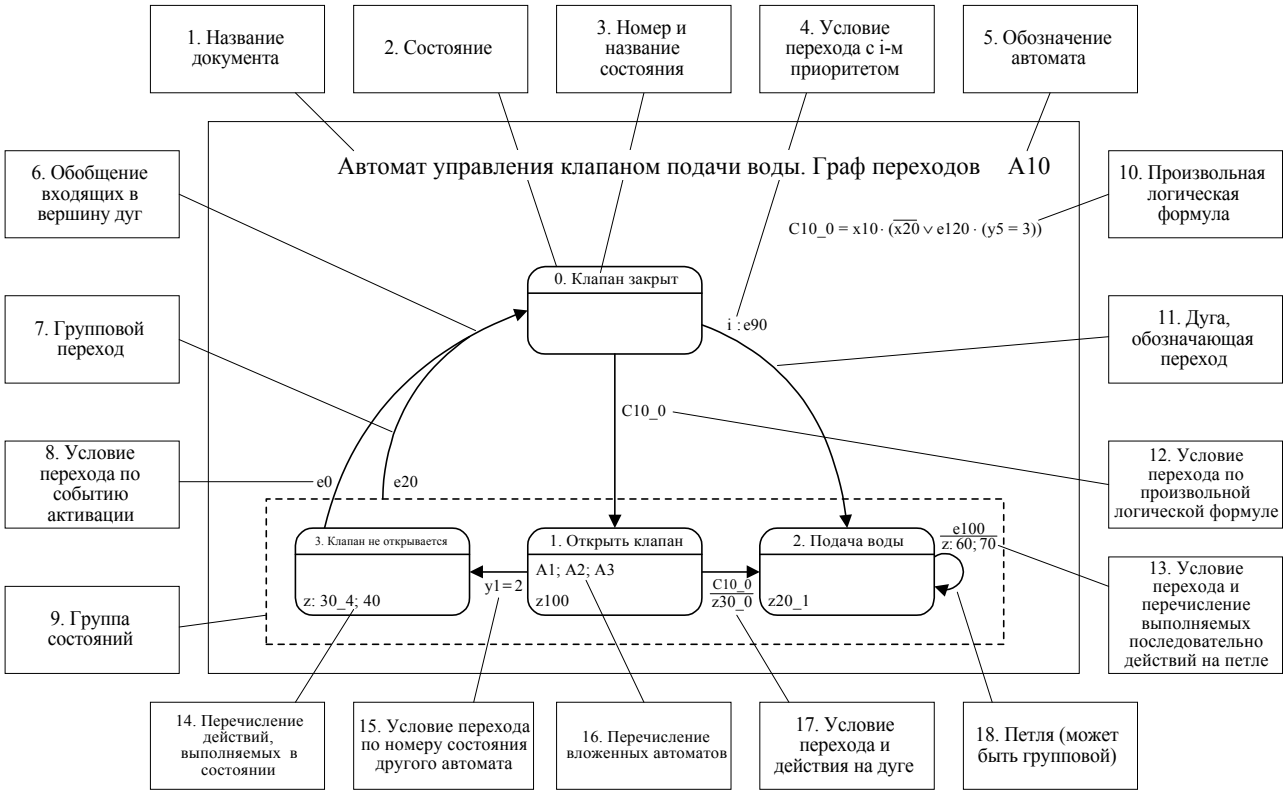

Рис. 2. Нотация, применяемая при построении графов переходов

На примере реализации графа переходов (рис. 3) рассмотрим особенности функционирования автоматов в событийных системах. В этом и дальнейших примерах схемы связей автоматов не приводятся - в рамках этой статьи физический смысл воздействий не представляет интереса. По этой же причине не приводятся тексты реализуемых отдельно функций обработчиков событий, входных переменных x() и выходных воздействий z().

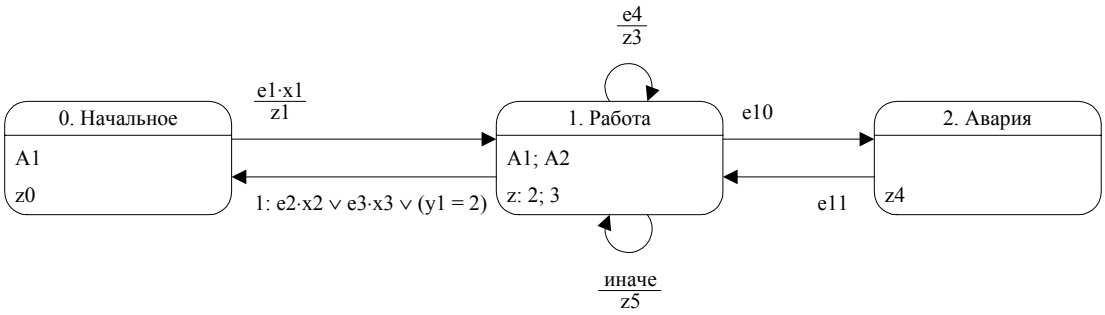

Рис. 3. Граф переходов автомата A0

Ниже приведен текст функции, формально построенной по графу переходов в соответствии с шаблоном [3].

Реализация автомата на языке Си

```
 1  int y0 = 0;

 2  void A0(int e)
    {
 3    int y_old = y0;

 4    switch(y0)
      {
 5      case 0:
 6        A1(e);
 7        if(e == 1 && x1()) { z1();                          y0 = 1; }
 8      break ;

        case 1:
 9        A1(e); A2(e);
10        if(e == 2 && x2() || e == 3 && x3() || (y1 == 2)) y0 = 0;
          else
          if(e == 10)                                       y0 = 2;
          else
11        if(e == 4) { z3(); }
12        else
          { z5(); }
        break;

        case 2:
          if(e == 11)                                       y0 = 1;
        break;
      }

13    if(y0 == y_old) return;

14    switch(y0)
      {
15      case 0:
16        A1(0);
17        z0();
        break;

        case 1:
18        A1(0) ; A2(0);
19        z2() ; z3();
        break;

        case 2:
          z4();
        break;
      }
    }
```

Приведем пояснения к помеченным номерами строкам программы.

1. В рамках примера объявим переменную состояния автомата A0 как глобальную. Так как начальным состоянием автомата считается состояние с номером 0, инициализируем переменную y0 этим значением. Таким образом, в отличие от диаграмм состояний, используемых в UML [7], нет необходимости в использовании специального обозначения для начального состояния.

2. В качестве параметра функции используется номер обрабатываемого автоматом события. При запуске автомат обрабатывает только одно событие. Каждое следующее событие начинает обрабатываться только после завершения обработки предыдущего. Отметим также, что обработка события автоматом весьма кратковременна.

3. Текущее состояние автомата запоминается во вспомогательной переменной.

4. Первый оператор switch запускает вложенные в текущее состояние автоматы и проверяет условия переходов на исходящих дугах и петлях.

5. Каждой вершине графа переходов соответствует своя метка case.

6. Вложенный автомат A1 запускается с тем же событием, с которым был запущен автомат A0. Автомат A1 может быть реализован аналогично, и не приводится. Отметим, что такая реализация одинакова для любых автоматов любого уровня вложенности.

7. Проверяется условие перехода на дуге из вершины 0 в вершину 1. Если условие перехода выполняется, то формируется выходное воздействие: вызывается функция z1(), а переменной состояния присваивается новое значение. Отметим, что обозначению "e1" (на графе) соответствует предикат "e == 1". Выходные воздействия, например указанная функция z1(), являются командами (действиями), выполняемыми кратковременно. Они, в свою очередь, могут управлять длительными процессами.

8. Оператор break соответствует петле, отражающей устойчивость состояния, которая для упрощения графа умалчивается [3]. Не умалчиваются только те петли, которые содержат действия.

9. Вложенные автоматы A1 и A2 запускаются последовательно с тем же событием, с которым был запущен головной автомат A0.

10. Условиями переходов могут быть произвольные логические формулы, содержащие комбинации входных воздействий. Как отмечалось ранее, события между собой ортогональны (они не обрабатываются одновременно), и поэтому для обеспечения непротиворечивости переходов достаточно ввести приоритет ("1:") только на одной из дуг, исходящих из вершины с номером 1. Приоритет используется для явного задания порядка проверки условий переходов. Отметим, что обозначению "(y1 = 2)" (на графе) соответствует предикат "(y1 == 2)".

11. Реализация петли, содержащей действие, выглядит так же, как и реализация дуги, но не содержит оператора, изменяющего значение переменной состояния. Действие z3() будет выполняться каждый раз, когда автомат запускается в состоянии 1 с событием e4, при условии, что не выполнилось условие перехода с более высоким приоритетом.

12. Петле с пометкой "иначе" соответствует последний оператор else.

13. Если состояние автомата не изменилось, то после выполнения первого оператора switch обработка события завершается. Таким образом, второй оператор switch выполняется только при изменении состояния. Также следует отметить, что в такой реализации автомат при обработке события выполняет не более одного перехода.

14. Второй оператор switch запускает вложенные автоматы с событием e0 и выполняет действия, указанные в вершинах графа.

15. Каждому состоянию, содержащему вложенные автоматы и/или выходные воздействия, соответствует своя метка case.

16. Запустить автомат A1, вложенный в нулевое состояние автомата A0, с событием e0.

17. Ввиду того, что второй оператор switch выполняется только при изменении состояния, следует, что выходные воздействия, указанные в вершинах графа переходов, будут формироваться однократно при переходе автомата в соответствующее состояние. В примере функция z0() выполняется при переходе автомата в состояние 0.

18. Последовательно запустить автоматы A1 и A2, вложенные в состояние 1 автомата A0, с событием e0.

19. Последовательно выполнить действия после перехода в состояние 1.

Обработка автоматом одного события - весьма кратковременный процесс при условии, что используемые функции, реализующие входные переменные и выходные воздействия также кратковременны.

Построим для рассматриваемого автомата пример протокола его работы, который можно было бы получить автоматически, добавив в текст программы вызовы функций протоколирования:

```
Запуск обработчика события e1
Запуск автомата A0 в состоянии 0 с событием e1
  Запуск автомата A1 в состоянии 0 с событием e1
    ...
  Завершение работы автомата A1 в состоянии 0
  Проверить x1 – истина
  Выполнить z1
  Автомат A0 перешел из состояния 0 в состояние 1
  Запуск автомата A1 в состоянии 0 с событием e0
    ...
  Завершение работы автомата A1 в состоянии 0
  Запуск автомата A2 в состоянии 0 с событием e0
    ...
  Завершение работы автомата A2 в состоянии 0
  Выполнить z2
  Выполнить z3
Завершение работы автомата A0 в состоянии 1
Завершение обработчика события e1
```

Из протокола следует, что автоматы, вложенные более чем в одно состояние головного автомата, могут запускаться дважды при обработке одного события. В примере при переходе автомата A0 из состояния 0 в состояние 1 вложенный в оба этих состояния автомат A1 сначала запускается с событием e1, а потом с событием e0.

Сравнение с диаграммами состояний Statecharts

Для анализа поведения одиночного объекта в UML используются диаграммы состояний (Statecharts diagram [7]). Покажем, что применяемые в SWITCH-технологии графы переходов обладают рядом преимуществ по сравнению с указанными диаграммами состояний:

– вместо специфических для диаграмм Statecharts понятий, таких как "действие при входе в состояние", "действие при выходе из состояния", "внутренний переход", "сторожевое условие", "событие-триггер" и т.д., использовано минимально необходимое число понятий, что позволяет применить такие классические модели теории автоматов, как автомат Мура, автомат Мили и смешанный автомат;

– рассматриваемые графы переходов реализуются по шаблону, в то время как вопрос о реализации диаграмм состояний в UML не рассматривается;

– применяются краткие символьные обозначения, которые расшифровываются в другом документе – схеме связей автомата [2,3]. Несмотря на то, что используемые в диаграммах Statecharts смысловые обозначения кажутся более понятными, при разработке программ с большим числом входных и выходных воздействий их применение практически невозможно [3];

– в качестве условий переходов используются произвольные логические формулы, в том числе содержащие события, объединенные "по ИЛИ", а не только отдельные события, помеченные "сторожевыми условиями";

– количество выходных воздействий, формируемых в вершинах, дугах и петлях не ограничено одним действием;

– нет необходимости использовать специфическое обозначение начальной вершины;

– отсутствуют специфические виды состояний, такие как параллельные, исторические и т.д., вместо которых используются вложенные автоматы, количество и уровень вложенности которых неограниченно, что невозможно в диаграммах Statecharts;

– кроме вложенности, автоматы могут взаимодействовать за счет вызываемости и обмена номерами состояний. Указанное взаимодействие автоматически протоколируется, что заменяет применяемые в UML диаграммы последовательности и кооперации, которые при сложной логике программ не могут быть построены вручную.

В предлагаемом подходе вложенные автоматы обладают свойством сохранения своего состояния вне зависимости от того, в каком состоянии находится головной автомат. Поэтому

отсутствует необходимость введения такой дополнительной сущности, как "историческое состояние".

На рис. 4 приведен граф переходов автомата A2, вложенного в состояние 1 автомата A0 (рис. 3). Из рассмотренной выше реализации следует, что состояние автомата A2 не изменяется, если автомат A0 находится в состоянии, отличном от состояния 1. При запуске автомата A2 с событием e0 ни одно из условий переходов этого автомата не будет выполняться. Таким образом, автомат A2 не изменяет своего состояния при обработке события e0. Поэтому при переходе автомата A0 в состояние 1, автомат A2 будет запущен с событием e0, но не изменит своего состояния.

Исходя из изложенного, можно утверждать, что поведение автомата A2 обладает свойствами "исторического состояния", используемого в диаграммах Statecharts, без применения дополнительных специфических обозначений.

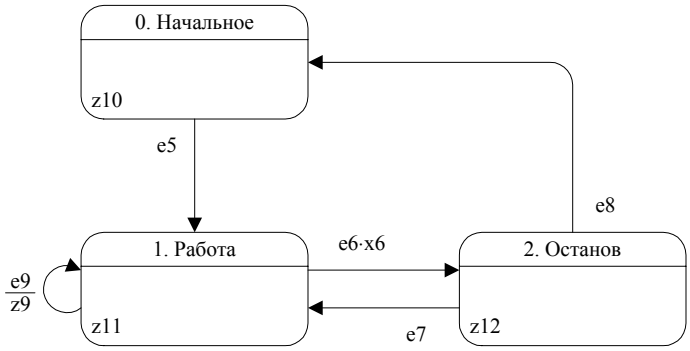

Рис. 4. Граф переходов автомата A2 без использования события e0

Рассмотрим случай, когда сохранение состояния вложенного автомата при изменении состояния головного автомата не требуется. Если необходимо, чтобы автомат A2 возвращался в начальное состояние каждый раз, когда автомат A0 переходит в состояние 1, следует выполнить его модификацию, добавив переходы из состояний 1 и 2 в состояние 0 по событию e0. Эти переходы могут быть изображены компактно в виде одной групповой дуги, которая используется только для упрощения изображения графа переходов (рис. 5). Рассмотренный граф переходов реализуется так же, как если бы групповая дуга с пометкой e0 исходила из каждой вершины в группе.

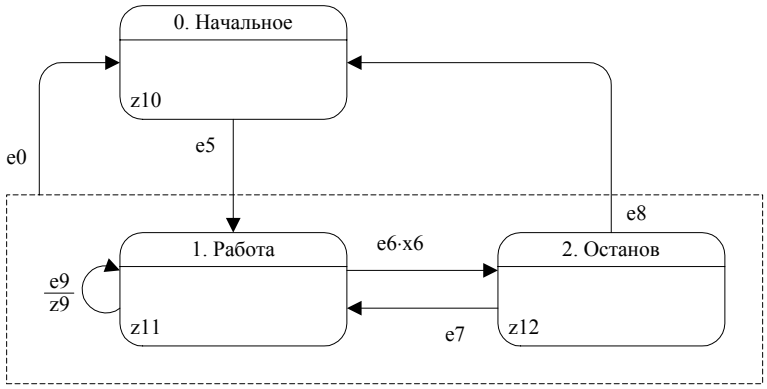

Рис. 5. Граф переходов автомата A2 с традиционным использованием события e0

В заключение отметим, что предлагаемый подход был разработан в ФГУП "НПО "Аврора"" (Санкт-Петербург) и в течение ряда лет преподается студентам кафедры

"Компьютерные технологии" Санкт-Петербургского государственного института точной механики и оптики.

Литература

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
2. Шалыто А.А., Туккель Н.И. От тьюрингова программирования к автоматному //Мир ПК. 2002. №2. С.144-149.
3. Шалыто А.А., Туккель Н.И. Программирование с явным выделением состояний //Мир ПК. 2001. №8. С. 116-121, №9. С. 132-138.
4. Секреты программирования игр /А. Ла Мот, Д. Ратклифф, М. Семинаторе и др. СПб.: Питер, 1995. 278 с.
5. Кук Д., Урбан Д., Хамилтон С. Unix и не только. Интервью с Кеном Томпсоном //Открытые системы. 1999. №4. С. 35-47.
6. Карпов Ю.Г. Теория алгоритмов и автоматов. СПб.: Геликон Плюс, 2000.
7. Рамбо Д., Якобсон А., Буч Г. UML. Специальный справочник. СПб.: Питер, 2002. 652 с.
8. Шалыто А.А., Туккель Н.И. Реализация вычислительных алгоритмов на основе автоматного подхода //Телекоммуникации и информатизация образования. 2001. №6. С. 35-53.