Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

А.А. Астафуров, А.А. Шалыто

Разработка и применение паттерна “Automata”

Проектная документация

Проект создан в рамках
“Движения за открытую проектную документацию”
http://is.ifmo.ru

Санкт-Петербург
2003

Введение

В последние годы в программировании весьма широко применяется объектно-ориентированный подход. Для структурирования и накопления базы знаний по объектно-ориентированному проектированию был введен термин “Паттерн” [1] – *описание взаимодействия объектов и классов для решения стандартной задачи проектирования*. В работе [2] было предложено использовать конечные автоматы для описания поведения широкого класса задач программирования. При этом реализация автоматов основана на применении конструкции “Switch”. В свою очередь, в работе [1] был предложен паттерн “State”, который также может быть использован для реализации конечных автоматов.

В настоящей работе анализируются недостатки этих реализаций, а для их устранения предложен паттерн “Automata”.

1. Постановка задачи

Цель настоящей работы состоит в разработке паттерна “Automata” для реализации конечных автоматов. Выполнено его сравнение с конструкциией “Switch” и паттерном “State”. Апробация предложенного паттерна произведена при реализации автомата управления светофором. Этот автомат был предложен в работе [3].

2. Требования к разработке паттерна “Automata”

После анализа материалов по паттернам [1] и по автоматному программированию (http://is.ifmo.ru) было решено создать паттерн, реализующий автомат, а затем описать подход к созданию автоматных приложений на базе этого паттерна.

Этот паттерн должен отвечать следующим требованиям.

1. Гибкость и возможность быстрого и “щадящего” перепроектирования приложения.
2. Возможность повторного использования.
3. Простота понимания.

Предлагаемый паттерн должен быть представлен набором абстрактных классов, при наследовании от которых можно реализовать конкретные автоматы. Помимо перечисленных требований паттерн должен обладать также еще одним свойством: возможностью динамической композиции и декомпозиции составляющих его объектов, что может позволить менять параметры автомата в ходе его работы.

Из изложенного следует, что эти требования не могут быть обеспечены при реализации автомата с помощью централизованной конструкции “Switch”.

3. Почему не паттерн “State”?

Опишем паттерн “State”, диаграмма классов которого приведена на рис. 1. Эта и другие диаграммы классов используют нотацию языка UML [3].

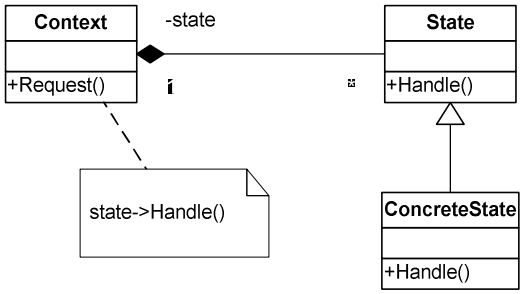

Рис. 1. Диаграмма классов паттерна “State”

В приведенной диаграмме:

- “Context”:
  - определяет интерфейс, представляющий интерес для клиентов;
  - хранит экземпляр подкласса “ConcreteState”, которым определяется текущее состояние;
- “State”:
  - определяет интерфейс для инкапсуляции поведения, ассоциированного с конкретным состоянием контекста “Context”;
- Подклассы “ConcreteState”:
  - каждый подкласс реализует поведение, ассоциированное с некоторым состоянием контекста “Context”

Недостаток этого паттерна состоит в том, что функция переходов из текущего состояния определяется непосредственно в самом классе “State”. Поэтому правила переходов задаются из контекста, что более удобно, хотя авторы работы [1] не раскрывают эти правила.

Обратим внимание, что при реализации автоматов каждый из них является контекстом для рассматриваемого паттерна.

4. Диаграмма классов паттерна “Automata”

Рассмотрим ряд абстракций, характеризующих автоматы.

1. Состояние (Класс “State”). Содержит метод `void Execute(int e)`, который вызывает все вложенные автоматы по списку.
2. Действие, выполняемое в состоянии или при изменении состояния (Класс “Action”). Этот класс содержит метод `void Do()`, инициирующий выполнение действия.

3. Условия перехода (Класс “Condition”). Условие перехода помечает дугу графа переходов. Условие перехода содержит метод `bool Test(int e)`, который проверяет все необходимые переменные и номера событий. Он возвращает значение `true`, если по дуге можно перейти в другое состояние.
4. Автомат (Класс “Automata”). Это абстракция содержит в себе все другие, в том числе и саму себя, так как автомат может содержать вложенные автоматы.
5. Все вложенные автоматы и выходные воздействия хранятся внутри состояния как атрибуты.
6. Все действия хранятся в объектах класса “Переход” (“Transition”), представляющих собой дуги графа переходов, которые, в свою очередь, хранятся в объекте класса “TransitionContainer”.

Диаграмма классов для паттерна “Automata” приведена на рис. 2.

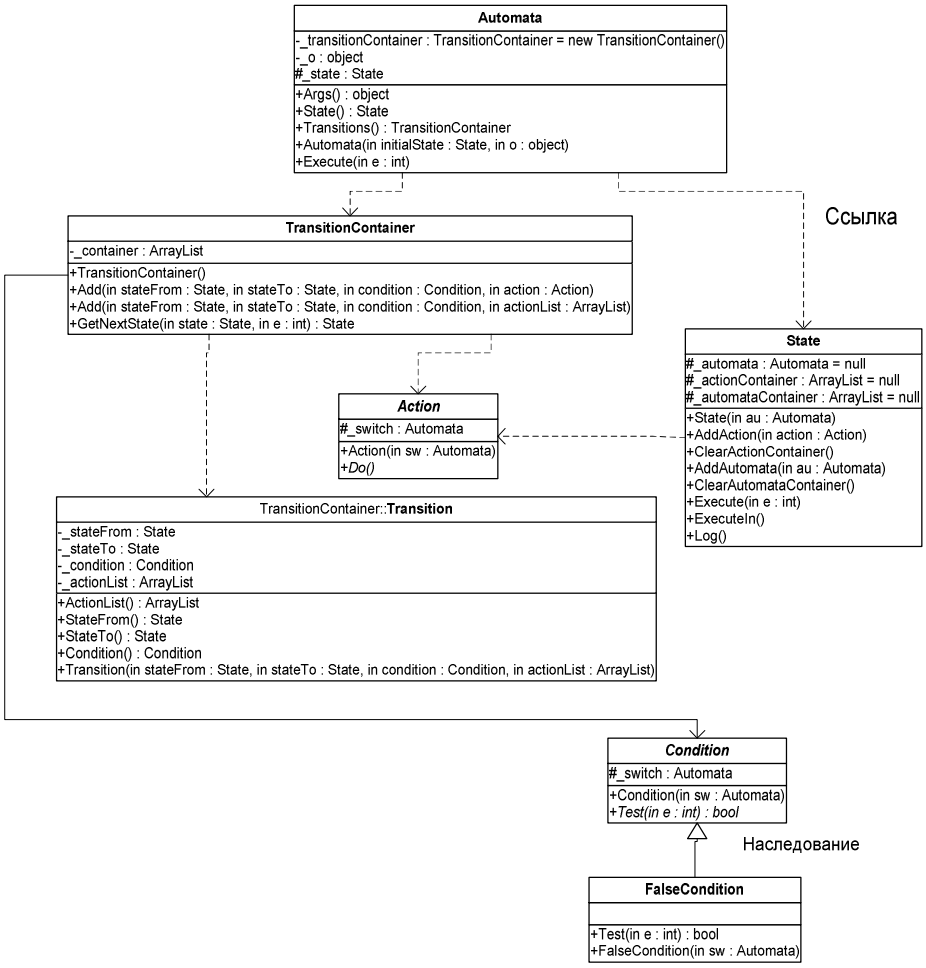

Рис. 2. Диаграмма классов для паттерна “Automata”

5. Класс “State”

5.1. Словесное описание

Абстрактный класс “State” является базовым для всех классов состояний. Каждый потомок этого класса должен реализовать метод `void Execute(int e)`, который будет вызван после перехода в это состояние.

5.2. Структурная схема класса

Рис. 3. Структурная схема класса “State”

6. Класс “Condition”

6.1. Словесное описание

Абстрактный класс “Condition” является базовым для всех классов условий переходов. Каждый потомок этого класса должен реализовать метод `bool Test()`, возвращающий значение `true`, если условие выполняется, и значение `false` в противном случае.

6.2. Структурная схема класса

Рис. 4. Структурная схема класса “Condition”

7. КЛАСС “Action”

7.1. Словесное описание

Абстрактный класс “Action” является базовым для всех классов действий. Каждый потомок этого класса должен реализовать метод `void Do()`, который будет вызван средой при необходимости выполнить действия.

7. 2. Структурная схема класса

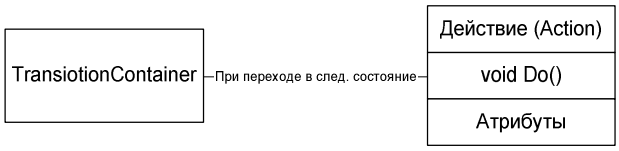

Рис. 5. Структурная схема класса “Action”

8. КЛАСС “Transition”

8.1. Словесное описание

Класс “Transition” задается четверкой объектов `State fromState`, `State toState`, `Condition condition`, `Action action`, которая задает переход из состояния `fromState` в состояние `toState` по условию `condition`, выполняя действие `action`. Если `condition` равен `null`, то из состояния `fromState` будет совершен переход в состояние `toState` в случае невыполнения ни одного из других условий перехода (аналог else в C/C++) или безусловный переход, если другие условия переходов отсутствуют.

8.2. Структурная схема класса

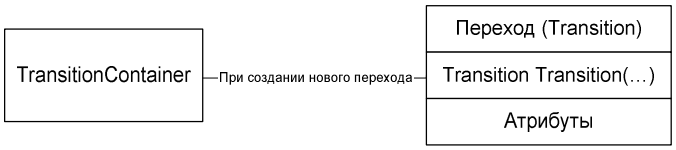

Рис. 6. Структурная схема класса “Transition”

9. Класс “TransitionContainer”

9.1. Словесное описание

Класс “TransitionContainer” хранит дуги графа переходов. Он содержит метод `bool Add(State fromState, State toState, Condition condition, Action action)`.

Этот метод возвращает значение `true`, если четверка была успешно добавлена, и значение `false`, если четверка добавлялась ранее. Это исключает появление одинаковых дуг в графе. Класс также содержит метод `State GetNexState(State state)`, который возвращает состояние, в которое возможен переход из состояния `state` и вызывает метод `void Do()` у объекта “Action”, соответствующего данному переходу.

9.2. Структурная схема класса

Рис. 7. Структурная схема класса “TransitionContainer”

10. Класс “Automata”

10.1. Словесное описание

Класс “Automata” является базовым для реализуемого автомата и содержит в себе основную логику автомата.

При реализации конструктора класса-потомка необходимо создать экземпляр класса “TransitionContainer” и с помощью метода `bool Add(…)` добавить в него все четверки объектов `fromState`, `toState`, `Condition`, `Action`. При этом, необходимо указать какое состояние является заключительным.

Класс содержит метод `Execute(int e)`, через который в автомат передаются события для его запуска. Этот метод, принадлежащий классу “Automata”, вызывает метод `Execute(int e)`, принадлежащий классу "State”. После этого класс “Automata” переходит в следующее состояние, вызвав метод `State TransitionContainer.GetNextState(state)`.

10.2. Структурная схема класса

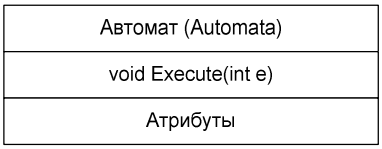

Рис. 8. Структурная схема класса “Automata”

11. Исходный код паттерна “Automata”

Файл State.cs

```
using System;
using System.Collections;
using System.Diagnostics;

namespace AutomataPattern
{
	public class State
	{
		protected Automata _automata = null;
		protected ArrayList _actionContainer = null;
		protected ArrayList _automataContainer = null;

		public State( Automata au )
		{
			_automata = au;
			_actionContainer = new ArrayList();
			_automataContainer = new ArrayList();
		}

		public void AddAction( Action action )
		{
			_actionContainer.Add( action );
		}

		public void ClearActionContainer()
		{
			_actionContainer.Clear();
		}

		public void AddAutomata( Automata au )
		{
			_automataContainer.Add( au );
		}

		public void ClearAutomataContainer()
		{
			_automataContainer.Clear();
		}

		public virtual void Execute( int e )
		{
			foreach ( Automata a in _automataContainer )
				a.Execute( e );
		}

		public virtual void ExecuteIn()
		{
			foreach ( Action a in _actionContainer )
				a.Do();
		}

		public virtual void Log()
		{
			Trace.WriteLine(String.Format("* Entering state: {0}", GetType().ToString()),
"State entry");
		}
	}
}
```

Файл Condition.cs

```
using System;

namespace AutomataPattern
{
	public abstract class Condition
	{
		protected Automata _switch;

		public Condition( Automata sw )
		{
			_switch = sw;
		}
```

```
		public abstract bool Test( int e );
	}
}
```

Файл Action.cs

```
using System;

namespace AutomataPattern
{
	public abstract class Action
	{
		protected Automata _switch;

		public Action( Automata sw )
		{
			_switch = sw;
		}

		public abstract void Do();
	}
}
```

Файл TransitionContainer.cs

```
using System;
using System.Collections;

namespace AutomataPattern
{
	public class TransitionContainer
	{
		private ArrayList _container;

		private class Transition
		{
			private State _stateFrom;
			private State _stateTo;
			private Condition _condition;
			private ArrayList _actionList;

			public ArrayList ActionList
			{
				get { return _actionList; }
				set { _actionList = value; }
			}

			public State StateFrom
			{
				get { return _stateFrom; }
				set { _stateFrom = value; }
			}

			public State StateTo
			{
				get { return _stateTo; }
				set { _stateTo = value; }
			}

			public Condition Condition
			{
				get { return _condition; }
				set { _condition = value; }
			}

			public Transition( State stateFrom, State stateTo, Condition condition, ArrayList
			actionList )
			{
				_stateFrom = stateFrom;
				_stateTo = stateTo;
				_condition = condition;
				_actionList = actionList;
			}
		}

		public TransitionContainer()
		{
			_container = new ArrayList();
```

```
		}

		public void Add( State stateFrom, State stateTo, Condition condition, Action action )
		{
			ArrayList actionList = new ArrayList();
			actionList.Add( action );
			Add(stateFrom, stateTo, condition, actionList);
		}

		public void Add( State stateFrom, State stateTo, Condition condition, ArrayList actionList
)
		{
			foreach ( Transition transition in _container )
				if ( (transition.StateFrom == stateFrom) && (transition.StateTo == stateTo)
&& (transition.Condition == condition) )
					throw new ApplicationException("Transition already exists");

			_container.Add(new Transition(stateFrom, stateTo, condition, actionList) );

		}

		public State GetNextState( State state, int e )
		{
			Transition _default = null;

			foreach (Transition transition in _container)
				if (transition.StateFrom == state)
				{
					if (transition.Condition == null)
						_default = transition;
					else if (transition.Condition.Test( e ))
					{
						if (transition.ActionList != null)
						{
							foreach ( Action a in transition.ActionList )
							{
								if ( a != null)
									a.Do();
							}
						}
						if (transition.StateFrom != transition.StateTo)
							transition.StateTo.ExecuteIn();
						return transition.StateTo;
					}
				}

			if (_default != null)
			{
				if (_default.ActionList != null)
				{
					foreach ( Action a in _default.ActionList )
					{
						a.Do();
					}
				}
				if (_default.StateFrom != _default.StateTo)
					_default.StateTo.ExecuteIn();
				return _default.StateTo;
			}
			return state;
		}
	}
}
```

Файл Automata.cs

```
using System;

namespace AutomataPattern
{
	public class Automata
	{
		private TransitionContainer _transitionContainer = new TransitionContainer();
		private object _o;
		protected State _state;
```

```
		public object Args
		{
			get { return _o; }
		}

		public State State
		{
			get { return _state; }
			set { _state = value; }
		}

		public TransitionContainer Transitions
		{
			get { return _transitionContainer; }
		}

		public Automata( State initialState, object o )
		{
			_o = o;
			_state = initialState;
		}

		public void Execute( int e )
		{
			_state.Execute( e );
			_state = _transitionContainer.GetNextState( _state, e );
			_state.Log();
		}
	}
}
```

Файл FalseCondition.cs (Реализация абстрактного класса Condition)

```
using System;

namespace AutomataPattern
{
	public class FalseCondition : Condition
	{
		public FalseCondition( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			return false;
		}
	}
}
```

12. Управление светофором

12.1. Постановка задачи

Рассматривается светофор [4] с тремя огнями (красным, желтым и зеленым), который снабжен кнопкой включения зеленого света для пешеходов (*) и дистанционным пультом управления. Визуализатор (рис. 9), кроме пульта управления, содержит имитатор внешних воздействий на светофор (панель *Хулиганы*).

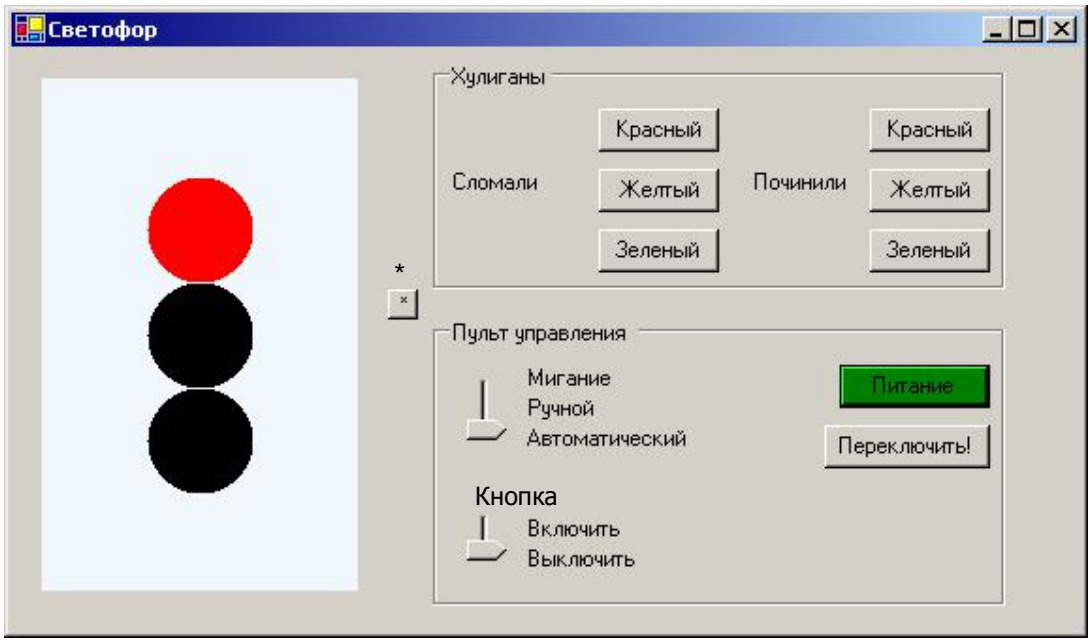

Рис. 9. Визуализатор работы светофора

На пульте управления расположены:

- переключатель режимов – автоматического, ручного, мигания;
- переключатель кнопки включения зеленого света;
- кнопка включения/выключения светофора (“Питание”);
- кнопка переключения зеленого и красного света в ручном режиме (“Переключить”).

Установка указанных переключателей объясняется тем, что светофор должен работать в нескольких режимах: автоматическом, ручном и мигания. Кроме того, имеется переключатель, который обеспечивает возможность включения и отключения кнопки включения зеленого света.

В автоматическом режиме светофор по таймеру последовательно переключается следующим образом: друг за другом загораются зеленый, желтый и красный огни, причем зеленый перед выключением фиксированное время мигает, а красный – горит вместе с желтым.

Если при этом кнопка включения зеленого света разблокирована (переключатель *Кнопка* находится в положении *Включить*), то светофор переключается с красного света на зеленый только при ее нажатии. Если пешеходы будут постоянно нажимать на эту кнопку, красный свет для них должен гореть не менее предусмотренного автоматическим режимом времени.

Ручной режим предполагает переключение с красного на зеленый свет (и наоборот) только по сигналу с пульта управления. Режим мигания означает, что

светофор мигает желтым светом и должен интерпретироваться участниками дорожно-транспортного движения как неработающий[1].

Кроме того, в светофоре предусмотрен контроль перегорания ламп. В случае, если из строя выходит лампа красного или зеленого света, светофор автоматически переходит в режим мигания. Если же перегорает и лампа желтого света, то светофор выключается и может быть включен после ремонта.

12.2. Схема связей

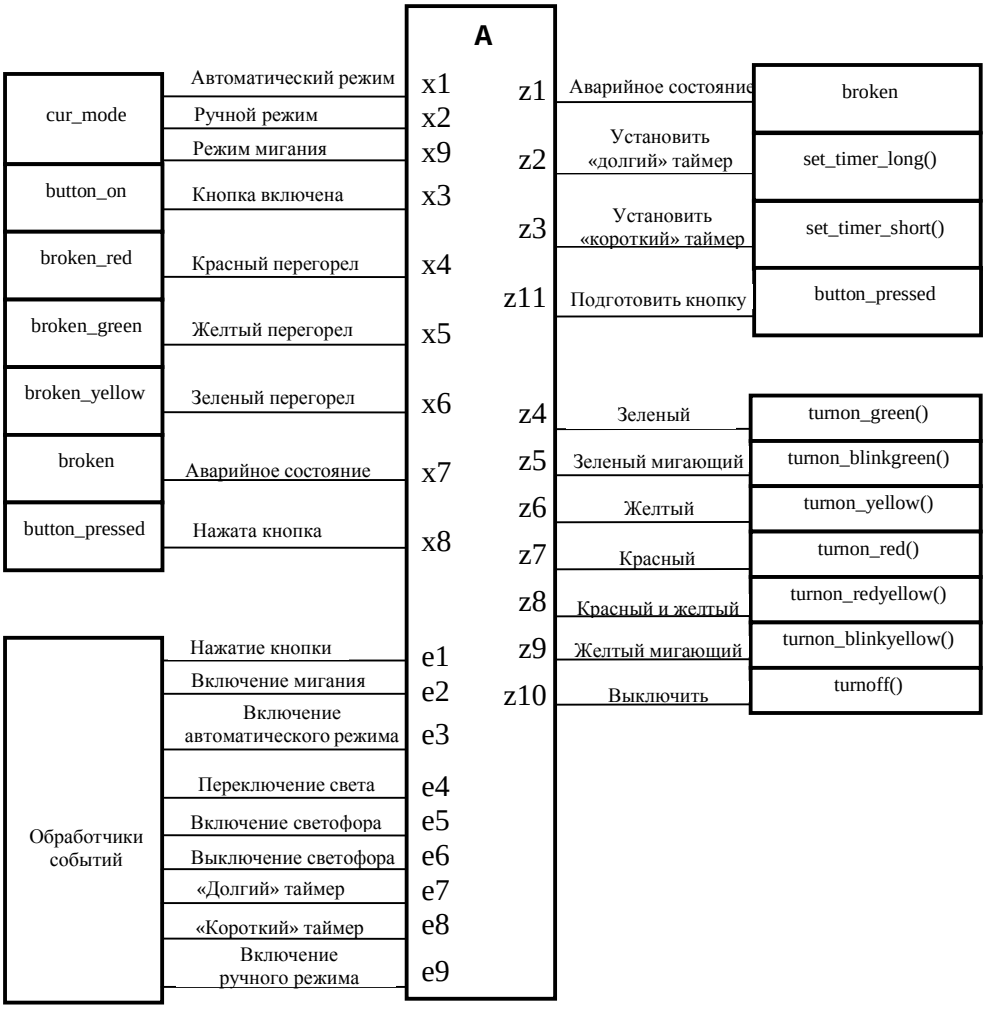

Рис. 10. Схема связей автомата управления светофором

[1] Правила дорожного движения. http://www.rules.ru

12.3. Граф переходов

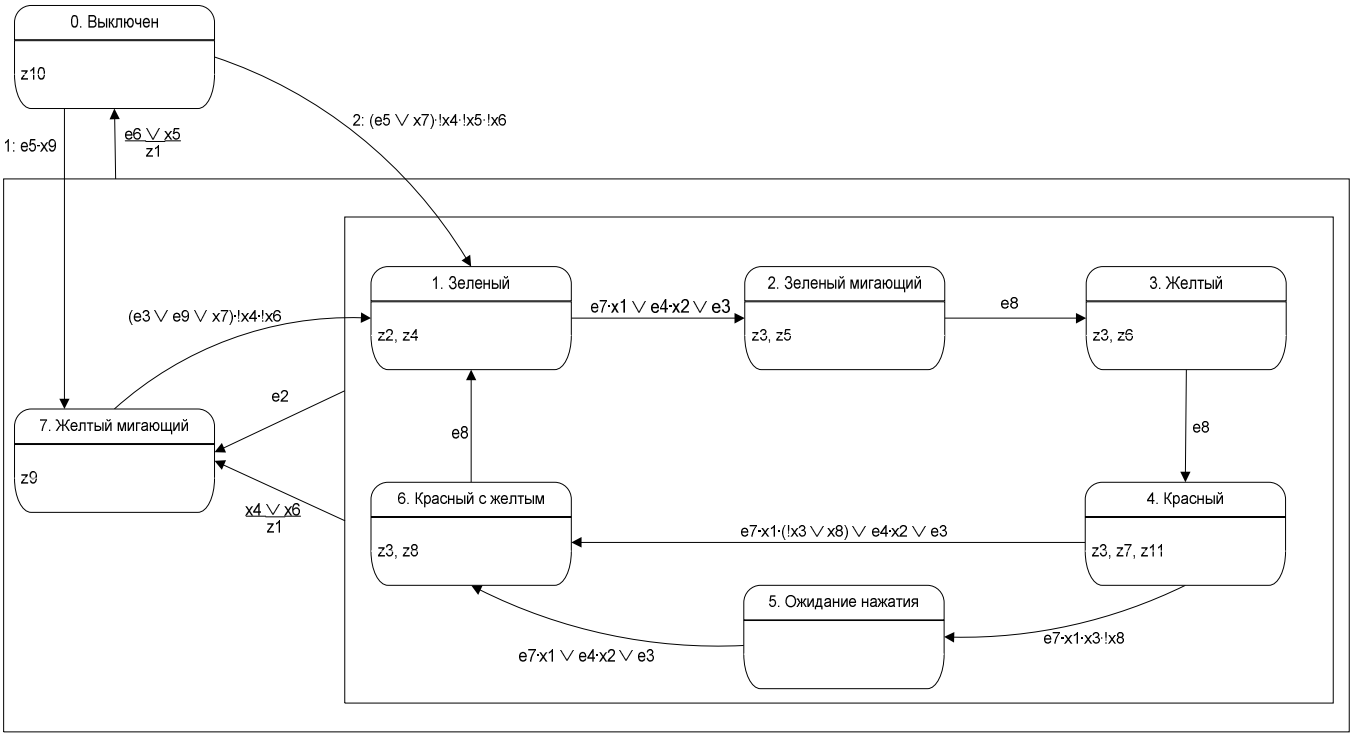

Рис. 11. Граф переходов автомата управления светофором

13. Протоколирование

Предлагаемый подход позволяет реализовать функцию протоколирования, которая поддерживается непосредственно паттерном.

Пример протокола:

```
I x7 Аварийное мигание NO
Перешли в состояние '0. Выключен'
I x4 Красный перегорел NO
I x5 Желтый перегорел NO
I x6 Зеленый перегорел NO
O z2 Установить 'долгий' таймер
O z4 Включить зеленый
Перешли в состояние '1. Зеленый'
I x1 Автоматический режим NO
I x2 Ручной режим YES
O z3 Установить 'короткий' таймер
O z5 Включить зеленый мигающий
Перешли в состояние '2. Зеленый мигающий'
O z3 Установить 'короткий' таймер
O z6 Включить желтый
Перешли в состояние '3. Желтый'
I x4 Красный перегорел NO
I x6 Зеленый перегорел NO
I x5 Желтый перегорел NO
Перешли в состояние '3. Желтый'
O z3 Установить 'короткий' таймер
O z7 Включить красный
O z11 Переход по кнопке
Перешли в состояние '4. Красный'
I x2 Ручной режим YES
O z3 Установить 'короткий' таймер
O z8 Включить красный и желтый
Перешли в состояние '6. Красный с желтым'
O z2 Установить 'долгий' таймер
O z4 Включить зеленый
Перешли в состояние '1. Зеленый'
I x1 Автоматический режим NO
I x2 Ручной режим YES
O z3 Установить 'короткий' таймер
O z5 Включить зеленый мигающий
Перешли в состояние '2. Зеленый мигающий'
O z3 Установить 'короткий' таймер
O z6 Включить желтый
Перешли в состояние '3. Желтый'
I x4 Красный перегорел NO
I x6 Зеленый перегорел NO
I x5 Желтый перегорел NO
Перешли в состояние '3. Желтый'
I x4 Красный перегорел YES
O z1 Аварийное состояние
O z9 Включить желтый мигающий
Перешли в состояние '7. Желтый мигающий'
I x7 Аварийное мигание YES
I x4 Красный перегорел YES
I x5 Желтый перегорел NO
Перешли в состояние '7. Желтый мигающий'
I x7 Аварийное мигание YES
I x4 Красный перегорел YES
I x5 Желтый перегорел NO
Перешли в состояние '7. Желтый мигающий'
I x7 Аварийное мигание YES
I x4 Красный перегорел NO
I x6 Зеленый перегорел NO
O z2 Установить 'долгий' таймер
O z4 Включить зеленый
Перешли в состояние '1. Зеленый'
I x2 Ручной режим YES
O z3 Установить 'короткий' таймер
O z5 Включить зеленый мигающий
Перешли в состояние '2. Зеленый мигающий'
I x4 Красный перегорел NO
I x6 Зеленый перегорел NO
```

```
I x5 Желтый перегорел NO
Перешли в состояние '2. Зеленый мигающий'
O z3 Установить 'короткий' таймер
O z6 Включить желтый
Перешли в состояние '3. Желтый'
O z3 Установить 'короткий' таймер
O z7 Включить красный
O z11 Переход по кнопке
Перешли в состояние '4. Красный'
```

Заключение

Предлагаемый подход кажется весьма громоздким. Однако, он имеет существенное достоинство – автоматы, построенные подобным образом, могут динамически менять свою конфигурацию.

Литература

1. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.
2. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
3. *Буч Г., Рамбо Д., Джекобсон А.* UML. Специальный справочник. СПб.: Питер, 2002.
4. *Дистель А.А., Кобак Д.А., Шалыто А.А.* Система управления дорожным светофором. Проектная документация. http://is.ifmo.ru

Приложение. Исходный код системы управления светофором

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Diagnostics;
using System.IO;
using AutomataPattern;


namespace MenuExample
{
	// Визуализатор светофора
	public class Form1 : System.Windows.Forms.Form
	{
		private System.Windows.Forms.Panel panel1;
		private Graphics _graphics;
		private System.Windows.Forms.Button button9;
		private System.Windows.Forms.Button button8;
		private System.Windows.Forms.Button button7;
		private System.Windows.Forms.Button button6;
		private System.Windows.Forms.Button button5;
		private System.Windows.Forms.Button button4;
		private System.Windows.Forms.Label label2;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.GroupBox groupBox1;
		private System.Windows.Forms.Label label7;
		private System.Windows.Forms.Label label6;
		private System.Windows.Forms.Label label5;
		private System.Windows.Forms.Label label4;
		private System.Windows.Forms.Label label3;
		private System.Windows.Forms.TrackBar trackBar2;
		private System.Windows.Forms.TrackBar trackBar1;
		private System.Windows.Forms.Button button3;
		private System.Windows.Forms.Button button2;
		private System.Windows.Forms.GroupBox groupBox2;
		private System.Windows.Forms.Button button1;
		private System.ComponentModel.IContainer components;
		private System.Windows.Forms.Timer blinkTimer;
		public System.Windows.Forms.Timer longTimer;
		public System.Windows.Forms.Timer shortTimer;
		protected Automata0 a0;
		protected Automata0.states currState;
		private bool _showLight;

		// Отображение модуля светофора
		public void DrawElement( int index, Brush brush )
		{
			int d = (panel1.Width > panel1.Height) ? panel1.Height / 3 : panel1.Width / 3;
				_graphics.FillEllipse( brush, panel1.Width / 2 - d / 2,
		(panel1.Height - d*3)/2 + d*index, d, d);
		}

		// Перерисовка светофора
		public void Redraw(Automata0.states what)
		{
			_graphics.FillRectangle( Brushes.AliceBlue, 0, 0, panel1.Width, panel1.Height);


			for (int i = 0; i < 3; i++)
				DrawElement(i, Brushes.Black);

			switch(what)
			{
				case Automata0.states.GREEN:
					_showLight = false;
					DrawElement(2, Brushes.Green);
					break;
```

```
                        case Automata0.states.BLINKGREEN:
                                if (_showLight)
                                        DrawElement(2, Brushes.Green);
                                else
                                        DrawElement(2, Brushes.Black);
                                blinkTimer.Start();
                                break;

                        case Automata0.states.BLINKYELLOW:
                                if (_showLight)
                                        DrawElement(1, Brushes.Yellow);
                                else
                                        DrawElement(1, Brushes.Black);
                                blinkTimer.Start();
                                break;

                        case Automata0.states.YELLOW:
                                _showLight = false;
                                DrawElement(1, Brushes.Yellow);
                                break;

                        case Automata0.states.RED:
                                _showLight = false;
                                DrawElement(0, Brushes.Red);
                                break;

                        case Automata0.states.REDYELLOW:
                                _showLight = false;
                                DrawElement(0, Brushes.Red);
                                DrawElement(1, Brushes.Yellow);
                                break;

                        case Automata0.states.OFF:
                                _showLight = false;
                                break;

                }
        }

        // Сменить состояние светофора
        public void ChangeLight(Automata0.states what)
        {
                currState = what;
                Redraw( what );
        }

        // Конструктор визулизатора
        public Form1()
        {
                // Инициализация протокола
                StreamWriter sw = new StreamWriter("test.log", false,
System.Text.Encoding.Default);

                sw.AutoFlush = true;

                TextWriterTraceListener myWriter = new
                        TextWriterTraceListener( sw );
                Trace.Listeners.Add(myWriter);

                // Инициализация начального состояния визуализатора
                currState = Automata0.states.OFF;

                InitializeComponent();

                _graphics = panel1.CreateGraphics();
                button2.BackColor = Color.Red;

                // Инстанционирование автомата
                a0 = new Automata0( null, this );
        }

        protected override void Dispose( bool disposing )
        {
                if( disposing )
                {
                        if (components != null)
                        {
                                components.Dispose();
                        }
                }
```

```
        base.Dispose( disposing );
}

// Генерируется автоматически
#region Windows Form Designer generated code
private void InitializeComponent()
{
        this.components = new System.ComponentModel.Container();
        this.panel1 = new System.Windows.Forms.Panel();
        this.button9 = new System.Windows.Forms.Button();
        this.button8 = new System.Windows.Forms.Button();
        this.button7 = new System.Windows.Forms.Button();
        this.button6 = new System.Windows.Forms.Button();
        this.button5 = new System.Windows.Forms.Button();
        this.button4 = new System.Windows.Forms.Button();
        this.label2 = new System.Windows.Forms.Label();
        this.label1 = new System.Windows.Forms.Label();
        this.groupBox1 = new System.Windows.Forms.GroupBox();
        this.label7 = new System.Windows.Forms.Label();
        this.label6 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.label3 = new System.Windows.Forms.Label();
        this.trackBar2 = new System.Windows.Forms.TrackBar();
        this.trackBar1 = new System.Windows.Forms.TrackBar();
        this.button3 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.groupBox2 = new System.Windows.Forms.GroupBox();
        this.button1 = new System.Windows.Forms.Button();
        this.blinkTimer = new System.Windows.Forms.Timer(this.components);
        this.longTimer = new System.Windows.Forms.Timer(this.components);
        this.shortTimer = new System.Windows.Forms.Timer(this.components);
        this.groupBox1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
        this.groupBox2.SuspendLayout();
        this.SuspendLayout();
        //
        // panel1
        //
        this.panel1.Location = new System.Drawing.Point(16, 16);
        this.panel1.Name = "panel1";
        this.panel1.Size = new System.Drawing.Size(168, 272);
        this.panel1.TabIndex = 0;
        //
        // button9
        //
        this.button9.Location = new System.Drawing.Point(232, 88);
        this.button9.Name = "button9";
        this.button9.Size = new System.Drawing.Size(64, 23);
        this.button9.TabIndex = 7;
        this.button9.Text = "Зеленый";
        this.button9.Click += new System.EventHandler(this.button9_Click);
        //
        // button8
        //
        this.button8.Location = new System.Drawing.Point(232, 56);
        this.button8.Name = "button8";
        this.button8.Size = new System.Drawing.Size(64, 23);
        this.button8.TabIndex = 6;
        this.button8.Text = "Желтый";
        this.button8.Click += new System.EventHandler(this.button8_Click);
        //
        // button7
        //
        this.button7.Location = new System.Drawing.Point(232, 24);
        this.button7.Name = "button7";
        this.button7.Size = new System.Drawing.Size(64, 23);
        this.button7.TabIndex = 5;
        this.button7.Text = "Красный";
        this.button7.Click += new System.EventHandler(this.button7_Click);
        //
        // button6
        //
        this.button6.Location = new System.Drawing.Point(88, 88);
        this.button6.Name = "button6";
        this.button6.Size = new System.Drawing.Size(64, 23);
        this.button6.TabIndex = 4;
        this.button6.Text = "Зеленый";
        this.button6.Click += new System.EventHandler(this.button6_Click);
```

```
//
// button5
//
this.button5.Location = new System.Drawing.Point(88, 56);
this.button5.Name = "button5";
this.button5.Size = new System.Drawing.Size(64, 23);
this.button5.TabIndex = 3;
this.button5.Text = "Желтый";
this.button5.Click += new System.EventHandler(this.button5_Click);
//
// button4
//
this.button4.Location = new System.Drawing.Point(88, 24);
this.button4.Name = "button4";
this.button4.Size = new System.Drawing.Size(64, 23);
this.button4.TabIndex = 2;
this.button4.Text = "Красный";
this.button4.Click += new System.EventHandler(this.button4_Click);
//
// label2
//
this.label2.Location = new System.Drawing.Point(168, 56);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(56, 23);
this.label2.TabIndex = 1;
this.label2.Text = "Починили";
//
// label1
//
this.label1.Location = new System.Drawing.Point(8, 56);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(56, 23);
this.label1.TabIndex = 0;
this.label1.Text = "Сломали";
//
// groupBox1
//
this.groupBox1.Controls.Add(this.button9);
this.groupBox1.Controls.Add(this.button8);
this.groupBox1.Controls.Add(this.button7);
this.groupBox1.Controls.Add(this.button6);
this.groupBox1.Controls.Add(this.button5);
this.groupBox1.Controls.Add(this.button4);
this.groupBox1.Controls.Add(this.label2);
this.groupBox1.Controls.Add(this.label1);
this.groupBox1.Location = new System.Drawing.Point(224, 8);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(304, 120);
this.groupBox1.TabIndex = 2;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Хклиганы";
this.groupBox1.Enter += new System.EventHandler(this.groupBox1_Enter);
//
// label7
//
this.label7.Location = new System.Drawing.Point(48, 120);
this.label7.Name = "label7";
this.label7.TabIndex = 8;
this.label7.Text = "Выключить";
//
// label6
//
this.label6.Location = new System.Drawing.Point(48, 104);
this.label6.Name = "label6";
this.label6.TabIndex = 7;
this.label6.Text = "Включить";
//
// label5
//
this.label5.Location = new System.Drawing.Point(48, 56);
this.label5.Name = "label5";
this.label5.TabIndex = 6;
this.label5.Text = "Автоматический";
//
// label4
//
this.label4.Location = new System.Drawing.Point(48, 40);
this.label4.Name = "label4";
this.label4.TabIndex = 5;
this.label4.Text = "Ручной";
```

```
this.label4.Click += new System.EventHandler(this.label4_Click);
//
// label3
//
this.label3.Location = new System.Drawing.Point(48, 24);
this.label3.Name = "label3";
this.label3.TabIndex = 4;
this.label3.Text = "Мигалка";
//
// trackBar2
//
this.trackBar2.Location = new System.Drawing.Point(16, 96);
this.trackBar2.Maximum = 1;
this.trackBar2.Name = "trackBar2";
this.trackBar2.Orientation = System.Windows.Forms.Orientation.Vertical;
this.trackBar2.Size = new System.Drawing.Size(45, 40);
this.trackBar2.TabIndex = 3;
this.trackBar2.TickStyle = System.Windows.Forms.TickStyle.None;
this.trackBar2.Scroll += new System.EventHandler(this.trackBar2_Scroll);
//
// trackBar1
//
this.trackBar1.Location = new System.Drawing.Point(16, 24);
this.trackBar1.Maximum = 2;
this.trackBar1.Name = "trackBar1";
this.trackBar1.Orientation = System.Windows.Forms.Orientation.Vertical;
this.trackBar1.Size = new System.Drawing.Size(45, 48);
this.trackBar1.TabIndex = 2;
this.trackBar1.TickStyle = System.Windows.Forms.TickStyle.None;
this.trackBar1.Scroll += new System.EventHandler(this.trackBar1_Scroll);
//
// button3
//
this.button3.Location = new System.Drawing.Point(208, 56);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(88, 23);
this.button3.TabIndex = 1;
this.button3.Text = "Переключить!";
this.button3.Click += new System.EventHandler(this.button3_Click);
//
// button2
//
this.button2.Location = new System.Drawing.Point(216, 24);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(80, 23);
this.button2.TabIndex = 0;
this.button2.Text = "Питание";
this.button2.Click += new System.EventHandler(this.button2_Click);
//
// groupBox2
//
this.groupBox2.Controls.Add(this.label7);
this.groupBox2.Controls.Add(this.label6);
this.groupBox2.Controls.Add(this.label5);
this.groupBox2.Controls.Add(this.label4);
this.groupBox2.Controls.Add(this.label3);
this.groupBox2.Controls.Add(this.trackBar2);
this.groupBox2.Controls.Add(this.trackBar1);
this.groupBox2.Controls.Add(this.button3);
this.groupBox2.Controls.Add(this.button2);
this.groupBox2.Location = new System.Drawing.Point(224, 144);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(304, 152);
this.groupBox2.TabIndex = 3;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Пульт управления";
//
// button1
//
this.button1.Location = new System.Drawing.Point(200, 128);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(16, 16);
this.button1.TabIndex = 1;
this.button1.Text = "*";
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
            //
            // blinkTimer
            //
            this.blinkTimer.Interval = 1000;
            this.blinkTimer.Tick += new System.EventHandler(this.blinkTimer_Tick);
            //
            // longTimer
            //
            this.longTimer.Tick += new System.EventHandler(this.longTimer_Tick);
            //
            // shortTimer
            //
            this.shortTimer.Tick += new System.EventHandler(this.shortTimer_Tick);
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(568, 309);
            this.Controls.Add(this.groupBox2);
            this.Controls.Add(this.groupBox1);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.panel1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.Paint += new System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
            this.groupBox1.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            this.groupBox2.ResumeLayout(false);
            this.ResumeLayout(false);

    }
    #endregion


    [STAThread]
    static void Main()
    {
            Application.Run(new Form1());
    }

    private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
    {
    }

    // Нажатие на кнопку поломки красного света
    private void button4_Click(object sender, System.EventArgs e)
    {
            a0.brokenred();
    }

    // Нажатие на кнопку поломки жклтого света
    private void button5_Click(object sender, System.EventArgs e)
    {
            a0.brokenyellow();
    }

    // Нажатие на кнопку поломки зеленого света
    private void button6_Click(object sender, System.EventArgs e)
    {
            a0.brokengreen();
    }

    // Нажатие на кнопку починки красного света
    private void button7_Click(object sender, System.EventArgs e)
    {
            a0.fixedred();
    }

    // Нажатие на кнопку починки желтого света
    private void button8_Click(object sender, System.EventArgs e)
    {
            a0.fixedyellow();
    }

    // Нажатие на кнопку починки зеленого света
    private void button9_Click(object sender, System.EventArgs e)
    {
            a0.fixedgreen();
    }
```

```
        // Изменение положения бегунка выбора режима работы
        private void trackBar1_Scroll(object sender, System.EventArgs e)
        {
                switch ( ((TrackBar)sender).Value )
                {
                        case 0 :
                                a0.auto_mode();
                                break;

                        case 1 :
                                a0.manual_mode();
                                break;

                        case 2 :
                                a0.blinking_mode();
                                break;
                }
        }

        // Нажатие на кнопку “Питание”
        private void button2_Click(object sender, System.EventArgs e)
        {
                a0.power_button();
                if (a0.on)
                        button2.BackColor = Color.Green;
                else
                        button2.BackColor = Color.Red;
        }

        // Нажатие на кнопку “Переключить”
        private void button3_Click(object sender, System.EventArgs e)
        {
                a0.switch_lights();
        }

        // Изменение положения бегунка включения/выключения кнопки
        private void trackBar2_Scroll(object sender, System.EventArgs e)
        {
                a0.switch_button();
        }

        // Нажатие на кнопку “*”
        private void button1_Click(object sender, System.EventArgs e)
        {
                a0.buttonpressed();
        }

        // Срабатывание “длинного” таймера
        private void longTimer_Tick(object sender, System.EventArgs e)
        {
                a0.long_time_elapsed();
        }

        // Срабатывание “короткого” таймера
        private void shortTimer_Tick(object sender, System.EventArgs e)
        {
                a0.short_time_elapsed();
        }

        // Срабатывание таймера мигания света
        private void blinkTimer_Tick(object sender, System.EventArgs e)
        {
                _showLight = !_showLight;
                Redraw( currState );
        }

        private void label4_Click(object sender, System.EventArgs e)
        {

        }

        private void groupBox1_Enter(object sender, System.EventArgs e)
        {

        }
}
```

```
// Ведение протокола

public class Logger
{
	public static void log_input(int n, string s, bool res)
	{
		Trace.WriteLine(String.Format("I x{0} {1} {2}", n, s, res ? "YES" : "NO" ), "State
		entry");
	}
	public static void log_output(int n, string s)
	{
		Trace.WriteLine(String.Format("O z{0} {1}", n, s), "State entry");

	}
	public static void log_string( string toLog )
	{
		Trace.WriteLine( toLog );
	}
}

// Формирование входных перенных

public class SupplyFuncs
{
	private Automata automata;
	public SupplyFuncs( Automata a )
	{
		automata = a;
	}

	public static bool x1(Automata automata)
	{
		bool res = ((Automata0)automata).cur_mode == Automata0.modes.AUTO;
			Logger.log_input(1, "Автоматический режим", res);
		return res;
	}

	public static bool x2(Automata automata)
	{
		bool res = ((Automata0)automata).cur_mode == Automata0.modes.MANUAL;
			Logger.log_input(2, "Ручной режим", res);
		return res;
	}

	public static bool x3(Automata automata)
	{
		bool res = ((Automata0)automata).button_on;
			Logger.log_input(3, "Кнопка включена", res);
		return res;
	}

	public static bool x4(Automata automata)
	{
		bool res = ((Automata0)automata).broken_red;
			Logger.log_input(4, "Красный перегорел", res);
		return res;
	}

	public static bool x5(Automata automata)
	{
		bool res = ((Automata0)automata).broken_yellow;
			Logger.log_input(5, "Желтый перегорел", res);
		return res;
	}

	public static bool x6(Automata automata)
	{
		bool res = ((Automata0)automata).broken_green;
			Logger.log_input(6, "Зеленый перегорел", res);
		return res;
	}

	public static bool x7(Automata automata)
	{
		bool res = ((Automata0)automata).broken;
			Logger.log_input(7, "Аварийное мигание", res);
		return res;
	}

	public static bool x8(Automata automata)
```

```
		{
			bool res = ((Automata0)automata).button_pressed;
				Logger.log_input(8, "Нажата кнопка 'переход'", res);
			return res;
		}

		public static bool x9(Automata automata)
		{
			bool res = ((Automata0)automata).cur_mode == Automata0.modes.BLINKING;
				Logger.log_input(9, "Режим мигалки", res);
			return res;
		}
	}

	// Классы, реализующие условия переходов

	public class Cond1 : Condition
	{
		public Cond1( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			return ((e == 5) || SupplyFuncs.x7(this._switch)) && !SupplyFuncs.x4(this._switch)
&& !SupplyFuncs.x5(this._switch) && !SupplyFuncs.x6(this._switch) && !SupplyFuncs.x9(this._switch);
		}
	}

	public class Cond2 : Condition
	{
		public Cond2( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			return (e == 5) && ( SupplyFuncs.x9(this._switch) || SupplyFuncs.x4(this._switch)
|| SupplyFuncs.x6(this._switch));
		}
	}
	public class Cond3 : Condition
	{
		public Cond3( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			return (e == 6) || SupplyFuncs.x5(this._switch);
		}
	}

	public class Cond4 : Condition
	{
		public Cond4( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			bool res = ( (e == 3) || (e == 9) || (SupplyFuncs.x7(this._switch) &&
!SupplyFuncs.x9(this._switch))  ) && !SupplyFuncs.x4(this._switch) && !SupplyFuncs.x6(this._switch);
			return res;
		}
	}

	public class Cond5 : Condition
	{
		public Cond5( Automata sw ) : base( sw )
		{
		}

		public override bool Test( int e )
		{
			return ( ( e == 7 ) && SupplyFuncs.x1(this._switch) ) || ( ( e == 4 ) &&
SupplyFuncs.x2(this._switch) ) || ( e == 3 );
		}
	}
```

```
        public class Cond6 : Condition
        {
                public Cond6( Automata sw ) : base( sw )
                {
                }

                public override bool Test( int e )
                {
                        return e == 8;
                }
        }

        public class Cond7 : Condition
        {
                public Cond7( Automata sw ) : base( sw )
                {
                }

                public override bool Test( int e )
                {
                        return ( e == 7 ) && (SupplyFuncs.x1(this._switch)) &&

(!SupplyFuncs.x3(this._switch) || SupplyFuncs.x3(this._switch) && SupplyFuncs.x8(this._switch)) ||

 (e == 4) && SupplyFuncs.x2(this._switch) || (e == 3);
                }
        }

        public class Cond8 : Condition
        {
                public Cond8( Automata sw ) : base( sw )
                {

                }

                public override bool Test( int e )
                {
                        return ( e == 7 ) && SupplyFuncs.x1(this._switch) && SupplyFuncs.x3(this._switch)

                        && !SupplyFuncs.x8(this._switch);
                }
        }


        public class Cond9 : Condition
        {
                public Cond9( Automata sw ) : base( sw )
                {
                }

                public override bool Test( int e )
                {
                        return (( e == 1 ) && SupplyFuncs.x1(this._switch)) || ( (e == 4) &&

                        SupplyFuncs.x2(this._switch) ) || (e == 3);
                }
        }

        public class Cond10 : Condition
        {
                public Cond10( Automata sw ) : base( sw )
                {
                }

                public override bool Test( int e )
                {
                        return SupplyFuncs.x4(this._switch) || SupplyFuncs.x6(this._switch);
                }
        }

        public class Cond11 : Condition
        {
                public Cond11( Automata sw ) : base( sw )
                {
                }

                public override bool Test( int e )
                {
                        return e == 2;
                }
```

```
}

// Классы, реализующие выходные воздействия

public class ActionZ1 : Action
{
	private Automata _a;
	public ActionZ1( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).broken = true;
			Logger.log_output(1, "Аварийное состояние");

	}
}

public class ActionZ2 : Action
{
	private Automata _a;
	public ActionZ2( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Form1)((Automata0)_a).Args).longTimer.Interval = 10000;
		((Form1)((Automata0)_a).Args).longTimer.Start();
		Logger.log_output(2, "Установить 'долгий' таймер");
	}
}

public class ActionZ3 : Action
{
	private Automata _a;
	public ActionZ3( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Form1)((Automata0)_a).Args).shortTimer.Interval = 6000;
		((Form1)((Automata0)_a).Args).shortTimer.Start();
		Logger.log_output(3, "Установить 'короткий' таймер");
	}
}

public class ActionZ4 : Action
{
	private Automata _a;
	public ActionZ4( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.GREEN);
			Logger.log_output(4, "Включить зеленый");
	}
}

public class ActionZ5 : Action
{
	private Automata _a;
	public ActionZ5( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.BLINKGREEN);
		Logger.log_output(5, "Включить зеленый мигающий");
	}
```

```
}

public class ActionZ6 : Action
{
	private Automata _a;
	public ActionZ6( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.YELLOW);
			Logger.log_output(6, "Включить желтый");
	}
}

public class ActionZ7 : Action
{
	private Automata _a;
	public ActionZ7( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.RED);
			Logger.log_output(7, "Включить красный");
	}
}

public class ActionZ8 : Action
{
	private Automata _a;
	public ActionZ8( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.REDYELLOW);
			Logger.log_output(8, "Включить красный и желтый");
	}
}

public class ActionZ9 : Action
{
	private Automata _a;
	public ActionZ9( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.BLINKYELLOW);
			Logger.log_output(9, "Включить желтый мигающий");
	}
}

public class ActionZ10 : Action
{
	private Automata _a;
	public ActionZ10( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).light(Automata0.states.OFF);
			Logger.log_output(10, "Погасить огни");
	}
}
```

```
public class ActionZ11 : Action
{
	private Automata _a;
	public ActionZ11( Automata a ) : base( a )
	{
		_a = a;
	}

	public override void Do()
	{
		((Automata0)_a).button_pressed = false;
			Logger.log_output(11, "Переход по кнопке");
	}
}


// Класс “Состояние”

public class State0 : State
{
	private string _id;
	private Automata0 _a;
	public State0( Automata0 a, string id ) : base(a)
	{
		_id = id;
		_a = a;
	}

	public override void Log()
	{
		Logger.log_string( "Вошли в состояние '"+_id+"'" );
	}
}


// Класс “Автомат”

public class Automata0 : Automata
{

	public State s0, s1, s2, s3, s4, s5, s6, s7, s8;

	public Automata0( State initialState, object o ) : base( initialState, o )
	{
		on = false;

		LOGGING = true;
		cur_mode = modes.AUTO;
		button_on = false;
		broken_red = false;
		broken_yellow = false;
		broken_green = false;
		broken = false;
		button_pressed = false;

		// Создание экземаляров класса State0
		s0 = new State0( this, "0. Выключен" );
		s1 = new State0( this, "1. Зеленый" );
		s2 = new State0( this, "2. Зеленый мигающий"  );
		s3 = new State0( this, "3. Желтый"  );
		s4 = new State0( this, "4. Красный"  );
		s5 = new State0( this, "5. Ожидание нажатия"  );
		s6 = new State0( this, "6. Красный с желтым"  );
		s7 = new State0( this, "7. Желтый мигающий"  );

		// Инициализация
		State = s0;

		// “Размещение” выходных воздействий в стостояниях
		s0.AddAction( new ActionZ10( this ) );

		s1.AddAction( new ActionZ2( this ) );
		s1.AddAction( new ActionZ4( this ) );

		s2.AddAction( new ActionZ3( this ) );
		s2.AddAction( new ActionZ5( this ) );

		s3.AddAction( new ActionZ3( this ) );
		s3.AddAction( new ActionZ6( this ) );
```

```
        s4.AddAction( new ActionZ3( this ) );
        s4.AddAction( new ActionZ7( this ) );
        s4.AddAction( new ActionZ11( this ) );

        s6.AddAction( new ActionZ3( this ) );
        s6.AddAction( new ActionZ8( this ) );

        s7.AddAction( new ActionZ9( this ) );

        // Формирование переходов
        Transitions.Add(s0, s1, new Cond1( this ), (Action) null);
        Transitions.Add(s1, s2, new Cond5( this ), (Action) null);
        Transitions.Add(s2, s3, new Cond6( this ), (Action) null);
        Transitions.Add(s3, s4, new Cond6( this ), (Action) null);
        Transitions.Add(s4, s6, new Cond7( this ), (Action) null);
        Transitions.Add(s4, s5, new Cond8( this ), (Action) null);
        Transitions.Add(s5, s6, new Cond9( this ), (Action) null);
        Transitions.Add(s6, s1, new Cond6( this ), (Action) null);
        Transitions.Add(s1, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s1, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s2, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s2, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s3, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s3, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s4, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s4, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s5, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s5, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s6, s7, new Cond11( this ), (Action) null);
        Transitions.Add(s6, s7, new Cond10( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s7, s1, new Cond4( this ), (Action) null);
        Transitions.Add(s0, s7, new Cond2( this ), (Action) null);

        Transitions.Add(s1, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s2, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s3, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s4, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s5, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s6, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
        Transitions.Add(s7, s0, new Cond3( this ), (Action) new ActionZ1( this ) );
}


// Формирование событий

// Включение автоматического режима (e3)
public void auto_mode()
{
        Execute(3);
        cur_mode = modes.AUTO;
}

// Включение ручного режима (e9)
public void manual_mode()
{
        Execute(9);
        cur_mode = modes.MANUAL;
}

// Включение режима мигания (e2)
public void blinking_mode()
{
        cur_mode = modes.BLINKING;
        Execute(2);
}


public void switch_button()
{
        button_on = !button_on;

}
```

```
// Включение/выключение светофора (e5, e6)
public void power_button()
{
	if(on)
	{
		on = false;
		Execute(6);
	}
	else
	{
		on = true;
		Execute(5);
	}
}

// Переключение света (e4)
public void switch_lights()
{
	Execute(4);
}

// Сломан красный
public void brokenred()
{
	broken_red = true;
	Execute(0);
}

// Сломан зеленый
public void brokengreen()
{
	broken_green = true;
	Execute(0);
}

// Сломан желтый
public void brokenyellow()
{
	broken_yellow = true;
	Execute(0);
}

// Починен красный
public void fixedred()
{
	broken_red = false;
	Execute(0);
}

// Починен зеленый
public void fixedgreen()
{
	broken_green = false;
	Execute(0);
}

// Починен желтый
public void fixedyellow()
{
	broken_yellow = false;
	Execute(0);
}

// Нажата кнопка перехода (e1)
public void buttonpressed()
{
	button_pressed = true;
	Execute(1);
}

// Истекла задержка долгого таймера (e7)
public void long_time_elapsed()
{
	Execute(7);
}
```

```
		// Истекла задержка короткого таймера
		public void short_time_elapsed()
		{
			Execute(8);
		}

		public void light(states what)
		{
			((Form1)this.Args).ChangeLight(what);
		}

		public modes cur_mode;
		public bool    blink_mode;
		public bool    button_on;
		public bool    broken_red;
		public bool    broken_yellow;
		public bool    broken_green;
		public bool    broken;
		public bool    button_pressed;
		public int             y0;

		// Вспомогательные переменные
		public bool           LOGGING;
		public bool           on;

		// Возможные значения cur_mode
		public enum modes {AUTO, MANUAL, BLINKING};

		// Идентификаторы таймеров
		public enum timers {LTIMER=1, STIMER=2};

		public enum states
		{
			RED,
			YELLOW,
			GREEN,
			REDYELLOW,
			BLINKGREEN,
			BLINKYELLOW,
			OFF
		};


	}
}
```