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

Computer Technique Department

A.H. Kirakozov, A.A. Shalyto, B.R. Yaminov

Car Alarm Control System

Object-oriented programming with explicit state selection

Project documentation

This project has been developed in the context of the
"Foundation for open project documentation"

http://is.ifmo.ru/

Saint-Petersburg

2006

Table of contents

1. Introduction

This project is an example of designing a car alarm system with automata approach using instrumental tool *Unimod*.

When designing a control system it seems that it is a good idea to use automata approach. If a system is controlled by automata then by looking at them it is easy to understand system's behaviour. Moreover, if some bug is detected, it is much easier to find the reason in automata transition graphs than in source code.

Instrumental tool *Unimod* provides an easy way to build automata based systems. When using this tool firstly we have to describe system states and transitions between them. A system transits from one state to another when event happens. While it is transiting some actions are called. Entities that can generate events are called **event providers**, and those which can perform actions are called **controlled objects**. In such way the whole system is splitted into independent parts and so it becomes easier to maintain.

To sum up, if we want to design a system using *Unimod*, we have to describe event providers, controlled objects and automata which will react to event by transiting from one state to another and calling corresponding actions of controlled objects. Implementations of actions and event throwing are written in *Java*.

2. Problem definition

Our aim is to design and build a simple car alarm control system. An example of such system is taken from a car alarm manual [1]. Let's describe it.

User controls the system with remote control unit with 3 buttons. First button turns alarm on, second one turns it off. Third button is used for "silent mode". Alarm system has a hit sensor with 2 levels (easy or hard hit). Current state is represented by blinking LED.

When car gets an easy kick it gives a warning by blinking headlights and making sound 3 times. If car gets a hard kick, or two easy kicks within 5 seconds, it begins alarming, i.e. it blinks headlights and makes sound for 15 seconds.

When alarm is turned on it is confirmed by 1 blink and sound. When it is turned off 2 blinks and sounds are produced. Third button of remote control enables silent mode for 3 seconds. If user turns on or off alarm system during this time period, no confirmation is produced.

LED represents current state of system. If alarm is turned on then LED is blinking, otherwise it is off.

Figure1 represents graphical user interface of the program.

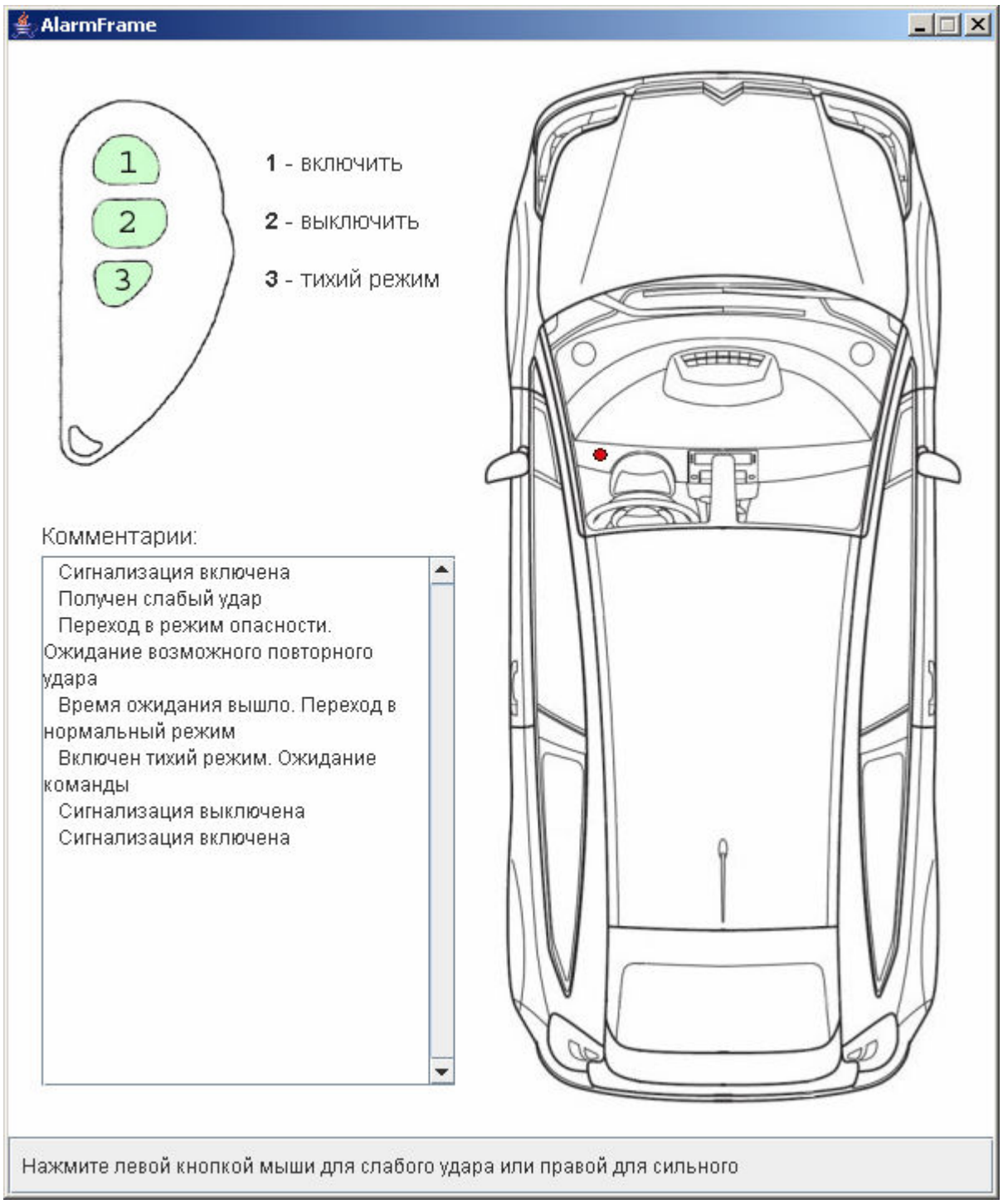

Figure 1: Graphical user interface of the program.

In the top left of the window on Figure 1 you can see remote control unit. You can click on its buttons. Also you can click on the car picture to simulate hits. Left-button click simulates easy hit and right-button is used for hard hit.

In the bottom of the window you can see a comment about an object which is under cursor of mouse at the moment. These objects are buttons and the car. Also in the car you can see red LED we described above. Comments about actions performed and states changed are printed into text area named “Комментарии”.

3. Formal script

Life cycle of the program can be divided into several states, in which program differently reacts to events. Let's describe a script of program's behaviour. States are indicated by `i` numbers and transitions are indicated by pairs `(i, j)` where transition is performed from state `i` to state `j`.

1. **Alarm system is off**

Initially alarm system is off.

(1, 2) Button "1" is pressed (event `e11`). Car lights blink one time (action `o1.z1`), LED begins to blink (`o4.z1`), all timers are stopped (`o2.z4`), and sound system is turned on (`o3.z6`). If sounds were on then a short sound is produced (`o3.z1`), otherwise all sounds are stopped (`o3.z5`).

2. **Alarm system is on**

(2, 3) An easy hit is received; the system gets into "Danger" state (`e21`). Lights blinks two times (`o1.z3`), a sound is performed (`o3.z3`). Countdown timer "2" is started for 5 seconds (`o2.z2`). If the car receives another hit during this period then system gets into "Alarming" state.

(2, 4) A hard hit is received; the system gets into "Alarming" state (`e22`). Lights begin to blink (`o1.z4`), alarming sounds are made (`o3.z4`), and countdown timer "1" is started for 15 seconds (`o2.z1`). System proceeds with alarming during this period.

3. **Danger**

(3, 4) An easy or hard hit is received (`e21`, `e22`). System gets into alarming state. Car lights begin to blink (`o1.z4`), alarming sounds are made (`o3.z4`), and countdown timer "1" is started for 15 seconds (`o2.z1`). System is in alarming state until the countdown finishes.

(3, 2) Countdown of timer number "2" has finished. System returns into regular state (`e32`).

4. **Alarming**

(4, 2) Countdown of timer "1" has finished. System returns into regular state (`e33`). Blinking of lights is stopped (`o1.z5`) and alarming sounds are stopped too (`o3.z5`).

4. Implementation in automata

Let's divide the system into objects. It is natural to make remote control and hit sensor as event providers and car lights, sound system and LED as controlled objects.

Figure 2 represents connectivity diagram.

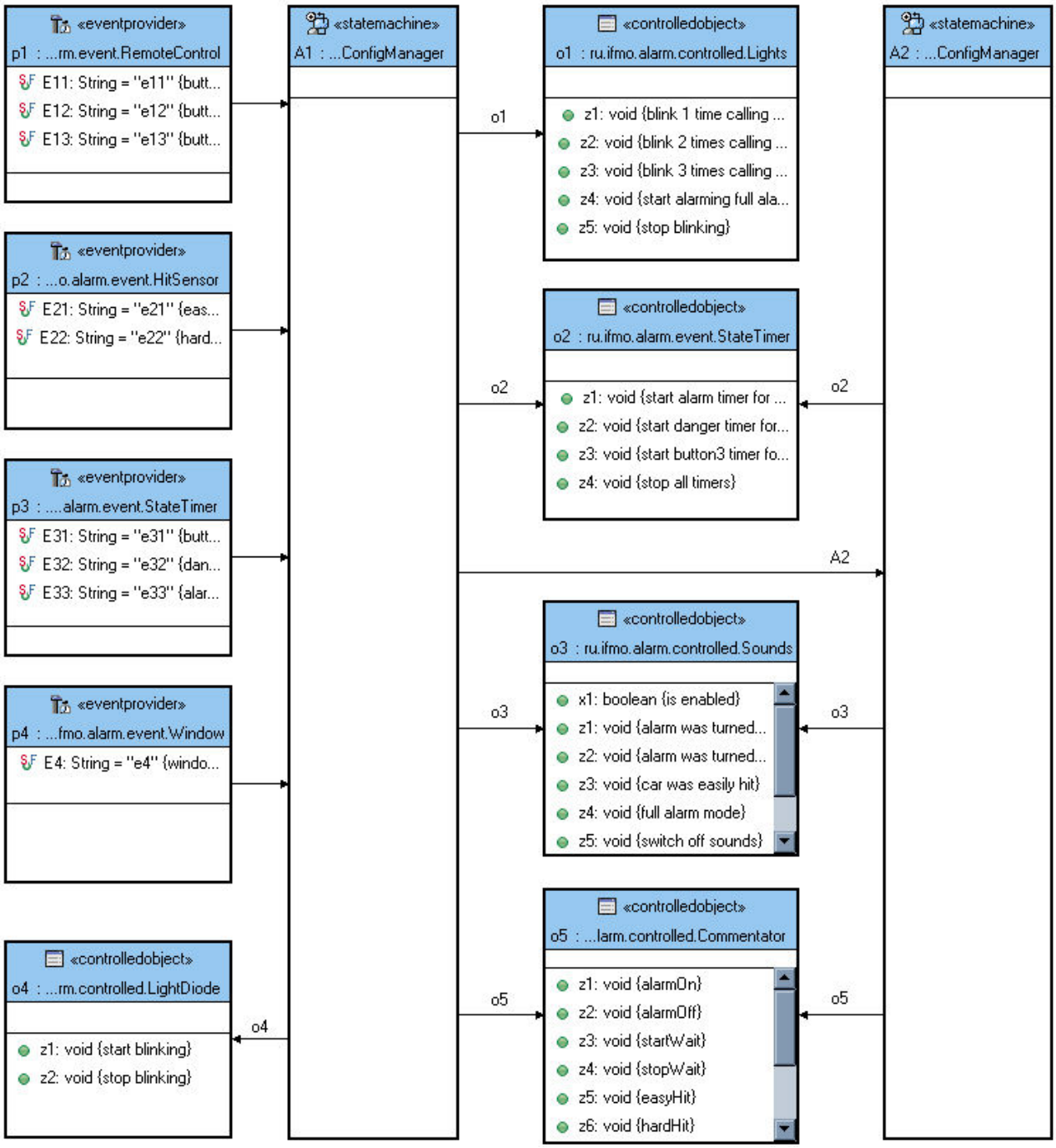

Figure 2: Connectivity diagram.

Also there is a timer as event provider (p3), a controlled object for starting timer (o2) and a controlled object for printing comments into comments field (o5).

4.1. Automata

As you can see on the connectivity diagram, we have two automata. *A1* is main one and *A2* is nested into several states of *A1*. We use nested automaton because it drastically simplifies transition diagrams of the system.

Figure 3 represents *A1* transitions diagram.

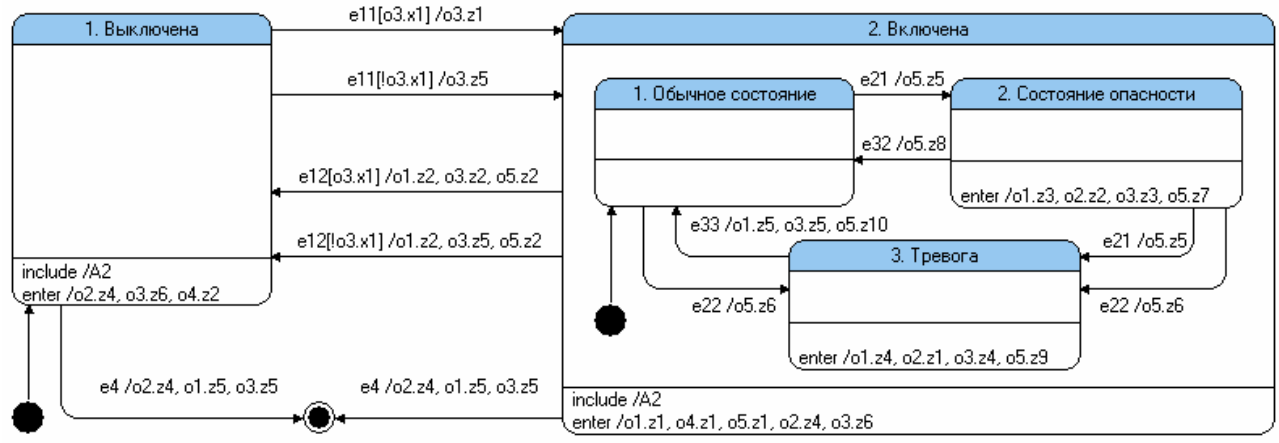

Figure 3: *A1* transition diagram.

The main states are "*1. Выключена"* and "*2. Включена"*. The first one is an initial one. Transitions between these states are activated when events of pressing buttons "1" and "2" happen. If sound system is on then when transiting, sounds are made. Sound system is turned on and off by *A2* automaton which is included in both main states.

The state "*2. Включена"* is compound. It contains a part of automaton which reacts to hits and timers. When an easy hit has been made, automaton transits from "*1. Обычное состояние"* to "*2. Состояние опасности"*. If no more hits are made during 5 seconds, then automaton returns back. Otherwise it transits into "*3. Тревога"*, and alarms for 15 seconds.

Using container state provides us a way to simplify transition diagram. For example, we can turn off alarm system from any state while we did not have to make "turning off" transitions from states "*1. Обычное состояние"*, "*2. Состояние опасности"* and "*3. Тревога"*.

The final state is reached when application's window is closed.

An *A2* automaton turns on and off sound system. When sound system is on any sound which is tried to be created is created and when it is off no sounds are made. Figure 4 represents transition diagram of automaton A2.

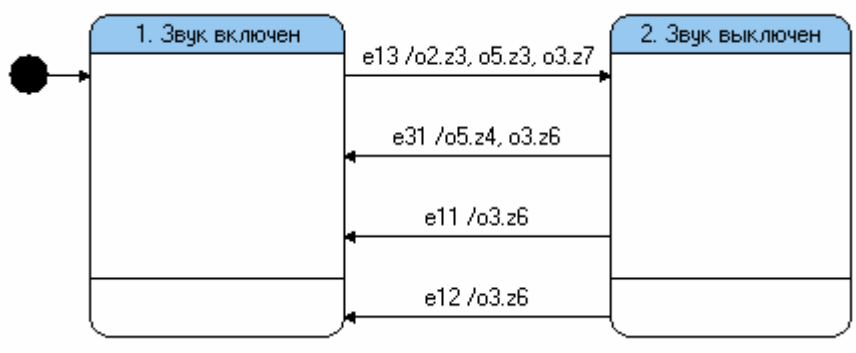

Figure 4: *A2* transition diagram.

Sounds are turned off by button "3". At the same time a timer is started, and in 3 seconds automaton transits back. Also it transits back when button "1" or "2" are pressed. So if user wants to turn alarm system on silently then user should press button "3" and within 3 seconds press button "1". Then no sounds would be made.

4.2. Event providers

In this section we will desribe event providers.

Event provider *p1*

This event provider generates events made by remote control. These events are:

e11 – button “1” was pressed;
e12 – button “2” was pressed;
e13 – button “3” was pressed.

Event provider *p2*

This event provider generates events made by hit sensor.

e21 – easy hit was detected;
e22 – hard hit was detected.

Event provider *p3*

This event provider is called to start countdown timer for different periods of time. When countdown is finished it generates one of three events (an event corresponding to period of time). The timer is started by controlled object “o2”.

e31 – countdown for timer “1” has finished (this timer is used for turning off sounds, 3 seconds);

e32 – countdown for timer “2” has finished (this timer is used in state “*2. Состояние опасности”* of automaton *A2*, 5 seconds);

e33 – countdown for timer “3” has finished (this timer is used in state “*3. Тревога”* of automaton *A2*, 15 seconds).

Event provider *p4*

This event provider generates event when application window is closed.

e4 – application window has been closed.

4.3. Controlled objects

Controlled object *o1*

This controlled object represents actions made by car lights.

z1 – blink one time;
z2 – blink two times;
z3 – blink three times;
z4 – begin alarming blinking;
z5 – stop any blinkings.

Controlled object *o2*

This controlled object starts timers from “p3”.

z1 – start timer “3” for 15 seconds;
z2 – start timer “2” for 5 seconds;
z3 – start timer “1” for 3 seconds;
z4 – stop all timers.

Controlled object *o3*

This controlled object represents actions of sound system.

x1 – boolean variable showing if sound are on;
z1 – generate a sound of turning on alarm system;
z2 – generate a sound of turning off alarm system;
z3 – generate a sound of danger (after easy hit);
z4 – begin alarming;
z5 – stop all sounds;
z6 – turn sound system on (allow making sounds);
z7 – turn sound system off (forbid making sounds).

Controlled object *o4*

This controlled object represents action of LED showing current condition of alarm system.

z1 – begin blinking;
z2 – stop blinking.

Controlled object *o5*

This controlled object prints comments upon actions of alarm system into comments field.

z1 – print comment when alarm system is turned on;
z2 – print comment when alarm system is turned off;
z3 – print comment when sounds are turned off;
z4 – print comment when countdown finishes and sounds are turned on again;
z5 – print comment when an easy hit was detected;
z6 – print comment when a hard hit was detected;
z7 – print comment when *A1* transits into “*2. Состояние опасности”*;
z8 – print comment when countdown for timer “2” has finished;
z9 – print comment when alarming is turned on;
z10 – print comment when alarming is finished.

5. Implementation

The system is implemented using *Unimod* tool. *Unimod* is a plugin for IDE *Eclipse* which provides an easy way to make automata based applications. Using *Unimod* you can make connectivity diagram and automata transitions diagrams (Figures 2 – 4). Then you only have to implement actions of event providers and controlled objects in *Java*.

There are two ways for making and running final working application in *Unimod*: *interpretating* and *compilating*.

5.1. Interpretating

Connectivity diagram and automata are coded into *XML* file by *Unimod*. You can see *XML* generated for our application in Appendix 2. When using interpretating way of running application, every time *Unimod* parses this *XML* file and interpretates automata. So to run the application you need all *Unimod* libraries, *XML* file generated and compiled classes implementing event providers and controlled objects. When application is running *Unimod* logs are printed into console.

You can see such logs for the following actions: run the application, press button "1" and close the window.

```
18:18:29,890 INFO  [Run] Start event [e11] processing. In state [/A1:Top]
18:18:29,890 INFO  [Run] Transition to go found [s1#1. Выключена##true]
18:18:29,890 INFO  [Run] Start on-enter action [o2.z4] execution
18:18:29,890 INFO  [Run] Finish on-enter action [o2.z4] execution
18:18:29,890 INFO  [Run] Start on-enter action [o3.z6] execution
18:18:30,953 INFO  [Run] Finish on-enter action [o3.z6] execution
18:18:30,968 INFO  [Run] Start on-enter action [o4.z2] execution
18:18:30,984 INFO  [Run] Finish on-enter action [o4.z2] execution
18:18:30,984 DEBUG [Run] Try transition [1. Выключена#2. Включена#e11#!o3.x1]
18:18:31,000 INFO  [Run] Start input action [o3.x1] calculation
18:18:31,000 INFO  [Run] Finish input action [o3.x1] calculation. Its value is
[true]
18:18:31,000 DEBUG [Run] Try transition [1. Выключена#2. Включена#e11#o3.x1]
18:18:31,000 INFO  [Run] Transition to go found [1. Выключена#2.
Включена#e11#o3.x1]
18:18:31,000 INFO  [Run] Start output action [o3.z1] execution
18:18:31,000 INFO  [Run] Finish output action [o3.z1] execution
18:18:31,000 INFO  [Run] Start on-enter action [o1.z1] execution
18:18:31,000 INFO  [Run] Finish on-enter action [o1.z1] execution
18:18:31,000 INFO  [Run] Start on-enter action [o4.z1] execution
18:18:31,000 INFO  [Run] Finish on-enter action [o4.z1] execution
18:18:31,000 INFO  [Run] Start on-enter action [o5.z1] execution
18:18:31,031 INFO  [Run] Finish on-enter action [o5.z1] execution
18:18:31,031 INFO  [Run] Start on-enter action [o2.z4] execution
18:18:31,031 INFO  [Run] Finish on-enter action [o2.z4] execution
18:18:31,031 INFO  [Run] Start on-enter action [o3.z6] execution
18:18:31,031 INFO  [Run] Finish on-enter action [o3.z6] execution
18:18:31,031 INFO  [Run] Transition to go found [s2#1. Обычное состояние##true]
18:18:31,031 INFO  [Run] Start event [e11] processing. In state [/A1:2.
Включена/A2:Top]
18:18:31,031 INFO  [Run] Transition to go found [s1#1. Звук включен##true]
18:18:31,109 INFO  [Run] Finish event [e11] processing. In state [/A1:2.
Включена/A2:1. Звук включен]
18:18:31,109 INFO  [Run] Finish event [e11] processing. In state [/A1:1. Обычное
состояние]
18:18:33,015 INFO  [Run] Start event [e4] processing. In state [/A1:1. Обычное
состояние]
18:18:33,015 DEBUG [Run] Try transition [2. Включена#s3#e4#true]
18:18:33,015 INFO  [Run] Transition to go found [2. Включена#s3#e4#true]
18:18:33,015 INFO  [Run] Start output action [o2.z4] execution
18:18:33,015 INFO  [Run] Finish output action [o2.z4] execution
18:18:33,015 INFO  [Run] Start output action [o1.z5] execution
18:18:33,015 INFO  [Run] Finish output action [o1.z5] execution
18:18:33,015 INFO  [Run] Start output action [o3.z5] execution
18:18:33,015 INFO  [Run] Finish output action [o3.z5] execution
18:18:33,140 INFO  [Run] State machine came to final state [/A1:s3]
18:18:33,140 INFO  [Run] Finish event [e4] processing. In state [/A1:s3]
```

Interpretating way of making application is not too good because you have to use all *Unimod* libraries to run the application, while for example a module parsing *XML* description seems to be excess for current task (task of making an alarm system).

5.2. Compilating

Unimod can generate a *Java* class from *XML* description. This class contains all the logic of the application and to use it you need less libraries than in the interpretating way. For example you do not have to parse *XML* anymore.

You can find the generated class for our application in package `ru.ifmo.alarm`, class name `Model1EventProcessor.java`. Let's describe its structure.

```
/**
 * This file was generated from model [Model1] on [Sun Dec 11 22:26:56 MSK
2005].
 * Do not change content of this file.
 */

package ru.ifmo.alarm;

import java.util.*;

import com.evelopers.common.exception.*;
import com.evelopers.unimod.core.stateworks.*;
import com.evelopers.unimod.runtime.*;
import com.evelopers.unimod.runtime.context.*;

public class Model1EventProcessor extends AbstractEventProcessor {
```

In the beginning of the file used packages are stated. Then fields of class are stated and among them you can find the following fields.

```
    private A1EventProcessor _A1;
    private A2EventProcessor _A2;
```

These fields are instances of classes representing automata *A1* and *A2*. Let's describe these classes.

```
    private class A1EventProcessor {

        // states
        private static final int Top = 1;
        private static final int s3 = 2;
        private static final int _1__Выключена = 3;
        private static final int _2__Включена = 4;
        private static final int _1__Обычное_состояние = 5;
        private static final int _2__Состояние_опасности = 6;
        private static final int _3__Тревога = 7;
        private static final int s2 = 8;
        private static final int s1 = 9;


        private int decodeState(String state) {
            if ("Top".equals(state)) {
                return Top;
            } else
            if ("s3".equals(state)) {
                return s3;
            } else
            if ("1. Выключена".equals(state)) {
                return _1__Выключена;
```

```
        } else
        ...
    }
```

Here states of *A1* automaton are described. State `Top` is a special state used by *Unimod* as initial. State `s1` is initial state of automaton; state `s3` is an ending state. State `s2` is an initial state in the part of automaton included into state «*2. Включена*» (Figure 3). Method `decodeState` decodes state's name into its id.

Then events and controlled objects are described.

```
// events
private static final int e33 = 1;
private static final int e4 = 2;
private static final int e22 = 3;
private static final int e11 = 4;
private static final int e32 = 5;
private static final int e21 = 6;
private static final int e12 = 7;

private int decodeEvent(String event) {
    ...
}

private ru.ifmo.alarm.controlled.Commentator o5;

private ru.ifmo.alarm.controlled.LightDiode o4;

private ru.ifmo.alarm.controlled.Sounds o3;

private ru.ifmo.alarm.controlled.Lights o1;

private ru.ifmo.alarm.controlled.StateTimerStarter o2;
```

The following method processes an event and transits into stable state.

```
private StateMachineConfig process(Event event,
        StateMachineContext context, StateMachinePath path,
        StateMachineConfig config) throws Exception {
    config = lookForTransition(event, context, path, config);

    config = transiteToStableState(context, path, config);

    // execute included state machines
    executeSubmachines(event, context, path, config);

    return config;
}
```

This method gets an event (`event`), current configuration of automata (`config`) and some other parameters. Then it looks for a transition and transits into a stable state. For example, initial state `s1` is not stable and automata can not be in this state, it only can go through it. After transiting automata lets included automata perform their transitions.

The following method executes included automata.

```
private void executeSubmachines(Event event,
        StateMachineContext context, StateMachinePath path,
        StateMachineConfig config) throws Exception {
    int state = decodeState(config.getActiveState());

    while (true) {
        switch (state) {
        case s3:
```

```
                return;
            case _1__Выключена:
                // 1. Выключена includes A2

                fireBeforeSubmachineExecution(context, event, path,
                        "1. Выключена", "A2");

                Model1EventProcessor.this.process(event, context,
                        new StateMachinePath(path, "1. Выключена","A2"));

                fireAfterSubmachineExecution(context, event, path,
                        "1. Выключена", "A2");

                return;
            case _2__Включена:
                // 2. Включена includes A2

                ...

                return;
            case _1__Обычное_состояние:

                state = _2__Включена;
                break;
            case _2__Состояние_опасности:

                state = _2__Включена;
                break;
            case _3__Тревога:

                ...
            }
        }
    }
```

As you can see that when *A1* is in state “1. Выключена” or “2. Включена”, events are passed to included automata *A2*. And if current state is included into “2. Включена”, then the loop is repeated for the parent state and finally event is also passed to *A2* on the second loop.

The following method makes transition into stable state.

```
    private StateMachineConfig transiteToStableState(
            StateMachineContext context, StateMachinePath path,
            StateMachineConfig config) throws Exception {

        int s = decodeState(config.getActiveState());
        Event event;

        switch (s) {
        case Top:

            fireComeToState(context, path, "s1");

            // s1->1. Выключена [true]/
            event = Event.NO_EVENT;
            fireTransitionFound(context, path, "s1", event,
                    "s1#1. Выключена##true");

            fireComeToState(context, path, "1. Выключена");

            // 1. Выключена []

            return new StateMachineConfig("1. Выключена");
```

```
            case _2__Включена:
                fireCompositeTargetState(context, path, "2. Включена");

                fireComeToState(context, path, "s2");

                // s2->1. Обычное состояние [true]/
                event = Event.NO_EVENT;
                fireTransitionFound(context, path, "s2", event,
                        "s2#1. Обычное состояние##true");

                fireComeToState(context, path, "1. Обычное состояние");

                // 1. Обычное состояние []

                return new StateMachineConfig("1. Обычное состояние");
            }

            return config;
        }
```

If automaton is in `Top` state, then it has just started its work and makes transition from `s1` to "*1. Выключена"*. If automaton is in state "*2. Включена"* then it transits into first stable state among included ones ("*1. Обычное состояние"*).

The following method finds transitions. It describes the main logic of automaton.

```
  private StateMachineConfig lookForTransition(Event event,
          StateMachineContext context, StateMachinePath path,
          StateMachineConfig config) throws Exception {

      boolean

      o3_x1 = false;

      BitSet calculatedInputActions = new BitSet(1);

      int s = decodeState(config.getActiveState());
      int e = decodeEvent(event.getName());

      while (true) {
          switch (s) {
          case _1__Выключена:

              switch (e) {
              case e11:

                  // 1. Выключена->2. Включена
                  // e11[!o3.x1]/o1.z1,o3.z5,o4.z1,o5.z1,o2.z4,o3.z6

                  fireTransitionCandidate(context, path, "1. Выключена",
                          event, "1. Выключена#2. Включена#e11#!o3.x1");

                  if (!isInputActionCalculated(calculatedInputActions,
                          _o3_x1)) {

                      fireBeforeInputActionExecution(context, path,
                              "1. Выключена#2. Включена#e11#!o3.x1",
                              "o3.x1");

                      o3_x1 = o3.x1(context);

                      fireAfterInputActionExecution(context, path,
                              "1. Выключена#2. Включена#e11#!o3.x1",
                              "o3.x1", new Boolean(o3_x1));
```

```
}

if (!o3_x1) {

    fireTransitionFound(context, path, "1. Выключена",
            event,
            "1. Выключена#2. Включена#e11#!o3.x1");

    fireBeforeOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#!o3.x1",
            "o1.z1");

    o1.z1(context);

    fireAfterOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#!o3.x1",
            "o1.z1");
    fireBeforeOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#!o3.x1",
            "o3.z5");

    o3.z5(context);

    fireAfterOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#!o3.x1",
            "o3.z5");

    ...


    fireComeToState(context, path, "2. Включена");

    // 2. Включена []
    return new StateMachineConfig("2. Включена");

}
// 1. Выключена->2. Включена
// e11[o3.x1]/o1.z1,o3.z1,o4.z1,o5.z1,o2.z4,o3.z6

fireTransitionCandidate(context, path, "1. Выключена",
        event, "1. Выключена#2. Включена#e11#o3.x1");

if (o3_x1) {

    fireTransitionFound(context, path, "1. Выключена",
            event, "1. Выключена#2. Включена#e11#o3.x1");

    fireBeforeOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#o3.x1",
            "o1.z1");

    o1.z1(context);

    fireAfterOutputActionExecution(context, path,
            "1. Выключена#2. Включена#e11#o3.x1",
            "o1.z1");
    ...

    fireComeToState(context, path, "2. Включена");

    // 2. Включена []
    return new StateMachineConfig("2. Включена");

}
```

```
            // transition not found
            return config;
        case e4:

            // 1. Выключена->s3 e4[true]/o2.z4,o1.z5,o3.z5

        ...

        default:

            // transition not found
            return config;
        }

    case _2__Включена:

        fireTransitionsOfSuperstate(context, path, "2. Включена",
                event);

        switch (e) {
        case e4:

            // 2. Включена->s3 e4[true]/o2.z4,o1.z5,o3.z5

            ...

        case e12:

            // 2. Включена->1. Выключена
            // e12[o3.x1]/o1.z2,o3.z2,o4.z2,o5.z2,o2.z4,o3.z6
            ...
            if (o3_x1) {
                ...
            }
            ...
            // 2. Включена->1. Выключена
            // e12[!o3.x1]/o1.z2,o3.z5,o4.z2,o5.z2,o2.z4,o3.z6
            ...
            if (!o3_x1) {
                ...
            }
            ...
            // transition not found
            return config;
        }

    case _1__Обычное_состояние:

        ...

    case _2__Состояние_опасности:

        ...

    case _3__Тревога:

        ...

    default:
        throw new EventProcessorException(
                "Incorrect stable state ["
                        + config.getActiveState()
                        + "] in state machine [A1]");
```

```
            }
        }
    }
```

In this method in nested `switch` operators current state is found and then possible transition is found. Then controlled object's actions are performed and new automata configuration is returned.

Structure of class `A2EventProcessor` is the same.

We finished describing of generated file.

To run application in compilating way we also have to create a small class which will run the application (see file `Main.java` in source files). To sum up, in compilating way of running application we need generated file, main class, compiled classes of event providers and controlled objects and some unimod libraries.

We compared sizes of needed files in interpretating and compilating modes and for interpretating mode it was 4 Mb while in compilating mode it was less that 1 Mb.

Conclusion

This project's aim was to show that automata based development of controlling systems is very simple and robust. The structure of system becomes viewable and easy to modify. Also we wanted to show that *Unimod* tool [2] is a great tool for developing automata based applications. It makes much work itself leaving to developer only task of creating right automata.

Literature

1. *Alarm System* MONGOOSE manual.
2. *Unimod*. http://unimod.sourceforge.net

Appendix 1. Example of protocol

This protocol describes automata actions for the following sequence of user actions. First, alarm system is turned on, then car is easily hit, and then it is hardly hit. After that alarm system is silently turned off and application window is closed.

```
Обработка события [e11 Нажатие кнопки 1] в состоянии [A1.start]
  [A1.start] ==> [A1.1 Выключена]
  [A1.1 Выключена] ==> [A1.2 Включена]
    [o1.z11 Мигнуть один раз]
    [o3.z31 Выдать короткий гудок]
    [o4.z41 Включить мигание светодиода]
      Обработка события [e11] в состоянии [A2.start]
        [A2.start] ==> [A2.1 Обычное состояние]
      Обработка события [e11] завершена в состоянии [A2.1 Обычное состояние]
Обработка события [e11] завершена в состоянии [A1.2 Включена]

Обработка события [e21 Слабый удар] в состоянии [A1.2 Включена]
      Обработка события [e21] в состоянии [A2.1 Обычное состояние]
        Проверка условия перехода [o1.x1 Подается сигнал тревоги]
          [o1.x1] = false
        [A2.1 Обычное состояние] ==> [A2.2 Состояние опасности]
          [o1.z13 Мигнуть три раза]
          [o3.z33 Выдать три коротких гудка]
          [o2.z25 Запустить таймер на пять секунд]
      Обработка события [e21] завершена в состоянии [A2.2 Состояние опасности]
Обработка события [e21] завершена в состоянии [A1.2 Включена]

Обработка события [e22 Сильный удар] в состоянии [A1.2. Включена]
      Обработка события [e22] в состоянии [A2.2. Состояние опасности]
        [A2.2 Состояние опасности] ==> [A2.1 Обычное состояние]
          [o1.z14 Запустить тревожный сигнал фар на 10 секунд]
          [o3.z34 Запустить тревожный сигнал сирены на 10 секунд]
      Обработка события [e22] завершена в состоянии [A2.1 Обычное состояние]
Обработка события [e22] завершена в состоянии [A1.2. Включена]

Обработка события [e31 Завершение таймера] в состоянии [A1.2. Включена]
      Обработка события [e31] в состоянии [A2.1 Обычное состояние]
      Обработка события [e31] завершена в состоянии [A2.1 Обычное состояние]
Обработка события [e31] завершена в состоянии [A1.2. Включена]

Обработка события [e13 Нажатие кнопки 3] в состоянии [A1.2. Включена]
  [A1.2. Включена] ==> [A1.4. Ожидание тихого выключения]
    [o2.z23 Запустить таймер на три секунды]
      Обработка события [e13] в состоянии [A2.start]
        [A2.start] ==> [A2.1 Обычное состояние]
      Обработка события [e13] завершена в состоянии [A2.1 Обычное состояние]
Обработка события [e13] завершена в состоянии [A1.4. Ожидание тихого выключения]

Обработка события [e12 Нажатие кнопки 2] в состоянии [A1.4. Ожидание тихого
выключения]
  [A1.4. Ожидание тихого выключения] ==> [A1.1. Выключена]
    [o1.z12 Мигнуть два раза]
    [o3.z35 Завершить звук]
    [o4.z42 Выключить светодиод]
Обработка события [e12] завершена в состоянии [A1.1. Выключена]

Обработка события [e31 Завершение таймера] в состоянии [A1.1. Выключена]
Обработка события [e31] завершена в состоянии [A1.1. Выключена]

Обработка события [e32 Закрытие окна программы] в состоянии [A1.1. Выключена]
  [A1.1. Выключена] ==> [A1.final]
```

```
    [o3.z35 Завершить звук]
  Автомат достиг конечного состояния
```
Обработка события [e32] завершена в состоянии [A1.final]

Appendix 2. Generated XML-description

```
<?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">
<model name="Model1">
  <controlledObject class="ru.ifmo.alarm.controlled.Lights" name="o1"/>
  <controlledObject class="ru.ifmo.alarm.event.StateTimer" name="o2"/>
  <controlledObject class="ru.ifmo.alarm.controlled.Sounds" name="o3"/>
  <controlledObject class="ru.ifmo.alarm.controlled.LightDiode" name="o4"/>
  <controlledObject class="ru.ifmo.alarm.controlled.Commentator" name="o5"/>
  <eventProvider class="ru.ifmo.alarm.event.RemoteControl" name="p1">
    <association clientRole="p1" targetRef="A1"/>
  </eventProvider>
  <eventProvider class="ru.ifmo.alarm.event.HitSensor" name="p2">
    <association clientRole="p2" targetRef="A1"/>
  </eventProvider>
  <eventProvider class="ru.ifmo.alarm.event.StateTimer" name="p3">
    <association clientRole="p3" targetRef="A1"/>
  </eventProvider>
  <eventProvider class="ru.ifmo.alarm.event.Window" name="p4">
    <association targetRef="A1"/>
  </eventProvider>
  <rootStateMachine>
    <stateMachineRef name="A1"/>
  </rootStateMachine>
  <stateMachine name="A1">
    <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
    <association clientRole="A1" supplierRole="o2" targetRef="o2"/>
    <association clientRole="A1" supplierRole="A2" targetRef="A2"/>
    <association clientRole="A1" supplierRole="o4" targetRef="o4"/>
    <association clientRole="A1" supplierRole="o1" targetRef="o1"/>
    <association clientRole="A1" supplierRole="o3" targetRef="o3"/>
    <association clientRole="A1" supplierRole="o5" targetRef="o5"/>
    <state name="Top" type="NORMAL">
      <state name="s3" type="FINAL"/>
      <state name="1. Выключена" type="NORMAL">
        <stateMachineRef name="A2"/>
        <outputAction ident="o2.z4"/>
        <outputAction ident="o3.z6"/>
        <outputAction ident="o4.z2"/>
      </state>
      <state name="2. Включена" type="NORMAL">
        <state name="1. Обычное состояние" type="NORMAL"/>
        <state name="2. Состояние опасности" type="NORMAL">
          <outputAction ident="o1.z3"/>
          <outputAction ident="o2.z2"/>
          <outputAction ident="o3.z3"/>
          <outputAction ident="o5.z7"/>
        </state>
        <state name="3. Тревога" type="NORMAL">
          <outputAction ident="o1.z4"/>
          <outputAction ident="o2.z1"/>
          <outputAction ident="o3.z4"/>
          <outputAction ident="o5.z9"/>
        </state>
        <state name="s2" type="INITIAL"/>
```

```
        <stateMachineRef name="A2"/>
        <outputAction ident="o1.z1"/>
        <outputAction ident="o4.z1"/>
        <outputAction ident="o5.z1"/>
        <outputAction ident="o2.z4"/>
        <outputAction ident="o3.z6"/>
      </state>
      <state name="s1" type="INITIAL"/>
    </state>
    <transition event="e11" guard="!o3.x1" sourceRef="1. Выключена" targetRef="2.
Включена">
      <outputAction ident="o3.z5"/>
    </transition>
    <transition event="e11" guard="o3.x1" sourceRef="1. Выключена" targetRef="2.
Включена">
      <outputAction ident="o3.z1"/>
    </transition>
    <transition event="e4" sourceRef="1. Выключена" targetRef="s3">
      <outputAction ident="o2.z4"/>
      <outputAction ident="o1.z5"/>
      <outputAction ident="o3.z5"/>
    </transition>
    <transition event="e12" guard="o3.x1" sourceRef="2. Включена" targetRef="1.
Выключена">
      <outputAction ident="o1.z2"/>
      <outputAction ident="o3.z2"/>
      <outputAction ident="o5.z2"/>
    </transition>
    <transition event="e12" guard="!o3.x1" sourceRef="2. Включена" targetRef="1.
Выключена">
      <outputAction ident="o1.z2"/>
      <outputAction ident="o3.z5"/>
      <outputAction ident="o5.z2"/>
    </transition>
    <transition event="e4" sourceRef="2. Включена" targetRef="s3">
      <outputAction ident="o2.z4"/>
      <outputAction ident="o1.z5"/>
      <outputAction ident="o3.z5"/>
    </transition>
    <transition event="e21" sourceRef="1. Обычное состояние" targetRef="2.
Состояние опасности">
      <outputAction ident="o5.z5"/>
    </transition>
    <transition event="e22" sourceRef="1. Обычное состояние" targetRef="3.
Тревога">
      <outputAction ident="o5.z6"/>
    </transition>
    <transition event="e32" sourceRef="2. Состояние опасности" targetRef="1.
Обычное состояние">
      <outputAction ident="o5.z8"/>
    </transition>
    <transition event="e22" sourceRef="2. Состояние опасности" targetRef="3.
Тревога">
      <outputAction ident="o5.z6"/>
    </transition>
    <transition event="e21" sourceRef="2. Состояние опасности" targetRef="3.
Тревога">
      <outputAction ident="o5.z5"/>
    </transition>
    <transition event="e33" sourceRef="3. Тревога" targetRef="1. Обычное
состояние">
      <outputAction ident="o1.z5"/>
      <outputAction ident="o3.z5"/>
      <outputAction ident="o5.z10"/>
```

```
    </transition>
    <transition sourceRef="s2" targetRef="1. Обычное состояние"/>
    <transition sourceRef="s1" targetRef="1. Выключена"/>
  </stateMachine>
  <stateMachine name="A2">
    <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
    <association clientRole="A2" supplierRole="o2" targetRef="o2"/>
    <association clientRole="A2" supplierRole="o3" targetRef="o3"/>
    <association clientRole="A2" supplierRole="o5" targetRef="o5"/>
    <state name="Top" type="NORMAL">
      <state name="s1" type="INITIAL"/>
      <state name="2. Звук выключен" type="NORMAL"/>
      <state name="1. Звук включен" type="NORMAL"/>
    </state>
    <transition sourceRef="s1" targetRef="1. Звук включен"/>
    <transition event="e11" sourceRef="2. Звук выключен" targetRef="1. Звук
включен">
      <outputAction ident="o3.z6"/>
    </transition>
    <transition event="e31" sourceRef="2. Звук выключен" targetRef="1. Звук
включен">
      <outputAction ident="o5.z4"/>
      <outputAction ident="o3.z6"/>
    </transition>
    <transition event="e12" sourceRef="2. Звук выключен" targetRef="1. Звук
включен">
      <outputAction ident="o3.z6"/>
    </transition>
    <transition event="e13" sourceRef="1. Звук включен" targetRef="2. Звук
выключен">
      <outputAction ident="o2.z3"/>
      <outputAction ident="o5.z3"/>
      <outputAction ident="o3.z7"/>
    </transition>
  </stateMachine>
   </model>
```