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

Т.Г. Магомедов, А.Б. Островский

Моделирование работы «умного» светофора

Проектная документация

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

Санкт–Петербург
2006

Оглавление

Введение

В качестве темы проекта авторы выбрали моделирование работы светофора. Обычные светофоры не имеют обратной связи – они просто изменяют цвета сигналов через заранее заданные промежутки времени. Если машин на дороге нет, то пешеходы все равно должны ждать зеленого сигнала. Если нет пешеходов, то машины должны останавливаться на красный свет.

«Умный» светофор, моделируемый в этом проекте, экономит время, как пешеходов, так и водителей за счет отсутствия ненужных ожиданий.

Проект выполнен в среде разработки *Eclipse* с помощью инструментального средства *UniMod* на языке *Java*.

Все средства разработки (*Eclipse, UniMod, Java*), используемые в этом проекте, многоплатформенные. Проект первоначально был написан в операционной системе *Linux*, но затем был запущен в операционной системе *Windows*.

1. Постановка задачи

Как отмечено выше, в проекте моделируется пешеходный переход с «умным» светофором. Последовательность переключения сигналов **обычного** светофора упрощенно можно представить следующим образом:

- машины едут;
- все (машины и пешеходы) стоят;
- пешеходы идут;
- все (машины и пешеходы) стоят;
- цикл начинается сначала.

«Умный» светофор работает не совсем так. Он располагает сведениями о том, есть ли на дороге машины, а на тротуаре – пешеходы. При наличии пешеходов и машин «умный» светофор работает как обычный. Если же машин нет, но пешеходы стоят на переходе, им будет гореть зеленый свет, пока не появится, по крайней мере, одна машина. После этого светофор перейдет в режим обычного.

Если есть машины на дороге, но нет пешеходов, собирающихся переходить дорогу, то машинам будет гореть зеленый свет, пока не появится пешеход. После этого «умный» светофор перейдет в режим обычного.

Если нет ни машин, ни пешеходов, то зеленый горит машинам (табл. 1), так как они быстрее и им требуется больше времени, чтобы затормозить или начать движение.

Таблица 1. Логика поведения «умного» светофора

	Пешеходов нет	**Пешеходы есть**
Машин нет	Зеленый свет машинам	Зеленый свет пешеходам
Машины есть	Зеленый свет машинам	Обычный светофор

Интерфейс окна программы представлен на рис.1 для случая, когда добавлена одна машина и один пешеход. Пользователь может изменять ситуацию на дороге при помощи кнопок добавления пешеходов (*add pedestrian – bottom, top*) и машин (*add car – left, right*).

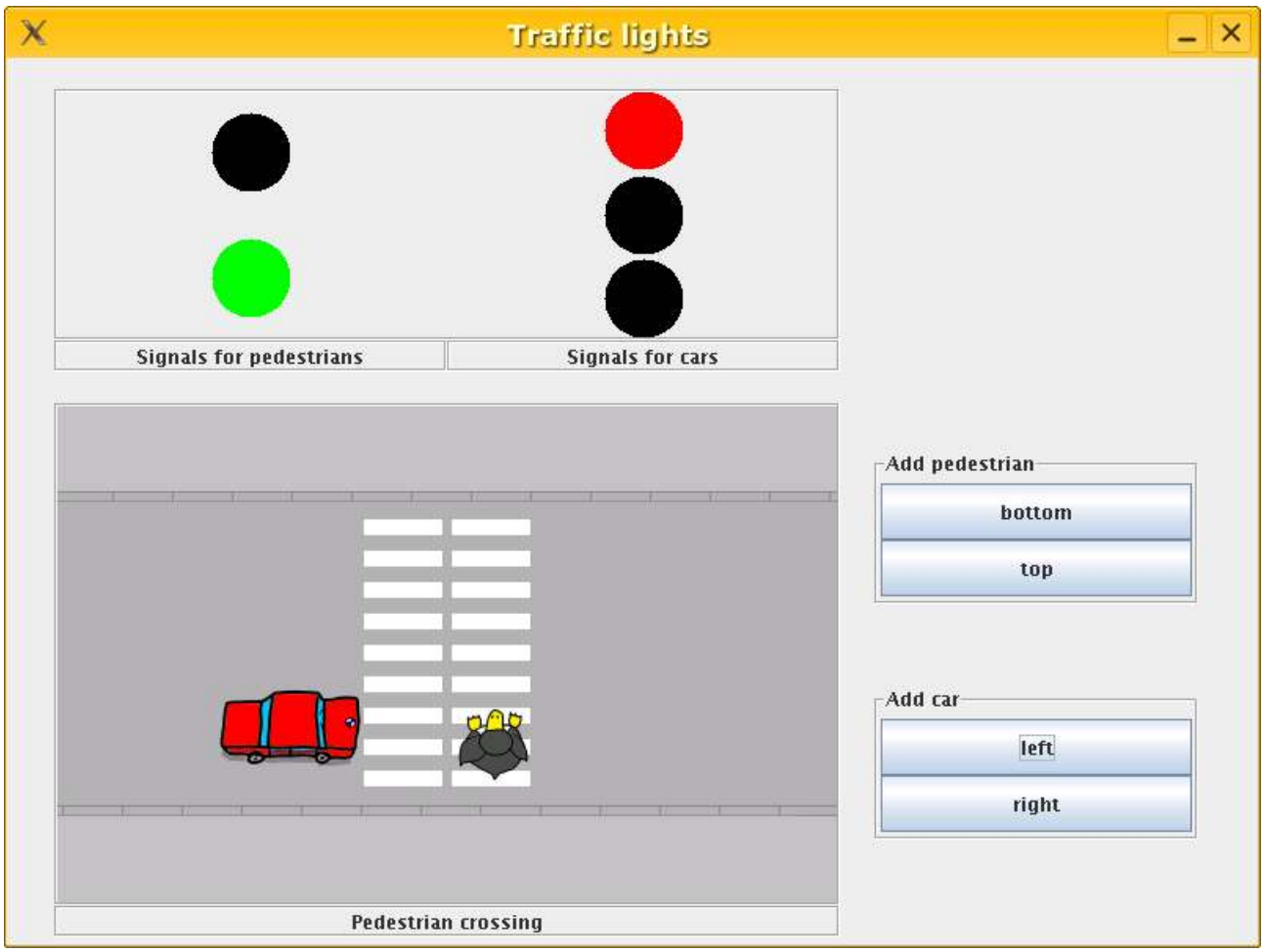

Рис.1. Окно программы

2. Схема связей, представленная в виде диаграммы классов

Схема связей, являющаяся диаграммой классов, представлена на рис. 2.

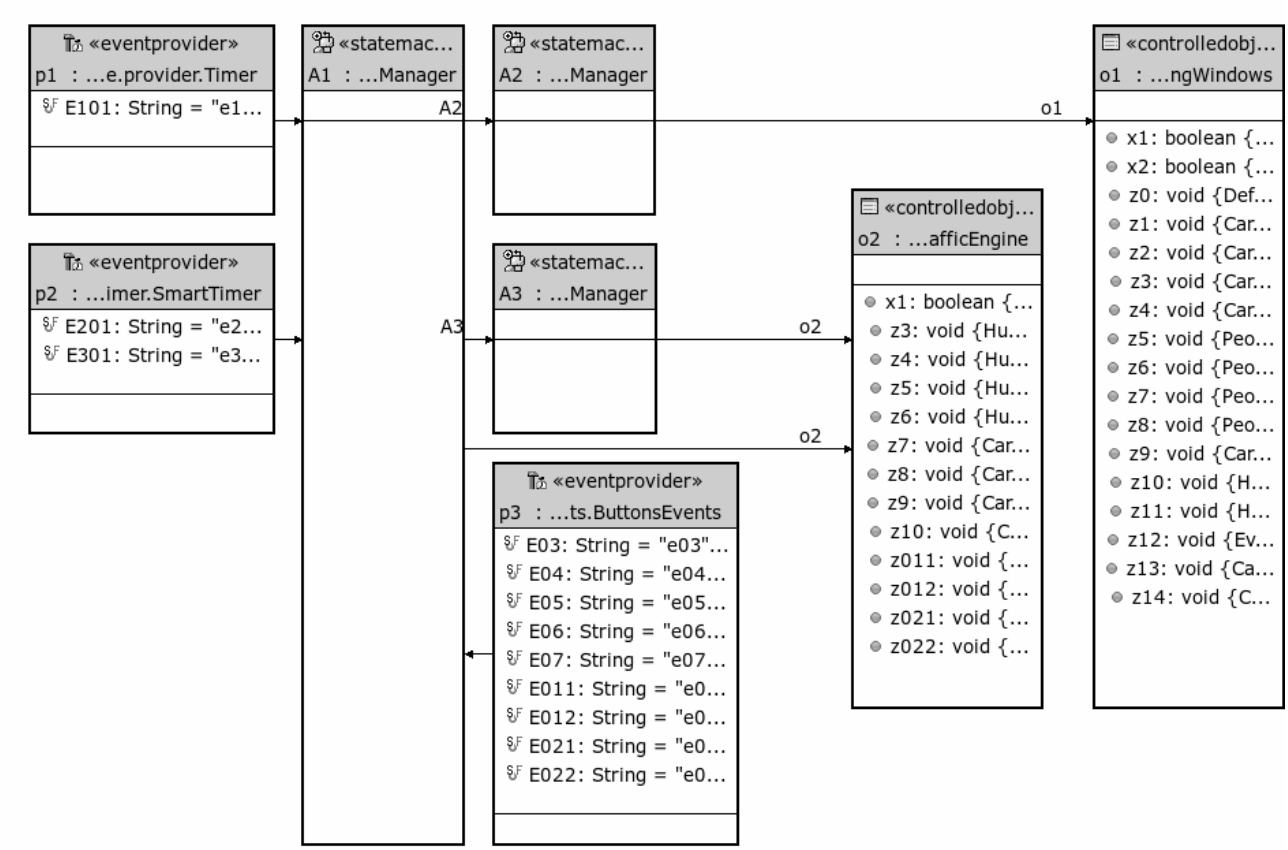

Рис.2. Диаграмма классов

Классы *p1*, *p2*, *p3* являются источниками событий. *A1, A2, A3* — автоматы. Автомат *A1* отвечает за события нажатий на кнопки, *A2* — за переключение сигналов светофора, *A3* — за движение пешеходов и машин. Классы *o1, o2* — объекты управления.

Каждый источник формирует события. Источник *p1* — это реализованный в инструментальном средстве *Unimod* таймер, посылающий события через равные промежутки времени. Источник *p2* — аналогичный ему таймер, который посылает два вида событий с разными интервалами. Источник *p3* формирует события, инициируемые нажатием кнопок, а также переключением сигналов светофора (табл. 2 – табл. 4).

Таблица 2. События источника событий *p1*

Событие	Описание
e101	Прошла секунда

Таблица 3. События источника событий *p2*

Событие	Описание
e201	Прошло пять секунд
e301	Прошла одна десятая секунды

Таблица 4. События источника событий *p3*

Событие	Описание
E03	Пешеходы должны начать идти
E04	Пешеходы должны начать торопиться
E05	Машины должны начать ехать
E06	Машины должны начать торопиться
E07	Все останавливаются
E011	Нажата кнопка «Добавить пешехода в “нижнюю” часть перехода»
E012	Нажата кнопка «Добавить пешехода в “верхнюю” часть перехода»
E021	Нажата кнопка «Добавить машину слева от перехода»
E022	Нажата кнопка «Добавить машину справа от перехода»

После нажатия кнопки добавления пешехода в нижнюю часть перехода, он появляется (рис.3).

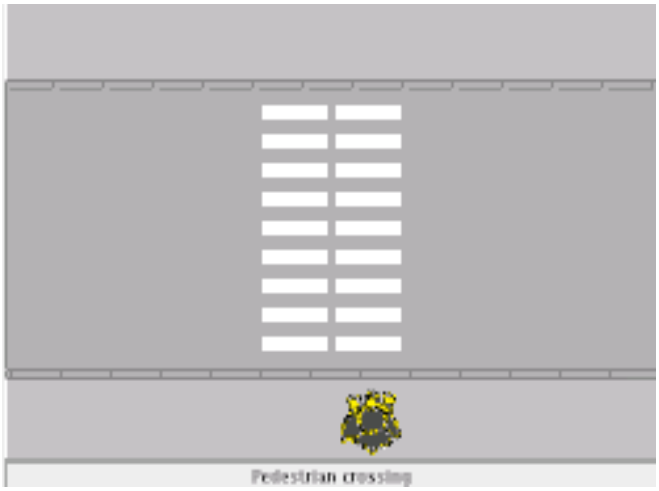

Рис. 3. Пешеход в «нижней» части перехода

После нажатия кнопки добавления пешехода в верхнюю часть перехода, он появляется (рис.4).

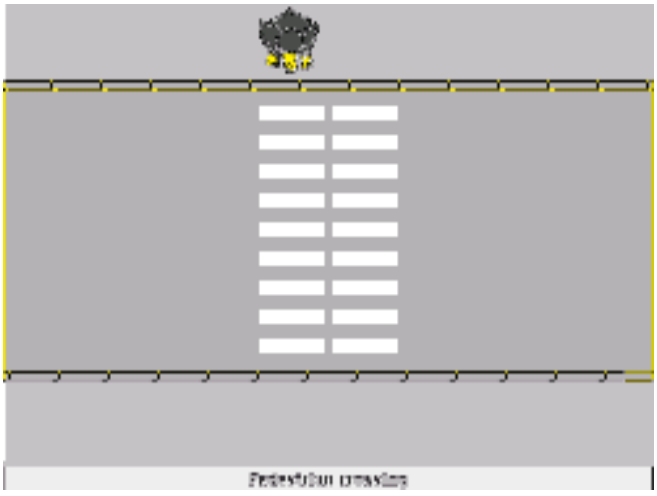

Рис. 4. Пешеход в «верхней» части перехода

Автомат *A1* получает события от всех источников – *p1*, *p2* и *p3*. Автомат *A1*

организован так, что автоматы *A2* и *A3* получают те же события, что и он, так как они вложены в автомат *A1*. Автоматы могут вызывать методы классов – объектов управления. Автомат *A2* вызывает методы объекта управления *o1*, а автоматы *A1* и *A3* — объекта управления *o2*.

3. Объекты управления

3.1. Объект управления графическим выводом *o1*

Входные и выходные воздействия объекта управления графическим выводом представлены в табл. 5.

Таблица 5. Входные и выходные воздействия объекта управления *o1*

Метод	Описание
x1	Есть ли пешеходы?
x2	Есть ли машины?
z0	Выставляются умолчания, открывается графическое окно вывода
z1	Гаснет зеленый свет машинам
z2	Загорается зеленый свет машинам
z3	Загорается желтый свет машинам
z4	Машинам гаснет желтый свет, загорается красный
z5	Пешеходам гаснет красный свет и включается зеленый
z6	Пешеходам загорается зеленый свет
z7	Пешеходам гаснет зеленый свет
z8	Пешеходам загорается красный, машинам – желтый и красный одновременно.
z9	Машинам загорается зеленый, желтый и красный гаснут
z10	Инициирует событие e03 (пешеходы могут идти)
z11	Инициирует событие e04 (пешеходам следует поторопиться)
z12	Инициирует событие e07 (пешеходам и машинам следует остановиться)
z13	Инициирует событие e05 (машины могут ехать)
z14	Инициирует событие e06 (машинам следует увеличить скорость)

3.2. Объект управления движением *o2*

Входные и выходные воздействия объекта управления движением представлены в табл. 6.

Таблица 6. Входные и выходные воздействия объекта управления *o2*

Метод	описание
x1	Ускорились ли пешеходы?
z3	Пешеходы делают шаг
z4	Пешеходы начинают ускоряться
z5	Пешеходы делают ускоренный шаг
z6	Пешеходы останавливаются
z7	Машины едут
z8	Машины едут и останавливаются перед пешеходным переходом.
z9	Машины начинают ускоряться
z10	Машины останавливаются
z011	Добавляется пешеход с нижней стороны пешеходного перехода
z012	Добавляется пешеход с верхней стороны пешеходного перехода
z021	Добавляется машина слева от перехода
z022	Добавляется машина справа от перехода

4. Автоматы

4.1. Главный автомат *A1*

4.1.1. Граф переходов

Граф переходов автомата *A1* представлен на рис. 5.

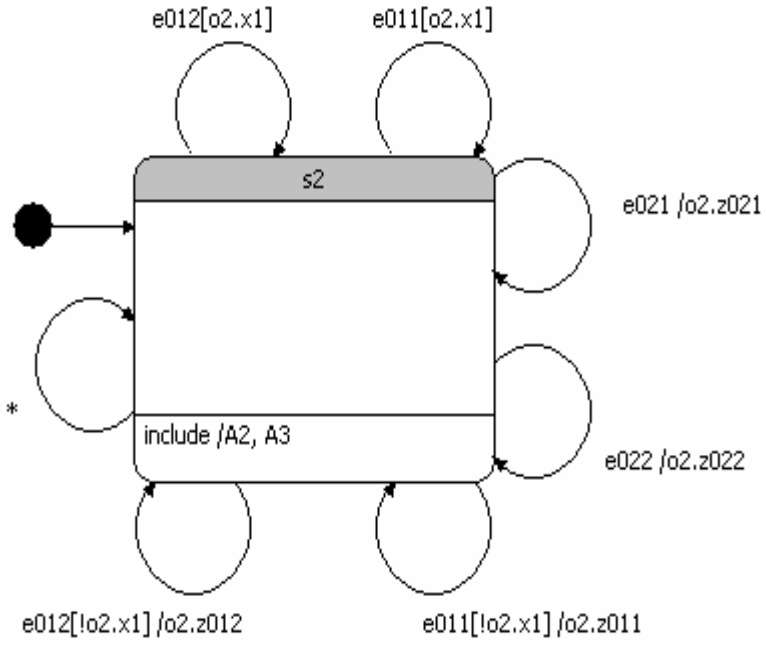

Рис. 5. Граф переходов автомата *A1*

4.1.2. Словесное описание

Автомат *A1* – главный автомат проекта. Ему передаются события от всех источников. У автомата *A1* одно состояние. В него вложены автоматы *A2* и *A3* – все события,

пришедшие автомату *A1*, передаются автоматам *A2* и *A3*. Петли обозначают запуск соответствующих методов объекта управления *o2* в ответ на события, связанные с нажатием на управляющие элементы (кнопки) графического интерфейса.

4.2. Автомат *A2* управления переключением светофора

4.2.1. Граф переходов

Граф переходов автомата *A2* представлен на рис. 6.

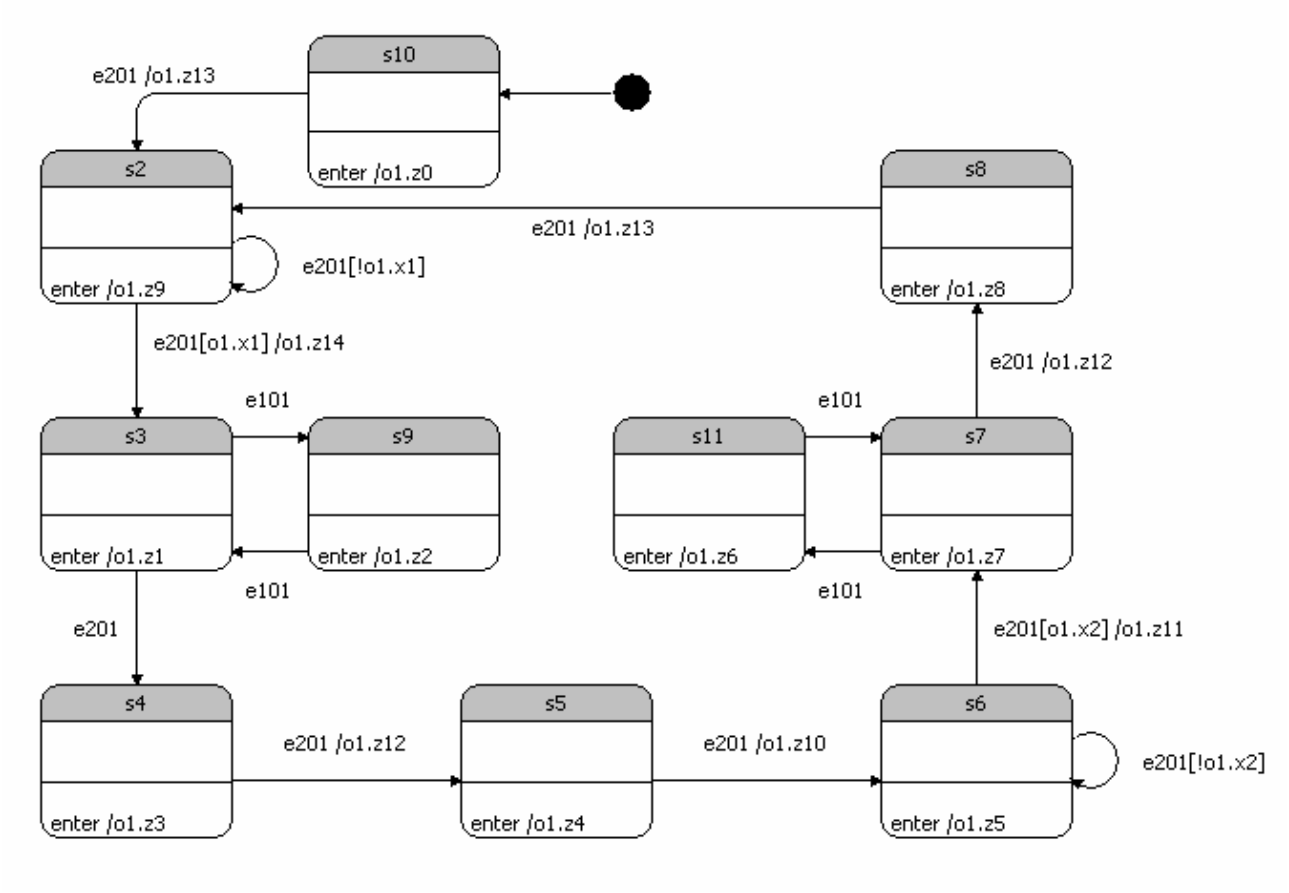

Рис. 6. Граф переходов автомата *A2*

4.2.2. Состояния автомата *A2*

Состояния автомата *A2* представлены в табл. 7.

Таблица 7. Состояния автомата *A2*

Состояние	Описание
s10	Начальное состояние автомата
s2	Машинам горит зеленый свет, пешеходам – красный
s3	Зеленый свет для машин гаснет
s9	Зеленый свет для машин загорается. Вместе с предыдущим состоянием реализует мигание светофора
s4	Для машин загорается желтый свет
s5	Для машин гаснет зеленый свет и загорается желтый
s6	Для пешеходов красный свет гаснет, зеленый загорается
s7	Для пешеходов зеленый свет гаснет
s11	Для пешеходов зеленый свет гаснет. Вместе с предыдущим состоянием реализует мигание зеленого света
s8	Пешеходам загорается красный, машинам – красный и желтый

4.2.3. Словесное описание

Граф переходов автомата содержит несколько циклов. Перед входом в цикл выполняется инициализация графического интерфейса в состоянии *s10*. Это действие необходимо выполнить один раз. Поэтому состояние *s10* не входит в цикл. Остальные состояния выбраны так, что каждое из них соответствует своей «конфигурации» светофора. Например, в состоянии *s2* машинам горит зеленый свет, пешеходам – красный. В состоянии *s6* машинам горит красный свет, а пешеходам – зеленый.

Переходы между состояниями выполняются по событиям таймера, задерживаясь на состоянии *s2*, если на дороге нет пешеходов, и состоянии *s6*, если на дороге нет машин. При входе в каждое состояние, которое помечено словом `enter`, запускается соответствующий метод объекта управления *o1*. Этот метод задает цвета сигналов светофора, соответствующие состоянию, из которого он вызван. На переходах вызываются выходные воздействия, посылающие события о том, что всё движение должно остановиться. При этом пешеходы могут идти, а машины – ехать, причем пешеходы или машины могут ускориться. Эти события примет автомат *A3*, управляющий движением пешеходов и машин по дороге.

4.3. Автомат *A3* управления движением пешеходов и машин по дороге

4.3.1. Граф переходов

Граф переходов автомата *A3* представлен на рис. 7.

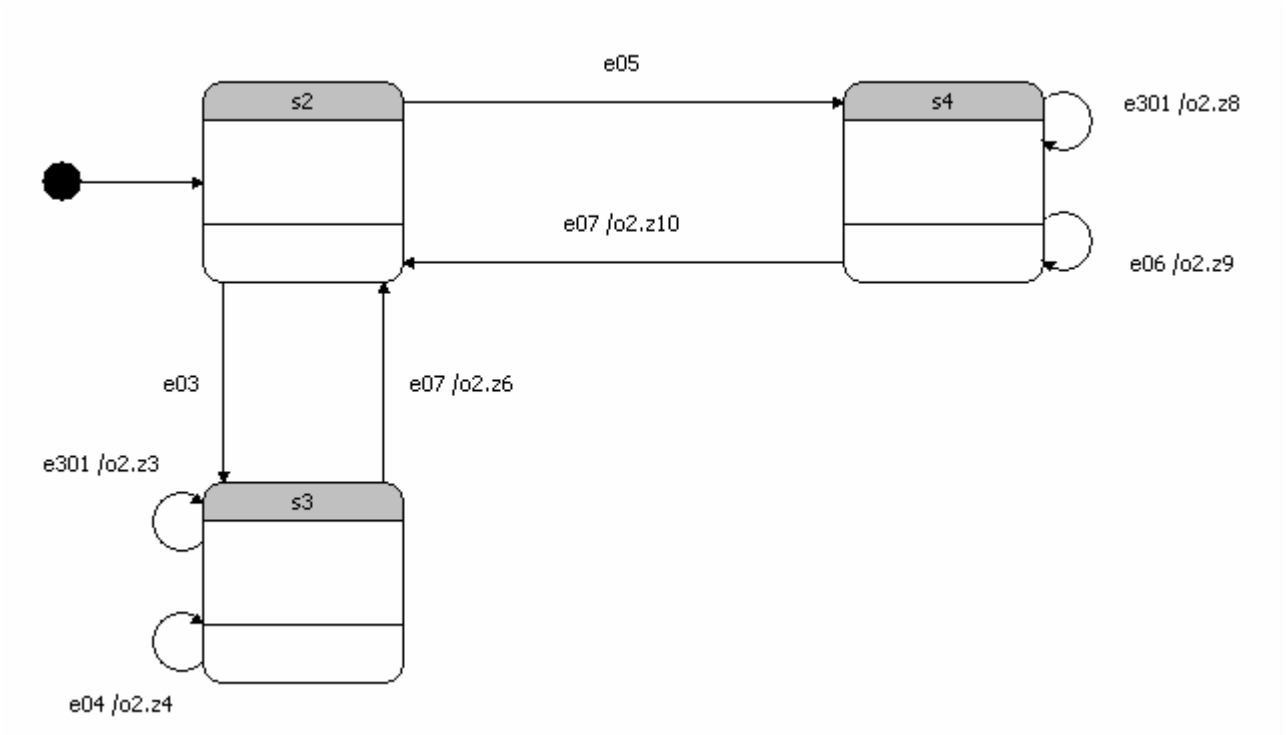

Рис. 7. Граф переходов автомата A3

4.3.2. Состояния автомата *A3*

Состояния автомата A2 представлены в табл. 8.

Таблица 8. Состояния автомата *A3*

Состояние	Описание
s2	Машины и пешеходы стоят
s3	Пешеходы переходят через дорогу
s4	Машины едут по дороге

4.3.3. Словесное описание

Этот автомат ответственен за моделирование движения через дорогу, управляемое сигналами светофора.

В состоянии *s2* и машины и пешеходы стоят на месте. По событию, которое было инициировано светофором, осуществляется переход в состояние *s3* (когда машины проезжают дорогу) или в состояние *s4* (когда пешеходы переходят через дорогу). В этих состояниях по «быстрому» таймеру выполняется перерисовка машин или пешеходов, и изменяются их координаты. По сигналу светофора о том, что следует поторопиться (начал мигать зеленый для пешеходов или машин), скорость пешеходов или машин увеличивается. Таким образом, после получения сигнала о том, что надо торопиться, изменение координат в методе *o2.z3* будет происходить быстрее.

5. Интерпретационный и компиляционный подходы

При реализации автоматов на основе интерпретационного подхода с использованием инструментального средства *Unimod* автоматы не преобразуются в *java*-код, а интерпретируются. Это удобно при написании программы, так как не надо думать о преобразовании автомата в код – он сам запускается на исполнение, как исходный код. При

этом классы – источники событий и объекты управления компилируются в *java* байт-код. Схема интерпретационного подхода представлена на рис. 8.

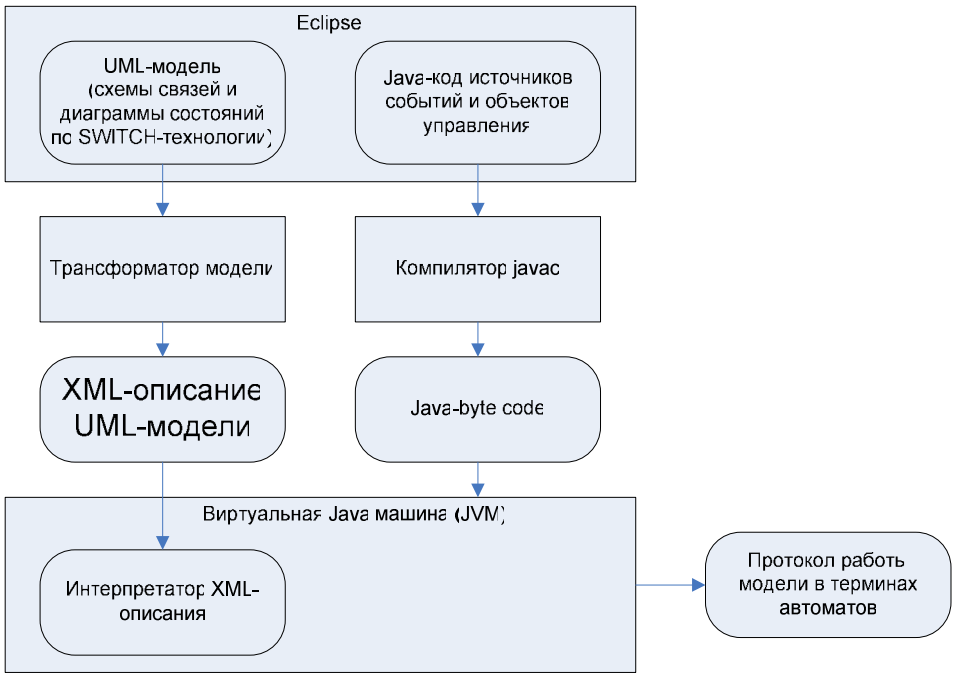

Рис 8. Структурная схема интерпретационного подхода

Интерпретатор по *XML*-описанию автомата имитирует его работу, запуская выходные воздействия уже как байт-код. При разработке программы не важно, как схемы связей и диаграммы состояний исполняются, но у интерпретационного подхода есть недостатки:

- низкое быстродействие;
- необходим интерпретатор;

Есть альтернатива интерпретационному подходу. По *XML*-описанию автомата можно построить изоморфный ему *java*-код, который потом будет скомпилирован в байт-код. Этот подход к запуску программы называется компиляционным. Схема компиляционного подхода представлена на рис. 9.

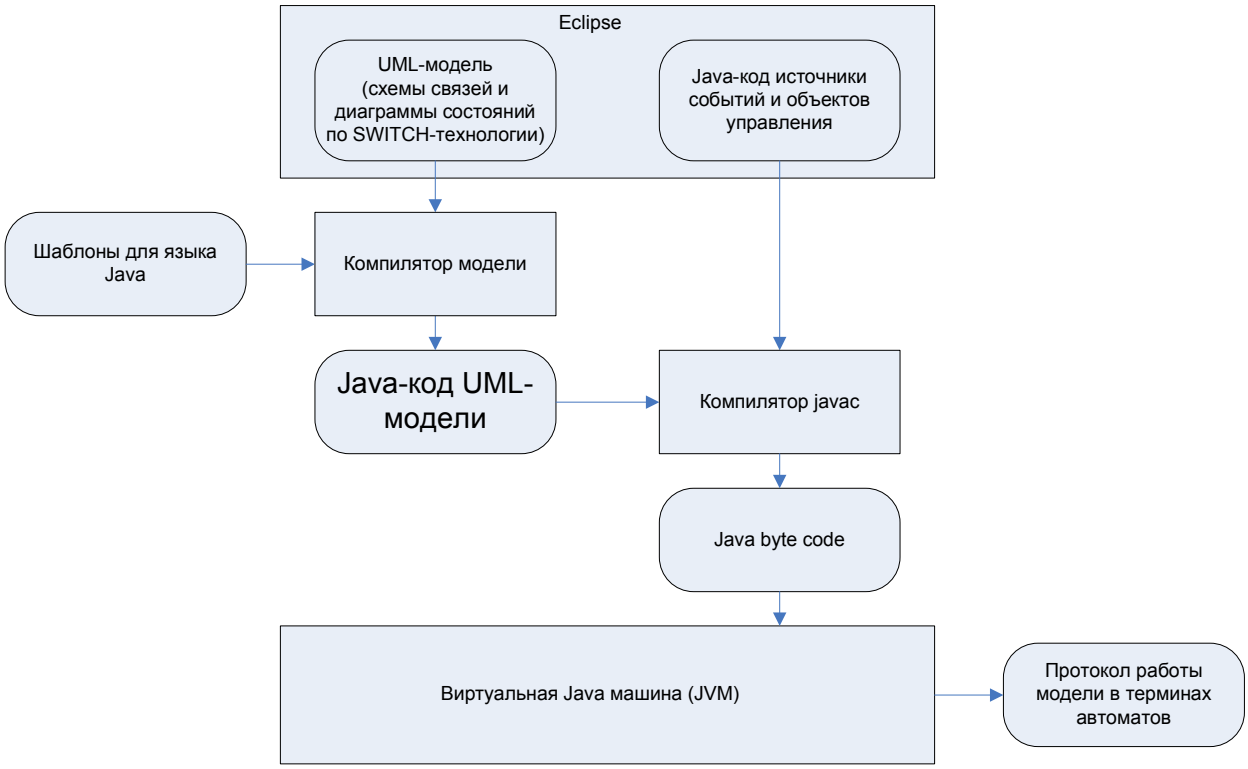

Рис. 9. Структурная схема компиляционного подхода

При компиляционном подходе *Unimod* генерирует исходный файл, соответствующий автомату `A1`. Вручную пишется java-файл, запускающий автомат. Необходимые программе классы источников событий, объектов управления, некоторые классы *Unimod*, файл, запускающий автомат компилируются и на выходе получается независимая программа, которую можно запустить всюду, где установлена *Java*.

Заключение

Описанная модель светофора в принципе пригодна для использования, если решить проблему с тем, как определить, есть ли у нас ожидающие пешеходы и машины.

Дальнейшее улучшение светофора может быть связано с зависимостью интервалов времени переключения светофора от отношения количества проезжающих машин к количеству пересекающих переход пешеходов, а так же касаться «взаимодействия» светофоров на перекрестке или соседних перекрестках.

Источники

1. http://is.ifmo.ru
2. *Шалыто А. А., Туккель Н. И.* SWITCH-технология — автоматный подход к созданию программного обеспечения и реактивных систем //Программирование. 2001. № 5, с.45–62. http://is.ifmo.ru/works/switch/1/
3. *Шалыто А. А.* Автоматно-ориентированное программирование //Материалы IX Всероссийской конференции по проблемам науки и высшей школы «Фундаментальные исследования в технических университетах». СПб.: изд-во Политехнического университета. 2005, с. 44–52. http://is.ifmo.ru/works/_politeh.pdf

Приложение 1. Исходные тексты программы, написанные вручную на языке *Java*

1. Main.java

```
import svet.co.DrawingWindows;
import svet.co.TrafficEngine;

import com.evelopers.unimod.adapter.standalone.provider.Timer;
import svetoforTimer.SmartTimer;
import FormEvents.ButtonsEvents;

import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.QueuedHandler;
import com.evelopers.unimod.runtime.ControlledObjectsManager;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.EventProvidersManager;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.common.exception.CommonException;


public class Main {

    public static void main(String[] args) {

        try {

            ModelEngine me = ModelEngine.createStandAlone(
                new QueuedHandler(),
                new Model1EventProcessor(),
                new ControlledObjectsManager() {
                    DrawingWindows o1 = new DrawingWindows();
                    TrafficEngine o2 = new TrafficEngine();
                    /*Sounds o3 = new Sounds();
                    LightDiode o4 = new LightDiode();
                    Commentator o5 = new Commentator();*/

                    public void init(ModelEngine engine) throws CommonException
{}
                    public void dispose() {}
                    public ControlledObject getControlledObject(String coName) {
                        if (coName.equals("o1")) { return o1; }
                        else if (coName.equals("o2")) { return o2; }
                        /*else if (coName.equals("o3")) { return o3; }
                        else if (coName.equals("o4")) { return o4; }
                        else if (coName.equals("o5")) { return o5; }*/
                        else {
                                System.out.println("No such object " + coName +
                                                   " (look Main.java)");
                                return null;
                        }
                    }
                },
                new EventProvidersManager() {
                    Timer p1 = new Timer();
                    SmartTimer p2 = new SmartTimer();
                    ButtonsEvents p3 = new ButtonsEvents();
```

```
                    public void init(ModelEngine engine) throws CommonException
{
                        p1.init(engine);
                        p2.init(engine);
                        p3.init(engine);
                    }
                    public void dispose() {
                        p1.dispose();
                        p2.dispose();
                        p3.dispose();
                    }
                    public EventProvider getEventProvider(String epName) {
                        if (epName.equals("p1")) { return p1; }
                        else if (epName.equals("p2")) { return p2; }
                        else { return p3; }
                }
            });

            me.start();

        } catch(CommonException e) {
            System.out.println(e.toString());
        }

    }

}
```

2. TrafficEngine.java

```
package svet.co;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;

public class TrafficEngine implements ControlledObject {
  public static boolean h_up, h_down, c_left, c_right, hurry, stopping;

  public static int yup, ydown, xleft, xright;

  /**
   *
   * @unimod.action.descr Add bottom human
   */
  public void z011(StateMachineContext context) {
    System.out.println("New human came from the bottom of the screen");
    h_up=true;
      DrawingWindows.sl.repaint(220, 145, 320, 460);

  }
  /**
   *
   * @unimod.action.descr Add top human
   */
  public void z012(StateMachineContext context) {
    System.out.println("New human came from the top of the screen");
    h_down=true;
    DrawingWindows.sl.repaint(220, 145, 320, 460);

  }

  /**
   *
   * @unimod.action.descr Add left car
   */
  public void z021(StateMachineContext context) {
    System.out.println("New car arrived from the left side of the road");
    c_right = true;
    DrawingWindows.sl.repaint();
  }

  /**
   *
   * @unimod.action.descr Add right car
   */
  public void z022(StateMachineContext context) {
    System.out.println("New car arrived from the right side of the road");
      c_left = true;
      DrawingWindows.sl.repaint();
  }
  /**
   *
   * @unimod.action.descr Humans step
   */
  public void z3(StateMachineContext context) {
    if (h_up || h_down) {
      byte v = 2;
      if (hurry)
        v = 4;
      if (h_up)
        yup += v;
```

```
    if (h_down)
      ydown += v;
    if (yup > 265) {
      h_up = false;
      yup = 0;
    }
    if (ydown > 265) {
      h_down = false;
      ydown = 0;
    }
    DrawingWindows.sl.repaint();
  }
}

/**
 * @unimod.action.descr Humans are beginning to make hurry
 */
public void z4(StateMachineContext context) {
  hurry = true;
}

/**
 *
 * @unimod.action.descr Humans run
 */
public void z5(StateMachineContext context) {
  if (h_up || h_down) {
    if (h_up)
      yup += 6;
    if (h_down)
      ydown += 6;
    if (yup > 265) {
      h_up = false;
      yup = 0;
    }
    if (ydown > 265) {
      h_down = false;
      ydown = 0;
    }
    DrawingWindows.sl.repaint();
  }
}

/**
 * @unimod.action.descr Humans stop
 */
public void z6(StateMachineContext context) {
  h_up = false;
  h_down = false;
  hurry = false;
  yup = 0;
  ydown = 0;
}

/**
 *
 * @unimod.action.descr Cars drive
 */
public void z7(StateMachineContext context) {
  if (c_left || c_right) {
    // byte v = 2;
    // if (hurry) v=4;
    if (c_left)
```

```
      xleft += 5;
    if (c_right)
      xright += 5;
    if (xright >= 410) {
      c_right = false;
      xright = 0;
    }
    if (xleft >= 410) {
      c_left = false;
      xleft = 0;
    }
    DrawingWindows.sl.repaint();
  }
}

/**
 *
 * @unimod.action.descr Cars drive and stop
 */
public void z8(StateMachineContext context) {
  if (c_left || c_right) {
    if (stopping) {
      if (c_left) {
        if (xleft <= 100)
          xleft = Math.min(xleft + 10, 100);
        if (xleft > 100)
          xleft += 10;
      }
      if (c_right) {
        if (xright <= 100)
          xright = Math.min(xright + 10, 100);
        if (xright > 100)
          xright += 10;
      }
    } else {
      if (c_left)
        xleft += 5;
      if (c_right)
        xright += 5;
    }
    if (xright >= 410) {
      c_right = false;
      xright = 0;
    }
    if (xleft >= 410) {
      c_left = false;
      xleft = 0;
    }
    DrawingWindows.sl.repaint();
  }
}

/**
 * @unimod.action.descr Cars are beginning to make hurry
 */
public void z9(StateMachineContext context) {
  stopping = true;
}

/**
 * @unimod.action.descr Cars stop
 */
public void z10(StateMachineContext context) {
```

```
    stopping = false;
  }

  /**
   * @unimod.action.descr humans are hurrying
   */
  public boolean x1(StateMachineContext context) {
    return TrafficEngine.hurry;
  }
}
```

ButtonsEvents.java

```
package FormEvents;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;

public class ButtonsEvents implements EventProvider {
  /**
   * @unimod.event.descr "Add bottom human" button is pressed
   */
  public static final String E011 = "e011";
  /**
   * @unimod.event.descr "Add top human" button is pressed
   */
  public static final String E012 = "e012";
  /**
   * @unimod.event.descr "Add left car" button is pressed
   */
  public static final String E021 = "e021";
  /**
   * @unimod.event.descr "Add right car" button is pressed
   */
  public static final String E022 = "e022";
  /**
   * @unimod.event.descr Humans can go
   */
  public static final String E03 = "e03";
  /**
   * @unimod.event.descr Humans should hurry up
   */
  public static final String E04 = "e04";
  /**
   * @unimod.event.descr Cars can drive
   */
  public static final String E05 = "e05";
  /**
   * @unimod.event.descr Cars should hurry up
   */
  public static final String E06 = "e06";
  /**
   * @unimod.event.descr Everyone stops
   */
  public static final String E07 = "e07";



  public static ModelEngine engine;
  public void init(ModelEngine engine) throws CommonException {
    ButtonsEvents.engine = engine;
  }  public void dispose() {
```

```
    // TODO Auto-generated method stub
  }
}
```

3. DrawingWindows.java

```
package svet.co;

import FormEvents.ButtonsEvents;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import javax.swing.*;
import javax.swing.border.*;

import java.awt.*;
import java.awt.event.*;


class MainWindow extends JFrame implements Runnable, ActionListener {
  static final long serialVersionUID = 0;

  static private int width = 800, height = 600;// razmery okna

  Color c1 = new Color(0x000000);

  Color c2 = new Color(0x000000);

  Color c3 = new Color(0x00ff00);

  Color c4 = new Color(0xff0000);

  Color c5 = new Color(0x000000);

  ImageIcon roadImg, humanUpImage, humanDownImage, carRightImage,
      carLeftImage;

  public JButton buttonHumanDown;

  public JButton buttonHumanUp;

  public JButton buttonCarLeft;

  public JButton buttonCarRight;

  class ImagePanel extends JPanel {
    /**
     *
     */
    private static final long serialVersionUID = -7150943989162204260L;

    public ImagePanel() {
      super();
    }

    public void paintComponent(Graphics g) {
      super.paintComponent(g); // paint background
      roadImg.paintIcon(this, g, 0, 0);// 25, 140,
      if (TrafficEngine.h_up)
        humanUpImage.paintIcon(this, g, 255, 270 - TrafficEngine.yup);
      if (TrafficEngine.h_down)
        humanDownImage.paintIcon(this, g, 195, 5 + TrafficEngine.ydown);
      if (TrafficEngine.c_right)
        carRightImage.paintIcon(this, g, 0 + TrafficEngine.xright, 180);
```

```
    if (TrafficEngine.c_left)
      carLeftImage.paintIcon(this, g, 400 - TrafficEngine.xleft, 90);
  }
}

class LightsPanel extends JPanel {
  /**
   *
   */
  private static final long serialVersionUID = 72694650675088650 46L;

  public LightsPanel() {
    super();
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g); // paint background
    Graphics offGr = g;
    width = this.getWidth();// 150;
    height = this.getHeight();// 350;
    int r = 25;
    int d = r * 2;
    offGr.setColor(c1);
    offGr.fillOval(width * 3 / 4 - r, height / 6 - r, d, d);

    offGr.setColor(c2);
    offGr.fillOval(width * 3 / 4 - r, height / 2 - r, d, d);

    offGr.setColor(c3);
    offGr.fillOval(width * 3 / 4 - r, height * 5 / 6 - r, d, d);

    offGr.setColor(c4);
    offGr.fillOval(width / 4 - r, height / 4 - r, d, d);

    offGr.setColor(c5);
    offGr.fillOval(width / 4 - r, height * 3 / 4 - r, d, d);
  }
}

MainWindow() {
  super("Traffic lights");
  String prefix = "/home/magomedov/workspace/Svetofor/res/";
  roadImg = new ImageIcon(prefix + "road.png");
  humanUpImage = new ImageIcon(prefix + "human_up.png");
  humanDownImage = new ImageIcon(prefix + "human_down.png");
  carRightImage = new ImageIcon(prefix + "car_right.png");
  carLeftImage = new ImageIcon(prefix + "car_left.png");
  System.getProperties().list(System.out);
  try {
    // Runtime.getRuntime().exec("nautilus .");
  } catch (Exception e) {
  }

  Container c = getContentPane();
  c.setLayout(null);

  ImagePanel imgPan = new ImagePanel();
  c.add(imgPan);
  imgPan.setBounds(30, 220, 500, 320);
  imgPan.setBorder(BorderFactory.createEtchedBorder());
  imgPan.setVisible(true);

  LightsPanel lghPan = new LightsPanel();
```

```
    c.add(lghPan);
    lghPan.setBounds(30, 20, 500, 160);
    lghPan.setBorder(BorderFactory.createEtchedBorder());
    lghPan.setVisible(true);

    JLabel roadLabel = new JLabel("Pedestrian crossing", JLabel.CENTER);
    roadLabel.setBounds(30, 540, 500, 20);
    roadLabel.setBorder(BorderFactory.createEtchedBorder());
    c.add(roadLabel);

    JLabel humanLabel = new JLabel("Signals for pedestrians", JLabel.CENTER);
    humanLabel.setBounds(30, 180, 250, 20);
    humanLabel.setBorder(BorderFactory.createEtchedBorder());
    c.add(humanLabel);

    JLabel carLabel = new JLabel("Signals for cars", JLabel.CENTER);
    carLabel.setBounds(280, 180, 250, 20);
    carLabel.setBorder(BorderFactory.createEtchedBorder());
    c.add(carLabel);

    JPanel humanButtons = new JPanel();
    c.add(humanButtons);
    humanButtons.setBounds(550, 250, 210, 100);
    humanButtons.setLayout(new GridLayout(2, 1));
    humanButtons.setBorder(new TitledBorder(BorderFactory
        .createEtchedBorder(), "Add pedestrian"));

    buttonHumanDown = new JButton("bottom");
    buttonHumanDown.addActionListener(this);
    buttonHumanDown
        .setToolTipText("Add pedestrian to the bottom of the road");
    humanButtons.add(buttonHumanDown);
    buttonHumanDown.setVisible(true);

    buttonHumanUp = new JButton("top");
    buttonHumanUp.addActionListener(this);
    buttonHumanUp.setToolTipText("Add pedestrian to the top of the road");
    humanButtons.add(buttonHumanUp);
    buttonHumanUp.setVisible(true);

    humanButtons.setVisible(true);

    JPanel carButtons = new JPanel();
    c.add(carButtons);
    carButtons.setBounds(550, 400, 210, 100);
    carButtons.setLayout(new GridLayout(2, 1));
    carButtons.setBorder(new TitledBorder(BorderFactory
        .createEtchedBorder(), "Add car"));// BorderFactory.createLineBorder(Col
or.BLACK));

    buttonCarLeft = new JButton("left");
    buttonCarLeft.addActionListener(this);
    buttonCarLeft.setToolTipText("Add left car to the road");
    carButtons.add(buttonCarLeft);
    buttonCarLeft.setVisible(true);

    buttonCarRight = new JButton("right");
    buttonCarRight.addActionListener(this);
    buttonCarRight.setToolTipText("Add right car to the road");
    carButtons.add(buttonCarRight);
    buttonCarRight.setVisible(true);

    carButtons.setVisible(true);
```

```
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(width, height);
    setLocation(100, 100);
    setBackground(Color.WHITE);
    setResizable(false);
    setVisible(true);
  }

  public void sendEvent(String event) {
    ButtonsEvents.engine.getEventManager().handle(new Event(event),
        StateMachineContextImpl.create());
  }

  public void actionPerformed(ActionEvent e) {

    if (e.getSource() == buttonHumanDown)
      sendEvent(ButtonsEvents.E011);
    if (e.getSource() == buttonHumanUp)
      sendEvent(ButtonsEvents.E012);
    if (e.getSource() == buttonCarLeft)
      sendEvent(ButtonsEvents.E021);
    if (e.getSource() == buttonCarRight)
      sendEvent(ButtonsEvents.E022);

  }

  public void free() {
  };

  public void paint(Graphics g) {
    super.paintComponents(g);
  }

  public void run() {
    repaint();
  }
}

public class DrawingWindows implements ControlledObject {

  static MainWindow sl = new MainWindow();

  TrafficEngine trEn = new TrafficEngine();

  /**
   * @unimod.action.descr Defaults are setting
   */
  public void z0(StateMachineContext context) {
    sl.paint(sl.getGraphics());
    sl.run();
    TrafficEngine.h_up = false;
    TrafficEngine.h_down = false;
    TrafficEngine.c_left = false;
    TrafficEngine.c_right = false;
    TrafficEngine.hurry = false;
    TrafficEngine.stopping = false;
    TrafficEngine.xleft = 0;
    TrafficEngine.xright = 0;
    TrafficEngine.yup = 0;
    TrafficEngine.ydown = 0;
  }
```

```
/**
 * @unimod.action.descr Cars - green turnes off
 */
public void z1(StateMachineContext context) {
  sl.c3 = Color.BLACK;
  sl.run();
}

/**
 *
 * @unimod.action.descr Cars - green turnes on
 */
public void z2(StateMachineContext context) {
  sl.c3 = Color.GREEN;
  sl.run();
}

/**
 * @unimod.action.descr Cars - yellow turns on
 */
public void z3(StateMachineContext context) {
  sl.c2 = Color.YELLOW;
  sl.run();
}

/**
 * @unimod.action.descr Cars - yellow turns off; red turnes on
 */
public void z4(StateMachineContext context) {
  sl.c2 = Color.BLACK;
  sl.c1 = Color.RED;
  sl.run();
}

/**
 * @unimod.action.descr People - red turnes off; green turnes on
 */
public void z5(StateMachineContext context) {
  sl.c4 = Color.BLACK;
  sl.c5 = Color.GREEN;
  sl.run();
}

/**
 * @unimod.action.descr People - green turnes on
 */
public void z6(StateMachineContext context) {
  sl.c5 = Color.GREEN;
  sl.run();
}

/**
 * @unimod.action.descr Peopel - green turnes off
 */
public void z7(StateMachineContext context) {
  sl.c5 = Color.BLACK;
  sl.run();
}

/**
 * @unimod.action.descr People - red turnes on; Cars - yellow turnes on; red
 *                       turnes on
 */
```

```
public void z8(StateMachineContext context) {
  sl.c4 = Color.RED;
  sl.c2 = Color.YELLOW;
  sl.c1 = Color.RED;
  sl.run();
}

/**
 * @unimod.action.descr Cars - green turnes on; yellow turnes off; red
 *                      turnes off
 */
public void z9(StateMachineContext context) {
  sl.c1 = Color.BLACK;
  sl.c2 = Color.BLACK;
  sl.c3 = Color.GREEN;
  sl.run();
}

/**
 * @unimod.action.descr Humans can go message
 */
public void z10(StateMachineContext context) {
  sl.sendEvent(ButtonsEvents.E03);
}

/**
 * @unimod.action.descr Humans should hurry message
 */
public void z11(StateMachineContext context) {
  sl.sendEvent(ButtonsEvents.E04);
}

/**
 * @unimod.action.descr Everyone stops message
 */
public void z12(StateMachineContext context) {
  sl.sendEvent(ButtonsEvents.E07);
}

/**
 * @unimod.action.descr Cars can drive message
 */
public void z13(StateMachineContext context) {
  sl.sendEvent(ButtonsEvents.E05);
}

/**
 * @unimod.action.descr Cars should hurry message
 */
public void z14(StateMachineContext context) {
  sl.sendEvent(ButtonsEvents.E06);
}

/**
 * @unimod.action.descr Are there any humans?
 */
public boolean x1(StateMachineContext context) {
  return TrafficEngine.h_down || TrafficEngine.h_up;
}
/**
 * @unimod.action.descr Are there any cars?
 */
public boolean x2(StateMachineContext context) {
```

```
    return TrafficEngine.c_left || TrafficEngine.c_right;
  }

}
```

Приложение 2. Установка и запуск приложения

На сайте http://is.ifmo.ru в разделе «Проекты» размещены проектная документация, Java-приложение и исходные тексты на языке *Java* программы «Моделирование работы «умного» светофора».

С сайта можно загрузить версию приложения, выполненную в рамках компиляционного подхода. Для запуска данного приложения также требуется только виртуальная машина *Java*, которую можно скачать с сайта *Sun Microsystems* (http://java.sun.com/j2se/1.5.0/download.jsp).

Исходные тексты программы предоставляются в двух вариантах:

- в виде файлов приложения, построенного по компиляционному подходу;
- в виде проекта для *Eclipse*.

Оба варианта содержат написанные вручную классы для объектов управления и источников событий. Отличие между двумя версиями исходных текстов состоит в том, что в одном случае в поставку включены *UML*-диаграммы и некоторые другие файлы для просмотра проекта в среде *Eclipse*, а в другом вместо диаграмм включен автоматически сгенерированный класс на языке *Java*.

Для работы с файлами проекта в среде *Eclipse*, сначала необходимо установить данную среду разработки. Загрузить последнюю версию *Eclipse SDK* можно по ссылке http://www.eclipse.org/downloads. Кроме того, необходимы следующие плагины:

- *GEF* (http://download.eclipse.org/tools/gef/downloads);
- *UniMod* (http://unimod.sourceforge.net/download.html).

Краткие инструкции по интеграции этих плагинов в *Eclipse* приведены на странице http://unimod.sourceforge.net/eclipse-plugin.html

. Рассматриваемый проект разрабатывался в среде *Eclipse* версии 3.1 с плагином *UniMod* версии 1.3.1.35 (релиз от 10.01.2006).

Чтобы открыть проект в среде разработки *Eclipse*, выберите пункт меню *File/Import...* (*Файл/Импорт...*). В открывшемся диалоге мастера выберите *Existing Project into Workspace* (Существующий проект в рабочую область) из списка возможных действий и нажмите кнопку *Next* > (*Далее* >). В окне выбора папки с файлами проекта укажите ту папку, в которую был распакован архив с исходными текстами, скачанный с сайта http://is.ifmo.ru. Нажмите кнопку *Finish (Завершить импорт)*. Подождите несколько секунд, в течение которых *Eclipse* в фоновом режиме скомпилирует *Java*-классы. Чтобы запустить приложение в режиме интерпретации, откройте файл с диаграммами *Svet.unimod*, щелкните правой клавишей мыши на графе переходов автомата *A1* и выберите пункт контекстного меню *Launch A1*! (Запустить *A1*!).