Санкт-Петербургский университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

И.М. Аничкин

Использование автоматного программирования при построении редактора графа переходов

Санкт-Петербург
2003

Содержание

Введение

Для алгоритмизации и программирования задач логического управления была предложена SWITCH-технология [1], которая в дальнейшем была развита применительно к разработке событийных систем. Данная технология предлагает проектировать и реализовывать алгоритмы управления, представляя их в виде системы взаимосвязанных автоматов, поведение которой формализуется с помощью системы взаимосвязанных графов переходов. Далее эти графы изоморфно преобразуются в текст программы, основу которого составляет конструкция *switch* языка C/C++ или ее аналоги в других языках программирования [2].

Для построения графов переходов предлагается использовать пакет *Microsoft Visio* со специальными шаблонами, а для преобразования графов в текст программы – конвертор *Visio2SWITCH* (программа *v2s*). Этот конвертор размещен на сайте http://is.ifmo.ru, раздел «Последователи».

Использование указанного конвертора имеет ряд недостатков. Во-первых, необходимость применения пакета *Visio*. Этот пакет предназначен для решения весьма широкого круга задач, и его использование только для рассматриваемой задачи экономически не оправдано. Во-вторых, преобразование графов переходов в текст программы производится другой программой – *v2s*, что является неудобным. И, наконец, выходным языком для программы *v2s* является язык *C/C++*, тогда как для ряда приложений необходимо иметь в качестве выходного другие языки программирования, например, язык *Pascal* (*Delphi*). Последний аргумент явился решающим для выполнения данного проекта.

В настоящей работе описывается первая версия программы, обеспечивающей редактирование одного графа переходов, в вершины которого могут быть вложены другие автоматы. Его логика также построена с использованием графов переходов, образующих взаимосвязанную систему.

Программа создана с помощью интегрированной среды разработки *Delphi*.

Кроме редактора графов приложение содержит также редактор схемы связей автомата, который в данной работе не рассматривается.

1. Постановка задачи

Целью работы является разработка редактора графа переходов для одного автомата на основе автоматного подхода. При этом в вершины этого графа могут быть вложены другие автоматы.

Первоначально автором был построен редактор графа переходов на основе объектного подхода без использования автоматов. В настоящей работе этот редактор применялся для создания графов переходов нового редактора (использующего как объекты, так и автоматы) и их преобразования в текст программы редактора.

2. Описание подхода

При разработке редактора использовался следующий подход.

1. Данные, относящиеся непосредственно к структуре графа переходов, а также некоторые функции по их обработке, выделяются в отдельный класс *TGraph*, который включает в себя все элементы графа. Этот класс для удобства содержит также некоторые данные и функции, не имеющие непосредственного отношения к структуре графа.
2. Данные, не имеющие непосредственного отношения к структуре графа, выделяются в другие классы.
3. Для каждого элемента графа определяется перечень операций, которые могут быть выполнены над этим элементом в ходе редактирования графа. Операции реализуются методами соответствующих классов.
4. Для каждого события, происходящего в системе, которое имеет значение для работы редактора (нажатие клавиши, передвижение мыши, нажатие кнопки и т.д.), выделяется идентификатор – номер, с которым это событие будет передаваться в функции, реализующие автоматы.
5. Кроме событий в системе используются также и входные переменные, являющиеся внутренними для редактора.
6. Выделяются основные состояния, в которых будет находиться система: нахождение курсора мыши вне окна отрисовки, нахождение курсора мыши над окном отрисовки, процесс редактирования и т.д.
7. В некоторых состояниях производится выделение вложенных автоматов. Для построения каждого из этих автоматов определяются состояния (перемещение объектов, изменение размеров прямоугольника и т.д.).
8. Выходные воздействия автоматов включают в себя:
   - операции, производимые над элементами графа;
   - изменение значений некоторых переменных, применяемых для описания состояний системы в тех случаях, когда с этой целью невозможно непосредственно использовать автомат с его состояниями (например, для выбора текущего элемента графа, когда число состояний заранее неизвестно).
9. Основная часть логики программы реализуется на автоматах. Оставшаяся часть, не имеющая прямого отношения к управлению редактором, выносится в функции выходных воздействий или, если возможно, в методы классов.

3. Структурная схема программы

Программа состоит из пяти основных составляющих:

- классы (неавтоматные);
- события;
- автоматы, реализующие логику работы редактора;
- глобальные переменные (включая входные);
- выходные воздействия.

Отметим, что автоматы выделены в отдельную составляющую, так как они только «обернуты» в классы для удобства работы.

Структурная схема программы редактора графов переходов приведена на рис. 1.

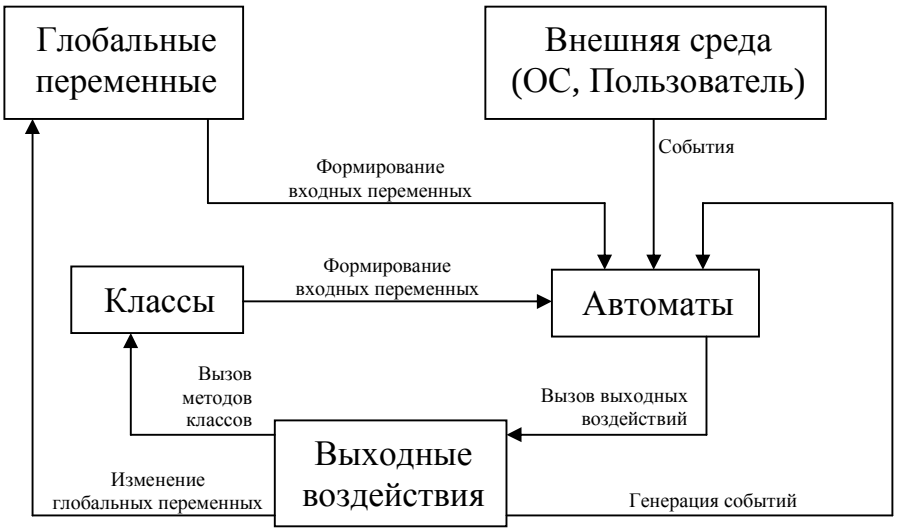

Рис.1. Структурная схема программы редактора

4. Диаграмма классов

На диаграмме классов изображены связи между объектами, которые соответствуют составляющим редактируемых графов переходов (рис.2).

Головной класс *TGraph*, представляющий граф переходов, содержит три списка (массива):

- состояний (*TState*);
- групп состояний (*TGroup*);
- дуг (*TArch*).

Дуга, в свою очередь, состоит из точек (*TPointObj*) и линий (*TLineObj*).

Кроме перечисленных классов программа содержит также другие классы, являющиеся в иерархии их предками (например, класс *TVertex*).

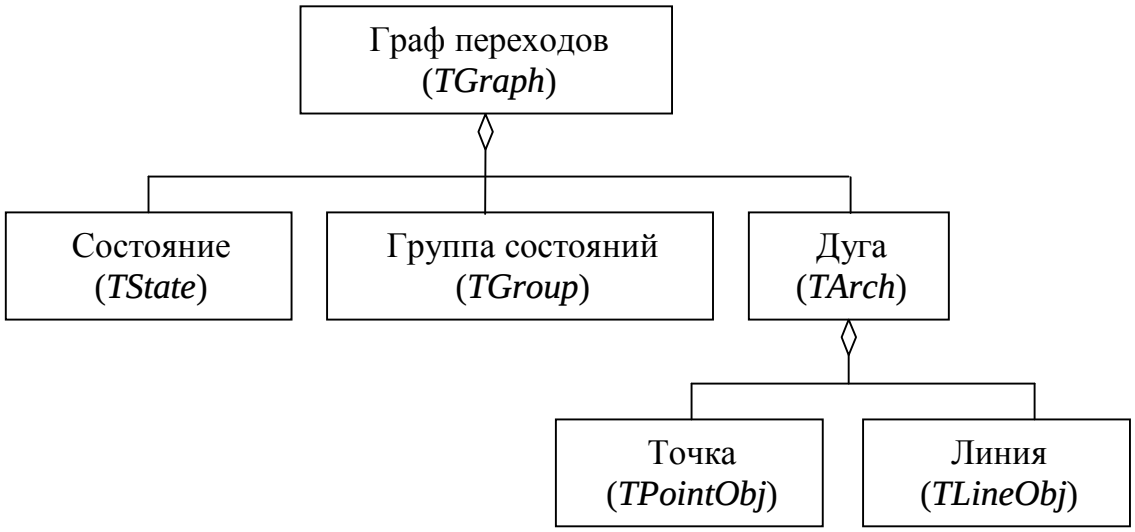

Рис. 2. Диаграмма классов

5. Иерархия классов

На рис. 3 приведена иерархия классов, отражающая наследование, в которой кроме классов, указанных на диаграмме (рис. 2), указаны также и ряд других классов, например класс *TVertex* – общий предок группы и состояния.

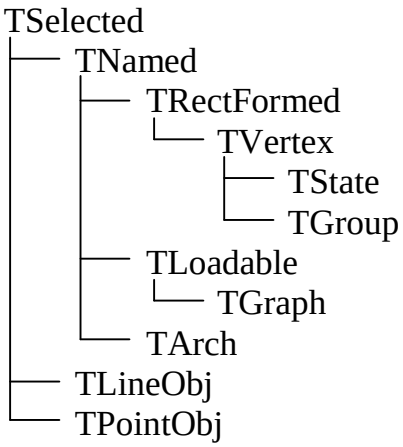

Рис. 3. Иерархия классов

6. Описание классов

6.1. Класс «Граф переходов»

Класс *TGraph* является головным классом в иерархии. Он содержит в себе все данные, относящиеся непосредственно к графу, а также методы работы с ними: добавление и удаление элементов, выделение и снятие выделения, вычисление расстояний между элементами и расстояния от точки до элемента, поиск элемента, ближайшего к заданной точке, определение различных признаков.

В этот класс входят также функции загрузки графа и сохранения его в файл, а также функция компиляции графа переходов в текст программы.

6.2. Класс «Вершина»

Класс *TVertex* является общим предком для классов *TState* и *TGroup*. Объект данного класса представляет собой прямоугольник, имеющий номер и название, к которому могут подключаться входящие и исходящие дуги.

Класс содержит методы для перемещения и изменения размеров объекта, а также для подключения и отключения дуг.

6.3. Класс «Состояние»

Класс *TState* представляет состояние автомата. Помимо атрибутов вершины он содержит список вызываемых из данного состояния автоматов и производимых при инициализации действий.

Каждое состояние может входить в ноль или более групп. Для включения в группу и исключения из нее имеются соответствующие методы.

6.4. Класс «Группа состояний»

Класс *TGroup* представляет собой группу состояний автомата. Каждая группа может включать в себя ноль или более состояний. Для добавления/удаления состояний имеются соответствующие методы.

6.5. Класс «Дуга»

Класс *TArch* представляет собой дугу графа переходов автомата. Дуга может начинаться и заканчиваться либо на каком-то состоянии, либо на пустом месте (в процессе редактирования такая ситуация допустима). Дуга включает в себя списки точек и линий, из которых она состоит, и методы работы с ними: добавление и удаление, получение координат точек и углов наклона линий, а также функцию перенаправления начала (конца) дуги на другую вершину.

Также дуга содержит условие перехода и список действий, производимых на нем. Расположение текста «условие перехода/действие» в окне отрисовки контролируется специальными атрибутами и методами, обеспечивающими определение координат текста и его перемещение.

6.6. Классы «Точка» и «Линия»

Классы *TLineObj* и *TPointObj* введены для удобства работы с отдельными элементами дуги. Точка содержит свои координаты, линия – угол наклона. В обоих классах есть методы для перемещения объекта, а также для подгонки координат (угла) в соответствие с расположением соседних элементов дуги.

7. Автоматы

Логика управления процессом редактирования графа переходов реализуется с помощью четырех взаимосвязанных автоматов. Нумерация автоматов выполнена следующим образом: номер вложенного автомата повторяет номер автомата, в который он вложен, после этого указывается порядковый номер этого автомата среди вложенных. Например, автомат A12 – это второй автомат, вложенный в автомат A1.

В силу того, что, как отмечалось во введении, редактор схемы связей автоматов не рассматривается, соответственно не описываются и автоматы, реализующие логику его работы. В качестве этих автоматов будут использоваться автомат A11 и вложенные в него автоматы.

Использование автоматов позволяет реализовывать обработчики событий иначе, чем это делается традиционно. При этом обработчик события содержит не функцию его обработки (с флагами, которые могут влиять на поведение других обработчиков), а вызов соответствующего автомата с этим событием [2]. Это повышает уровень централизации логики программы и упрощает внесение изменений при повторном ее использовании.

В графах переходов используются весьма длинные символические обозначения входных переменных и выходных воздействий. Это объясняется тем, что функции выходных воздействий, вызываемые из одного и того же автомата и выполняющие сходные действия, удобно называть сходными именами для лучшего понимания алгоритма работы программы пользователем, читающим программную документацию, или самим разработчиком при последующей модификации программы. При использовании таких обозначений очевидно, например, что функции z12305 – z12312 вызываются из автомата A123, отвечающего за редактирование графа, и выполняют сходные действия (в данном случае – изменение размеров прямоугольника, изображающего вершину графа). Если нумеровать функции произвольным образом, то, по мнению автора, понимание алгоритма работы программы окажется более трудным.

Отметим, что приведенные ниже графы построены с помощью редактора, разработанного в настоящей работе.

7.1. Автомат A12 (GraView) - автомат просмотра графа

7.1.1. Словесное описание

Автомат отслеживает происходящие события – перемещения курсора мыши, нажатия на кнопки мыши, нажатия некоторых клавиш. В зависимости от положения курсора мыши (над окном редактирования или вне его) могут осуществляться различные действия. Например, при нажатии на кнопку мыши в окне редактирования может происходить выделение текущего элемента и некоторые действия, связанные с редактированием графа.

В данном автомате реализуется также редактирование текста на дугах и вершинах (списков вызываемых автоматов, выходных воздействий и условий переходов).

7.1.2. Схема связей

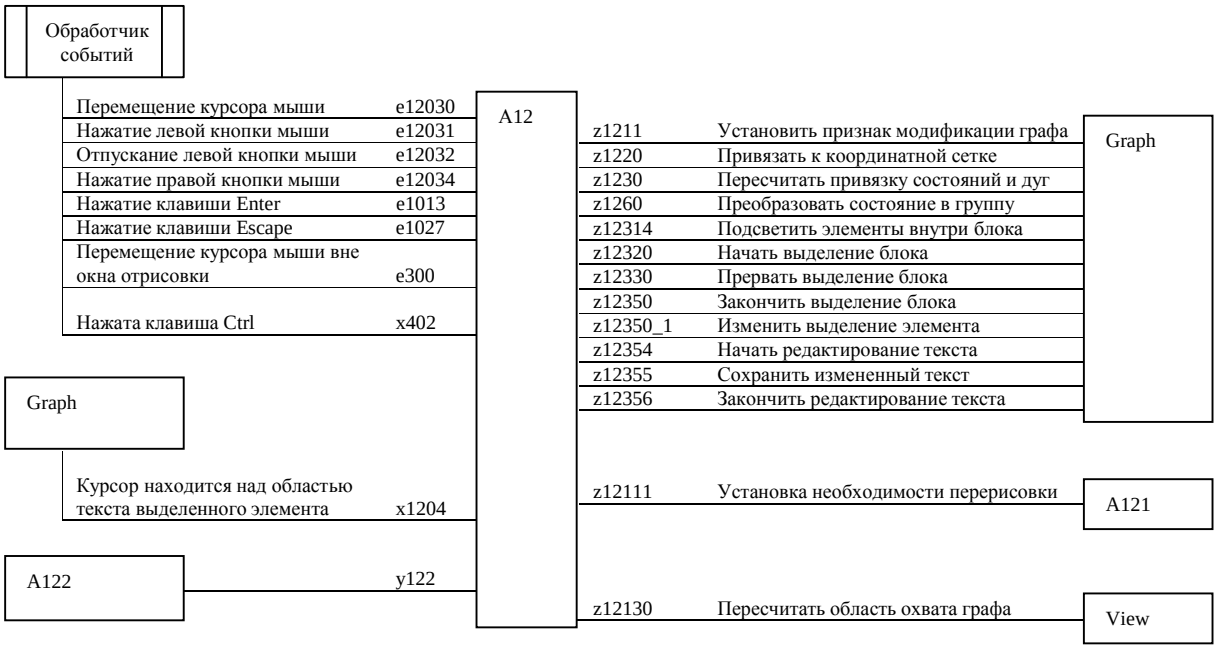

Рис. 4. Схема связей автомата просмотра графа A12

7.1.3. Граф переходов

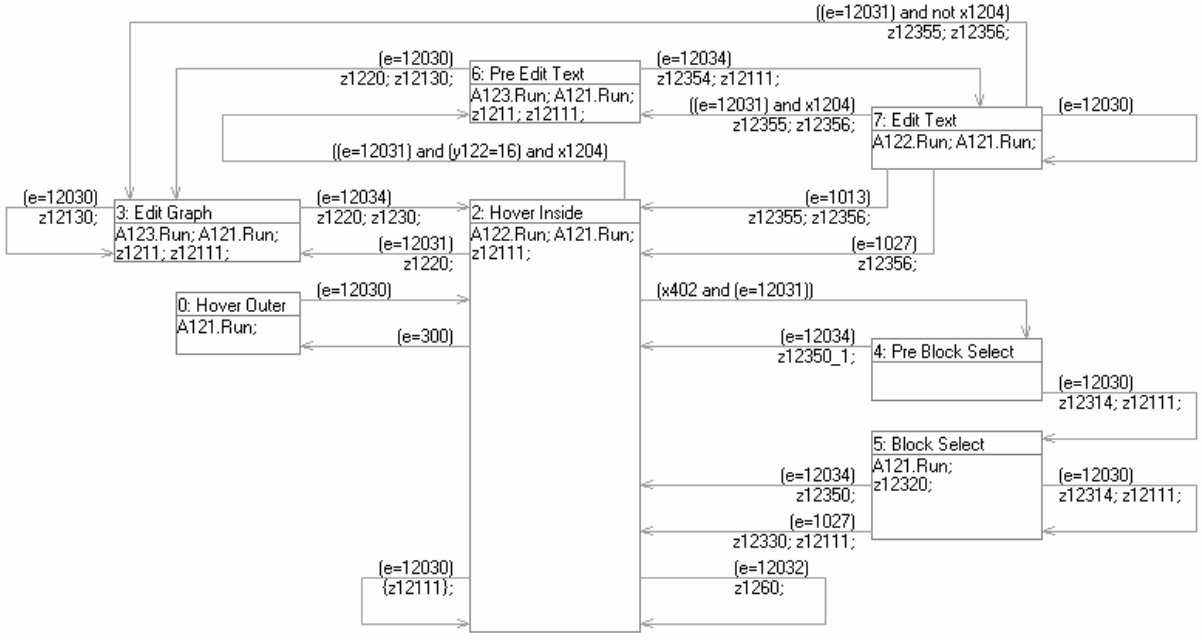

Рис. 5. Граф переходов автомата просмотра графа A12

7. 2. Автомат A121 (GraVDraw) - автомат отрисовки графа

7. 2. 1. Словесное описание

Автомат производит перерисовку окна отображения при необходимости обновить его содержимое. Отрисовка осуществляется по системному событию *onIdle*, когда система бездействует.

7. 2. 2. Схема связей

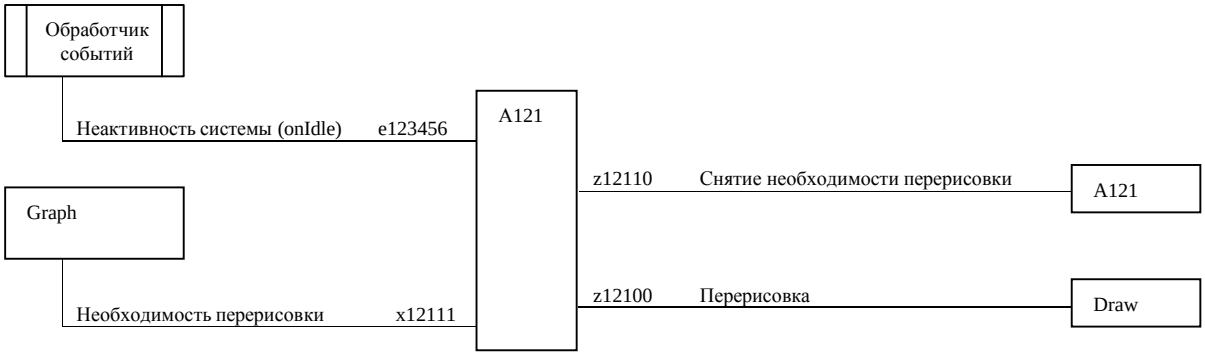

Рис.6. Схема связей автомата отрисовки графа A121

7.2.3. Граф переходов

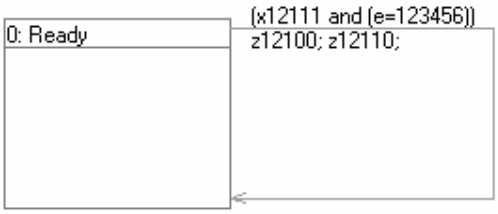

Рис.7. Граф переходов автомата отрисовки графа A121

7.3. Автомат A122 (GraVHilight) - автомат подсветки элементов графа

7.3.1. Словесное описание

Автомат осуществляет подсветку элемента графа при нахождении курсора над элементом, а также устанавливает форму курсора в зависимости от того, над каким элементом и над какой его частью находится курсор. Интуитивно понятно, что стрелка вправо – влево предлагает произвести изменение горизонтальных размеров объекта, курсор в виде буквы `I` – редактирование текста, и т.д.

7.3.2. Схема связей

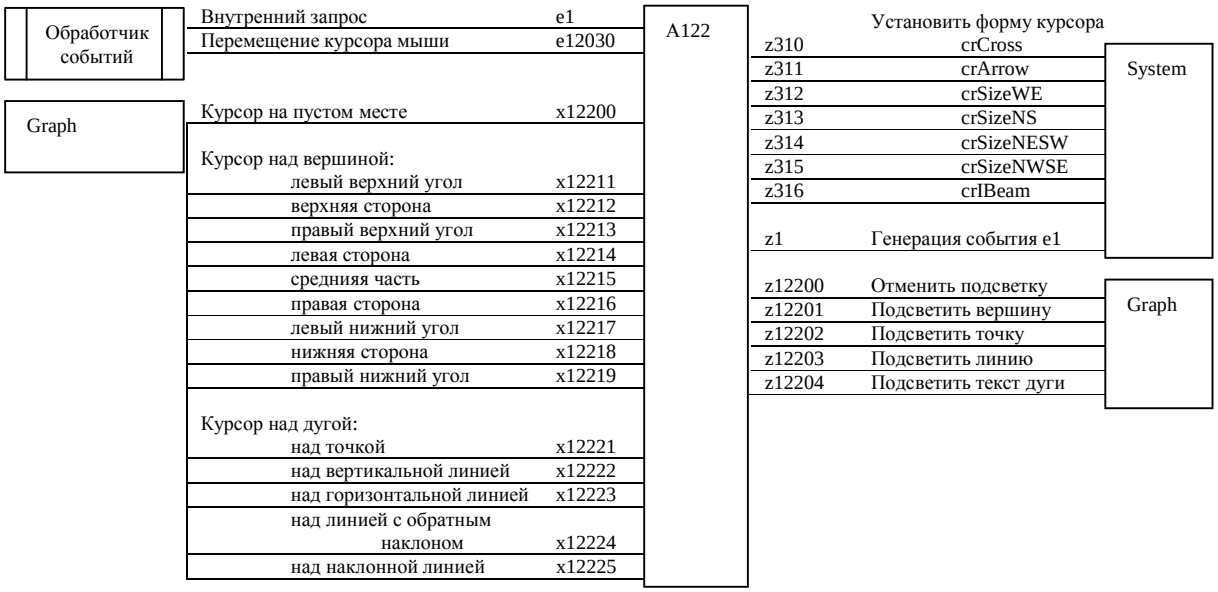

Рис.8. Схема связей автомата подсветки элементов графа

7.3.3. Граф переходов

Рис.9. Граф переходов автомата подсветки элементов графа

7.4. Автомат A123 (GraVEdit) - автомат редактирования графа

7.4.1. Словесное описание

Автомат осуществляет действия по редактированию элементов графа, их блоковому выделению, а также по добавлению новых элементов. Автомат «умеет» перемещать выделенные объекты, изменять размер вершин (состояний и групп), создавать новые дуги, точки и состояния, а также осуществлять блоковое выделение объектов.

7. 4. 2. Схема связей

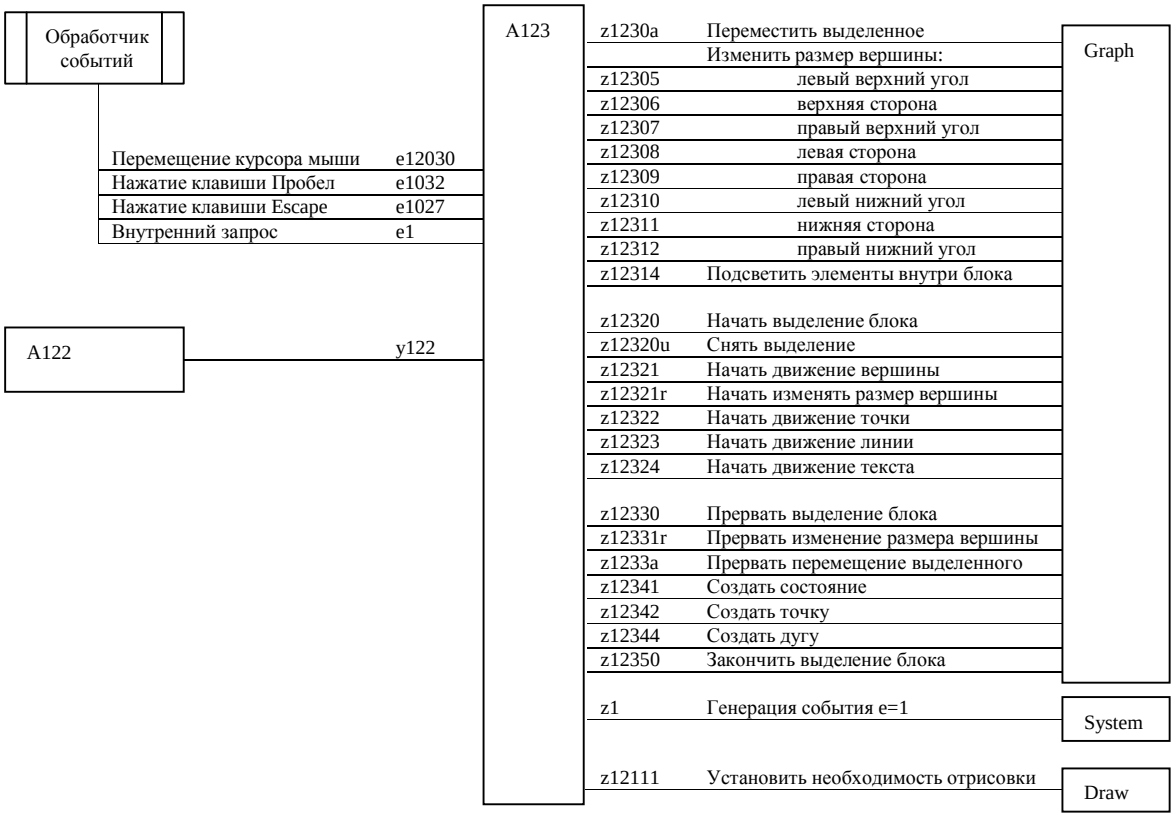

Рис.10. Схема связей автомата редактирования графа

7.4.3. Граф переходов

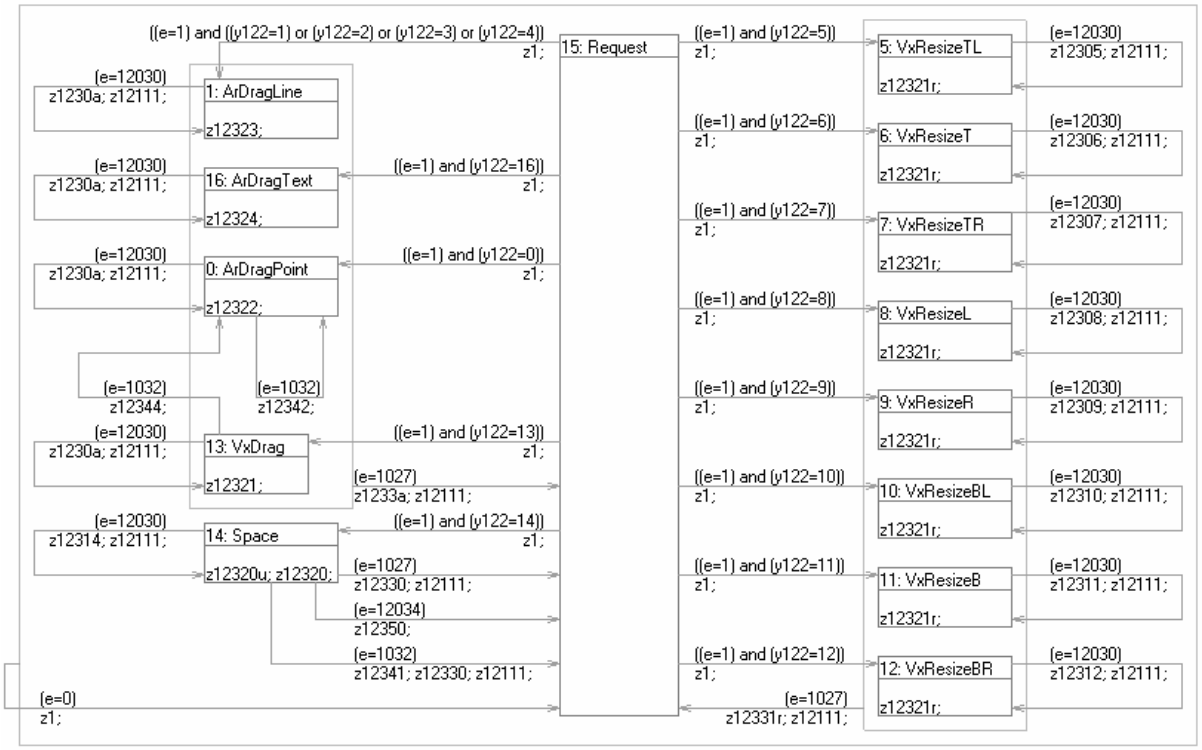

Рис.11. Граф переходов автомата редактирования графа

8. Особенности приложения

8.1. Особенности работы системы автоматов

В автоматах *A122* и *A123* используется специальное событие *e1*, генерируемое только при вызове функции *z1*. В отличие от других событий, используемых в системе автоматов редактора, событие *e1* является внутренним, и никакие внешние причины не могут явиться непосредственной причиной его возникновения. Это событие используется следующим образом.

Автомат *A122* находится в одном из своих стабильных состояний, отличных от состояния *15 (Request)*. При перемещении мыши генерируется событие *e12030*, которое переводит автомат *A122* в состояние *Request* с одновременным вызовом функции *z1*. При следующем запуске автомата *A122* с событием *e1*, сгенерированным при вызове этой функции, произойдет переход автомата в одно из стабильных состояний в зависимости от положения курсора мыши над окном редактора графов.

Автомат *A123* работает аналогичным образом. Стабильное состояние автомата *A123*, в которое осуществляется переход из состояния *Request*, выбирается в зависимости от состояния автомата *A122*.

8.2. Протоколирование

Приложение 1 содержит фрагмент протокола работы редактора. Этот протокол записан при тестовом запуске программы, в ходе которого были произведены несколько элементарных действий по редактированию графа переходов, например выделение объекта, перемещение его и снятие выделения.

8.3. Генерация кода

Как отмечалось выше, редактор графа построен с использованием той же технологии, по которой он сам позволяет создавать и редактировать графы переходов – на базе автоматного подхода. Помимо собственно редактирования графа, приложение позволяет также его «компилировать» – преобразовывать в текст программы на языке *Object Pascal* (*Delphi*).

Исходный текст программы редактора размещен на сайте http://is.ifmo.ru в разделе «Проекты». Логика работы редактора описывается системой графов переходов, созданных с помощью этого же редактора. В *Приложении 2* приведен код, сгенерированный встроенным в программу компилятором по указанной системе графов переходов.

Функция, реализующая встроенный компилятор, приведена в *Приложении 3.*

Заключение

Особенность настоящей работы состоит в том, что редактор графов переходов построен по той же технологии, по которой с его помощью можно строить программы – на базе системы взаимосвязанных автоматов, изображаемых в виде графов переходов. Данная технология позволяет создавать приложения, логика работы которых, по сравнению с приложениями, построенными по другим технологиям, вполне понятна. Это, в свою очередь, позволяет легко отлаживать программы, давая возможность получать правильно работающие программные продукты, удовлетворяющие предъявляемым к ним требованиям.

Рассмотренный пример не претендует на роль законченного приложения, готового к широкому использованию, а представляет собой работающий экспериментальный образец. Работа над совершенствованием этого приложения продолжается.

Подробнее о SWITCH-технологии и примерах ее использования можно прочитать на сайте http://is.ifmo.ru.

Литература

1. Шалыто А. А. Алгоритмизация и программирование для систем логического управления и "реактивных" систем //Автоматика и телемеханика. 2001. № 1. http://is.ifmo.ru, раздел "Статьи".

2. Шалыто А.А., Туккель Н.И. SWITCH-технология: автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5. http://is.ifmo.ru, раздел "Статьи".

Приложение 1. Пример протокола

```
-----=====[     1 ]=====-----
{ Automate _A12: Executed in state # 0 with event # 0
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 0 in state # 0
} Automate _A12: Finished execution of event # 0 in state # 0

  -----=====[     2 ]=====-----
{ Automate _A12: Executed in state # 0 with event # 300
{ Automate _A121: Executed in state # 0 with event # 300
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 300 in state # 0
} Automate _A12: Finished execution of event # 300 in state # 0

  -----=====[     3 ]=====-----
{ Automate _A12: Executed in state # 0 with event # 300
{ Automate _A121: Executed in state # 0 with event # 300
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 300 in state # 0
} Automate _A12: Finished execution of event # 300 in state # 0

  -----=====[     4 ]=====-----
{ Automate _A12: Executed in state # 0 with event # 12030
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 12030 in state # 0
T Automate _A12: Proceeded from state # 0 to state # 2
{ Automate _A122: Executed in state # 0 with event # 0
} Automate _A122: Finished execution of event # 0 in state # 0
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 0 in state # 0
* z12111 - set redraw needed
} Automate _A12: Finished execution of event # 12030 in state # 2

  -----=====[     5 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 123456
{ Automate _A122: Executed in state # 0 with event # 123456
} Automate _A122: Finished execution of event # 123456 in state # 0
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 123456 in state # 2

  -----=====[     6 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12030
{ Automate _A122: Executed in state # 0 with event # 12030
* z1 - generate e=1
T Automate _A122: Proceeded from state # 0 to state # 15
* z12111 - set redraw needed
} Automate _A122: Finished execution of event # 12030 in state # 15
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
} Automate _A12: Finished execution of event # 12030 in state # 2

  -----=====[     7 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 1
{ Automate _A122: Executed in state # 15 with event # 1
> x12221 - is cursor on arch Point - returned TRUE
T Automate _A122: Proceeded from state # 15 to state # 0
* z12202 - hilight nearest point
} Automate _A122: Finished execution of event # 1 in state # 0
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 1 in state # 2
```

```
...
  ----=====[    22 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12030
{ Automate _A122: Executed in state # 13 with event # 12030
* z1 - generate e=1
T Automate _A122: Proceeded from state # 13 to state # 15
* z12111 - set redraw needed
} Automate _A122: Finished execution of event # 12030 in state # 15
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
} Automate _A12: Finished execution of event # 12030 in state # 2

  ----=====[    23 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 1
{ Automate _A122: Executed in state # 15 with event # 1
> x12221 - is cursor on arch Point - returned FALSE
> x12222 - is cursor on Arch LineV - returned FALSE
> x12223 - is cursor on Arch LineH - returned FALSE
> x12224 - is cursor on Arch LineB - returned FALSE
> x12225 - is cursor on Arch LineS - returned FALSE
> x12226 - is cursor on Text - returned FALSE
> x12212 - is cursor on vertex Top - returned FALSE
> x12213 - is cursor on vertex TopRight - returned FALSE
> x12214 - is cursor on vertex Left - returned FALSE
> x12216 - is cursor on vertex Right - returned FALSE
> x12217 - is cursor on vertex BottomLeft - returned FALSE
> x12218 - is cursor on vertex Bottom - returned FALSE
> x12219 - is cursor on vertex BottomRight - returned FALSE
> x12215 - is cursor on vertex Center - returned TRUE
T Automate _A122: Proceeded from state # 15 to state # 13
* z12201 - hilight nearest vertex
} Automate _A122: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 1 in state # 2

  ----=====[    24 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12031
{ Automate _A122: Executed in state # 13 with event # 12031
} Automate _A122: Finished execution of event # 12031 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12031
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12031 in state # 0
> x402 - is Ctrl pressed - returned FALSE
* z1220 - perform snap to grid
T Automate _A12: Proceeded from state # 2 to state # 3
{ Automate _A123: Executed in state # 0 with event # 0
* z1 - generate e=1
T Automate _A123: Proceeded from state # 0 to state # 15
} Automate _A123: Finished execution of event # 0 in state # 15
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 0 in state # 0
* z1211 - set graph is modified
* z12111 - set redraw needed
} Automate _A12: Finished execution of event # 12031 in state # 3

  ----=====[    25 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 0
{ Automate _A123: Executed in state # 15 with event # 0
* z1 - generate e=1
} Automate _A123: Finished execution of event # 0 in state # 15
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 0 in state # 0
} Automate _A12: Finished execution of event # 0 in state # 3

  ----=====[    26 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1
{ Automate _A123: Executed in state # 15 with event # 1
* z1 - generate e=1
T Automate _A123: Proceeded from state # 15 to state # 13
* z12321 - start vertex drag
```

```
} Automate _A123: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
} Automate _A12: Finished execution of event # 1 in state # 3

  ----=====[    27 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1
{ Automate _A123: Executed in state # 13 with event # 1
} Automate _A123: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
} Automate _A12: Finished execution of event # 1 in state # 3

  ----=====[    28 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1
{ Automate _A123: Executed in state # 13 with event # 1
} Automate _A123: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
} Automate _A12: Finished execution of event # 1 in state # 3

  ----=====[    29 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 12030
{ Automate _A123: Executed in state # 13 with event # 12030
* z1230a - drag selected
* z12111 - set redraw needed
} Automate _A123: Finished execution of event # 12030 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
* z12130 - recalculate confine rect
} Automate _A12: Finished execution of event # 12030 in state # 3

...

  ----=====[    32 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 123456
{ Automate _A123: Executed in state # 13 with event # 123456
} Automate _A123: Finished execution of event # 123456 in state # 13
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
} Automate _A12: Finished execution of event # 123456 in state # 3

  ----=====[    33 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 12030
{ Automate _A123: Executed in state # 13 with event # 12030
* z1230a - drag selected
* z12111 - set redraw needed
} Automate _A123: Finished execution of event # 12030 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
* z12130 - recalculate confine rect
} Automate _A12: Finished execution of event # 12030 in state # 3

...

  ----=====[    38 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 12034
{ Automate _A123: Executed in state # 13 with event # 12034
} Automate _A123: Finished execution of event # 12034 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12034
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12034 in state # 0
* z1220 - perform snap to grid
* z1220 - stick archs and states
T Automate _A12: Proceeded from state # 3 to state # 2
{ Automate _A122: Executed in state # 13 with event # 0
} Automate _A122: Finished execution of event # 0 in state # 13
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned TRUE
```

```
} Automate _A121: Finished execution of event # 0 in state # 0
* z12111 - set redraw needed
} Automate _A12: Finished execution of event # 12034 in state # 2

  ----=====[    39 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12030
{ Automate _A122: Executed in state # 13 with event # 12030
* z1 - generate e=1
T Automate _A122: Proceeded from state # 13 to state # 15
* z12111 - set redraw needed
} Automate _A122: Finished execution of event # 12030 in state # 15
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
} Automate _A12: Finished execution of event # 12030 in state # 2

  ----=====[    40 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 1
{ Automate _A122: Executed in state # 15 with event # 1
> x12221 - is cursor on arch Point - returned FALSE
> x12222 - is cursor on Arch LineV - returned FALSE
> x12223 - is cursor on Arch LineH - returned FALSE
> x12224 - is cursor on Arch LineB - returned FALSE
> x12225 - is cursor on Arch LineS - returned FALSE
> x12226 - is cursor on Text - returned FALSE
> x12212 - is cursor on vertex Top - returned FALSE
> x12213 - is cursor on vertex TopRight - returned FALSE
> x12214 - is cursor on vertex Left - returned FALSE
> x12216 - is cursor on vertex Right - returned FALSE
> x12217 - is cursor on vertex BottomLeft - returned FALSE
> x12218 - is cursor on vertex Bottom - returned FALSE
> x12219 - is cursor on vertex BottomRight - returned FALSE
> x12215 - is cursor on vertex Center - returned TRUE
T Automate _A122: Proceeded from state # 15 to state # 13
* z12201 - hilight nearest vertex
} Automate _A122: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 1 in state # 2

  ----=====[    41 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 123456
{ Automate _A122: Executed in state # 13 with event # 123456
} Automate _A122: Finished execution of event # 123456 in state # 13
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 123456 in state # 2

  ----=====[    42 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 123456
{ Automate _A122: Executed in state # 13 with event # 123456
} Automate _A122: Finished execution of event # 123456 in state # 13
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 123456 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 123456 in state # 2

  ----=====[    43 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12031
{ Automate _A122: Executed in state # 13 with event # 12031
} Automate _A122: Finished execution of event # 12031 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12031
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 12031 in state # 0
> x402 - is Ctrl pressed - returned FALSE
* z1220 - perform snap to grid
T Automate _A12: Proceeded from state # 2 to state # 3
{ Automate _A123: Executed in state # 13 with event # 0
* z1 - generate e=1
T Automate _A123: Proceeded from state # 13 to state # 15
```

```
} Automate _A123: Finished execution of event # 0 in state # 15
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 0 in state # 0
* z1211 - set graph is modified
* z12111 - set redraw needed
} Automate _A12: Finished execution of event # 12031 in state # 3

  -----=====[    44 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1
{ Automate _A123: Executed in state # 15 with event # 1
* z1 - generate e=1
T Automate _A123: Proceeded from state # 15 to state # 13
* z12321 - start vertex drag
} Automate _A123: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
} Automate _A12: Finished execution of event # 1 in state # 3

  -----=====[    45 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1
{ Automate _A123: Executed in state # 13 with event # 1
} Automate _A123: Finished execution of event # 1 in state # 13
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
} Automate _A12: Finished execution of event # 1 in state # 3

  -----=====[    46 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 123456
{ Automate _A123: Executed in state # 13 with event # 123456
} Automate _A123: Finished execution of event # 123456 in state # 13
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
} Automate _A12: Finished execution of event # 123456 in state # 3

  -----=====[    47 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 12030
{ Automate _A123: Executed in state # 13 with event # 12030
* z1230a - drag selected
* z12111 - set redraw needed
} Automate _A123: Finished execution of event # 12030 in state # 13
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
* z12130 - recalculate confine rect
} Automate _A12: Finished execution of event # 12030 in state # 3

...

  ----------[    55 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 123456
{ Automate _A123: Executed in state # 13 with event # 123456
} Automate _A123: Finished execution of event # 123456 in state # 13
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
} Automate _A12: Finished execution of event # 123456 in state # 3

  ----------[    56 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 1027
{ Automate _A123: Executed in state # 13 with event # 1027
* z1233a - abort selection drag
* z1230a - drag selected
* z1220 - perform snap to grid
* z12111 - set redraw needed
T Automate _A123: Proceeded from state # 13 to state # 15
} Automate _A123: Finished execution of event # 1027 in state # 15
{ Automate _A121: Executed in state # 0 with event # 1027
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1027 in state # 0
```

```
} Automate _A12: Finished execution of event # 1027 in state # 3

  ----=====[    57 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 123456
{ Automate _A123: Executed in state # 15 with event # 123456
} Automate _A123: Finished execution of event # 123456 in state # 15
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
} Automate _A12: Finished execution of event # 123456 in state # 3

  ----=====[    58 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 123456
{ Automate _A123: Executed in state # 15 with event # 123456
} Automate _A123: Finished execution of event # 123456 in state # 15
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 123456 in state # 0
} Automate _A12: Finished execution of event # 123456 in state # 3

  ----=====[    59 ]=====-----
{ Automate _A12: Executed in state # 3 with event # 12034
{ Automate _A123: Executed in state # 15 with event # 12034
} Automate _A123: Finished execution of event # 12034 in state # 15
{ Automate _A121: Executed in state # 0 with event # 12034
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 12034 in state # 0
* z1220 - perform snap to grid
* z1220 - stick archs and states
T Automate _A12: Proceeded from state # 3 to state # 2
{ Automate _A122: Executed in state # 13 with event # 0
} Automate _A122: Finished execution of event # 0 in state # 13
{ Automate _A121: Executed in state # 0 with event # 0
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 0 in state # 0
* z12111 - set redraw needed
} Automate _A12: Finished execution of event # 12034 in state # 2

  ----=====[    60 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 12030
{ Automate _A122: Executed in state # 13 with event # 12030
* z1 - generate e=1
T Automate _A122: Proceeded from state # 13 to state # 15
* z12111 - set redraw needed
} Automate _A122: Finished execution of event # 12030 in state # 15
{ Automate _A121: Executed in state # 0 with event # 12030
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 12030 in state # 0
} Automate _A12: Finished execution of event # 12030 in state # 2

  ----=====[    61 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 1
{ Automate _A122: Executed in state # 15 with event # 1
> x12221 - is cursor on arch Point - returned FALSE
> x12222 - is cursor on Arch LineV - returned FALSE
> x12223 - is cursor on Arch LineH - returned FALSE
> x12224 - is cursor on Arch LineB - returned FALSE
> x12225 - is cursor on Arch LineS - returned FALSE
> x12226 - is cursor on Text - returned FALSE
> x12212 - is cursor on vertex Top - returned FALSE
> x12213 - is cursor on vertex TopRight - returned FALSE
> x12214 - is cursor on vertex Left - returned FALSE
> x12216 - is cursor on vertex Right - returned FALSE
> x12217 - is cursor on vertex BottomLeft - returned FALSE
> x12218 - is cursor on vertex Bottom - returned FALSE
> x12219 - is cursor on vertex BottomRight - returned FALSE
> x12215 - is cursor on vertex Center - returned FALSE
> x12211 - is cursor on vertex TopLet - returned FALSE
> x12200 - is cursor on Empty Space - returned TRUE
T Automate _A122: Proceeded from state # 15 to state # 14
* z12200 - hilight nothing
} Automate _A122: Finished execution of event # 1 in state # 14
{ Automate _A121: Executed in state # 0 with event # 1
> x12111 - is redraw needed - returned TRUE
} Automate _A121: Finished execution of event # 1 in state # 0
```

```
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 1 in state # 2
  ----=====[    62 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 123456
{ Automate _A122: Executed in state # 14 with event # 123456
} Automate _A122: Finished execution of event # 123456 in state # 14
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned TRUE
* z12100 - redraw
* z12110 - clear redraw needed
} Automate _A121: Finished execution of event # 123456 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 123456 in state # 2

  ----=====[    63 ]=====-----
{ Automate _A12: Executed in state # 2 with event # 123456
{ Automate _A122: Executed in state # 14 with event # 123456
} Automate _A122: Finished execution of event # 123456 in state # 14
{ Automate _A121: Executed in state # 0 with event # 123456
> x12111 - is redraw needed - returned FALSE
} Automate _A121: Finished execution of event # 123456 in state # 0
> x402 - is Ctrl pressed - returned FALSE
} Automate _A12: Finished execution of event # 123456 in state # 2
```

Приложение 2. Фрагмент программы (автоматы), сгенерированный редактором

```
unit aA12;

// ================================================ //
// ==                INTERFACE                   == //
// ================================================ //
interface

{$include defines.inc}

uses
    aAuto;

type

// ================ TAuto ======================== //
    TA12 = class (TAuto)
    protected
        procedure P_Run (e: integer); override;
    end;

var
    A12: TA12;

// ================================================ //
// ==                IMPLEMENTATION              == //
// ================================================ //
implementation

uses
    X, Zed, Logger,
    aA121, aA122, aA123;

// ================ TAuto ======================== //

    // run
    procedure TA12.P_Run (e: integer);
    label
        l_End;
    var
        oldState: integer;
        {$ifdef GRAPH_LOOP_LOGGING} someArch: integer; {$endif}
    begin

        oldState := State;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}

        {$ifdef GRAPH_EVENTS_LOGGING}
            log_exec ('_A12', oldState, e);
        {$endif}

        case (State) of
            0:  begin
                A121.Run(e);
                if (e=12011) then begin
                    z1250;
                    State := 0;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}
                end else
                if (e=12012) then begin
                    z1250;
                    State := 0;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 1; {$endif}
                end else
                if (e=12021) then begin
                    z1241;
                    State := 1;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 2; {$endif}
                end else
                if (e=12030) then begin
```

```
            State := 2;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 5; {$endif}
        end else
        if (e=12027) then begin
            z1247; z12100; z1211;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 10; {$endif}
        end else
        if (e=12028) then begin
            z1248; z12100; z1211;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 11; {$endif}
        end else
        if (e=12001) then begin
            z12100;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 14; {$endif}
        end else
        if (e=12010) then begin
            z12100; z1211;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 15; {$endif}
        end else;
    end;
    1:  begin
        A121.Run(e);
        if (e=12024) then begin
            z1244;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 3; {$endif}
        end else
        if (e=12020) then begin
            z1240; z12111;
            State := 1;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 4; {$endif}
        end else
        if (e=1027) then begin
            z1249; z12100;
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 12; {$endif}
        end else;
    end;
    2:  begin
        A122.Run(e); A121.Run(e);
        if (e=300) then begin
            State := 0;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 6; {$endif}
        end else
        if (e=12030) then begin
            {z12111};
            State := 2;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 7; {$endif}
        end else
        if (x402 and (e=12031)) then begin
            State := 4;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 8; {$endif}
        end else
        if ((e=12031) and (A122.state=16) and x1204) then begin
            State := 6;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 21; {$endif}
        end else
        if (e=12031) then begin
            z1220;
            State := 3;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 22; {$endif}
        end else
        if (e=12032) then begin
            z1260;
            State := 2;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 30; {$endif}
        end else;
    end;
    3:  begin
        A123.Run(e); A121.Run(e);
        if (e=12034) then begin
            z1220; z1230;
            State := 2;
```

```
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 9; {$endif}
    end else
    if (e=12030) then begin
        z12130;
        State := 3;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 13; {$endif}
    end else;
end;
4:  begin
    if (e=12030) then begin
        z12314; z12111;
        State := 5;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 16; {$endif}
    end else
    if (e=12034) then begin
        z12350_1;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 18; {$endif}
    end else;
end;
5:  begin
    A121.Run(e);
    if (e=12030) then begin
        z12314; z12111;
        State := 5;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 17; {$endif}
    end else
    if (e=12034) then begin
        z12350;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 19; {$endif}
    end else
    if (e=1027) then begin
        z12330; z12111;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 20; {$endif}
    end else;
end;
6:  begin
    A123.Run(e); A121.Run(e);
    if (e=12034) then begin
        z12354; z12111;
        State := 7;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 23; {$endif}
    end else
    if (e=12030) then begin
        z1220; z12130;
        State := 3;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 24; {$endif}
    end else;
end;
7:  begin
    A122.Run(e); A121.Run(e);
    if ((e=12031) and not x1204) then begin
        z12355; z12356;
        State := 3;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 25; {$endif}
    end else
    if (e=12030) then begin
        State := 7;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 26; {$endif}
    end else
    if (e=1027) then begin
        z12356;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 27; {$endif}
    end else
    if (e=1013) then begin
        z12355; z12356;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 28; {$endif}
    end else
    if ((e=12031) and x1204) then begin
        z12355; z12356;
        State := 6;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 29; {$endif}
    end else;
```

```
            end;
            else begin
                {$ifdef GRAPH_ERRORS_LOGGING}
                    log_write (LOG_GRAPH_ERROR, 'ERROR IN _A12: unknown state
                                                                    number!', 0);
                {$endif}
            end;
        end;

        if (oldState = State) then begin
            {$ifdef GRAPH_LOOP_LOGGING}
                if (someArch <> 0) then log_loop ('_A12', oldState, someArch);
            {$endif}
            goto l_End;
        end;

        begin
            {$ifdef GRAPH_TRANS_LOGGING}
                log_trans ('_A12', oldState, State);
            {$endif}
        end;

        case (State) of
            0:  begin
                A121.Run(0);
            end;
            1:  begin
                A121.Run(0);
                z1211;
            end;
            2:  begin
                A122.Run(0); A121.Run(0);
                z12111;
            end;
            3:  begin
                A123.Run(0); A121.Run(0);
                z1211; z12111;
            end;
            4:  begin
            end;
            5:  begin
                A121.Run(0);
                z12320;
            end;
            6:  begin
                A123.Run(0); A121.Run(0);
                z1211; z12111;
            end;
            7:  begin
                A122.Run(0); A121.Run(0);
            end;
        end;

        l_End:
        {$ifdef GRAPH_ENDS_LOGGING}
            log_end ('_A12', State, e);
        {$endif}

    end;
end.
```

```
unit aA121;

// ================================================ //
// ==                INTERFACE                   == //
// ================================================ //
interface

{$include defines.inc}

uses
    aAuto;

type

// ================ TAuto ======================== //
    TA121 = class (TAuto)
    protected
        procedure P_Run (e: integer); override;
    end;

var
    A121: TA121;

// ================================================ //
// ==                IMPLEMENTATION              == //
// ================================================ //
implementation

uses
    X, Zed, Logger
    ;

// ================ TAuto ======================== //

    // run
    procedure TA121.P_Run (e: integer);
    label
        l_End;
    var
        oldState: integer;
        {$ifdef GRAPH_LOOP_LOGGING} someArch: integer; {$endif}
    begin

        oldState := State;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}

        {$ifdef GRAPH_EVENTS_LOGGING}
            log_exec ('_A121', oldState, e);
        {$endif}

        case (State) of
            0:  begin
                if (x12111 and (e=123456)) then begin
                    z12100; z12110;
                    State := 0;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}
                end else;
            end;
            else begin
                {$ifdef GRAPH_ERRORS_LOGGING}
                    log_write (LOG_GRAPH_ERROR, 'ERROR IN _A121: unknown state
                                                                    number!', 0);
                {$endif}
            end;
        end;

        if (oldState = State) then begin
            {$ifdef GRAPH_LOOP_LOGGING}
                if (someArch <> 0) then log_loop ('_A121', oldState, someArch);
            {$endif}
            goto l_End;
        end;

        begin
            {$ifdef GRAPH_TRANS_LOGGING}
                log_trans ('_A121', oldState, State);
            {$endif}
```

```
        end;

        case (State) of
            0:  begin
            end;
        end;

        l_End:
        {$ifdef GRAPH_ENDS_LOGGING}
            log_end ('_A121', State, e);
        {$endif}

    end;

end.




unit aA122;

// ============================================== //
// ==                INTERFACE                 == //
// ============================================== //
interface

{$include defines.inc}

uses
    aAuto;

type

// ================ TAuto ======================== //
    TA122 = class (TAuto)
    protected
        procedure P_Run (e: integer); override;
    end;

var
    A122: TA122;

// ============================================== //
// ==                IMPLEMENTATION            == //
// ============================================== //
implementation

uses
    X, Zed, Logger
    ;

// ================ TAuto ======================== //

    // run
    procedure TA122.P_Run (e: integer);
    label
        l_End;
    var
        oldState: integer;
        {$ifdef GRAPH_LOOP_LOGGING} someArch: integer; {$endif}
    begin

        oldState := State;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}

        {$ifdef GRAPH_EVENTS_LOGGING}
            log_exec ('_A122', oldState, e);
        {$endif}

        case (State) of
            0:  begin
                if (e=12030) then begin
                    z1;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 15; {$endif}
                end else;
```

```
end;
1:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 16; {$endif}
    end else;
end;
2:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 17; {$endif}
    end else;
end;
3:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 18; {$endif}
    end else;
end;
4:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 19; {$endif}
    end else;
end;
5:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 20; {$endif}
    end else;
end;
6:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 21; {$endif}
    end else;
end;
7:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 22; {$endif}
    end else;
end;
8:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 23; {$endif}
    end else;
end;
9:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 24; {$endif}
    end else;
end;
10:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 25; {$endif}
    end else;
end;
11:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 26; {$endif}
    end else;
```

```
end;
12:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 27; {$endif}
    end else;
end;
13:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 28; {$endif}
    end else;
end;
14:  begin
    if (e=12030) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 30; {$endif}
    end else;
end;
15:  begin
    if ((e=1) and x12221) then begin
        z311;
        State := 0;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}
    end else
    if ((e=1) and x12222) then begin
        z312;
        State := 1;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 1; {$endif}
    end else
    if ((e=1) and x12223) then begin
        z313;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 2; {$endif}
    end else
    if ((e=1) and x12224) then begin
        z314;
        State := 3;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 3; {$endif}
    end else
    if ((e=1) and x12225) then begin
        z315;
        State := 4;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 4; {$endif}
    end else
    if ((e=1) and x12226) then begin
        z316;
        State := 16;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 5; {$endif}
    end else
    if ((e=1) and x12212) then begin
        z313;
        State := 6;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 6; {$endif}
    end else
    if ((e=1) and x12213) then begin
        z314;
        State := 7;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 7; {$endif}
    end else
    if ((e=1) and x12214) then begin
        z312;
        State := 8;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 8; {$endif}
    end else
    if ((e=1) and x12216) then begin
        z312;
        State := 9;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 9; {$endif}
    end else
    if ((e=1) and x12217) then begin
        z314;
        State := 10;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 10; {$endif}
```

```
        end else
        if ((e=1) and x12218) then begin
            z313;
            State := 11;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 11; {$endif}
        end else
        if ((e=1) and x12219) then begin
            z315;
            State := 12;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 12; {$endif}
        end else
        if ((e=1) and x12215) then begin
            z311;
            State := 13;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 13; {$endif}
        end else
        if ((e=1) and x12211) then begin
            z315;
            State := 5;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 14; {$endif}
        end else
        if ((e=1) and x12200) then begin
            z310;
            State := 14;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 31; {$endif}
        end else
        if (e=12030) then begin
            z1;
            State := 15;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 32; {$endif}
        end else;
    end;
    16:  begin
        if (e=12030) then begin
            z1;
            State := 15;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 29; {$endif}
        end else;
    end;
    else begin
        {$ifdef GRAPH_ERRORS_LOGGING}
            log_write (LOG_GRAPH_ERROR, 'ERROR IN _A122: unknown state
                                                          number!', 0);
        {$endif}
    end;
end;

if (oldState = State) then begin
    {$ifdef GRAPH_LOOP_LOGGING}
        if (someArch <> 0) then log_loop ('_A122', oldState, someArch);
    {$endif}
    goto l_End;
end;

begin
    {$ifdef GRAPH_TRANS_LOGGING}
        log_trans ('_A122', oldState, State);
    {$endif}
end;

case (State) of
    0:  begin
        z12202;
    end;
    1:  begin
        z12203;
    end;
    2:  begin
        z12203;
    end;
    3:  begin
        z12203;
    end;
    4:  begin
        z12203;
    end;
    5:  begin
```

```
                z12201;
            end;
            6:  begin
                z12201;
            end;
            7:  begin
                z12201;
            end;
            8:  begin
                z12201;
            end;
            9:  begin
                z12201;
            end;
            10:  begin
                z12201;
            end;
            11:  begin
                z12201;
            end;
            12:  begin
                z12201;
            end;
            13:  begin
                z12201;
            end;
            14:  begin
                z12200;
            end;
            15:  begin
                z12111;
            end;
            16:  begin
                z12204;
            end;
        end;

        l_End:
        {$ifdef GRAPH_ENDS_LOGGING}
            log_end ('_A122', State, e);
        {$endif}

    end;

end.
```

```
unit aA123;

// ================================================ //
// ==                INTERFACE                   == //
// ================================================ //
interface

{$include defines.inc}

uses
    aAuto;

type

// ================ TAuto ======================== //
    TA123 = class (TAuto)
    protected
        procedure P_Run (e: integer); override;
    end;

var
    A123: TA123;
```

```
// ============================================== //
// ==                IMPLEMENTATION             == //
// ============================================== //
implementation

uses
    X, Zed, Logger,
    aA122;

// ================ TAuto ======================== //

    // run
    procedure TA123.P_Run (e: integer);
    label
        l_End;
    var
        oldState: integer;
        {$ifdef GRAPH_LOOP_LOGGING} someArch: integer; {$endif}
    begin

        oldState := State;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}

        {$ifdef GRAPH_EVENTS_LOGGING}
            log_exec ('_A123', oldState, e);
        {$endif}

        case (State) of
            0:  begin
                if (e=0) then begin
                    z1;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 15; {$endif}
                end else
                if (e=1027) then begin
                    z1233a; z12111;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 30; {$endif}
                end else
                if (e=12030) then begin
                    z1230a; z12111;
                    State := 0;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 45; {$endif}
                end else
                if (e=1032) then begin
                    z12342;
                    State := 0;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 62; {$endif}
                end else;
            end;
            1:  begin
                if (e=0) then begin
                    z1;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 16; {$endif}
                end else
                if (e=1027) then begin
                    z1233a; z12111;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 31; {$endif}
                end else
                if (e=12030) then begin
                    z1230a; z12111;
                    State := 1;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 46; {$endif}
                end else;
            end;
            2:  begin
                if (e=0) then begin
                    z1;
                    State := 15;
                    {$ifdef GRAPH_LOOP_LOGGING} someArch := 17; {$endif}
                end else
                if (e=1027) then begin
                    z1233a; z12111;
                    State := 15;
```

```
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 32; {$endif}
    end else
    if (e=12030) then begin
        z1230a; z12111;
        State := 2;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 47; {$endif}
    end else;
end;
3:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 18; {$endif}
    end else
    if (e=1027) then begin
        z1233a; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 33; {$endif}
    end else
    if (e=12030) then begin
        z1230a; z12111;
        State := 3;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 48; {$endif}
    end else;
end;
4:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 19; {$endif}
    end else
    if (e=1027) then begin
        z1233a; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 34; {$endif}
    end else
    if (e=12030) then begin
        z1230a; z12111;
        State := 4;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 49; {$endif}
    end else;
end;
5:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 20; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 35; {$endif}
    end else
    if (e=12030) then begin
        z12305; z12111;
        State := 5;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 50; {$endif}
    end else;
end;
6:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 21; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 36; {$endif}
    end else
    if (e=12030) then begin
        z12306; z12111;
        State := 6;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 51; {$endif}
    end else;
end;
7:  begin
```

```
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 22; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 37; {$endif}
    end else
    if (e=12030) then begin
        z12307; z12111;
        State := 7;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 52; {$endif}
    end else;
end;
8:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 23; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 38; {$endif}
    end else
    if (e=12030) then begin
        z12308; z12111;
        State := 8;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 53; {$endif}
    end else;
end;
9:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 24; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 39; {$endif}
    end else
    if (e=12030) then begin
        z12309; z12111;
        State := 9;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 54; {$endif}
    end else;
end;
10:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 25; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 40; {$endif}
    end else
    if (e=12030) then begin
        z12310; z12111;
        State := 10;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 55; {$endif}
    end else;
end;
11:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 26; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 41; {$endif}
```

```
    end else
    if (e=12030) then begin
        z12311; z12111;
        State := 11;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 56; {$endif}
    end else;
end;
12:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 27; {$endif}
    end else
    if (e=1027) then begin
        z12331r; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 42; {$endif}
    end else
    if (e=12030) then begin
        z12312; z12111;
        State := 12;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 57; {$endif}
    end else;
end;
13:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 28; {$endif}
    end else
    if (e=1027) then begin
        z1233a; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 43; {$endif}
    end else
    if (e=12030) then begin
        z1230a; z12111;
        State := 13;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 58; {$endif}
    end else
    if (e=1032) then begin
        z12344;
        State := 0;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 61; {$endif}
    end else;
end;
14:  begin
    if (e=0) then begin
        z1;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 29; {$endif}
    end else
    if (e=1027) then begin
        z12330; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 44; {$endif}
    end else
    if (e=12030) then begin
        z12314; z12111;
        State := 14;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 59; {$endif}
    end else
    if (e=12034) then begin
        z12350;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 63; {$endif}
    end else
    if (e=1032) then begin
        z12341; z12330; z12111;
        State := 15;
        {$ifdef GRAPH_LOOP_LOGGING} someArch := 64; {$endif}
    end else;
end;
15:  begin
    if ((e=1) and (A122.state=0)) then begin
        z1;
        State := 0;
```

```
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}
end else
if ((e=1) and (A122.state=1)) then begin
    z1;
    State := 1;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 1; {$endif}
end else
if ((e=1) and (A122.state=2)) then begin
    z1;
    State := 2;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 2; {$endif}
end else
if ((e=1) and (A122.state=3)) then begin
    z1;
    State := 3;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 3; {$endif}
end else
if ((e=1) and (A122.state=4)) then begin
    z1;
    State := 4;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 4; {$endif}
end else
if ((e=1) and (A122.state=5)) then begin
    z1;
    State := 5;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 5; {$endif}
end else
if ((e=1) and (A122.state=6)) then begin
    z1;
    State := 6;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 6; {$endif}
end else
if ((e=1) and (A122.state=7)) then begin
    z1;
    State := 7;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 7; {$endif}
end else
if ((e=1) and (A122.state=8)) then begin
    z1;
    State := 8;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 8; {$endif}
end else
if ((e=1) and (A122.state=9)) then begin
    z1;
    State := 9;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 9; {$endif}
end else
if ((e=1) and (A122.state=10)) then begin
    z1;
    State := 10;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 10; {$endif}
end else
if ((e=1) and (A122.state=11)) then begin
    z1;
    State := 11;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 11; {$endif}
end else
if ((e=1) and (A122.state=12)) then begin
    z1;
    State := 12;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 12; {$endif}
end else
if ((e=1) and (A122.state=13)) then begin
    z1;
    State := 13;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 13; {$endif}
end else
if ((e=1) and (A122.state=14)) then begin
    z1;
    State := 14;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 14; {$endif}
end else
if (e=0) then begin
    z1;
    State := 15;
    {$ifdef GRAPH_LOOP_LOGGING} someArch := 60; {$endif}
end else
```

```
        if ((e=1) and (A122.state=16)) then begin
            z1;
            State := 16;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 65; {$endif}
        end else;
    end;
    16:  begin
        if (e=0) then begin
            z1;
            State := 15;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 66; {$endif}
        end else
        if (e=12030) then begin
            z1230a; z12111;
            State := 16;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 67; {$endif}
        end else
        if (e=1027) then begin
            z1233a; z12111;
            State := 15;
            {$ifdef GRAPH_LOOP_LOGGING} someArch := 68; {$endif}
        end else;
    end;
    else begin
        {$ifdef GRAPH_ERRORS_LOGGING}
            log_write (LOG_GRAPH_ERROR, 'ERROR IN _A123: unknown state
                                                              number!', 0);
        {$endif}
    end;
end;

if (oldState = State) then begin
    {$ifdef GRAPH_LOOP_LOGGING}
        if (someArch <> 0) then log_loop ('_A123', oldState, someArch);
    {$endif}
    goto l_End;
end;

begin
    {$ifdef GRAPH_TRANS_LOGGING}
        log_trans ('_A123', oldState, State);
    {$endif}
end;

case (State) of
    0:  begin
        z12322;
    end;
    1:  begin
        z12323;
    end;
    2:  begin
        z12323;
    end;
    3:  begin
        z12323;
    end;
    4:  begin
        z12323;
    end;
    5:  begin
        z12321r;
    end;
    6:  begin
        z12321r;
    end;
    7:  begin
        z12321r;
    end;
    8:  begin
        z12321r;
    end;
    9:  begin
        z12321r;
    end;
    10:  begin
        z12321r;
```

```
            end;
            11:  begin
                z12321r;
            end;
            12:  begin
                z12321r;
            end;
            13:  begin
                z12321;
            end;
            14:  begin
                z12320u; z12320;
            end;
            15:  begin
            end;
            16:  begin
                z12324;
            end;
        end;

        l_End:
        {$ifdef GRAPH_ENDS_LOGGING}
            log_end ('_A123', State, e);
        {$endif}

    end;

end.
```

Приложение 3. Текст программы компилятора

```
// COMPILE TO STREAM
function TGraph.CompileToStream (Stream: TStream): boolean;
    // SR__y_N__A_N_dot_state : 'yN=i' -> 'AN.state=i'
    function SR__y_N__A_N_dot_state (const S: String): String;
    var
        i, v, code: integer;
        c: char;
        St: String;
    begin
        Result := S;
        i := 1;
        while (TRUE) do begin
            if (i > length (Result)) then
                break;
            c := Result [i];
            if (c = 'y') then begin
                St := Copy (Result, i+1, length (Result)-i);
                Val (St, v, code);
                if ((code <> 0) and (code < length (St)) and (St [code] = '='))
                then begin
                    Delete (Result, i, code);
                    Insert (Format ('A%d.state', [v]), Result, i);
                end;
            end;
            inc (i);
        end;
    end;


    // SR__e_N__e_equal_N : 'ei' -> 'e=i'
    function SR__e_N__e_equal_N (const S: String): String;
    var
        i, v, code: integer;
        c: char;
        St: String;
    begin
        Result := S;
        i := 1;
        while (TRUE) do begin
            if (i > length (Result)) then
                break;
            c := Result [i];
            if (c = 'e') then begin
                St := Copy (Result, i+1, length (Result)-i);
```

```
            Val (St, v, code);
            if ((code = 0) or (code > 1)) then begin
                if (code = 0) then
                    code := length (St)+1;
                Delete (Result, i, code);
                Insert (Format ('e=%d', [v]), Result, i);
            end;
        end;
        inc (i);
    end;
end;


// SR__A_N_separator__A_N_dot_Run_semicolon
function SR__A_N_separator__A_N_dot_Run_semicolon (const S: String): String;
var
    i, v, code: integer;
    c: char;
    St: String;
begin
    Result := S;
    i := 1;
    while (TRUE) do begin
        if (i > length (Result)) then
            break;
        c := Result [i];
        if (c = 'A') then begin
            St := Copy (Result, i+1, length (Result)-i);
            Val (St, v, code);
            if ((code <> 0) and (
                ((code < length (St)+1) and (
                    (St [code] = ',') or
                    (St [code] = ';') or
                    (St [code] = ' ')
                )) or
                (code = length (St)+1)
            )) then begin
                Delete (Result, i, code+1);
                if ((i < length (Result)+1) and (Result [i] <> ' ')) then
                    Insert (' ', Result, i);
                Insert (Format ('A%d.Run;', [v]), Result, i);
            end;
        end;
        inc (i);
    end;
end;


// OptimalCondition
function OptimalCondition (S: String): String;
var
    SS: TStringList;
    i: integer;
begin
    SS := TStringList .Create ();

    SS .Text := S;

    if (SS .Count = 0) then Result := '' else
    if (SS .Count = 1) then Result := SS .Strings [0] else begin
        Result := '';
        for i := 0 to SS .Count-1 do begin
            Result := Result + #13#10'                    ' +
            SS .Strings [i];
        end;
        Result := Result + #13#10'                ';
    end;

    SS .Free ();

    Result := SR__e_N__e_equal_N (Result);
    Result := SR__y_N__A_N_dot_state (Result);
    //
    Result := StringReplace (Result, ' && ', ' and ', [rfReplaceAll]);
    Result := StringReplace (Result, ' & ', ' and ', [rfReplaceAll]);

    Result := StringReplace (Result, '&& ', ' and ', [rfReplaceAll]);
```

```
    Result := StringReplace (Result, ' &&', ' and ', [rfReplaceAll]);
    Result := StringReplace (Result, '&&', ' and ', [rfReplaceAll]);

    Result := StringReplace (Result, '& ', ' and ', [rfReplaceAll]);
    Result := StringReplace (Result, ' &', ' and ', [rfReplaceAll]);
    Result := StringReplace (Result, '&', ' and ', [rfReplaceAll]);
    //
    Result := StringReplace (Result, ' || ', ' or ', [rfReplaceAll]);
    Result := StringReplace (Result, ' |', ' or ', [rfReplaceAll]);

    Result := StringReplace (Result, '|| ', ' or ', [rfReplaceAll]);
    Result := StringReplace (Result, ' ||', ' or ', [rfReplaceAll]);
    Result := StringReplace (Result, '||', ' or ', [rfReplaceAll]);

    Result := StringReplace (Result, '| ', ' or ', [rfReplaceAll]);
    Result := StringReplace (Result, ' |', ' or ', [rfReplaceAll]);
    Result := StringReplace (Result, '|', ' or ', [rfReplaceAll]);
    //
    Result := StringReplace (Result, '! ', 'not ', [rfReplaceAll]);
    Result := StringReplace (Result, '!', 'not ', [rfReplaceAll]);
end;


// OptimalAction20__case1
function OptimalAction20__case1 (S: String): String;
var
    SS: TStringList;
    i: integer;
begin
    SS := TStringList .Create ();

    SS .Text := S;

    if (SS .Count = 0) then Result := '' else begin
        Result := '';
        for i := 0 to SS .Count-1 do begin
            Result := Result + #13#10'                  ' +
            SS .Strings [i];
        end;
    end;

    SS .Free ();
end;


// OptimalAction16_case2
function OptimalAction16_case2 (S: String): String;
var
    SS: TStringList;
    i: integer;
begin
    SS := TStringList .Create ();

    SS .Text := S;

    if (SS .Count = 0) then Result := '' else begin
        Result := '';
        for i := 0 to SS .Count-1 do begin
            Result := Result + #13#10'               ' + SS .Strings [i];
        end;
    end;

    SS .Free ();
end;


// OptimalActionAuto16_e
function OptimalActionAuto16_e (S: String): String;
begin
    Result := OptimalAction16_case2 (S);
    Result := SR__A_N_separator__A_N_dot_Run_semicolon (Result);
    Result := StringReplace (Result, 'Run', 'Run(e)', [rfReplaceAll]);
end;


// OptimalActionAuto16_0
function OptimalActionAuto16_0 (S: String): String;
```

```
    begin
        Result := OptimalAction16_case2 (S);
        Result := SR__A_N_separator__A_N_dot_Run_semicolon (Result);
        Result := StringReplace (Result, 'Run', 'Run(0)', [rfReplaceAll]);
    end;




var
    i, j, k: integer;
    ic, jc, kc: integer;
    St: TState;
    Gr: TGroup;
    Ar: TArch;
    S, _ANAME: String;
    SStream: TStringStream;


begin
    // header
    //_ANAME := '_ANAME';
    _ANAME := ExtractFilename (Self .FileName);
    i := Pos ('_', _ANAME);
    SetLength (_ANAME, i-1);
    _ANAME := '_' + _ANAME;
    SStream := TStringStream (Stream);
    S := ''
    {+#13#10'    procedure Run (e: integer);'
    +#13#10}+'    label'
    +#13#10'        l_End;'
    +#13#10'    var'
    +#13#10'        oldState: integer;'
    +#13#10'        {$ifdef GRAPH_LOOP_LOGGING} someArch: integer; {$endif}'
    +#13#10'    begin'
    +#13#10''
    +#13#10'        oldState := State;'
    +#13#10'        {$ifdef GRAPH_LOOP_LOGGING} someArch := 0; {$endif}'
    +#13#10''
    +#13#10'        {$ifdef GRAPH_EVENTS_LOGGING}'
    +#13#10'            log_exec ('''+_ANAME+''', oldState, e);'
    +#13#10'        {$endif}'
    +#13#10''
    +#13#10'        case (State) of'
    ;
    // first case
    SStream .WriteString (S);
    ic := StateCount-1;
    for i := 0 to ic do
    begin
        St := State [i];
        S := Format (#13#10'            %d:  begin%s',
              [St .No, OptimalActionAuto16_e (St.Automates)]);
        SStream .WriteString (S);
        // proceed groups first
        kc := St .GroupCount-1;
        for k := 0 to kc do begin
            Gr := St .Group [k];
            jc := Gr .OutArchCount-1;
            for j := 0 to jc do begin
                Ar := Gr .OutArch [j];
                S := ''
                +Format (#13#10'                if (%s) then begin%s',
                   [OptimalCondition (Ar.Condition),
                    OptimalAction20__case1 (Ar .Action)])

                +Format (#13#10'                    State := %d;', [Ar .Ei])
                +Format (#13#10'                    {$ifdef GRAPH_LOOP_LOGGING}
                   someArch := %d; {$endif}', [Ar .No])
                +#13#10'                end else';
                SStream .WriteString (S);
            end;
        end;
        // than state inself
        jc := St .OutArchCount-1;
        for j := 0 to jc do begin
```

```
            Ar := St .OutArch [j];
            S := ''
            +Format (#13#10'                if (%s) then begin%s',
               [OptimalCondition (Ar.Condition),
                OptimalAction20__case1 (Ar .Action)])
            +Format (#13#10'                    State := %d;', [Ar .Ei])
            +Format (#13#10'                    {$ifdef GRAPH_LOOP_LOGGING}
               someArch := %d; {$endif}', [Ar .No])
            +#13#10'                end else';
            SStream .WriteString (S);
        end;
        S := ';'#13#10'            end;';
        SStream .WriteString (S);
    end;
    // trans
    S := ''
    +#13#10'            else begin'
    +#13#10'                {$ifdef GRAPH_ERRORS_LOGGING}'
    +#13#10'                    log_write (LOG_GRAPH_ERROR, ''ERROR IN '
                               +_ANAME+': unknown state number!'', 0);'
    +#13#10'                {$endif}'
    +#13#10'            end;'
    +#13#10'        end;'
    +#13#10''
    +#13#10'        if (oldState = State) then begin'
    +#13#10'            {$ifdef GRAPH_LOOP_LOGGING}'
    +#13#10'                if (someArch <> 0) then log_loop ('''+_ANAME+''',
                               oldState, someArch);'
    +#13#10'            {$endif}'
    +#13#10'            goto l_End;'
    +#13#10'        end;'
    +#13#10''
    +#13#10'        begin'
    +#13#10'            {$ifdef GRAPH_TRANS_LOGGING}'
    +#13#10'                log_trans ('''+_ANAME+''', oldState, State);'
    +#13#10'            {$endif}'
    +#13#10'            {{$ifdef DEBUG_FRAME}'
    +#13#10'            {    update_debug ();'
    +#13#10'            {{$endif}'
    +#13#10'        end;'
    +#13#10''
    +#13#10'        case (State) of'
    ;
    SStream .WriteString (S);
    // second case
    ic := StateCount-1;
    for i := 0 to ic do
    begin
        St := State [i];
        S := Format (#13#10'            %d:  begin', [St .No]);
        SStream .WriteString (S);
            S := Format ('%s', [OptimalActionAuto16_0 (St .Automates)]);
            S := S + Format ('%s', [OptimalAction16_case2 (St .InitialAction)]);
            SStream .WriteString (S);
        S := #13#10'            end;';
        SStream .WriteString (S);
    end;
    // tail
    S := ''
    +#13#10'        end;'
    +#13#10''
    +#13#10'        l_End:'
    +#13#10'        {$ifdef GRAPH_ENDS_LOGGING}'
    +#13#10'            log_end ('''+_ANAME+''', State, e);'
    +#13#10'        {$endif}'
    +#13#10''
    +#13#10'    end;'
    +#13#10''
    {}
    +#13#10'end.'
    +#13#10''
    +#13#10''
    {}
    ;
    SStream .WriteString (S);
    Result := TRUE;
end;
```