Санкт-Петербургский государственный институт точной механики и оптики (технический университет)

Кафедра «Компьютерные технологии»

Е. А. Заякин, А. А. Шалыто

Метод устранения повторных фрагментов кода при реализации конечных автоматов

Объектно-ориентированное программирование с явным выделением состояний

Разработан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2003

Содержание

Введение

SWITCH-технология [1,2] (http://is.ifmo.ru), предназначенная для алгоритмизации задач управления, основана на применении графов переходов конечных автоматов.

В рамках этой технологии граф переходов отображается формально и изоморфно в текст программы. Однако изоморфизм нарушается при выделении в графе переходов хотя бы одной группы состояний [3], обладающих одинаковыми переходами, так как при этом на графе одинаковые дуги заменяются одной дугой, но в тексте программы каждая из них реализуется отдельно.

Поэтому первой задачей, рассматриваемой в настоящей работе, является обеспечение изоморфизма и в этом случае. Указанная задача может быть решена процедурно [4]. В настоящей работе для этой цели использован объектно-ориентированный подход.

Другая задача устранения повторных фрагментов кода состоит в том, что если в SWITCH-технологии одинаковые выходные действия, помечающие входящие в вершину дуги, переносились в вершину и назывались «Действиями в вершине», то в настоящей работе, как и в работе [3], наряду с такой оптимизацией, называемой «Действиями при входе в вершину», предлагается применять также другой вид оптимизации - «Действия при выходе из вершины», которое заменяет одинаковые выходные действия, выполняемые на исходящих из этой вершины дугах.

Отметим, что в графах переходов возможна и третья разновидность выходных воздействий – «Действия на дугах (и петлях)», которые не могут быть перенесены в вершины.

Еще один вид оптимизации связан с вынесением одинаковых «Действий при входе в вершину» из всех вершин группы, если такие действия имеются. Аналогично можно поступить и с «Действиями при выходе из вершины».

Цель настоящей работы состоит в устранении повторных фрагментов кода на основе объектно-ориентированного подхода и, в частности, в модификации паттерна «State» [5]. Это выполняется за счет введения дополнительных классов для групп «родственных» состояний.

Предлагаемый подход позволяет также упростить повторное использование кода (*code reusing*), реализующего автоматы.

Программа создана с помощью системы разработки Delphi 7.0.

1. Пример

В настоящей работе предлагаемый метод иллюстрируется примером перемещения персонажа по имени *App* по домикам.

При этом домики располагаются в вершинах графа переходов, а персонаж перемещается из домика в домик по дугам, связывающим эти вершины.

Визуализатор этой программы (рис. 1) содержит четыре кнопки («Событие 1»,

«Событие 2», «Событие 3» и «Событие 4»), обеспечивающие перемещения персонажа по графу. Используются действия из следующих групп:

- уменьшение и увеличения размера персонажа;
- установка желтого, белого и зеленого цветов для окраски персонажа;
- открытие и закрытие рта.

При этом отметим, что действия из перечисленных групп могут происходить одновременно.

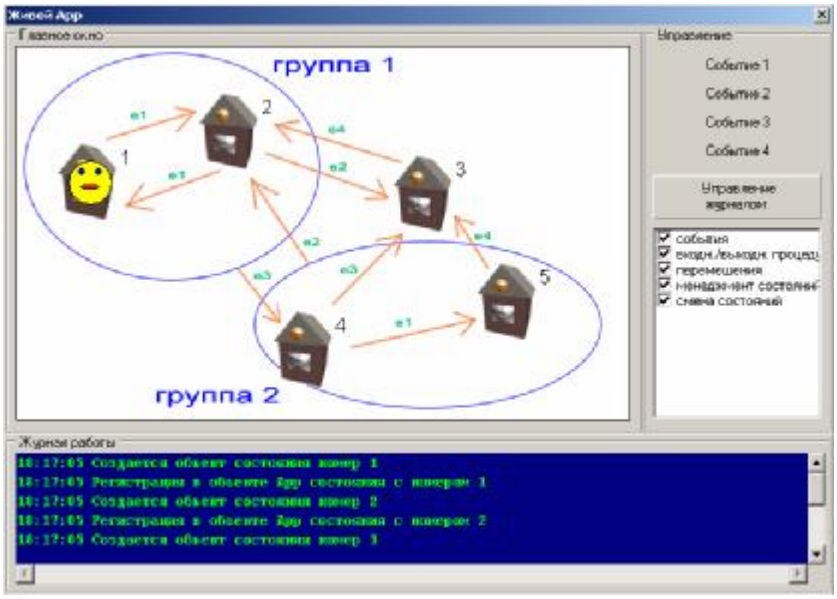

Рис. 1. Внешний вид программы

Обратим внимание, что на рис.1 имеются групповые переходы, как из первой группы, так и из второй группы.

2.Диаграмма классов состояний

Для решения поставленной задачи предлагается каждое состояние выделить в отдельный класс. Кроме состояний, в отдельные классы выделяются и их группы, если состояния имеют одинаковые переходы или в состояниях выполняются одинаковые действия. Кроме того, выделяется абстрактный базовый класс «Состояние», который за счет наследования определяет общий интерфейс всех потомков.

Иерархия классов состояний позволяет, кроме наследования, использовать также и полиморфизм.

На рис. 2 приведена диаграмма классов состояний для рассматриваемого примера.

Отметим, что термин «диаграмма классов состояний» является оригинальным.

Кроме того, отметим, что эта диаграмма отличается от фрагмента паттерна «State» [5], связанного с состояниями, наличием классов, соответствующих группам «родственных» состояний. Состояния названы «родственными», так как они обладают одинаковыми переходами или формируют одинаковые выходные воздействия.

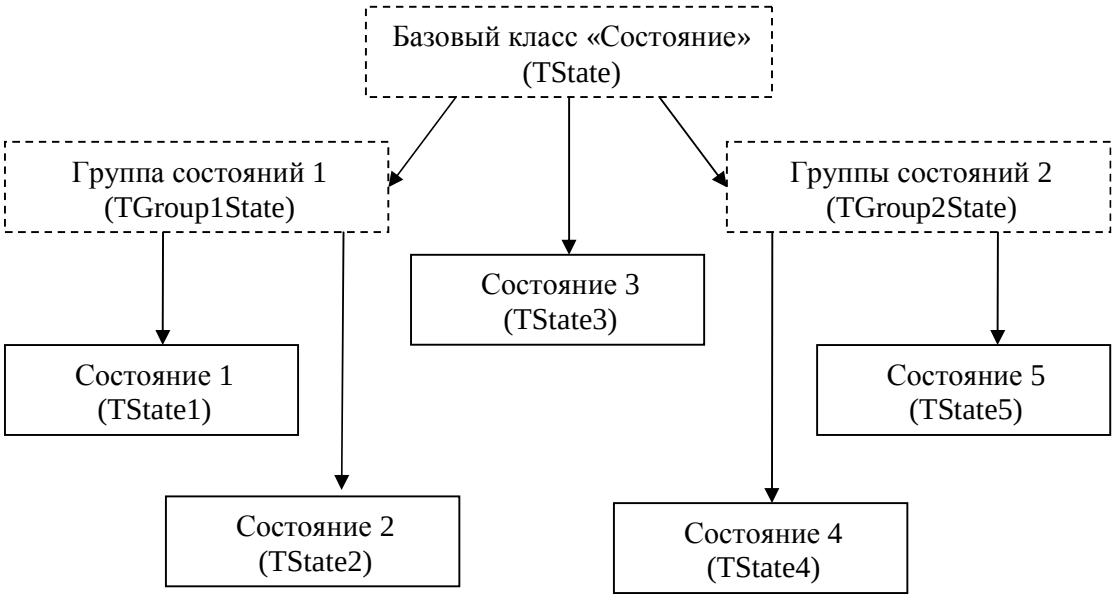

Рис. 2. Диаграмма классов состояний

Оптимизация обеспечивается, если все классы состояний реализуют виртуальный метод *ProcessEvent*, шаблон которого имеет следующий вид:

```
процедура НазваниеСотояния.ОбработатьСобытие
  ( номер_события )
begin
  case номер_события of

    <значение1>: begin
      ВыполнитьДействиеНаПереходе();
      ЗаменитьСостояниеНа(номер_нового_состояния);
    end;

    <значение2>: begin
      ВыполнитьДействиеНаПереходе();
      ЗаменитьСостояниеНа(номер_нового_состояния);
    end;

    ...

    else ПередатьУправлениеПредку();
  end;
end;
```

Если действие на переходе отсутствует, то вместо «ВыполнитьДействиеНаПереходе();» в тексте программы ставится символ «;».

Рассмотренный шаблон применим для случая, когда используются входные воздействия только одного типа – события (входные переменные не применяются). Шаблон не зависит от языка программирования.

Отметим, что для используемого в работе языка *Паскаль* в конструкции ***case*** служебное слово ***else*** эквивалентно служебному слову ***default*** в конструкции ***SWITCH*** языка *C*. Передача управления предку в языке *Паскаль* выполняет служебное слово ***inherited.***

Обратим внимание, что в данной работе выбор осуществляется по событию [6], а не по состоянию, как это делается в SWITCH-технологии. Отметим также, что рамках данного подхода, в отличие от работы [4], где паттерн «State» является первичным, исходным является граф переходов, а всё предлагаемое – его реализацией.

3.Метод устранения повторных фрагментов кода

Предлагаемый метод состоит из следующих этапов:

- разработка структурной схемы приложения;
- проектирование каждого автомата на основе SWITCH-технологии. При этом для каждого автомата разрабатываются словесное описание, схема связей и граф переходов;
- разработка диаграммы классов *приложения*, которая детализирует его структурную схему в части реализации автоматов;
- реализация каждого автомата состоит из следующих подэтапов:
    - разрабатывается диаграмма классов *состояний*, аналогичная приведенной на рис.2;
    - разрабатываются два модуля, первый из которых содержит класс, обеспечивающий переходы между состояниями, а также определяет абстрактный базовый класс «Состояние». Второй модуль содержит классы всех состояний и групп состояний;
- реализация функций выходных воздействий;
- написание вспомогательных модулей программы, например, визуализатора и связи с операционной системой;
- сборка проекта;
- отладка с помощью логов, описывающих поведение программы в терминах автоматов;
- разработка проектной документации.

4.Структурная схема приложения

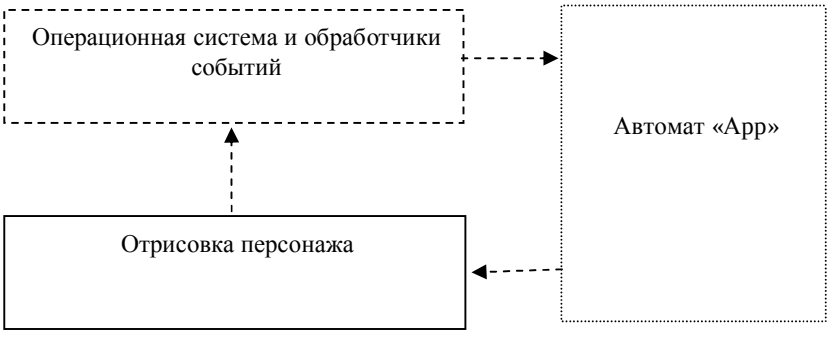

Рис. 3. Структурная схема приложения

5. Проектирование автомата

5.1. Словесное описание

Функциональность автомата приведена в разд. 1.

5.2. Схема связей автомата

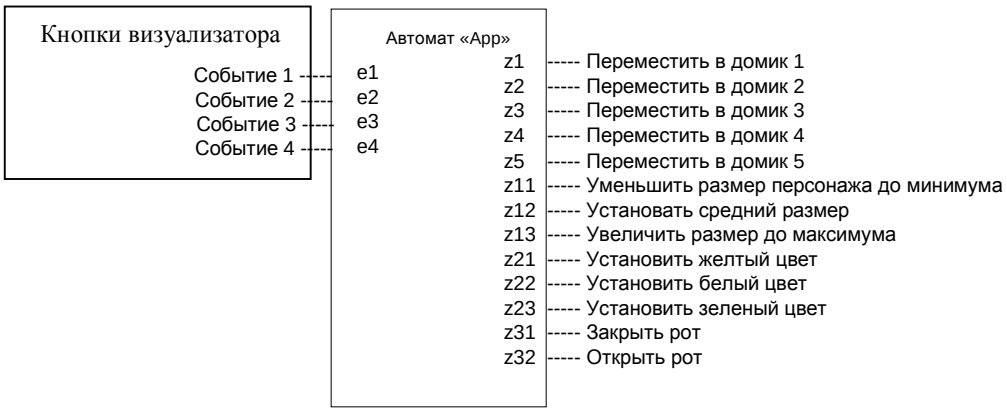

Рис. 4. Схема связей автомата

Все события поступают от кнопок визуализатора (рис. 1), а все выходные воздействия реализуются в классе *TPhoto*.

5.3. Граф переходов автомата «App»

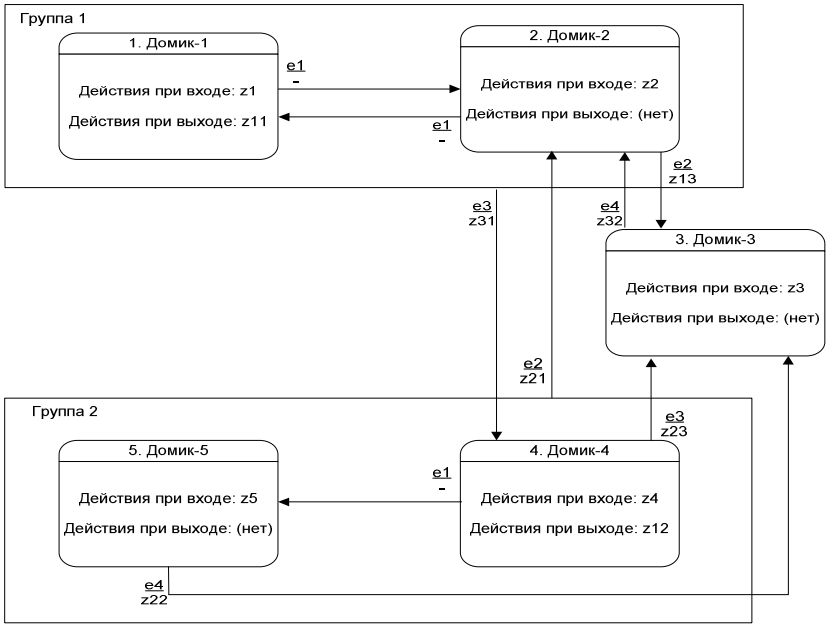

Рис. 5. Граф переходов автомата «App»

6.Диаграмма классов

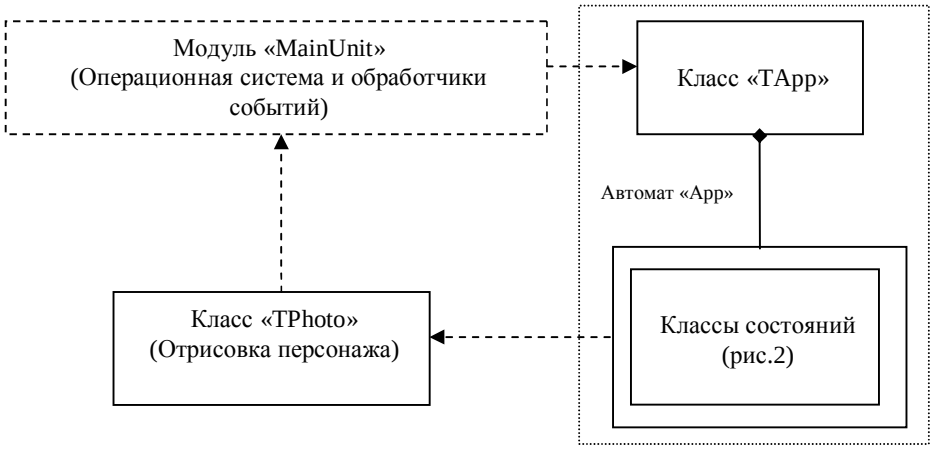

Рис. 6. Диаграмма классов примера

При реализации класс «TApp» (рис. 6) и абстрактный базовый класс «Состояние» (рис. 2) образуют один модуль, а классы, соответствующие состояниям и группам состояний (рис.2) – другой модуль.

7.Класс «TApp»

7.1.Словесное описание

Класс обеспечивает реализацию переходов в автомате. Он хранит информацию обо всех зарегистрированных в нем состояниях (объектах) вместе с указателем на текущее состояние.

7.2.Названия методов класса

Методы имеют следующие названия:

- `CurrentAutomataState` – возвращает текущее состояние;
- `SetState` – устанавливает новое состояние;
- `RegisterState` – регистрирует состояние в автомате;
- `ProcessEvent` – передает событие методу *ProcessEvent* текущего (другого) объекта «Состояние».

8.Класс «TState»

8.1.Словесное описание

Базовый класс, описывающий общие свойства состояний автомата.

8.2.Названия методов класса

Методы имеют следующие названия:

- `Number` – номер состояния в автомате;
- `Left` – X-координата на плоскости;

- `Top` – Y-координата на плоскости;
- `ProcessEvent` – основная функция обработки (шаблон приведен в разд. 2);
- `Enter` – действие при входе в вершину;
- `Exit` – действие при выходе из вершины.

8.Классы-наследники «TState»

В примере выделено семь классов-наследников, реализуемых по шаблону (разд. 2).

9.Модуль, реализующий выходные воздействия

Все выходные воздействия реализованы в этом модуле.

10.Класс «TPhoto»

10.1.Словесное описание

Класс, осуществляющий отображение главного персонажа на экране. Объект этого класса управляется с помощью выходных действий автомата.

10.2.Названия методов класса

Методы имеют следующие названия:

- `Left` – X-координата на плоскости;
- `Top` – Y-координата на плоскости;
- `Smile` – коэффициент «открытости рта»;
- `Color` – цвет персонажа;
- `Size` – коэффициент размера игрока;
- `WalkTo` –метод, перемещающий персонаж в заданную точку.

11.Модуль взаимодействия с операционной системой «MainUnit»

В модуле описана реализация взаимодействия с операционной системой, а также реакции на события.

12.Исходные тексты программы

12.1.Файл «PhotoUnit.pas»

```
//Модуль содержит класс, который реализует
//отрисовку "физиономии" App'а.

unit PhotoUnit;
```

```
interface
uses Graphics, ExtCtrls, Forms, Windows;
type
  TPhoto = class(TObject)
  private
    fX,fY: integer;
    fSmile, fSize: double;
    fColor: TColor;
    appBody: TShape;
    appEyes: array [0..1] of TShape;
    appMouth: TShape;
    procedure setXPos(x: integer);
    procedure setYPos(y: integer);
    procedure setSmile(s: double);
    procedure setColor(c: TColor);
    procedure setSize(s: double);
    procedure Update;
  public
    property Left: integer read fX write setXPos;       //X-координата
    property Top: integer read fY write setYPos;        //Y-координата
    property Smile: double read fSmile write setSmile;  //"Улыбчивость"
    property Color: TColor read fColor write setColor;  //Цвет
    property Size: double read fSize write setSize;     //Размер
    constructor Create;
    procedure WalkTo(newX, newY: integer);  //Перемещает изображение в (x,y)
  end;

implementation
uses MainUnit;

procedure TPhoto.setXPos(x: integer);
begin
  fX:=x;
  Update;
end;

procedure TPhoto.setYPos(y: integer);
begin
  fY:=y;
  Update;
end;

procedure TPhoto.setSmile(s: double);
begin
  fSmile:=s;
  Update;
end;

procedure TPhoto.setSize(s: double);
begin
  fSize:=s;
  Update;
end;

procedure TPhoto.setColor(c: TColor);
begin
  fColor:=c;
  Update;
end;

//Создание объекта-"Фото"
constructor TPhoto.Create;
var
  i: integer;
begin
  fColor:=clYellow;
  fSize:=0.5;
  fSmile:=0;
  fX:=0;
  fY:=0;
  appBody:=TShape.Create(MainForm);
  appBody.Shape:=stEllipse;
  appBody.Parent:=MainForm.ViewPanel;
  for i := 0 to 1 do
  begin
    appEyes[i]:=TShape.Create(MainForm);
    appEyes[i].Shape:=stEllipse;
    appEyes[i].Parent:=MainForm.ViewPanel;
    appEyes[i].Width:=6;
    appEyes[i].height:=6;
```

```
    appEyes[i].Brush.Color:=clBlue;
  end;
  appMouth:=TShape.Create(MainForm);
  appMouth.Shape:=stEllipse;
  appMouth.Parent:=MainForm.ViewPanel;
  appMouth.Brush.Color:=clRed;
  appMouth.Width:=16;

  Update;
end;

//Перерисовывает изображение Аппа
procedure TPhoto.Update;
begin
  appBody.Brush.Color:=fColor;
  appBody.Width:=25+round(fSize*25);
  appBody.Height:=25+round(fSize*25);
  appBody.Left:=fX-appBody.Width div 2;
  appBody.Top:=fY-appBody.height div 2;
  appEyes[0].Left:=fX-3-appBody.Width div 5;
  appEyes[0].Top:=fY-appBody.Width div 3;
  appEyes[1].Left:=-3+fX+appBody.Width div 5;
  appEyes[1].Top:=fY-appBody.Width div 3;
  appMouth.Top:=fY+2;//+appBody.Height div 10;
  appMouth.Left:=fX-appMouth.Width div 2;
  appMouth.Height:=4+round(fSmile*8);
end;

//Перемещает изображение в (x,y)
procedure TPhoto.WalkTo(newX, newY: integer);
var
  i, oldX, oldY: integer;
begin
  MainForm.ControlPanel.Enabled:=false;
  oldX:=fX;
  oldY:=fY;
  for i := 0 to 10 do
  begin
    fX:=oldX+round(i*(newX-oldX)/10);
    fY:=oldY+round(i*(newY-oldY)/10);
    Update;
    Application.ProcessMessages;
    Sleep(90);
  end;
  MainForm.ControlPanel.Enabled:=true;
end;

end.
```

12.2. Файл «MainUnit.pas»

```
//Модуль содержит описание структур, необходимых
//для функционирования в среде «Windows»

unit MainUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Buttons, CheckLst, StateUnit, AutomataUnit,
  PhotoUnit;

type
  TMainForm = class(TForm)
    ControlPanel: TGroupBox;
    LogPanel: TGroupBox;
    LogMemo: TMemo;
    ViewPanel: TGroupBox;
    ViewImage: TImage;
    Sig1Btn: TSpeedButton;
    Sig2Btn: TSpeedButton;
    Sig3Btn: TSpeedButton;
    Sig4Btn: TSpeedButton;
    Label1: TLabel;
    Bevel1: TBevel;
```

```
    Checks: TCheckListBox;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Sig3BtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;  //Главное окно программы
  buffer: TBitmap;      //"Теневой" буфер
  App: TApp;            //Конечный автомат "Апп"
  AppPhoto: TPhoto;     //Изображение Аппа

const
  //Типы сообщений журнала
  ltEvent            = 0;
  ltEnterExit         = 1;
  ltMoves             = 2;
  ltStateManagement   = 3;
  ltStateChange       = 4;

//Делегирует событие автомату
procedure ProcessEvent(Event: integer);

//Выводит заданную строку в журнал программы
procedure Log(messageType: integer; messageText: string);

implementation

{$R *.dfm}

//Инициализация оконной системы и создание объектов
procedure TMainForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  buffer:=TBitmap.Create;
  buffer.Width:=ViewImage.Width;
  buffer.Height:=ViewImage.Height;
  for i := 0 to Checks.Items.Count-1 do
    Checks.Checked[i]:=true;

  //Создание объекта "автомат"
  App:=TApp.Create;

  //Содание и регистрация "домиков"-состояний
  App.RegisterState(TState1.Create(App,70,130));
  App.RegisterState(TState2.Create(App,188,88));
  App.RegisterState(TState3.Create(App,350,140));
  App.RegisterState(TState4.Create(App,250,270));
  App.RegisterState(TState5.Create(App,420,230));

  //Инициализация "фото" Аппа
  AppPhoto:=TPhoto.Create;
  AppPhoto.Left:=App.CurrentAutomataState.Left;
  AppPhoto.Top:=App.CurrentAutomataState.Top;
end;

//Освобождение занятых системных ресурсов
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  buffer.Free;
  AppPhoto.Free;
  App.Free; //Уничтожение объекта-автомата
end;

//Обновляет буфер перед переброской его на экран
procedure UpdateBuffer;
begin
  with buffer do
  begin
    canvas.Pen.Color:=clBlack;
    canvas.Pen.Width:=1;
    canvas.Brush.Color:=clWhite;
    canvas.Rectangle(0,0,width,height);
  end;
end;
```

```
//Обновляет вид на экране
procedure UpdateView;
begin
  UpdateBuffer;
  MainForm.ViewImage.Canvas.Draw(0,0,buffer);
end;

//Делегирует событие автомату
procedure ProcessEvent(Event: integer);
begin
  App.ProcessEvent(Event);
end;

//Реакция на нажатие кнопки
procedure TMainForm.Sig3BtnClick(Sender: TObject);
begin
  Log(ltEvent,'Отправлено событие '+
    inttostr((sender as TSpeedButton).Tag));
  ProcessEvent((sender as TSpeedButton).Tag);
end;

//Выводит заданную строку в журнал программы
procedure Log(messageType: integer; messageText: string);
begin
  if MainForm.Checks.Checked[messageType] then
    MainForm.LogMemo.Lines.Add(timetostr(now)+' '+messageText)
end;

end.
```

12.3. Файл «AutomataUnit.pas»

```
//Модуль содержит описание объекта-автомата "Апп".
//Также в этом модуле определяется базовый класс "Дом-состояние"
//и реализуются все общие для его потомков методы.

unit AutomataUnit;
interface
uses SysUtils, ExtCtrls, StdCtrls, Graphics;
type
  TState = class;

  //Класс «Автомат "Апп"»
  TApp = class(TObject)

  private
    states: array of TState;
    currentState: TState;

  public
    //Для просмотра и управления
    function GetRegisteredState(i: integer): TState;
    property CurrentAutomataState: TState read currentState;

    //Устанавливает новое состояние
    procedure SetState(stateIndex: integer);

    //Регистрирует объект состояния
    procedure RegisterState(state: TState);
    destructor Destroy; override;

    //Делегирует событие текущему состоянию
    procedure ProcessEvent(Event: integer);
  end;

  //Класс "Дом-состояние"
  TState = class(TObject)

  protected
    fX, fY: integer;  //Позиции "домика" на экране
    fIndex: integer;  //Номер состояния в автомате - заполняется автоматически
    fImage: TImage;   //Изображение "домика"
    fLabel: TLabel;   //Номерной знак "домика"
    fOwner: TApp;
```

```
  public
    property Number: integer read fIndex; //Номер состояния в автомате
    property Left: integer read fX;       //Координата X "домика" на плоскости
    property Top: integer read fY;        //Координата Y "домика" на плоскости

    procedure ProcessEvent(Event: integer); virtual; //Функция обработки события
    procedure Enter; virtual; //"Действие при входе в состояние"
    procedure Exit; virtual;  //"Действие при выходе из состояния"

    constructor Create(app: TApp; x,y: integer);
    destructor Destroy; override;
  end;

implementation
uses MainUnit;
var
  StateIndex: integer = 1;



//  ---  Описание объекта-автомата “App” --- //

//Возвращает зарегистрированное состояние номер i
function TApp.GetRegisteredState(i: integer): TState;
begin
  result:=states[i-1];
end;

//Регистрирует объект «Состояние» в своем списке состояний
procedure TApp.RegisterState(state: TState);
begin
  log(ltStateManagement,'Регистрация в объекте App состояния с номером '
    +inttostr(state.Number));
  setLength(states,length(states)+1);
  States[high(states)]:=state;
  if length(states) = 1 then currentState:=state;
end;

//Деструктор автомата - удаляет все зарегистрированные состояния
destructor TApp.Destroy;
var
  i: integer;
begin
  for i := 0 to high(states) do
    states[i].Free;
  setlength(states,0);
  inherited;
end;

//Процедура обработки события - делегирует событие текущему состоянию
procedure TApp.ProcessEvent(Event: integer);
begin
  currentState.ProcessEvent(Event);
end;

//Устанавливает новое состояние автомата
procedure TApp.SetState(stateIndex: integer);
begin
  //Вызываем функцию "Действие при выходе из состояния"
  log(ltEnterExit,'Выполняем выходную процедуру:');
  currentState.Exit;

  //Устанавливаем новое состояние
  currentState:=states[stateIndex-1];
  log(ltStateChange,'Автомат App изменил состояние на состояние '+inttostr(stateIndex));

  // Вызываем функцию "Действие при входе из состояния"
  log(ltEnterExit,'Выполняем входную процедуру:');
  currentState.Enter;
end;


//  ---  Описание базового объекта “State” --- //

//Деструктор объекта
destructor TState.Destroy;
begin
  log(ltStateManagement,'Уничтожается объект состояния номер '+inttostr(fIndex));
  inherited;
end;
```

```
//Конструктор объекта
constructor TState.Create(app: TApp; x,y: integer);
begin
  fIndex:=StateIndex;
  inc(StateIndex);
  fX:=x;
  fY:=y;
  fOwner:=app;
  log(ltStateManagement,'Создается объект состояния номер '+inttostr(fIndex));
  fImage:=TImage.Create(MainForm);
  fImage.AutoSize:=true;
  fImage.Picture.LoadFromFile('house.bmp');
  fImage.Transparent:=true;
  fImage.Parent:=MainForm.ViewPanel;
  fImage.Left:=fX-fImage.Width div 2;
  fImage.Top:=fY-fImage.Height div 2;
  fLabel:=TLabel.Create(MainForm);
  fLabel.Parent:=MainForm.ViewPanel;
  fLabel.Caption:=inttostr(fIndex);
  FLabel.Left:=fX+25;
  fLabel.Transparent:=true;
  FLabel.Top:=fY-30;
  fLabel.Font.Name:='Arial';
  fLabel.Font.Color:=clBlack;
  fLabel.Font.Size:=12;
end;

//Функция обработки события
procedure TState.ProcessEvent(Event: integer);
begin
  //Ничего не делать - это базовый класс
end;

//Функция "Действие при входе в состояние"
procedure TState.Enter;
begin
  //Ничего не делать - это базовый класс
end;

//Функция "Действие при выходе из состояния"
procedure TState.Exit;
begin
  //Ничего не делать - это базовый класс
end;


end.
```

12.4. Файл «StateUnit.pas»

```
//Описания классов состояний автомата “App”.
//Любой класс, описываемый в данном модуле - наследник
//класса «TState»

unit StateUnit;
interface
uses SysUtils, Graphics, AutomataUnit;

  procedure z1;
  procedure z2;
  procedure z3;
  procedure z4;
  procedure z5;
  procedure z11;
  procedure z12;
  procedure z13;
  procedure z21;
  procedure z22;
  procedure z23;
  procedure z31;
  procedure z32;

type
```

```
  //Описание классов состояний и групп состояний

  TState3 = class(TState)
  public
    procedure ProcessEvent(Event: integer); override;
    procedure Enter; override;
  end;

  TGroup1State = class(TState)
  public
    procedure ProcessEvent(Event: integer); override;
  end;

  TState1 = class(TGroup1State)
  public
    procedure ProcessEvent(Event: integer); override;
    procedure Enter; override;
    procedure Exit; override;
  end;

  TState2 = class(TGroup1State)
  public
    procedure ProcessEvent(Event: integer); override;
    procedure Enter; override;
  end;

  TGroup2State = class(TState)
  public
    procedure ProcessEvent(Event: integer); override;
  end;

  TState4 = class(TGroup2State)
  public
    procedure ProcessEvent(Event: integer); override;
    procedure Enter; override;
    procedure Exit; override;
  end;

  TState5 = class(TGroup2State)
  public
    procedure ProcessEvent(Event: integer); override;
    procedure Enter; override;
  end;


implementation
uses MainUnit;


//  ---  Реализация функций выходных воздействий  --- //

procedure z1;
var
  s: TState;
begin
  Log(ltEnterExit,'..Выполняется процедура перемещения к домику 1 (z1)');
  s:=App.GetRegisteredState(1);
  AppPhoto.WalkTo(s.Left,s.Top);
end;

procedure z2;
var
  s: TState;
begin
  Log(ltEnterExit,'..Выполняется процедура перемещения к домику 2 (z2)');
  s:=App.GetRegisteredState(2);
  AppPhoto.WalkTo(s.Left,s.Top);
end;

procedure z3;
var
  s: TState;
begin
  Log(ltEnterExit,'..Выполняется процедура перемещения к домику 3 (z3)');
  s:=App.GetRegisteredState(3);
  AppPhoto.WalkTo(s.Left,s.Top);
end;
```

```
procedure z4;
var
  s: TState;
begin
  Log(ltEnterExit,'..Выполняется процедура перемещения к домику 4 (z4)');
  s:=App.GetRegisteredState(4);
  AppPhoto.WalkTo(s.Left,s.Top);
end;

procedure z5;
var
  s: TState;
begin
  Log(ltEnterExit,'..Выполняется процедура перемещения к домику 5 (z5)');
  s:=App.GetRegisteredState(5);
  AppPhoto.WalkTo(s.Left,s.Top);
end;

procedure z11;
begin
  Log(ltEnterExit,'..Выполняется процедура z11 - уменьшение до минимального размера');
  AppPhoto.Size:=0;
end;

procedure z12;
begin
  Log(ltEnterExit,'..Выполняется процедура z12 - уставнока среднего размера');
  AppPhoto.Size:=0.5;
end;

procedure z13;
begin
  Log(ltEnterExit,'..Выполняется процедура z13 - увеличение до максимального размера');
  AppPhoto.Size:=1;
end;

procedure z21;
begin
  Log(ltEnterExit,'..Выполняется процедура z21 - установка желтого цвета');
  AppPhoto.Color:=clYellow;
end;

procedure z22;
begin
  Log(ltEnterExit,'..Выполняется процедура z22 - установка белого цвета');
  AppPhoto.Color:=clWhite;
end;

procedure z23;
begin
  Log(ltEnterExit,'..Выполняется процедура z23 - установка зеленого цвета');
  AppPhoto.Color:=clLime;
end;

procedure z31;
begin
  Log(ltEnterExit,'..Выполняется процедура z31 - "губы сжаты"');
  AppPhoto.Smile:=0;
end;

procedure z32;
begin
  Log(ltEnterExit,'..Выполняется процедура z32 - "рот широко открыт"');
  AppPhoto.Smile:=1;
end;


//  ---  Описание состояния 3 --- //

procedure TState3.ProcessEvent(Event: integer);
begin
  case Event of
    4: begin
         z32;
         fOwner.SetState(2);
       end;
    else inherited;
  end;
end;
```

```
procedure TState3.Enter;
begin
  z3;
end;


//  ---  Описание первой группы состояний --- //

procedure TGroup1State.ProcessEvent(Event: integer);
begin
  case Event of
    3: begin
         z31;
         fOwner.SetState(4);
       end;
    else inherited;
  end;
end;

procedure TState1.ProcessEvent(Event: integer);
begin
  case Event of
    1: begin
         ;
         fOwner.SetState(2);
       end;
    else inherited; //Передать управление своему предку - группе 1
  end;
end;

procedure TState1.Enter;
begin
  z1;
end;

procedure TState1.Exit;
begin
  z11;
end;

procedure TState2.ProcessEvent(Event: integer);
begin
  case Event of
    1: begin
         ;
         fOwner.SetState(1);
       end;
    2: begin
         z13;
         fOwner.SetState(3);
       end;
    else inherited;  //Передать управление своему предку - группе 1
  end;
end;

procedure TState2.Enter;
begin
  z2;
end;


//  ---  Описание второй группы состояний --- //

procedure TGroup2State.ProcessEvent(Event: integer);
begin
  case Event of
    2: begin
         z21;
         fOwner.SetState(2);
       end;
    else inherited;
  end;
end;

procedure TState4.ProcessEvent(Event: integer);
begin
  case Event of
    1: begin
         ;
         fOwner.SetState(5);
```

```
       end;
    3: begin
         z23;
         fOwner.SetState(3);
       end;
    else inherited;  //Передать управление своему предку - группе 2
  end;
end;

procedure TState4.Enter;
begin
  z4;
end;

procedure TState4.Exit;
begin
  z12;
end;

procedure TState5.ProcessEvent(Event: integer);
begin
  case Event of
    4: begin
         z22;
         fOwner.SetState(3);
       end;
    else inherited;  //Передать управление своему предку - группе 2
  end;
end;

procedure TState5.Enter;
begin
  z5;
end;

end.
```

13. Пример протоколирования

Рассмотрим следующий сценарий:

1. Начальное положение – домик 1.
2. Событие 1.
3. Переход в домик 2.
4. Событие 3.
5. Переход в домик 4.
6. Событие 1.
7. Переход в домик 5.
8. Событие 4.
9. Переход в домик 3.
10. Событие 4.
11. Переход в домик 2.
12. Событие 1.
13. Возвращение в домик 1.

Протокол, соответствующий этому сценарию, имеет следующий вид:

```
12:04:37 Создается объект состояния номер 1
12:04:37 Регистрация в объекте App состояния с номером 1
12:04:37 Создается объект состояния номер 2
12:04:37 Регистрация в объекте App состояния с номером 2
12:04:37 Создается объект состояния номер 3
12:04:37 Регистрация в объекте App состояния с номером 3
12:04:37 Создается объект состояния номер 4
12:04:37 Регистрация в объекте App состояния с номером 4
```

```
12:04:37 Создается объект состояния номер 5
12:04:37 Регистрация в объекте App состояния с номером 5
12:05:32 Отправлено событие 1
12:05:32 Выполняем выходную процедуру:
12:05:32 ..Выполняется процедура z11 - уменьшение до минимального размера
12:05:32 Автомат App изменил состояние на состояние 2
12:05:32 Выполняем входную процедуру:
12:05:32 ..Выполняется процедура перемещения к домику 2 (z2)
12:06:32 Отправлено событие 3
12:06:32 ..Выполняется процедура z31 - "губы сжаты"
12:06:32 Выполняем выходную процедуру:
12:06:32 Автомат App изменил состояние на состояние 4
12:06:32 Выполняем входную процедуру:
12:06:32 ..Выполняется процедура перемещения к домику 4 (z4)
12:07:12 Отправлено событие 1
12:07:12 Выполняем выходную процедуру:
12:07:12 ..Выполняется процедура z12 - установка среднего размера
12:07:12 Автомат App изменил состояние на состояние 5
12:07:12 Выполняем входную процедуру:
12:07:12 ..Выполняется процедура перемещения к домику 5 (z5)
12:07:43 Отправлено событие 4
12:07:43 ..Выполняется процедура z22 - установка белого цвета
12:07:43 Выполняем выходную процедуру:
12:07:43 Автомат App изменил состояние на состояние 3
12:07:43 Выполняем входную процедуру:
12:07:43 ..Выполняется процедура перемещения к домику 3 (z3)
12:08:07 Отправлено событие 4
12:08:07 ..Выполняется процедура z32 - "рот широко открыт"
12:08:07 Выполняем выходную процедуру:
12:08:07 Автомат App изменил состояние на состояние 2
12:08:07 Выполняем входную процедуру:
12:08:07 ..Выполняется процедура перемещения к домику 2 (z2)
12:08:33 Отправлено событие 1
12:08:33 Выполняем выходную процедуру:
12:08:33 Автомат App изменил состояние на состояние 1
12:08:33 Выполняем входную процедуру:
12:08:33 ..Выполняется процедура перемещения к домику 1 (z1)
```

14. Заключение

Из рассмотренного примера следует, что предложенный метод обеспечивает достижение поставленной в работе цели – устранение повторных фрагментов кода, что достигается, однако, за счет его усложнения – введения новых классов. Отметим, что рассмотренная задача решена также и в работе [4] с использованием процедурного программирования.

15. Литература

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Шалыто А.А., Туккель Н.И.,* SWITCH-технология – автоматный подход к созданию программного обеспечения» реактивных систем //Программирование. 2001, №5.
3. *Рамбо Д., Якобсон А., Буч Г.* UML. Специальный справочник. СПб.: Питер, 2002.
4. *Марков С.М.., Шалыто А.А* .Система управления травоядным существом для игры “Terrarium”. http://is.ifmo.ru/. Раздел «Проекты».

5. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб.: Питер, 2001.
6. *Мартин Р.* Designing Object-Oriented C++ Applications Using The Booch Method. NJ.: Prentice Hall, 1993.