Saint-Petersburg State University
of Information Technologies, Mechanics and Optics
Computer Technologies Department

T. G. Magomedov, A. B. Ostrovskiy

Simulation of smart traffic lights

Programming with switch-technology and *UniMod* developer environment

Project documentation

Project was created under
«Foundation for open project documentation»
http://is.ifmo.ru

Saint – Petersburg
2006

Contents

Introduction 3
1. Target setting 3
2. Communication circuit 3
3. Controlled objects 5
3.1. Graphic output controlled object *o1* 5
3.2. Motion controlled object *o2* 6
4. Automata 6
4.1. Main automaton *A1* 6
4.1.1. State chart 6
4.1.2. Verbal description 6
4.2. Automaton controlled switching of traffic light *A2* 7
4.2.1. State-chart 7
4.2.2. States of *A2* 7
4.2.3. Verbal description 8
4.3. Automaton *A3* responsible for motion pedestrians and cars 8
4.3.1. State-chart 8
4.3.2. States of automata *A3* 9
4.3.3. Verbal description 9
5. Interpretation and compilation approaches. 9
6. Resume 11
Literature 11
Appendix. Source Code 12
1. `Main.java` file 12
2. `TrafficEngine.java` file 14
3. `DrawingWindows.java` file 19

Introduction

Usual traffic-light does not have feedback – it's only changing signals in established time interval. If there are not cars on the road, foot-passenger need to wait for green light. If there are not foot-passengers, drivers have to stop anyway.

«Clever» traffic-light, modeled in this project, saves time for foot-passengers as well as drivers due to minimizing idle wait.

Project is developed in *Eclipse* IDE using *UniMod* tool on *Java* language.

Every developing tool (*Eclipse, UniMod, Java*), used in project, is multiplatform. Originally program was developed in *Linux* operation system, and then was executed in *Microsoft* © *Windows*™.

1. Target setting

As mentioned above, in this work is modeled pedestrian crossing with «clever» traffic-light. sequence of switching signals of usual traffic light simplistically may be represented in the following way:

- drivers go;
- both (drivers and foot-passengers) wait;
- foot-passengers go;
- both (drivers and foot-passengers) wait;
- repeating cycle.

«Clever» traffic-light works not in the least like that. It's possession of information about cars on the road and foot-passengers on pedestrian crossing. In presence of some foot-passengers and cars, traffic-light works like common traffic-light. If there are no cars, but foot-passengers are waiting on pedestrian crossing, it will be indicating green light for them until at least one car appears, and vice versa.

If there are neither cars no foot-passengers, green light will be indicating for cars because they need more time to slow down and to pick up speed to continue the motion.

Table 1. Functional logic for «clever» traffic-light

	No foot-passengers	**Foot-passengers exist**
No cars.	Green light for cars	Green light for foot-passengers
Cars exist.	Green light for cars	Usual traffic-light mode

2. Communication circuit

Communication circuit as class diagram is presented at fig. 1.

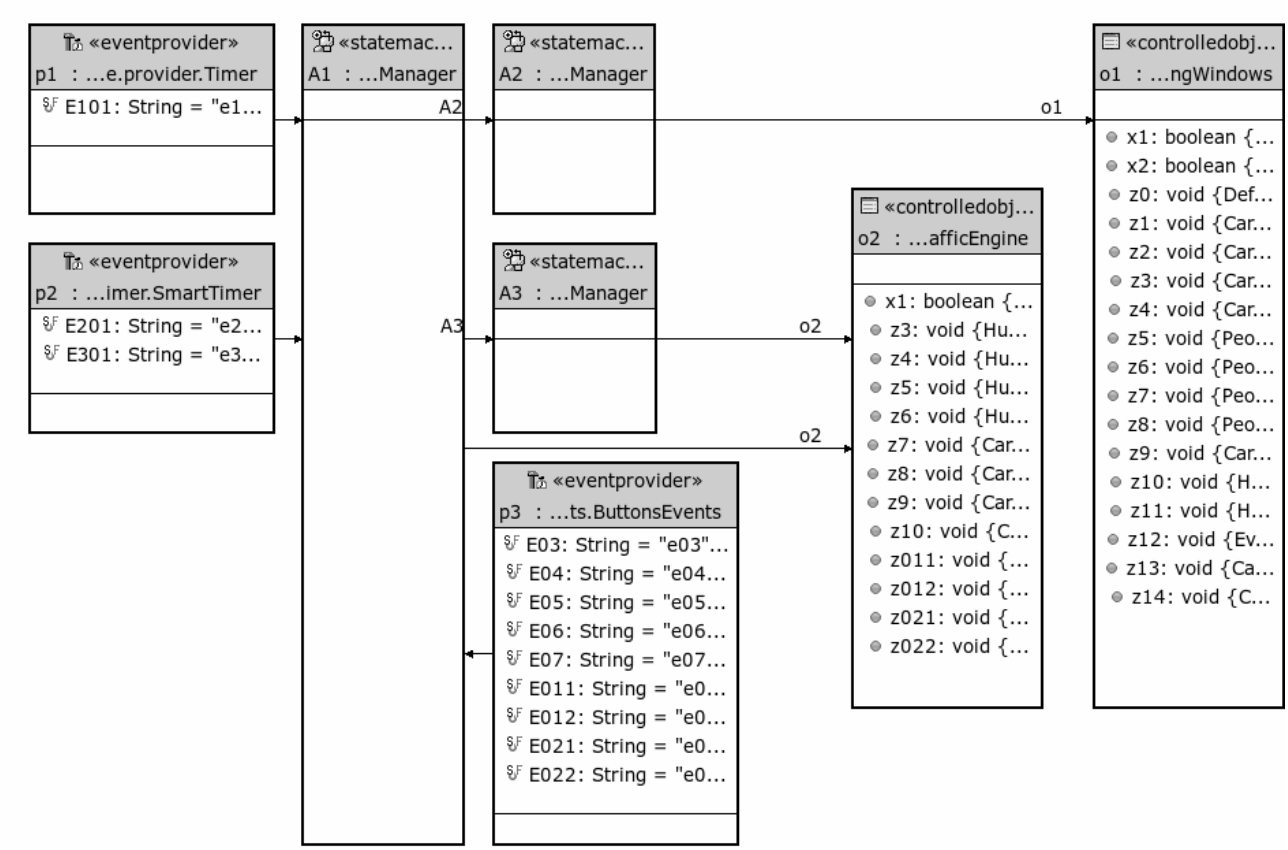

Fig.1. Class diagram

At this figure *p1, p2, p3* classes are event providers. *A1, A2, A3* — automaton. *o1, o2* classes — controlled objects.

Every provider supplies events. Provider *p1* is implemented in *Unimod tool* timer, sending events in an equal time interval. Provider *p2* is similar timer, that sends two kinds of events in a different time interval. Provider *p3* supplies events, initiated buttons pressing as well as traffic-light signal switching. (table 2 – table 4)

Table 2. Events of event provider p1.

Event	Description
e101	One second elapsed

Table 3. Events of event provider p2

Event	Description
e201	Five seconds elapsed
e301	1/10 of second elapsed

Table 4. Events of event provider p3

Event	Description
e03	Foot-passengers may go
e04	Foot-passengers need to hurry
e05	Cars may go
e06	Cars need to hurry
E07	Both (Foot-passengers and drivers) stop.
E011	Button «Add pedestrian from the bottom» pressed
E012	Button «Add pedestrian from the top» pressed
E021	Button «Add car from the left» pressed
e022	Button «Add car from the right» pressed

Automaton *A1* receives events for all providers – *p1*, *p2*, and *p3*. *A1* is organized in such way, than automata *A2* and *A3* receives the same events, as *A1*. Automata may call class methods – controlled objects' methods. Automaton *A2* calls methods of controlled objects *o1*, *A1* and *A3* methods of controlled objects *o2*.

3. Controlled objects

3.1. Graphic output controlled object *o1*

Input and output actions of controlled object responsible for rendering are presented in table 5.

Table 5. Input and output actions of controlled object o1

Method	Description
x1	If any foot-passenger exist?
x2	If any cars exist?
z0	Defaults are set, graphic output window opened
z1	Green light for drivers become dim
z2	Green light for drivers light up
z3	Yellow light for drivers light up
z4	Yellow light for drivers become dim, red light up
z5	Red light for foot-passengers become dim, green light up
z6	Green light for foot-passengers light up
z7	Green light for foot-passengers become dim
z8	Red light signal for foot-passengers light up, red yellow signals for drivers light up simultaneously.
z9	Green light signal for driver light up, red and yellow signal become dim
z10	Event e03 is initialized (foot-passengers can go)
z11	Event e04 is initialized (foot-passengers should hurry up)
z12	Event e07 is initialized (foot-passengers and drivers must stop)
z13	Event e05 is initialized (drivers can go)
z14	Event e06 is initialized (drivers should hurry up)

3.2. Motion controlled object *o2*

Input and output actions of controlled object responsible for motion are presented in table 6.

Table 6. Input and output actions of controlled object o2

Method	Description
x1	If foot-passengers hasten?
z3	Foot-passengers make a step
z4	Foot-passengers starts hasten
z5	Foot-passengers make fast step
z6	Foot-passengers stop
z7	Cars drive
z8	Cars drive until pedestrian cross
z9	Cars starts hasten
z10	Cars stops
z011	New foot-passenger appears in the bottom side
z012	New foot-passenger appears in the top side
z021	New car appears in the left side
z022	New car appears in the right side

4. Automata

4.1. Main automaton *A1*

4.1.1. State chart

State-chart for `A1` presented at fig. 2.

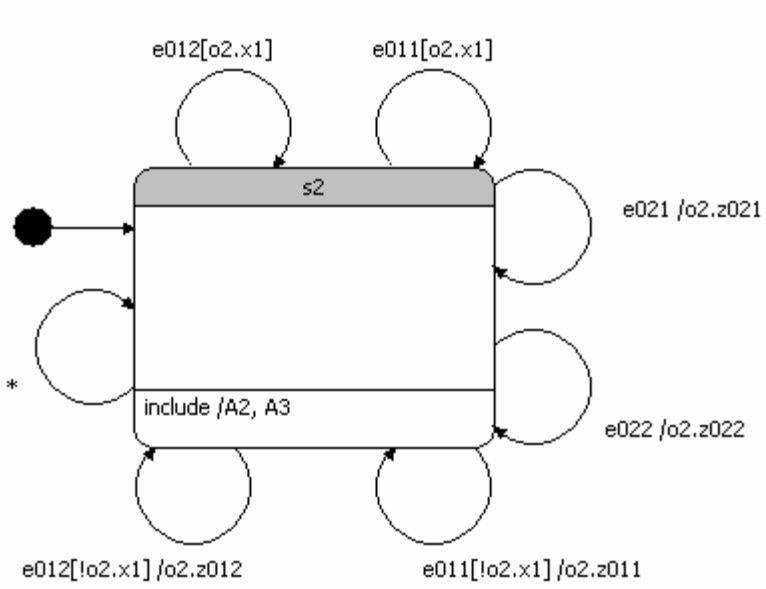

Fig. 2. State chart for *A1*

4.1.2. Verbal description

Automaton A1 – main project automaton. Event from every providers are passed to it. Automata *A1* has one state. Automata A2 and A3 are included into it. All events received by automata *A1*

automatically passed to *A2* and *A3*. Loops indicate invoking of corresponding methods of controlled object o2 in response to events, related with pressing on control elements (buttons) of graphical interface.

4.2. Automaton controlled switching of traffic light *A2*

4.2.1. State-chart

State-chart of automaton `A2` is presented at fig. 3.

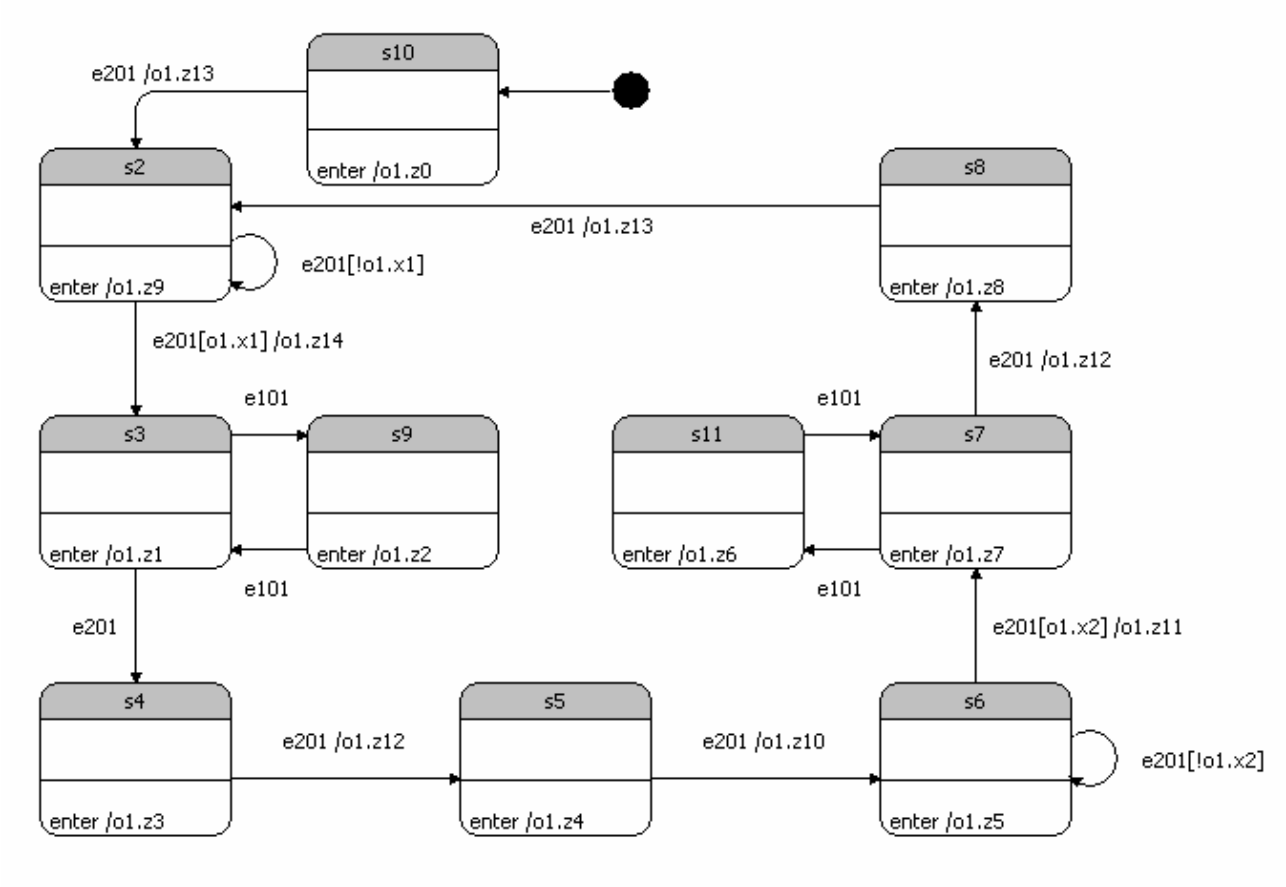

Fig. 3. State chart for *A2*

4.2.2. States of *A2*

States of automaton `A2` are presented at table 7.

Table 7. States of automaton A2

State	Description
s10	Start state
s2	Green signal for drivers glow, for foot-passengers - red
s3	Green signal for drivers become dim
s9	Green signal for drivers light up. This and previous state implements blinking
s4	Yellow signal for drivers light up
s5	Green signal for drivers become dim and yellow signal light up
s6	Red signal for foot-passengers become dim, green signal light up
s7	Green signal for foot-passengers light up
s11	Green signal for foot-passengers become dim. With previous state implements blinking
s8	Red signal for foot-passengers light up, for drivers – red and yellow signals

4.2.3. Verbal description

Automaton transition graph contains several cycles. Before entering the cycle, initialization of graphical interface performs in s10 state. This action need to be performed once. The rest states assigned such way, that each state agree too its own configuration of traffic-light. For example, in s2 state green signal for drivers, red signal indicates for foot-passengers. In s6 state red signal glow for drivers, green signal indicates for foot-passengers.

Transitions between states performs by timer events, lingering on s2 state, if there an no foot-passengers and in s6 state if there no cars on the road. When entering in each state than marked by word `enter`, corresponding method of o1 controlled objected invoked. This method sets colors for traffic-light signals, corresponding to invoking state. On transitions output actions invoked, sending events about necessity of motion stop. This actions receives automaton *A3*, controlled motion of cars and foot-passengers across pedestrian crossing.

4.3. Automaton *A3* responsible for motion pedestrians and cars

4.3.1. State-chart

State-chart of automata *A3* is presented at fig. 4.

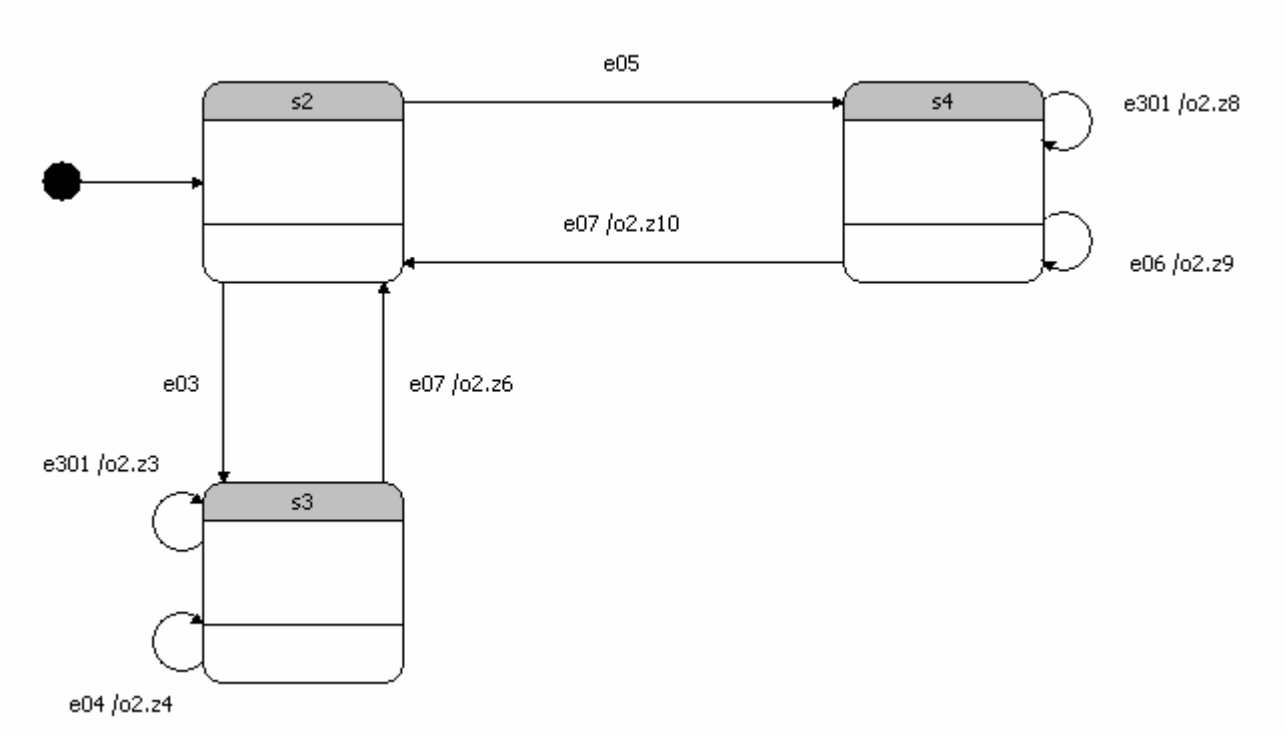

Fig. 4. State-chart of *A3*

4.3.2. States of automata *A3*

States of automata A2 are presented in table 8.

Table 8. States of automaton A3

State	Description
s2	Both cars and foot-passengers are stoped
s3	Foot-passengers are crossing pedestrian
s4	Drivers are crossing pedestrian

4.3.3. Verbal description

This automaton is responsible for modeling of motion across the pedestrian crossing, conducting by traffic-light signals.

In s2 state both cars and foot-passengers are stopped. By event, than was initiated by traffic-light, switch to state s3 (if cars are crossing the pedestrian crossing) or to state s4 (if foot-passengers are crossing the pedestrian crossing) realized. It those states by “fast” timer redrawing of cars and foot-passengers is realized, and those coordinates changes. By receiving a signal about necessity of hasten (green light start blinking), speed of foot-passengers and cars increase.

5. Interpretation and compilation approaches.

Automata implementation on basis of interpretative approach using Unimod, automata isn’t transformed into java-code. It is convenient during writing program, because programmer needn’t worry of automata to java-code transformation – it executing itself. At the save time event providers and controlled objects are compiled into java byte-code. Scheme of interpretative approach is presented of fig. 5.

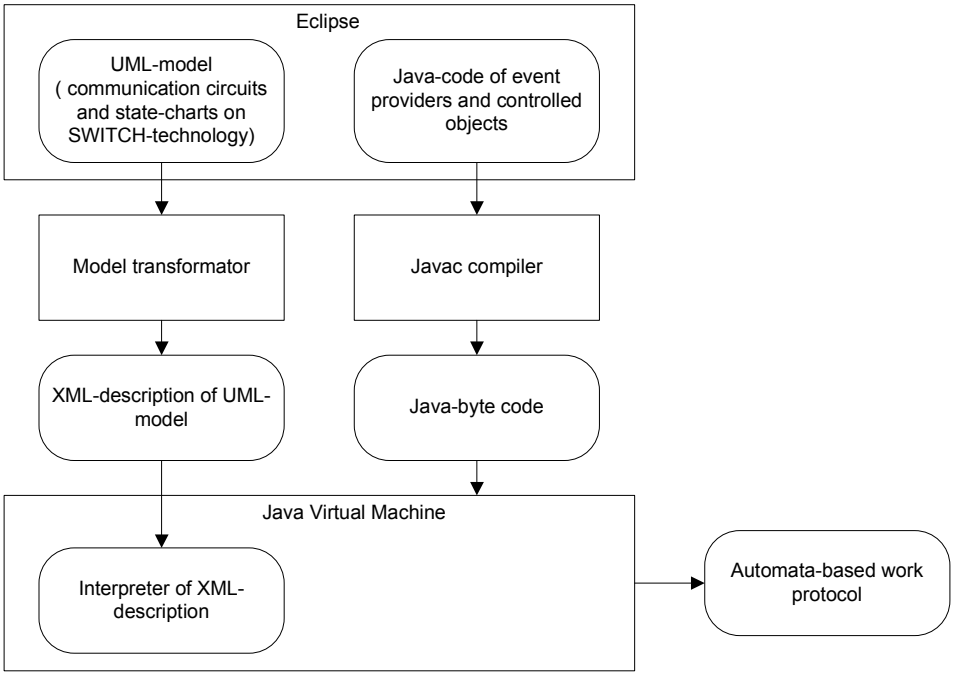

Fig 5. Structural scheme of interpretation approach

Interpreter using automaton XML-description, emulate its operating, launching automata as byte-code. During program writing it is not important, how connection schemes and state diagrams are executed, but interpretative approach has disadvantages:

- slow response;
- interpreter is necessary.

There is alternative of interpretative approach. Using automaton XML-descriptor can be built isomorphous java code, which will be compiled into byte-code. This approach to program executing is called compilation approach. Scheme of compilation approach is presented of fig. 6.

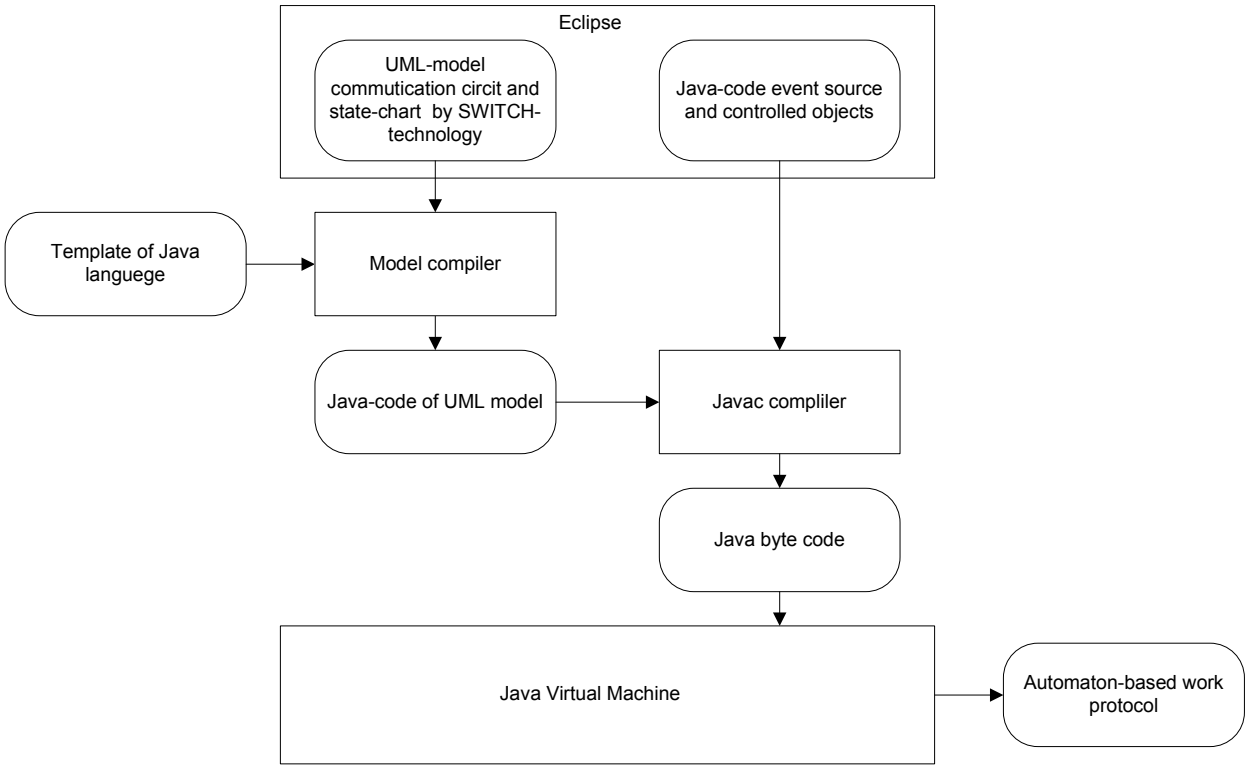

Fig. 6. Structural scheme of compilation approach

Using compilation approach Unimod generates source file, corresponding to automaton *A1*. *Java*-code, launching automaton, need to writed by hand. Classes of event providers, controlled objects, some *Unimod* classes and launching class, necessary for program, are compiled and on the outlet we have independent program, which can be executed everywhere, where *Java* is installed.

6. Resume

Described model of traffic-light basically is suitable for usage, if technical problem (recognition of waiting cars and foot-passengers) will be solved.

Further improvement of «clever» traffic-light can be in the area of time dependence interval for switching. Also it may be concerned cooperation traffic-lights on the crossroad and neighbor crossroads.

Literature

1. *Shalyto A. A.* SWITCH-technology. Algorithmic and developing logic control problems. Saint-Petersburg.: Nauka, 1998. http://is.ifmo.ru/books/switch/1
2. *Shalyto A. A., Tuckel N.I* SWITCH-technology – automata approach in creation software for “reactive” systems //Programming. 2001,.№ 5 http://is.ifmo.ru/works/switch/1/

Appendix. Source Code

1. `Main.java` file

```
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 file

```
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 file

```
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;
  }

}
```