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

Computer Technologies Department

A. V. Vokin, I. A. Pimenov, A. A. Shalyto

Modeling of the Car's Auto Gear

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

As shown in this work, *SWITCH*-technology, suggested in [1, 2], is, as we think, the most natural decision for a wide classes of problems to control events-based systems. That's why its application is expedient for solving problems of developing simulators of such systems.

The goal of this this work – modeling car auto gear basing on *SWITCH*-technology and *UniMod development environment*, that is used for automata programming.

You can find more on this technology at http://is.ifmo.ru, and more about *UniMod development environment* – at http://unimod.sourceforge.net .

Application is created using IDE *Eclipse 3.1*. At the same time *UniMod is a plug-in to mentioned above IDE*. Release 1.2.15 was used, that supports *JDK 1.5*.

1. Target setting

Goal of this project – is to build model to imitate auto car gear. It should follows the down below requirements.

1. Management of car auto gear is accomplished with three actions:

   а) switching on the ignition;

   б) switching direction;

   в) holding on accelerator pedal.

2. Management system, that is made basing on finite automata, should provide control over number of rotations and in-time gear switching.
3. It is thought that engine is unable to work with more then 3000 rotations a minute.
4. Engine is turned on by pressing button *ON*.
5. User chooses direction using panel *DIRECTION*:

   a) Forward direction – *FORWARD*;

   b) Backward direction – *BACKWARD*;

   c) Neutral – *NEUTRAL*. In this case engine is turned on, but is not moving.

6. While keeping *GAS* button pressed on, engine is increasing rotations number. At the moment number of 2000 rotations is reached, switching on a higher gear is happened;
7. User may stop engine, only in case being on *NEUTRAL* gear, by pressing the button *OFF*.

Application screenshot is shown on Fig. 1

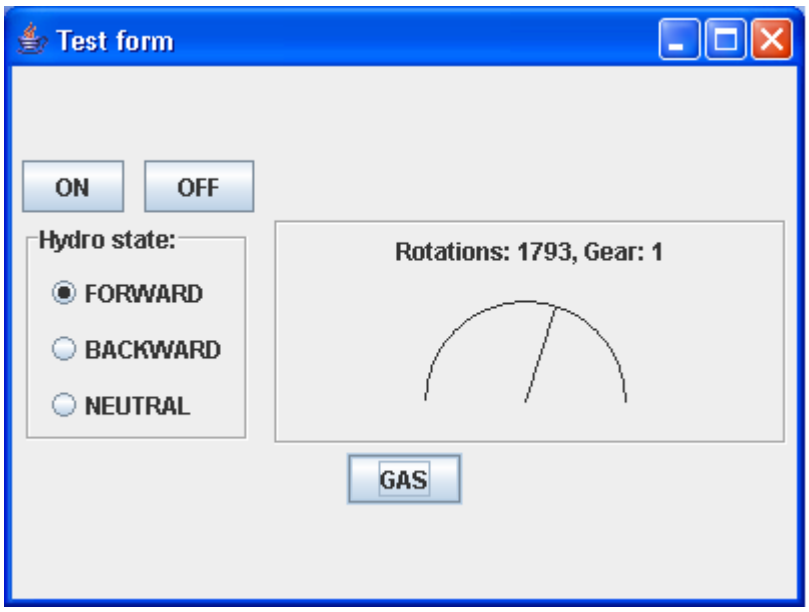

Fig. 1. Example of running application

2. Performance script

2.1 Informal script text

Car's engine is turned off, that's why all user interactions, except turning the engine on, doesn't change vehicle state.

User turns engine on. Tachometer's arrow gets up to 3000 transactions in minute. Now vehicle gives response on pushing down gas – it increases its transactions (tachometers arrow moves right). At this moment, even if number of transactions will exceed 3000 transactions in a minute. Turning gear to another state would not happen, because we don't know the direction where we are moving (auto gear's state in neutral).

After user has chosen moving direction, indicator appears on the form, ,that shows us number of engines transactions and state of auto gear. Now, if we chose forward direction, then after exceeding 3000 transactions in a minute switching to next gear state happens. Now turning the engine off or changing direction can be done only through neutral gear state.

And if we are exceeding 3000 transactions a minute, but we are at 5 gear state, than there no switching will happen.

2.2. Formal script

After running application, to turn on engine user have to press ON button. To turn engine off and exit program, user should press OFF button. Then we should chose directions using buttons FORWARD or BACKWARD. To increase number of transactions user should press GAS button. When exceeding 2000 transactions, switching to higher gear state happens (but only if it was not five gear state). While GAS pedal is not pressed engine decreases transactions and when they are just about 2000 switching to a lower gear state happens(but only of it was not first gear state).

3. Class diagram

On fig. 2 you can see class diagram for model of car auto gear that is made with *SWITCH*-technology.

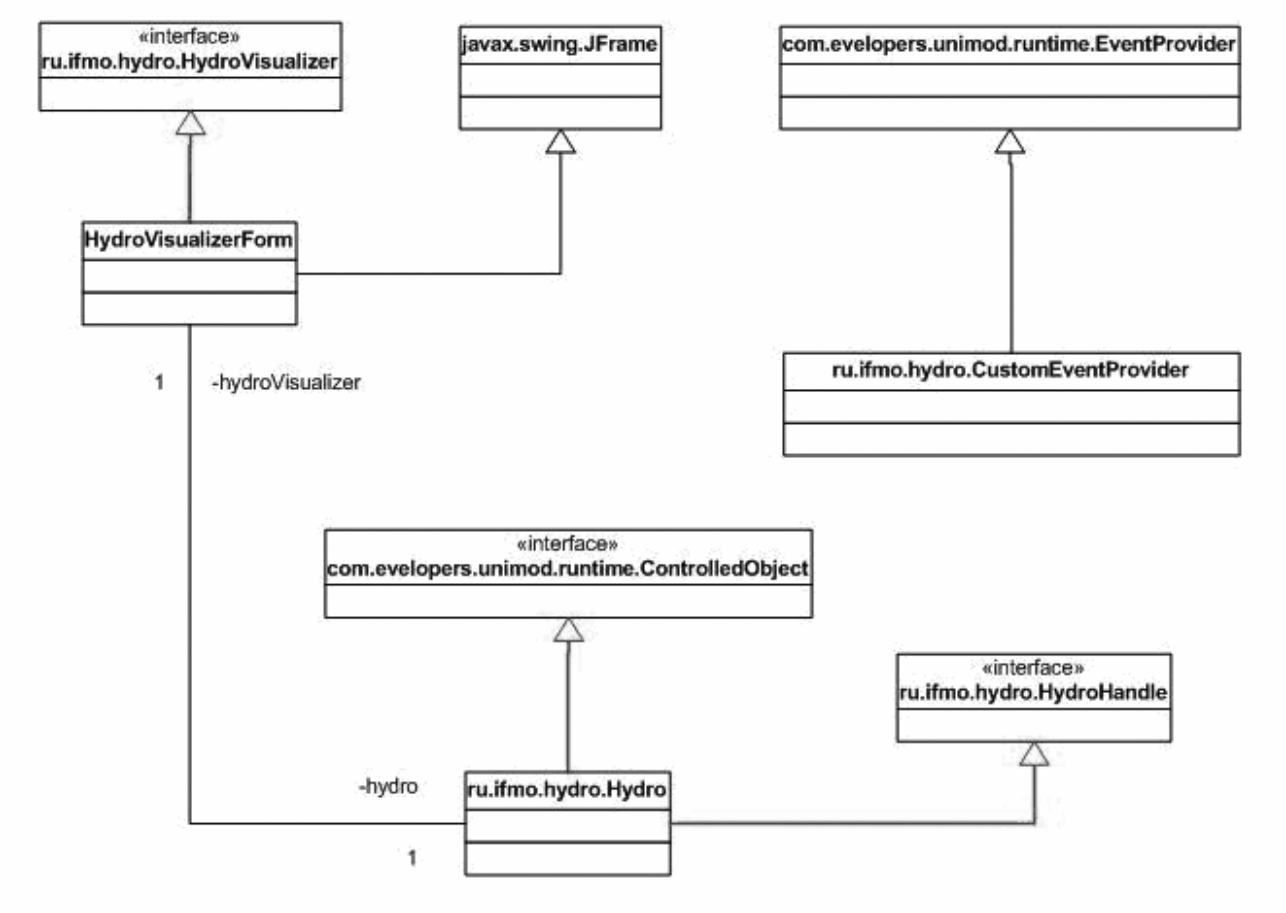

Fig.2. Class diagram

Let's shortly describe how classes on fig. 2 are connected:

- object of class `Hydro` – control object;
- interface `HydroHandler` – consists of methods, that affect user interactions to automata;
- interface `ControlledObject` – Interface of package *UniMod*. Tells us, that implemented class from it is an controlled object;
- object `CutromEventProvider` – event provider;
- interface `EventProvider` – interface of package *UniMod*. Tells us that class that implements it is event provider;
- object of class `HydroVisualizerForm` - realizes user interactions with application;
- interface `HydroVisualizer` – consists of methods, within GUI lets user control automata;
- class `JFrame` – class of *Swing* package that implements window.

Controlled object in this project is object of class `Hydro`. As it is thought in *UniMod* package Hydro implements interface ControlledObject. Also this class implements interface HydroHandle, that has methods to control class Hydro (user actions). For visualizing state of controlled object class HydroVisualizerForm is used. HydroVisualizerForm inherits window class JFrame from package

`Swing and implements interface HydroVisualizer, that has visualization methods.`

All events are set up in class `CustomEventProvider`, that, as it is thought in *UniMod* implements interface `EventProvider`. Class `CustomEventProvider` has all actions that happen in this project.

4. Classes descriptions

4.1. Interface `HydroVisualizer`

It consists of methods for visualizing states of finite automata, that simulates auto gear behavior and user interactions.

4.1.1. Methods descriptions

1. `Method public void setRotations(int rotations)` – sets engine rotations for drawing tachometer on the user form.
2. `Method public void setGear(int gear)` – sets number of current gear on visualizer.
3. `Method public void setStatus(int status)` – turns engine on or off.

4.2. Class `HydroVisualizerForm`

Implements `HydroVisualizer` interface and extends `JFrame` class from package *`Swing`*. Consists of elements of automata manipulation, that simulates car auto gear.

4.3 Class `JFrame`

Standard class from package *SWING* for drawing graphics window.

4.4. Class `CustomEventProvider`

Controlled object – auto gear. Implements interface `EventProvider` from package *UniMod*. Consists of variables, that describes auto gear state at this time:

- engine rotations;
- gear.

4.5. Class `EventProvider`

Interface from package *UniMod*, that implements all event providers.

4.6. Class `Hydro`

Implements interface `HydroHandler` and interface `ControlledObject` from package *UniMod*. Shows automata states. There the following attributes of this class:

- engine condition (turned on/off);
- movement direction (forward/backward);
- current gear;
- engine rotations.

4.7. Interface `HydroHandler`

Consists of methods for transmitting messages to automata, that are invoked by user interactions. Lets list used methods:

- `public void stop()` – user tried to turn engine off.
- `public void start()` – user turned engine on.
- `public void gasPedalPressed()` – user pressed accelerate pedal.
- `public void gasPedalReleased()` – user released accelerate pedal.
- `public void switchToForward()` – user selected *FORWARD* direction.
- `public void switchToBackward()` – user selected *BACKWARD* direction.

4.8. Class `ControlledObject`

Interface of package *UniMod*, that implements all controlled objects.

5. Automata `Automata`

5.1. Description

The automata emulates work of auto car gear.

5.2. Connections diagram

On the fig.3 you can see connections diagram for event provider, automata and controlled object.

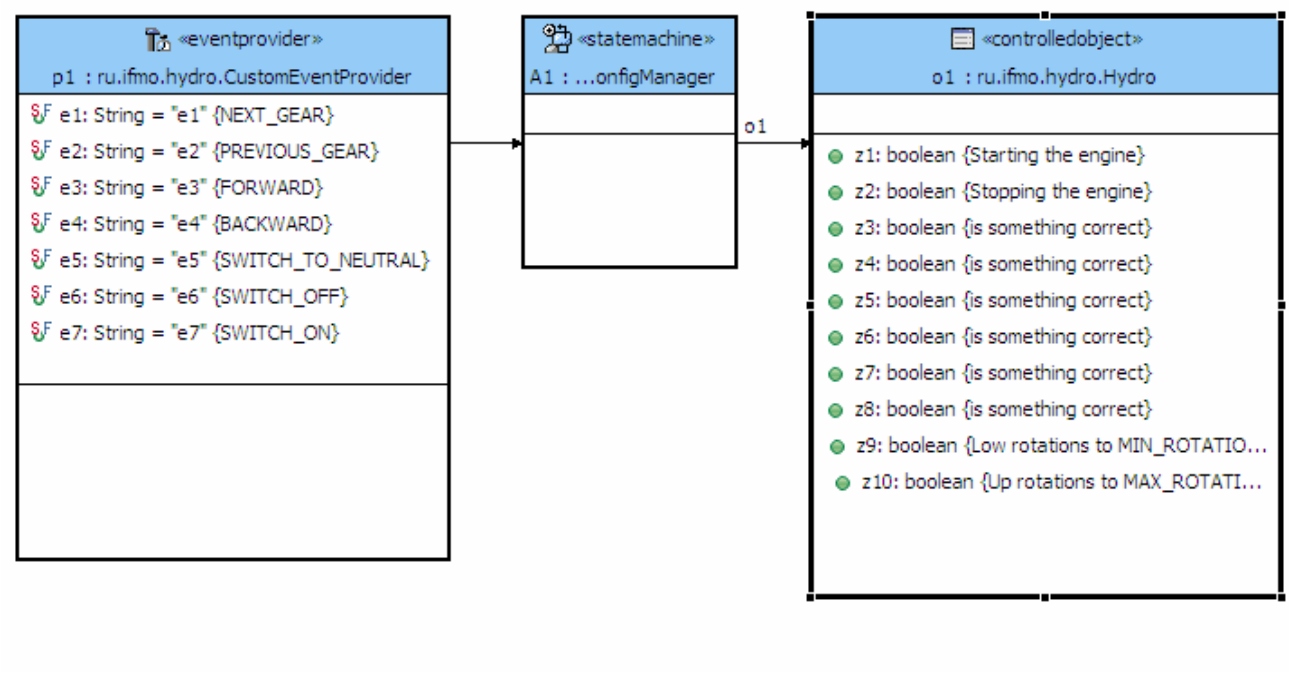

Fig. 3 Automata connections diagram

5.3. Transitions graph

On fig. 4 you can see transitions graph for automata

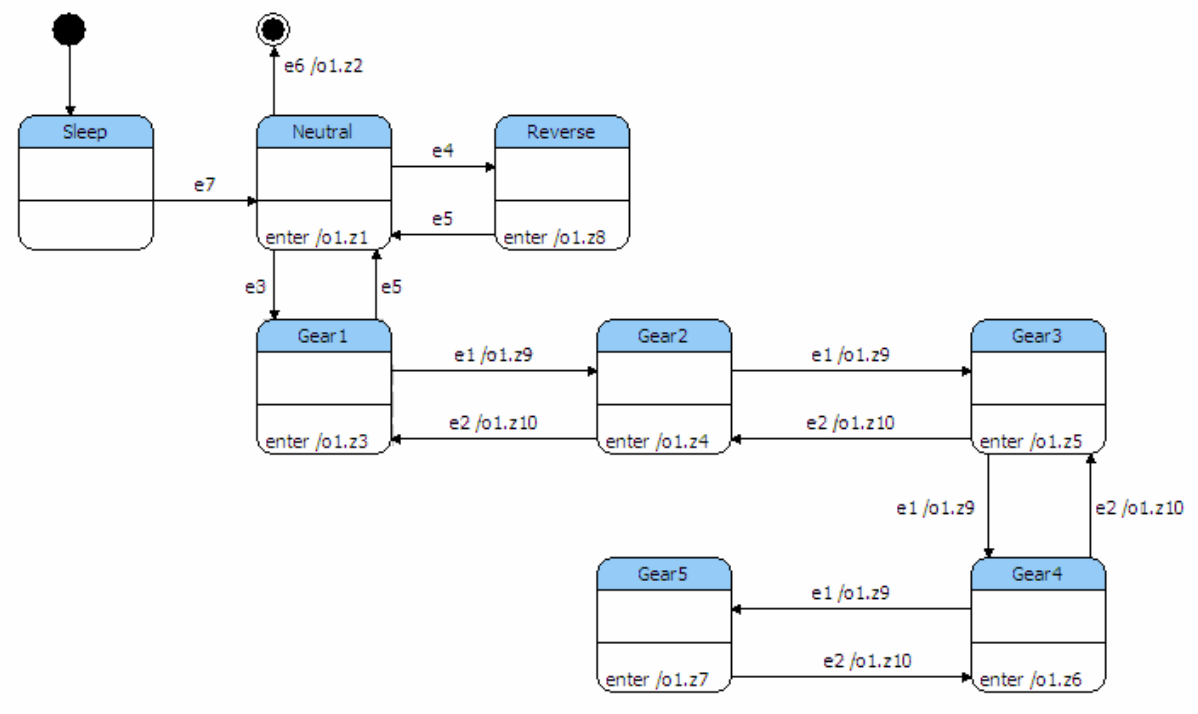

Fig. 4 Transition graph

6. Implementation

6.1 Interpretation

Interpretation way is thought as using xml-description of automata at every step of execution. This way has *UniMod* package in requirements.

In Application 1 you can see xml automata description that was created with the help of the *UniMod* package.

In Application 2 you can see source codes of events providers and controlled objects.

6.2 Compilation

Compilation allows creating Java class from xml description that represents automata. Following that we can avoid using UniMod libraries for launching application.

In Application 3 you can see source codes that are generated by *UniMod* package on transitions graph.

In Application 4 you can see sources of application interfaces.

Resume

SWITCH-technology can be used more than effective with the help of *UniMod* package, while developing applications.

At the same moment diagrams are used not only on designing application step, but also when debugging and working on feedback.

Package *UniMod* gives you ability to check for formal properties of transmission graph, the last one allows you to (despite of other products of the same type) get right working applications at first step.

Literature

1. *Shalito A. A.* SWITCH-technology. Algorithmic and developing logic control problems . Saint-Petersburg.: Nauka, 1998. http://is.ifmo.ru/books/switch/1
2. *Shalito 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/

Application1. XML automata description

```
01 <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE model PUBLIC "-//eVelopers Corp.//DTD
State machine model V1.0//EN" "http://www.evelopers.com/dtd/unimod/statemachine.dtd">
02 <model name="Model1">
03   <controlledObject class="ru.ifmo.hydro.Hydro" name="o1"/>
04   <eventProvider class="ru.ifmo.hydro.CustomEventProvider" name="p1">
05     <association clientRole="p1" targetRef="Automata"/>
06   </eventProvider>
07   <rootStateMachine>
08     <stateMachineRef name="Automata"/>
09   </rootStateMachine>
10   <stateMachine name="Automata">
11     <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
12     <association clientRole="Automata" supplierRole="o1" targetRef="o1"/>
13     <state name="Top" type="NORMAL">
14       <state name="Gear3" type="NORMAL">
15         <outputAction ident="o1.z5"/>
16       </state>
17       <state name="Neutral" type="NORMAL">
18         <outputAction ident="o1.z1"/>
19       </state>
20       <state name="Gear1" type="NORMAL">
21         <outputAction ident="o1.z3"/>
22       </state>
23       <state name="Sleep" type="NORMAL"/>
24       <state name="Reverce" type="NORMAL">
25         <outputAction ident="o1.z8"/>
26       </state>
27       <state name="Gear2" type="NORMAL">
28         <outputAction ident="o1.z4"/>
29       </state>
30       <state name="Gear5" type="NORMAL">
31         <outputAction ident="o1.z7"/>
32       </state>
33       <state name="s1" type="INITIAL"/>
34       <state name="Gear4" type="NORMAL">
35         <outputAction ident="o1.z6"/>
36       </state>
37       <state name="s2" type="FINAL"/>
38     </state>
39     <transition event="e2" sourceRef="Gear3" targetRef="Gear2">
40       <outputAction ident="o1.z10"/>
41     </transition>
42     <transition event="e1" sourceRef="Gear3" targetRef="Gear4">
43       <outputAction ident="o1.z9"/>
44     </transition>
45     <transition event="e3" sourceRef="Neutral" targetRef="Gear1"/>
46     <transition event="e4" sourceRef="Neutral" targetRef="Reverce"/>
47     <transition event="e6" sourceRef="Neutral" targetRef="s2">
```

```
48        <outputAction ident="o1.z2"/>
49      </transition>
50      <transition event="e5" sourceRef="Gear1" targetRef="Neutral"/>
51      <transition event="e1" sourceRef="Gear1" targetRef="Gear2">
52        <outputAction ident="o1.z9"/>
53      </transition>
54      <transition event="e7" sourceRef="Sleep" targetRef="Neutral"/>
55      <transition event="e5" sourceRef="Reverce" targetRef="Neutral"/>
56      <transition event="e2" sourceRef="Gear2" targetRef="Gear1">
57        <outputAction ident="o1.z10"/>
58      </transition>
59      <transition event="e1" sourceRef="Gear2" targetRef="Gear3">
60        <outputAction ident="o1.z9"/>
61      </transition>
62      <transition event="e2" sourceRef="Gear5" targetRef="Gear4">
63        <outputAction ident="o1.z10"/>
64      </transition>
65      <transition sourceRef="s1" targetRef="Sleep"/>
66      <transition event="e1" sourceRef="Gear4" targetRef="Gear5">
67        <outputAction ident="o1.z9"/>
68      </transition>
69      <transition event="e2" sourceRef="Gear4" targetRef="Gear3">
70        <outputAction ident="o1.z10"/>
71      </transition>
72    </stateMachine>
73  </model>
```

Application2. Source codes for events providers and controlled objects

`CustomEventProvider.java`

Events provider. That are generated with the use if user.

```
01  package ru.ifmo.hydro;
02
03  import com.evelopers.common.exception.CommonException;
04  import com.evelopers.unimod.runtime.EventProvider;
05  import com.evelopers.unimod.runtime.ModelEngine;
06
07  public class CustomEventProvider implements EventProvider {
08
09      /**
10       * @unimod.event.descr NEXT_GEAR
11       */
12      public static final String e1 = "e1";
13      /**
14       * @unimod.event.descr PREVIOUS_GEAR
15       */
16      public static final String e2 = "e2";
17      /**
18       * @unimod.event.descr FORWARD
19       */
20      public static final String e3 = "e3";
21      /**
22       * @unimod.event.descr BACKWARD
23       */
24      public static final String e4 = "e4";
25      /**
26       * @unimod.event.descr SWITCH_TO_NEUTRAL
27       */
28      public static final String e5 = "e5";
29      /**
30       * @unimod.event.descr SWITCH_OFF
31       */
32      public static final String e6 = "e6";
33      /**
34       * @unimod.event.descr SWITCH_ON
```

```
35      */
36     public static final String e7 = "e7";
37 
38     public void init(ModelEngine engine) throws CommonException {
39         Hydro.init(engine);
40     }
41 
42     public void dispose() {
43         // TODO Auto-generated method stub
44 
45     }
46 
47 }
```

Hydro.java

Control object. Represents car auto gear by itself.

```
001 package ru.ifmo.hydro;
002 
003 import com.evelopers.unimod.core.stateworks.Event;
004 import com.evelopers.unimod.runtime.ControlledObject;
005 import com.evelopers.unimod.runtime.ModelEngine;
006 import com.evelopers.unimod.runtime.context.StateMachineContext;
007 import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
008 
009 
010 
011 public class Hydro implements ControlledObject, HydroHandle {
012 
013     public static final int MAX_ROTATIONS = 2000;
014     public static final int MIN_ROTATIONS = 3000;
015 
016     private  ModelEngine engine;
017 
018     private static Hydro o1;
019 
020     private HydroVisualizer hydroVisualizer;
021 
022     private volatile int rotations = 0;
023     public volatile int gear = 0;
024     private volatile boolean gasPedalPressed;
025     private volatile boolean switchedOn;
026 
027     private CustomThread customThread;
028 
029     public Hydro() {
030         System.out.println("Default constructor");
031     }
032 
033     private Hydro(ModelEngine engine) {
034         System.out.println("Constructor");
035         this.engine = engine;
036         hydroVisualizer = new HydroVisualizerForm("Эмуляция работы АКПП", this);
037         hydroVisualizer.setStatus(0);
038         customThread = new CustomThread();
039     }
040 
041     public static void init(ModelEngine engine){
042         o1 = new Hydro(engine);
043     }
044 
045     private void lowGear() {
046         notify(CustomEventProvider.e2);
047     }
048 
049     private void upGear() {
050         notify(CustomEventProvider.e1);
051     }
```

```
052
053     private void checkRotations() {
054         if (o1.rotations >= MAX_ROTATIONS) {
055             if (o1.gear > 0 && o1.gear < 5) {
056                 o1.upGear();
057             }
058         } else if (o1.rotations <= MIN_ROTATIONS) {
059             if (o1.gear > 1) {
060                 o1.lowGear();
061             }
062         }
063         if (o1.rotations < MIN_ROTATIONS) {
064             o1.rotations = MIN_ROTATIONS;
065         }
066         if (o1.rotations > MAX_ROTATIONS) {
067             o1.rotations = MAX_ROTATIONS;
068         }
069     }
070
071     public void notify(String event){
072         o1.engine.getEventManager().handle(new Event(event),
StateMachineContextImpl.create());
073     }
074
075     public void start() {
076         o1.notify(CustomEventProvider.e7);
077     }
078
079     public void stop() {
080         o1.notify(CustomEventProvider.e6);
081     }
082
083     public void gasPedalPressed() {
084         o1.gasPedalPressed = true;
085     }
086
087     public void gasPedalReleased() {
088         o1.gasPedalPressed = false;
089     }
090
091     public void switchToForward() {
092         notify(CustomEventProvider.e3);
093     }
094
095     public void switchToBackward() {
096         notify(CustomEventProvider.e4);
097     }
098
099     public void switchToNeutral() {
100         notify(CustomEventProvider.e5);
101     }
102
103     /**
104      * @unimod.action.descr is something correct
105      */
106     public boolean z3(StateMachineContext context) {
107         o1.gear = 1;
108         o1.hydroVisualizer.setGear(o1.gear);
109         return true;
110     }
111
112     /**
113      * @unimod.action.descr is something correct
114      */
115     public boolean z4(StateMachineContext context) {
116         o1.gear = 2;
117         o1.hydroVisualizer.setGear(o1.gear);
118         return true;
119     }
120
121     /**
```

```
122      * @unimod.action.descr is something correct
123      */
124     public boolean z5(StateMachineContext context) {
125         o1.gear = 3;
126         o1.hydroVisualizer.setGear(o1.gear);
127         return true;
128     }
129 
130     /**
131      * @unimod.action.descr is something correct
132      */
133     public boolean z6(StateMachineContext context) {
134         o1.gear = 4;
135         o1.hydroVisualizer.setGear(o1.gear);
136         return true;
137     }
138 
139     /**
140      * @unimod.action.descr is something correct
141      */
142     public boolean z7(StateMachineContext context) {
143         o1.gear = 5;
144         o1.hydroVisualizer.setGear(o1.gear);
145         return true;
146     }
147 
148     /**
149      * @unimod.action.descr is something correct
150      */
151     public boolean z8(StateMachineContext context) {
152         o1.gear = -1;
153         o1.hydroVisualizer.setGear(o1.gear);
154         return true;
155     }
156 
157     /**
158      * @unimod.action.descr Starting the engine
159      */
160     public boolean z1(StateMachineContext context) {
161         o1.switchedOn = true;
162         o1.rotations = MIN_ROTATIONS;
163         o1.gear = 0;
164         o1.hydroVisualizer.setGear(o1.gear);
165         o1.hydroVisualizer.setStatus(1);
166 
167         if (!o1.customThread.isAlive()) {
168             o1.customThread.start();
169         }
170         return false;
171     }
172 
173     /**
174      * @unimod.action.descr Stopping the engine
175      */
176     public boolean z2(StateMachineContext context) {
177         o1.switchedOn = false;
178         o1.rotations = 0;
179         o1.hydroVisualizer.setRotations(o1.rotations);
180         o1.hydroVisualizer.setStatus(0);
181         System.exit(0);
182         return false;
183     }
184 
185     public class CustomThread extends Thread {
186         public void run() {
187             System.out.println("In the customThread.run");
188             while (o1.switchedOn) {
189                 if (o1.gasPedalPressed) {
190                     o1.rotations++;
191                 } else {
192                     o1.rotations--;
```

```
193                 }
194                 if (o1.rotations < MIN_ROTATIONS) {
195                     o1.rotations = MIN_ROTATIONS;
196                 }
197                 if (o1.gear != 0) {
198                     checkRotations();
199                 }
200                 o1.hydroVisualizer.setRotations(o1.rotations);
201                 try {
202                     Thread.sleep(10);
203                 } catch (InterruptedException e) {
204                     e.printStackTrace();
205                 }
206             }
207 
208         }
209     }
210 
211     /**
212      * @unimod.action.descr Low rotations to MIN_ROTATIONS value
213      */
214     public boolean z9(StateMachineContext context) {
215         o1.rotations = MIN_ROTATIONS;
216         return true;
217     }
218 
219     /**
220      * @unimod.action.descr Up rotations to MAX_ROTATIONS value
221      */
222     public boolean z10(StateMachineContext context) {
223         o1.rotations = MAX_ROTATIONS;
224         return true;
225     }
226 }
```

Application 3. Source code of *Java*-class representing the automata

```
001 package ru.ifmo.hydro;
002 
003 /**
004  * This file was generated from model [Model1] on [Tue Mar 07 14:21:15 MSK 2006].
005  * Do not change content of this file.
006  */
007 
008 //
009 
010 import java.util.*;
011 
012 import com.evelopers.common.exception.*;
013 import com.evelopers.unimod.core.stateworks.*;
014 import com.evelopers.unimod.runtime.*;
015 import com.evelopers.unimod.runtime.context.*;
016 
017 
018 public class Model1EventProcessor extends AbstractEventProcessor {
019 
020     private ModelStructure modelStructure;
021 
022     private static final int A1 = 1;
023 
024     private int decodeStateMachine(String sm) {
025 
026         if ("A1".equals(sm)) {
027             return A1;
028         }
029 
```

```
030         return -1;
031     }
032 
033     private A1EventProcessor _A1;
034 
035     public Model1EventProcessor() {
036         modelStructure = new Model1ModelStructure();
037 
038         _A1 = new A1EventProcessor();
039     }
040 
041     public ModelStructure getModelStructure() {
042         return modelStructure;
043     }
044 
045     public void setControlledObjectsMap(ControlledObjectsMap controlledObjectsMap) {
046         super.setControlledObjectsMap(controlledObjectsMap);
047 
048         _A1.init(controlledObjectsMap);
049     }
050 
051     protected StateMachineConfig process(
052             Event event, StateMachineContext context,
053             StateMachinePath path, StateMachineConfig config) throws SystemException
{
054 
055         // get state machine from path
056         int sm = decodeStateMachine(path.getStateMachine());
057 
058         try {
059             switch (sm) {
060                 case A1:
061                     return _A1.process(event, context, path, config);
062                 default:
063                     throw new EventProcessorException("Unknown state machine [" +
path.getStateMachine() + "]");
064             }
065         } catch (Exception e) {
066             if (e instanceof SystemException) {
067                 throw (SystemException)e;
068             } else {
069                 throw new SystemException(e);
070             }
071         }
072     }
073 
074     protected StateMachineConfig transiteToStableState(
075             StateMachineContext context,
076             StateMachinePath path, StateMachineConfig config) throws SystemException
{
077 
078         // get state machine from path
079         int sm = decodeStateMachine(path.getStateMachine());
080 
081         try {
082             switch (sm) {
083                 case A1:
084                     return _A1.transiteToStableState(context, path, config);
085                 default:
086                     throw new EventProcessorException("Unknown state machine [" +
path.getStateMachine() + "]");
087             }
088         } catch (Exception e) {
089             if (e instanceof SystemException) {
090                 throw (SystemException)e;
091             } else {
092                 throw new SystemException(e);
093             }
094         }
095     }
096 
```

```
097     private class Model1ModelStructure implements ModelStructure {
098 
099         private Map configManagers = new HashMap();
100 
101         private Model1ModelStructure() {
102             configManagers.put("A1", new
com.evelopers.unimod.runtime.config.DistinguishConfigManager());
103         }
104 
105         public StateMachinePath getRootPath() throws EventProcessorException {
106             return new StateMachinePath("A1");
107         }
108 
109         public StateMachineConfigManager getConfigManager(String stateMachine) throws
EventProcessorException {
110             return (StateMachineConfigManager)configManagers.get(stateMachine);
111         }
112 
113         public StateMachineConfig getTopConfig(String stateMachine) throws
EventProcessorException {
114             int sm = decodeStateMachine(stateMachine);
115 
116             switch (sm) {
117                 case A1:
118                     return new StateMachineConfig("Top");
119                 default:
120                     throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
121             }
122         }
123 
124         public boolean isFinal(String stateMachine, StateMachineConfig config) throws
EventProcessorException {
125             // get state machine from path
126             int sm = decodeStateMachine(stateMachine);
127             int state;
128 
129             switch (sm) {
130                 case A1:
131                     state = _A1.decodeState(config.getActiveState());
132                     switch (state) {
133                         case A1EventProcessor.s2:
134                             return true;
135                         default:
136                             return false;
137                     }
138                 default:
139                     throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
140             }
141         }
142     }
143 
144 
145 
146 
147 
148 
149 
150 
151     private class A1EventProcessor {
152 
153         // states
154         private static final int Top = 1;
155         private static final int s5 = 2;
156         private static final int Sleep = 3;
157         private static final int s1 = 4;
158         private static final int s9 = 5;
159         private static final int s7 = 6;
160         private static final int Gear1 = 7;
161         private static final int s4 = 8;
```

```
162         private static final int s2 = 9;
163         private static final int s10 = 10;
164         private static final int s8 = 11;
165 
166         private int decodeState(String state) {
167 
168             if ("Top".equals(state)) {
169                 return Top;
170             } else
171 
172                 if ("s5".equals(state)) {
173                     return s5;
174                 } else
175 
176                     if ("Sleep".equals(state)) {
177                         return Sleep;
178                     } else
179 
180                         if ("s1".equals(state)) {
181                             return s1;
182                         } else
183 
184                             if ("s9".equals(state)) {
185                                 return s9;
186                             } else
187 
188                                 if ("s7".equals(state)) {
189                                     return s7;
190                                 } else
191 
192                                     if ("Gear1".equals(state)) {
193                                         return Gear1;
194                                     } else
195 
196                                         if ("s4".equals(state)) {
197                                             return s4;
198                                         } else
199 
200                                             if ("s2".equals(state)) {
201                                                 return s2;
202                                             } else
203 
204                                                 if ("s10".equals(state)) {
205                                                     return s10;
206                                                 } else
207 
208                                                     if ("s8".equals(state)) {
209                                                         return s8;
210                                                     }
211 
212             return -1;
213         }
214 
215         // events
216         private static final int e4 = 1;
217         private static final int e2 = 2;
218         private static final int e7 = 3;
219         private static final int e5 = 4;
220         private static final int e1 = 5;
221         private static final int e3 = 6;
222         private static final int e6 = 7;
223 
224         private int decodeEvent(String event) {
225 
226             if ("e4".equals(event)) {
227                 return e4;
228             } else
229 
230                 if ("e2".equals(event)) {
231                     return e2;
232                 } else
```

```
233
234                     if ("e7".equals(event)) {
235                         return e7;
236                     } else
237
238                         if ("e5".equals(event)) {
239                             return e5;
240                         } else
241
242                             if ("e1".equals(event)) {
243                                 return e1;
244                             } else
245
246                                 if ("e3".equals(event)) {
247                                     return e3;
248                                 } else
249
250                                     if ("e6".equals(event)) {
251                                         return e6;
252                                     }
253
254             return -1;
255         }
256
257         private ru.ifmo.hydro.Hydro o1;
258
259         private void init(ControlledObjectsMap controlledObjectsMap) {
260             o1 = (ru.ifmo.hydro.Hydro)controlledObjectsMap.getControlledObject("o1");
261         }
262
263         private StateMachineConfig process(Event event, StateMachineContext context,
StateMachinePath path, StateMachineConfig config) throws Exception {
264             config = lookForTransition(event, context, path, config);
265
266             config = transiteToStableState(context, path, config);
267
268             // execute included state machines
269             executeSubmachines(event, context, path, config);
270
271             return config;
272         }
273
274         private void executeSubmachines(Event event, StateMachineContext context,
StateMachinePath path, StateMachineConfig config) throws Exception {
275             int state = decodeState(config.getActiveState());
276
277             while (true) {
278                 switch (state) {
279                     case s5:
280
281                         return;
282                     case Sleep:
283
284                         return;
285                     case s1:
286
287                         return;
288                     case s9:
289
290                         return;
291                     case s7:
292
293                         return;
294                     case Gear1:
295
296                         return;
297                     case s4:
298
299                         return;
300                     case s2:
301
```

```
302                         return;
303                     case s10:
304 
305                         return;
306                     case s8:
307 
308                         return;
309                     default:
310                         throw new EventProcessorException("State with name [" +
config.getActiveState() + "] is unknown for state machine [A1]");
311                 }
312             }
313         }
314 
315         private StateMachineConfig transiteToStableState(StateMachineContext context,
StateMachinePath path, StateMachineConfig config) throws Exception {
316 
317             int s = decodeState(config.getActiveState());
318             Event event;
319 
320             switch (s) {
321                 case Top:
322 
323 
324                     fireComeToState(context, path, "s1");
325 
326                     // s1->Sleep [true]/
327                     event = Event.NO_EVENT;
328                     fireTransitionFound(context, path, "s1", event,
"s1#Sleep##true");
329 
330 
331                     fireComeToState(context, path, "Sleep");
332 
333                     // Sleep []
334 
335                     return new StateMachineConfig("Sleep");
336             }
337 
338             return config;
339         }
340 
341         private StateMachineConfig lookForTransition(Event event, StateMachineContext
context, StateMachinePath path, StateMachineConfig config) throws Exception {
342 
343 
344 
345 
346 
347             BitSet calculatedInputActions = new BitSet(0);
348 
349             int s = decodeState(config.getActiveState());
350             int e = decodeEvent(event.getName());
351 
352             while (true) {
353                 switch (s) {
354                     case s5:
355 
356 
357                         switch (e) {
358                             case e5:
359 
360                                 // s5->s4 e5[true]/
361 
362                                 fireTransitionCandidate(context, path, "s5", event,
"s5#s4#e5#true");
363 
364 
365 
366 
367 
```

```
368                                 fireTransitionFound(context, path, "s5", event,
"s5#s4#e5#true");
369
370
371                                 fireComeToState(context, path, "s4");
372
373                                 // s4 [o1.z1]
374                                 fireBeforeOutputActionExecution(context, path,
"s5#s4#e5#true", "o1.z1");
375
376                                 o1.z1(context);
377
378                                 fireAfterOutputActionExecution(context, path,
"s5#s4#e5#true", "o1.z1");
379                                 return new StateMachineConfig("s4");
380
381
382                             default:
383
384
385                                 // transition not found
386                                 return config;
387                         }
388
389                     case Sleep:
390
391
392                         switch (e) {
393                             case e7:
394
395                                 // Sleep->s4 e7[true]/
396
397                                 fireTransitionCandidate(context, path, "Sleep",
event, "Sleep#s4#e7#true");
398
399
400
401
402
403                                 fireTransitionFound(context, path, "Sleep", event,
"Sleep#s4#e7#true");
404
405
406                                 fireComeToState(context, path, "s4");
407
408                                 // s4 [o1.z1]
409                                 fireBeforeOutputActionExecution(context, path,
"Sleep#s4#e7#true", "o1.z1");
410
411                                 o1.z1(context);
412
413                                 fireAfterOutputActionExecution(context, path,
"Sleep#s4#e7#true", "o1.z1");
414                                 return new StateMachineConfig("s4");
415
416
417                             default:
418
419
420                                 // transition not found
421                                 return config;
422                         }
423
424                     case s9:
425
426
427                         switch (e) {
428                             case e2:
429
430                                 // s9->s10 e2[true]/o1.z10
431
```

```
                                fireTransitionCandidate(context, path, "s9", event,
"s9#s10#e2#true");
                                fireTransitionFound(context, path, "s9", event,
"s9#s10#e2#true");
                                fireBeforeOutputActionExecution(context, path,
"s9#s10#e2#true", "o1.z10");
                                o1.z10(context);
                                fireAfterOutputActionExecution(context, path,
"s9#s10#e2#true", "o1.z10");
                                fireComeToState(context, path, "s10");
                                // s10 [o1.z6]
                                fireBeforeOutputActionExecution(context, path,
"s9#s10#e2#true", "o1.z6");
                                o1.z6(context);
                                fireAfterOutputActionExecution(context, path,
"s9#s10#e2#true", "o1.z6");
                                return new StateMachineConfig("s10");

                            default:

                                // transition not found
                                return config;
                        }

                    case s7:

                        switch (e) {
                            case e2:

                                // s7->Gear1 e2[true]/o1.z10

                                fireTransitionCandidate(context, path, "s7", event,
"s7#Gear1#e2#true");
                                fireTransitionFound(context, path, "s7", event,
"s7#Gear1#e2#true");
                                fireBeforeOutputActionExecution(context, path,
"s7#Gear1#e2#true", "o1.z10");
                                o1.z10(context);
                                fireAfterOutputActionExecution(context, path,
"s7#Gear1#e2#true", "o1.z10");
                                fireComeToState(context, path, "Gear1");
                                // Gear1 [o1.z3]
                                fireBeforeOutputActionExecution(context, path,
"s7#Gear1#e2#true", "o1.z3");
                                o1.z3(context);
```

```
492
493                                 fireAfterOutputActionExecution(context, path,
"s7#Gear1#e2#true", "o1.z3");
494                                 return new StateMachineConfig("Gear1");
495
496
497                             case e1:
498
499                                 // s7->s8 e1[true]/o1.z9
500
501                                 fireTransitionCandidate(context, path, "s7", event,
"s7#s8#e1#true");
502
503
504
505
506
507                                 fireTransitionFound(context, path, "s7", event,
"s7#s8#e1#true");
508
509                                 fireBeforeOutputActionExecution(context, path,
"s7#s8#e1#true", "o1.z9");
510
511                                 o1.z9(context);
512
513                                 fireAfterOutputActionExecution(context, path,
"s7#s8#e1#true", "o1.z9");
514
515                                 fireComeToState(context, path, "s8");
516
517                                 // s8 [o1.z5]
518                                 fireBeforeOutputActionExecution(context, path,
"s7#s8#e1#true", "o1.z5");
519
520                                 o1.z5(context);
521
522                                 fireAfterOutputActionExecution(context, path,
"s7#s8#e1#true", "o1.z5");
523                                 return new StateMachineConfig("s8");
524
525
526                             default:
527
528
529                                 // transition not found
530                                 return config;
531                         }
532
533                     case Gear1:
534
535
536                         switch (e) {
537                             case e5:
538
539                                 // Gear1->s4 e5[true]/
540
541                                 fireTransitionCandidate(context, path, "Gear1",
event, "Gear1#s4#e5#true");
542
543
544
545
546
547                                 fireTransitionFound(context, path, "Gear1", event,
"Gear1#s4#e5#true");
548
549
550                                 fireComeToState(context, path, "s4");
551
552                                 // s4 [o1.z1]
553                                 fireBeforeOutputActionExecution(context, path,
```

```
"Gear1#s4#e5#true", "o1.z1");
554
555                                 o1.z1(context);
556
557                                 fireAfterOutputActionExecution(context, path,
"Gear1#s4#e5#true", "o1.z1");
558                                 return new StateMachineConfig("s4");
559
560
561                             case e1:
562
563                                 // Gear1->s7 e1[true]/o1.z9
564
565                                 fireTransitionCandidate(context, path, "Gear1",
event, "Gear1#s7#e1#true");
566
567
568
569
570
571                                 fireTransitionFound(context, path, "Gear1", event,
"Gear1#s7#e1#true");
572
573                                 fireBeforeOutputActionExecution(context, path,
"Gear1#s7#e1#true", "o1.z9");
574
575                                 o1.z9(context);
576
577                                 fireAfterOutputActionExecution(context, path,
"Gear1#s7#e1#true", "o1.z9");
578
579                                 fireComeToState(context, path, "s7");
580
581                                 // s7 [o1.z4]
582                                 fireBeforeOutputActionExecution(context, path,
"Gear1#s7#e1#true", "o1.z4");
583
584                                 o1.z4(context);
585
586                                 fireAfterOutputActionExecution(context, path,
"Gear1#s7#e1#true", "o1.z4");
587                                 return new StateMachineConfig("s7");
588
589
590                             default:
591
592
593                                 // transition not found
594                                 return config;
595                         }
596
597                     case s4:
598
599
600                         switch (e) {
601                             case e4:
602
603                                 // s4->s5 e4[true]/
604
605                                 fireTransitionCandidate(context, path, "s4", event,
"s4#s5#e4#true");
606
607
608
609
610
611                                 fireTransitionFound(context, path, "s4", event,
"s4#s5#e4#true");
612
613
614                                 fireComeToState(context, path, "s5");
```

```
615
616                                 // s5 [o1.z8]
617                                 fireBeforeOutputActionExecution(context, path,
"s4#s5#e4#true", "o1.z8");
618
619                                 o1.z8(context);
620
621                                 fireAfterOutputActionExecution(context, path,
"s4#s5#e4#true", "o1.z8");
622                                 return new StateMachineConfig("s5");
623
624
625                             case e3:
626
627                                 // s4->Gear1 e3[true]/
628
629                                 fireTransitionCandidate(context, path, "s4", event,
"s4#Gear1#e3#true");
630
631
632
633
634
635                                 fireTransitionFound(context, path, "s4", event,
"s4#Gear1#e3#true");
636
637
638                                 fireComeToState(context, path, "Gear1");
639
640                                 // Gear1 [o1.z3]
641                                 fireBeforeOutputActionExecution(context, path,
"s4#Gear1#e3#true", "o1.z3");
642
643                                 o1.z3(context);
644
645                                 fireAfterOutputActionExecution(context, path,
"s4#Gear1#e3#true", "o1.z3");
646                                 return new StateMachineConfig("Gear1");
647
648
649                             case e6:
650
651                                 // s4->s2 e6[true]/o1.z2
652
653                                 fireTransitionCandidate(context, path, "s4", event,
"s4#s2#e6#true");
654
655
656
657
658
659                                 fireTransitionFound(context, path, "s4", event,
"s4#s2#e6#true");
660
661                                 fireBeforeOutputActionExecution(context, path,
"s4#s2#e6#true", "o1.z2");
662
663                                 o1.z2(context);
664
665                                 fireAfterOutputActionExecution(context, path,
"s4#s2#e6#true", "o1.z2");
666
667                                 fireComeToState(context, path, "s2");
668
669                                 // s2 []
670                                 return new StateMachineConfig("s2");
671
672
673                             default:
674
675
```

```
676                                 // transition not found
677                                 return config;
678                             }
679
680                         case s10:
681
682
683                             switch (e) {
684                                 case e2:
685
686                                     // s10->s8 e2[true]/o1.z10
687
688                                     fireTransitionCandidate(context, path, "s10", event,
"s10#s8#e2#true");
689
690
691
692
693
694                                     fireTransitionFound(context, path, "s10", event,
"s10#s8#e2#true");
695
696                                     fireBeforeOutputActionExecution(context, path,
"s10#s8#e2#true", "o1.z10");
697
698                                     o1.z10(context);
699
700                                     fireAfterOutputActionExecution(context, path,
"s10#s8#e2#true", "o1.z10");
701
702                                     fireComeToState(context, path, "s8");
703
704                                     // s8 [o1.z5]
705                                     fireBeforeOutputActionExecution(context, path,
"s10#s8#e2#true", "o1.z5");
706
707                                     o1.z5(context);
708
709                                     fireAfterOutputActionExecution(context, path,
"s10#s8#e2#true", "o1.z5");
710                                     return new StateMachineConfig("s8");
711
712
713                                 case e1:
714
715                                     // s10->s9 e1[true]/o1.z9
716
717                                     fireTransitionCandidate(context, path, "s10", event,
"s10#s9#e1#true");
718
719
720
721
722
723                                     fireTransitionFound(context, path, "s10", event,
"s10#s9#e1#true");
724
725                                     fireBeforeOutputActionExecution(context, path,
"s10#s9#e1#true", "o1.z9");
726
727                                     o1.z9(context);
728
729                                     fireAfterOutputActionExecution(context, path,
"s10#s9#e1#true", "o1.z9");
730
731                                     fireComeToState(context, path, "s9");
732
733                                     // s9 [o1.z7]
734                                     fireBeforeOutputActionExecution(context, path,
"s10#s9#e1#true", "o1.z7");
735
```

```
736                                 o1.z7(context);
737
738                                 fireAfterOutputActionExecution(context, path,
"s10#s9#e1#true", "o1.z7");
739                                 return new StateMachineConfig("s9");
740
741
742                             default:
743
744
745                                 // transition not found
746                                 return config;
747                         }
748
749                     case s8:
750
751
752                         switch (e) {
753                             case e2:
754
755                                 // s8->s7 e2[true]/o1.z9
756
757                                 fireTransitionCandidate(context, path, "s8", event,
"s8#s7#e2#true");
758
759
760
761
762
763                                 fireTransitionFound(context, path, "s8", event,
"s8#s7#e2#true");
764
765                                 fireBeforeOutputActionExecution(context, path,
"s8#s7#e2#true", "o1.z9");
766
767                                 o1.z9(context);
768
769                                 fireAfterOutputActionExecution(context, path,
"s8#s7#e2#true", "o1.z9");
770
771                                 fireComeToState(context, path, "s7");
772
773                                 // s7 [o1.z4]
774                                 fireBeforeOutputActionExecution(context, path,
"s8#s7#e2#true", "o1.z4");
775
776                                 o1.z4(context);
777
778                                 fireAfterOutputActionExecution(context, path,
"s8#s7#e2#true", "o1.z4");
779                                 return new StateMachineConfig("s7");
780
781
782                             case e1:
783
784                                 // s8->s10 e1[true]/o1.z9
785
786                                 fireTransitionCandidate(context, path, "s8", event,
"s8#s10#e1#true");
787
788
789
790
791
792                                 fireTransitionFound(context, path, "s8", event,
"s8#s10#e1#true");
793
794                                 fireBeforeOutputActionExecution(context, path,
"s8#s10#e1#true", "o1.z9");
795
796                                 o1.z9(context);
```

```
797
798                                 fireAfterOutputActionExecution(context, path,
"s8#s10#e1#true", "o1.z9");
799
800                                 fireComeToState(context, path, "s10");
801
802                                 // s10 [o1.z6]
803                                 fireBeforeOutputActionExecution(context, path,
"s8#s10#e1#true", "o1.z6");
804
805                                 o1.z6(context);
806
807                                 fireAfterOutputActionExecution(context, path,
"s8#s10#e1#true", "o1.z6");
808                                 return new StateMachineConfig("s10");
809
810
811                             default:
812
813
814                                 // transition not found
815                                 return config;
816                         }
817
818                     default:
819                         throw new EventProcessorException("Incorrect stable state ["
+ config.getActiveState() + "] in state machine [A1]");
820                 }
821             }
822         }
823
824
825     }
826
827     private static boolean isInputActionCalculated(BitSet calculatedInputActions, int
k) {
828         boolean b = calculatedInputActions.get(k);
829
830         if (!b) {
831             calculatedInputActions.set(k);
832         }
833
834         return b;
835     }
836
837 }
```

Application 4. Source code of the application interfaces

HydroVisualizerForm.java

```
001 package ru.ifmo.hydro;
002
003 import javax.swing.*;
004 import javax.swing.border.Border;
005 import javax.swing.border.TitledBorder;
006 import java.awt.*;
007 import java.awt.event.ActionEvent;
008 import java.awt.event.ActionListener;
009 import java.awt.event.MouseListener;
010 import java.awt.event.MouseEvent;
011
012 public class HydroVisualizerForm extends JFrame implements HydroVisualizer,
ActionListener, MouseListener {
013     public static final long serialVersionUID = 1234567890521;
014     public static final String OFF = "ВЫКЛ";
015     public static final String ON = "ВКЛ";
```

```
016     public static final String FORWARD = "ВПЕРЕД";
017     public static final String BACKWARD = "НАЗАД";
018     public static final String NEUTRAL = "НЕЙТРАЛЬ";
019     public static final String GAS = "ГАЗ";
020     public static final String HYDRO_STATE = "НАПРАВЛЕНИЕ";
021 
022     private JButton btnRotations;
023     private JButton btnOn;
024     private JButton btnOff;
025     private JRadioButton rbnForward;
026     private JRadioButton rbnBackward;
027     private JRadioButton rbnNeutral;
028     private TachometerPanel tachometerPanel;
029 
030     private Hydro hydro;
031 
032     public HydroVisualizerForm(String title, Hydro hydro) {
033         super(title);
034 
035         this.hydro = hydro;
036 
037         setSize(400, 300);
038 
039         getContentPane().setLayout(new GridBagLayout());
040         GridBagConstraints c = new GridBagConstraints();
041         c.insets = new Insets(2, 5, 2, 5);
042 
043         c.gridx = 0;
044         c.gridy = 0;
045         btnOn = new JButton(ON);
046         btnOn.addActionListener(this);
047         getContentPane().add(btnOn, c);
048 
049         c.gridx = 1;
050         btnOff = new JButton(OFF);
051         btnOff.addActionListener(this);
052         getContentPane().add(btnOff, c);
053 
054         JPanel switchPanel = new JPanel(new GridBagLayout());
055         Border etched = BorderFactory.createEtchedBorder();
056         TitledBorder titledBorder = BorderFactory.createTitledBorder(etched,
HYDRO_STATE + ":");
057         switchPanel.setBorder(titledBorder);
058 
059         c.gridx = 0;
060         c.gridy = 1;
061         c.gridwidth = 2;
062         getContentPane().add(switchPanel, c);
063 
064         c.gridx = 2;
065         c.weightx = 1.0;
066         c.fill = GridBagConstraints.BOTH;
067         tachometerPanel = new TachometerPanel();
068         tachometerPanel.setBorder(BorderFactory.createEtchedBorder());
069         getContentPane().add(tachometerPanel, c);
070 
071 
072         rbnNeutral = new JRadioButton(NEUTRAL);
073         rbnNeutral.addActionListener(this);
074         rbnForward = new JRadioButton(FORWARD);
075         rbnForward.addActionListener(this);
076         rbnBackward = new JRadioButton(BACKWARD);
077         rbnBackward.addActionListener(this);
078         ButtonGroup buttonGroup = new ButtonGroup();
079         buttonGroup.add(rbnNeutral);
080         buttonGroup.add(rbnForward);
081         buttonGroup.add(rbnBackward);
082 
083         c.gridx = 0;
084         c.weightx = 0.0;
085         c.gridy = 0;
```

```
086         c.anchor = GridBagConstraints.WEST;
087         switchPanel.add(rbnForward, c);
088 
089         c.gridy = 1;
090         switchPanel.add(rbnBackward, c);
091 
092         c.gridy = 2;
093         switchPanel.add(rbnNeutral, c);
094 
095         c.gridy = 2;
096         c.gridwidth = 4;
097         c.anchor = GridBagConstraints.CENTER;
098         c.fill = GridBagConstraints.NONE;
099         c.weightx = 1.0;
100         btnRotations = new JButton(GAS);
101         btnRotations.addMouseListener(this);
102         getContentPane().add(btnRotations, c);
103 
104         setVisible(true);
105 
106         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
107     }
108 
109     public void setRotations(int rotations) {
110         tachometerPanel.setRotations(rotations);
111         repaint();
112     }
113 
114     public void setGear(int gear){
115         tachometerPanel.setGear(gear);
116     }
117 
118     public void setStatus(int status) {
119     }
120 
121     public void actionPerformed(ActionEvent ae) {
122         if (ae.getSource() == btnOn) {
123             hydro.start();
124         }
125         if (ae.getSource() == btnOff) {
126             hydro.stop();
127         }
128 
129         if (ae.getSource() == rbnForward) {
130             hydro.switchToForward();
131         }
132         if (ae.getSource() == rbnBackward) {
133             hydro.switchToBackward();
134         }
135         if (ae.getSource() == rbnNeutral) {
136             hydro.switchToNeutral();
137         }
138     }
139 
140     public void mouseClicked(MouseEvent e) {
141         //To change body of implemented methods use File | Settings | File Templates.
142     }
143 
144     public void mousePressed(MouseEvent e) {
145         hydro.gasPedalPressed();
146     }
147 
148     public void mouseReleased(MouseEvent e) {
149         hydro.gasPedalReleased();
150     }
151 
152     public void mouseEntered(MouseEvent e) {
153         //To change body of implemented methods use File | Settings | File Templates.
154     }
155 
156     public void mouseExited(MouseEvent e) {
```

```
157         //To change body of implemented methods use File | Settings | File Templates.
158     }
159 }
```

TachometrPanel.java

```
01 package ru.ifmo.hydro;
02
03 import javax.swing.*;
04 import java.awt.*;
05
06 /**
07  * Created by Andrey Vokin.
08  * User: admin
09  * Date: 28.12.2005
10  * Time: 1:00:40
11  */
12 public class TachometerPanel extends JPanel {
13
14     public static final long serialVersionUID = 1234567890561;
15
16     public static final int RADIUS_X = 50;
17     public static final int RADIUS_Y = 50;
18     public static final int CENTER_X = 125;
19     public static final int CENTER_Y = 90;
20
21     public static final String ROTATIONS = "Обороты";
22     public static final String GEAR = "передача";
23
24     /**
25      * Maximal value of the engine rotations.
26      */
27     public static final int MAX_ROTATIONS = 3000;
28
29     private JLabel lblRotations;
30
31     private int rotations;
32
33     private int gear;
34
35     public TachometerPanel() {
36         super();
37
38         lblRotations = new JLabel();
39         add(lblRotations);
40     }
41
42     public void drawLine(Graphics g, double angle) {
43         int y = CENTER_Y - (int) (Math.sin(Math.PI * angle / 180) * RADIUS_Y);
44         int x = CENTER_X - (int) (Math.cos(Math.PI * angle / 180) * RADIUS_X);
45         g.drawLine(CENTER_X, CENTER_Y, x, y);
46     }
47
48     public void paint(Graphics g) {
49         super.paint(g);
50         double angle = 1.0 * rotations / MAX_ROTATIONS * 180;
51         g.drawArc(CENTER_X - RADIUS_X, CENTER_Y - RADIUS_Y, 2 * RADIUS_X, 2 *
RADIUS_Y, 0, 180);
52         drawLine(g, angle);
53         g.drawOval(200, 150, 2, 2);
54     }
55
56     private void write() {
57         lblRotations.setText(ROTATIONS + ": " + rotations + ", " + GEAR + ": " +
gear);
58     }
59
60     public void setRotations(int rotations) {
61         this.rotations = rotations;
62         write();
```

```
63      }
64
65      public void setGear(int gear) {
66          this.gear = gear;
67          write();
68      }
69  }
```