Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования

Кафедра “Компьютерные технологии”

В. В. Каширин, А. А. Шалыто

Имитация работы причального контейнерного крана

Программирование на базе switch-технологии и среды разработки UniMod

Проектная документация

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

Санкт-Петербург
2007

ОГЛАВЛЕНИЕ

Введение

Как показано в настоящей работе, *SWITCH*-технология, предложенная в работах [1, 2], является, пожалуй, наиболее естественным решением для широкого класса задач управления событийными системами. Поэтому ее применение целесообразно для задач построения имитаторов подобных систем.

Цель настоящей работы – моделирование работы контейнерного крана на основе *SWITCH*-технологии и инструментального средства *UniMod*, предназначенного для поддержки автоматного программирования.

Более подробно ознакомиться с этой технологией можно на сайте http://is.ifmo.ru, а с инструментальным средством *UniMod* – на сайте http://unimod.sourceforge.net .

Программа создана с помощью среды разработки *Eclipse 3.1*. При этом *UniMod* является плагином к указанной среде разработки. Использовался релиз инструментального средства *UniMod 1.3.38* .

1. Описание проекта

1.1. Описание работы крана - контейнерного перегружателя

В данном проекте рассматривается контейнеропогрузочный кран, задачей которого является разгрузка вновь прибывшего к причалу контейнеровоза.

Кран представляет собой перемещающуюся вдоль причала конструкцию, состоящую из основного корпуса, стрелы и передвигающегося вдоль стрелы захвата. С помощью захвата можно переносить контейнеры с контейнеровоза на отгрузочную площадку, обозначенную на причале вертикальным рядом прямоугольников. Для ускорения процесса разгрузки шаг перемещения захвата и крана был выбран равным половине длины стандартного контейнера. При этом отпала необходимость каждый раз тщательно нацеливать захват на очередной контейнер. Теперь достаточно лишь один раз откалибровать кран относительно любого из контейнеров на вновь прибывшем судне для того, чтобы затем быстро наводить захват на остальные контейнеры.

1.2. Постановка задачи

Цель данной работы – построение имитационной модели органов управления причальным контейнерным краном.

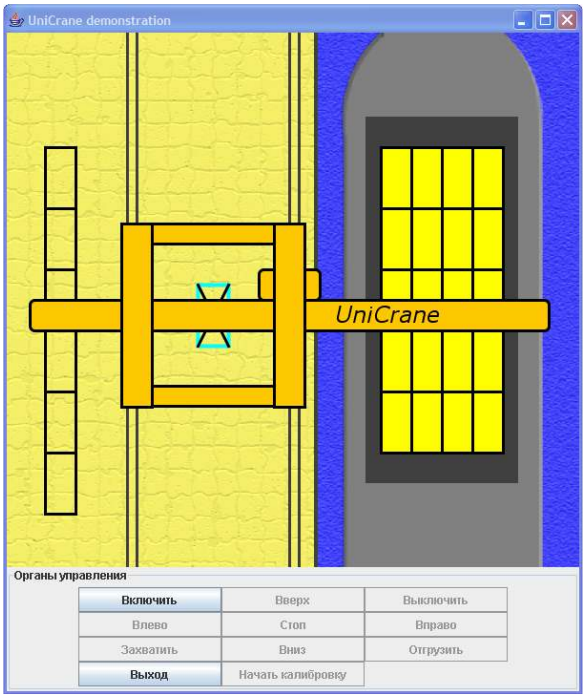

Рис.1. Пример окна работающей программы

На рис. 1 изображено окно приложения в начальном состоянии. В верхней части экрана отображается причал, контейнеровоз, контейнерный кран

и площадка для отгруженных контейнеров (вид сверху). В нижней части экрана расположены органы управления краном.

Вначале работы кран не откалиброван относительно площадки с контейнерами на судне, и, пользуясь стандартным шагом крана и захвата, невозможно нацелиться ни на один контейнер (рис. 2).

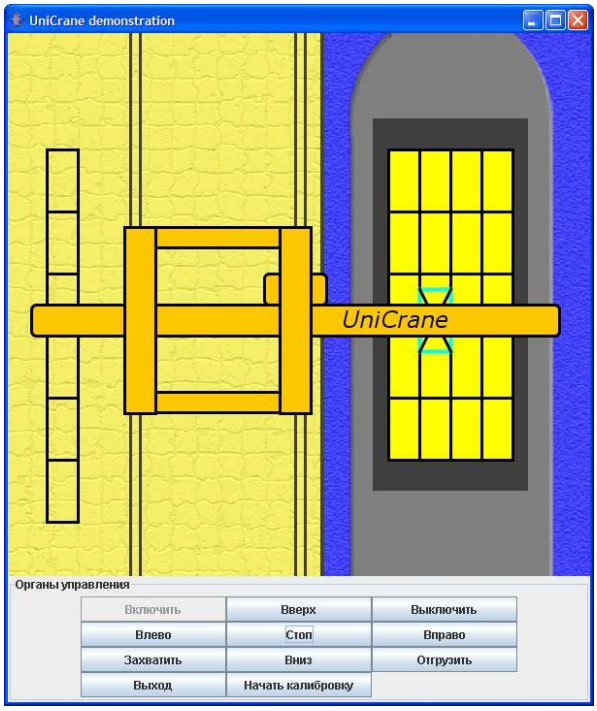

Рис.2. Пример не откалиброванного положения крана

Начав калибровку, изменив этим шаг хода, можно подстроить кран под положение площадки с контейнерами. Далее, отключив калибровочный режим, можно быстро переносить контейнеры на отведенную для них площадку на причале (рис. 3).

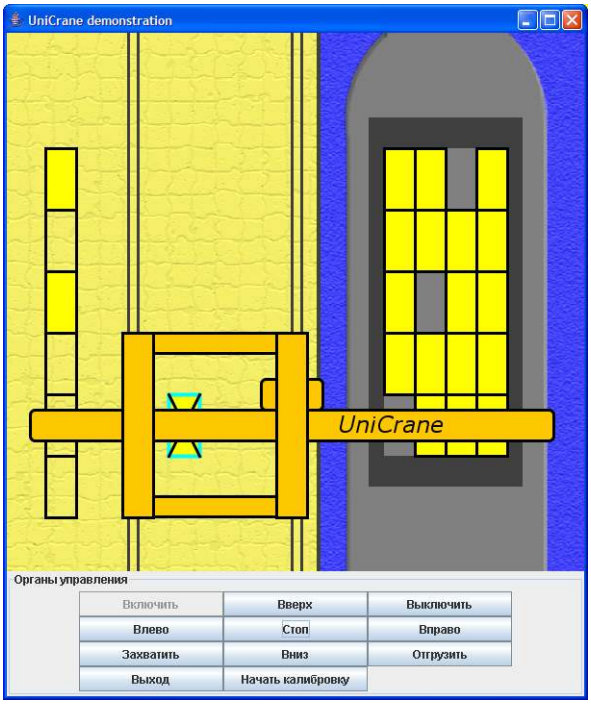

Рис.3. Рабочее состояние программы

Формально, требования к имитационной модели выглядят следующим образом.

1. Управление краном должно выполняться с помощью следующих действий:
   а) включение и выключение крана;
   б) изменение местоположения крана и захвата контейнера;
   в) калибровка положения захвата относительно контейнеров;
   г) захват и отгрузка контейнеров.
2. Система управления, реализуемая на основе конечных автоматов, должна обеспечить контроль за тем, чтобы:
   а) перемещение крана и захвата производились только в допустимой области;
   б) захват и отгрузка контейнеров осуществлялась только в предположенных для этого зонах.
3. Кран запускается нажатием кнопки *Включить*.
4. Пользователь вызывает перемещение крана или захвата:
   а) движение крана вверх – *Вверх*;
   б) движение крана вниз – *Вниз*;
   в) движение захвата влево – *Влево*;
   г) движение захвата вправо – *Вправо*.
5. Остановка крана или захвата происходит либо при достижении краном или захватом границы допустимой зоны перемещения, либо при нажатии пользователем кнопки *Стоп*.
6. Пользователь может изменить направление движения крана или захвата, только предварительно совершив их остановку.
7. Пользователь может захватить контейнер с помощью кнопки *Захватить* и опустить контейнер с помощью кнопки *Отгрузить*.

8. Пользователь может уменьшить шаг крана для того, чтобы откалибровать положение крана относительно нынешнего положения площадки с контейнерами на судне.

2. Проектирование

2.1. Диаграмма связей

Проектирование программы выполнено с помощью инструментального средства *UniMod*, которое позволяет построить диаграмму классов приложения (схему связей), а также автоматы, описывающие поведение приложения.

По-сути схема связей является диаграммой классов *UML*, однако классы располагаются не как обычно сверху вниз, а слева направо. Диаграмма связей изображена на рис. 4.

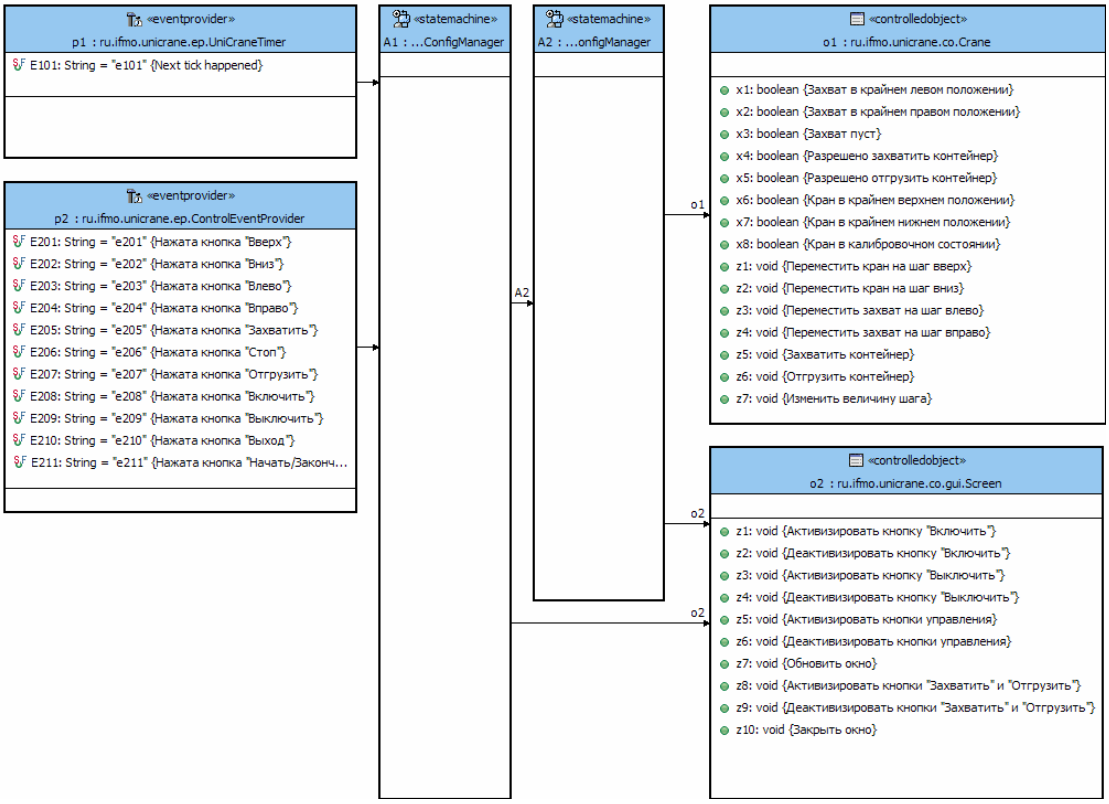

Рис.4. Диаграмма связей

На этой схеме слева направо изображены: поставщики событий, автоматы, объекты управления.

3. Поставщики событий

В этом разделе описываются поставщики событий (*Event providers*).

3.1. Поставщик событий `p1:UniCraneTimer`

Этот поставщик событий предназначен для синхронизации работы программы со временем. Поставщик `UniCraneTimer` генерирует событие `e101` каждые 500 миллисекунд.

Поставщик событий `UniCraneTimer` является расширением стандартного поставщика событий `Timer`, входящего в пакет *UniMod*, который может генерировать события лишь с интервалом равным тысяче миллисекунд.

3.2. Поставщик событий `p2:ControlEventProvider`

Этот поставщик реализует события, связанные с вызовом пользователем различных функций органов управления.

События, генерируемые поставщиком `ControlEventProvider`:

- `e201` – была нажата кнопка *Вверх*;
- `e202` – была нажата кнопка *Вниз*;
- `e203` – была нажата кнопка *Влево*;
- `e204` – была нажата кнопка *Вправо*;
- `e205` – была нажата кнопка захвата контейнера *Захватить*;
- `e206` – была нажата кнопка *Стоп*;
- `e207` – была нажата кнопка отгрузки контейнера *Отгрузить*;
- `e208` – была нажата кнопка запуска крана *Включить*;
- `e209` – была нажата кнопка отключения крана *Выключить*;
- `e210` – была нажата кнопка выхода из программы *Выход*;
- `e211` – была нажата кнопка включения/отключения калибровки *Начать/Закончить калибровку*.

4. Объекты управления

В данном разделе описываются объекты управления (*Controlled Objects*).

4.1. Объект управления `o1:Crane`

Этот объект реализует методы управления краном. Методы, предоставленные объектом управления `Crane`, делятся на входные переменные:

- `x1` – захват находится в крайнем левом положении;
- `x2` – захват находится в крайнем правом положении;
- `x3` – захват пуст;
- `x4` – разрешено захватить контейнер;
- `x5` – разрешено отгрузить контейнер;
- `x6` – кран находится в крайнем верхнем положении;
- `x7` – кран находится в крайнем нижнем положении;
- `x8` – кран в калибровочном состоянии;

и выходные воздействия:

- `z1` – переместить кран на шаг вверх;
- `z2` – переместить кран на шаг вниз;
- `z3` – переместить захват на шаг влево;
- `z4` – переместить захват на шаг вправо;

- `z5` – захватить контейнер;
- `z6` – отгрузить контейнер;
- `z7` – изменить величину шага.

4.2. Объект управления `o2:Screen`

Этот объект реализует методы управления графическим интерфейсом приложения. Выходные воздействия объекта управления `Screen`:

- `z1` – активизировать кнопку *Включить*;
- `z2` – деактивизировать кнопку *Включить*;
- `z3` – активизировать кнопку *Выключить*;
- `z4` – деактивизировать кнопку *Выключить*;
- `z5` – активизировать кнопки управления;
- `z6` – деактивизировать кнопки управления;
- `z7` – обновить окно;
- `z8` – активизировать кнопки *Захватить* и *Отгрузить*;
- `z9` – деактивизировать кнопки *Захватить* и *Отгрузить*;
- `z10` – закрыть окно.

5. Автоматы

В этом разделе описываются автоматы.

5.1. Автомат *A1*

5.1.1. Описание

Автомат *A1* управляет графическим интерфейсом программы.

5.1.2. Принцип работы

При запуске приложение инициализируется. После этого при нажатии на кнопку *Включить* начинается моделирование. При нажатии кнопки *Выключить* моделирование останавливается, и автомат возвращается в состояние инициализации.

5.1.3. Состояния

Автомат *A1* имеет следующие состояния.

1. *Кран выключен* – в этом состоянии инициализируется приложение:
   - после вхождения в это состояние изменяются названия кнопок, инициализируется окно приложения;
   - по нажатию кнопки *Включить* автомат переходит в состояние *Кран включен*.
2. *Кран включен* – состояние, в котором происходит моделирование:
   - после вхождения в это состояние изменяются названия кнопок, запускается вложенный автомат *A2*, осуществляющий моделирование.
   - по нажатию кнопки *Выключить* прекращается моделирование, и автомат переходит в состояние *Кран выключен*.

При нажатии кнопки *Выход* автомат из любого состояния переходит в конечное.

5.1.4. Граф переходов

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

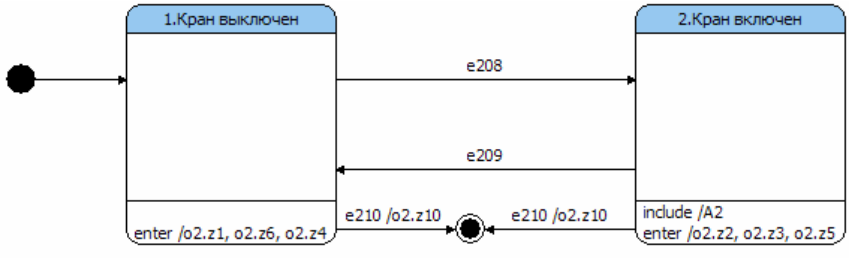

Рис.5. Автомат *A1*

5.2 Автомат *A2*

5.2.1. Описание

Автомат *A2* управляет основным циклом программы и отвечает собственно за моделирование.

5.2.2. Принцип работы

В основном состоянии кран ожидает команд от органов управления. При получении команды проверяется разрешимость вызываемых действий, и если нет конфликтов, то команда начинает выполняться.

5.2.3. Состояния

Автомат *A2* имеет следующие состояния.

1. *Кран остановлен* – в этом состоянии кран ожидает команд:
   - по событию `e201` (нажата кнопка *Вверх*) автомат переходит в состояние *Кран перемещается вверх;*
   - по событию `e202` (нажата кнопка *Вниз*) автомат переходит в состояние *Кран перемещается вниз;*
   - по событию `e203` (нажата кнопка *Влево*) автомат переходит в состояние *Захват перемещается влево;*
   - по событию `e204` (нажата кнопка *Вправо*) автомат переходит в состояние *Захват перемещается вправо;*
   - по событию `e205` (нажата кнопка *Захватить*) при выполнении условий `o1.x3` (захват пуст) и `o1.x4` (разрешено захватить контейнер) автомат переходит в состояние *Захватить контейнер.* При невыполнении условий `o1.x3` или `o1.x4` автомат остается в состоянии *Кран остановлен;*
   - по событию `e207` (нажата кнопка *Отгрузить*) при выполнении условий `!o1.x3` (захват не пуст) и `o1.x5` (разрешено отгрузить контейнер) автомат переходит в состояние *Отгрузить контейнер.* При условиях `o1.x3` (захват пуст) или `!o1.x5` (не разрешено отгрузить контейнер) автомат остается в состоянии *Отгрузить контейнер;*

- по событию `e211` (нажата кнопка *Начать/Закончить калибровку*) при выполнении условия `!o1.x8` (кран не в калибровочном состоянии) автомат переходит в состояние *Начать калибровку*, а при выполнении условия `o1.x8` (кран в калибровочном состоянии) автомат переходит в состояние *Закончить калибровку.*

2. *Кран перемещается вверх* – состояние, в котором кран движется вверх:
   - по событию `e101` проверяется, не достигнута ли крайняя верхняя точка, допустимая для перемещения. Если ответ отрицательный, автомат вызывает перемещение объекта кран на единицу вверх, и обновляет экран. В противном случае автомат переходит в состояние *Кран остановлен*;
   - по событию `e206` (нажата кнопка *Стоп*) автомат переходит в состояние *Кран остановлен*.
3. *Кран перемещается вниз* – состояние, в котором кран движется вниз:
   - по событию `e101` проверяется, не достигнута ли крайняя нижняя точка, допустимая для перемещения. Если ответ отрицательный, то автомат вызывает перемещение объекта кран на единицу вниз, и обновляет экран. В противном случае автомат переходит в состояние *Кран остановлен*;
   - по событию `e206` (нажата кнопка *Стоп*) автомат переходит в состояние *Кран остановлен*.
4. *Захват перемещается вправо* – состояние, в котором захват движется вправо:
   - по событию `e101` проверяется, не достигнута ли крайняя правая точка, допустимая для перемещения. Если ответ отрицательный, автомат вызывает перемещение захвата объекта кран на единицу вправо, и обновляет экран. В противном случае автомат переходит в состояние *Кран остановлен*;
   - по событию `e206` (нажата кнопка *Стоп*) автомат переходит в состояние *Кран остановлен*.
5. *Захват перемещается влево* – состояние, в котором захват движется влево:
   - по событию `e101` проверяется, не достигнута ли крайняя левая точка, допустимая для перемещения. Если ответ отрицательный, автомат вызывает перемещение захвата объекта кран на единицу влево, и обновляет экран. В противном случае автомат переходит в состояние *Кран остановлен*;
   - по событию `e206` (нажата кнопка *Стоп*) автомат переходит в состояние *Кран остановлен*.

6. Захватить контейнер
   - при входе в состояние автомат вызывает метод `o1.z5` объекта кран по захвату контейнера, и возвращается в состояние *Кран остановлен*.

7. Отгрузить контейнер
   - при входе в состояние автомат вызывает метод `o1.z6` объекта кран по отгрузке контейнера, и возвращается в состояние *Кран остановлен*.

8. Начать калибровку
   - при входе в состояние автомат вызывает метод `o1.z7` объекта кран по изменению шага хода и метод `o2.z9` объекта Screen, изменяющий кнопки, и возвращается в состояние *Кран остановлен*.

9. Закончить калибровку

- при входе в состояние автомат вызывает метод `o1.z7` объекта кран по изменению шага хода и метод `o1.z8` объекта Screen, изменяющий кнопки, и возвращается в состояние *Кран остановлен*.

5.2.4. Граф переходов

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

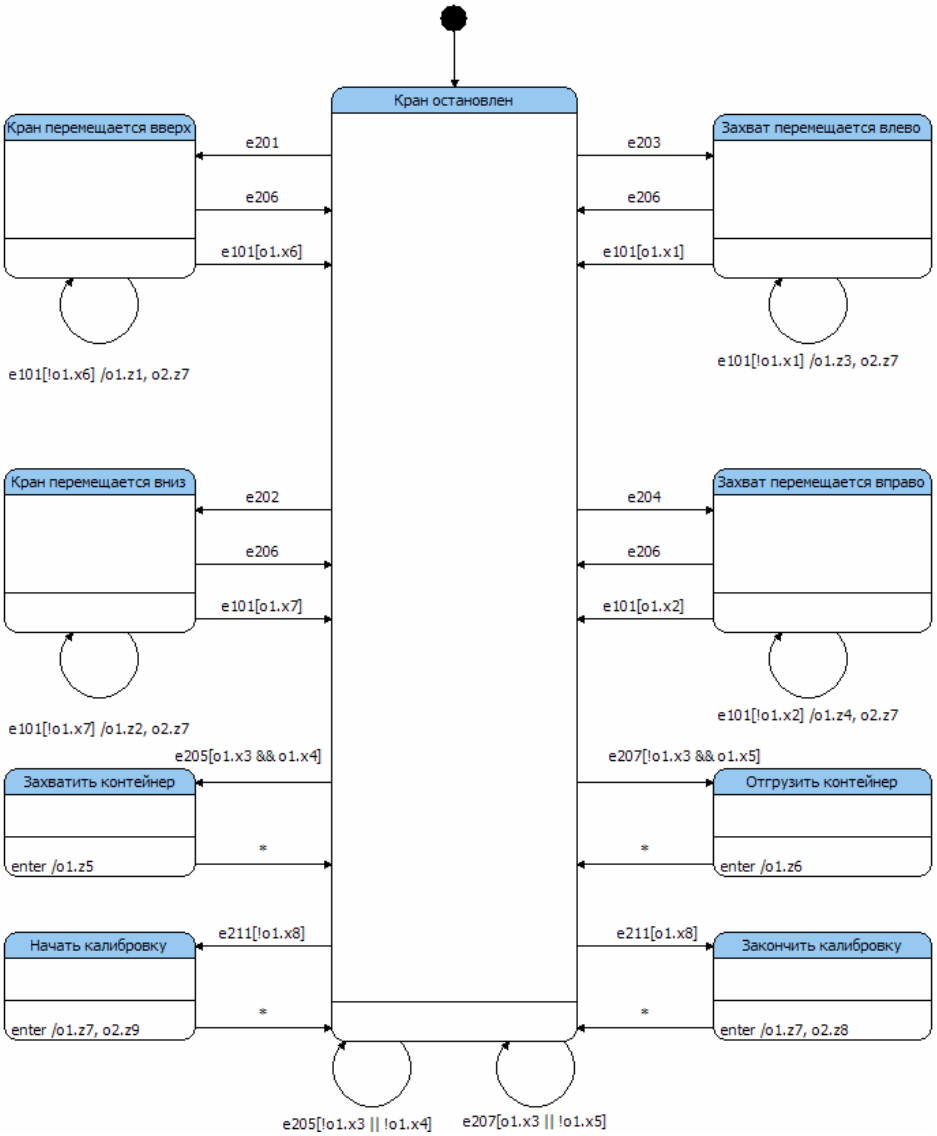

Рис.6. Граф переходов автомата *A2*

6. Реализация

Проект реализован на языке *Java* и содержит следующие пакеты:

- `ru.ifmo.unicrane.co` – объект управления:
  - `Crane.java` – объект управления кран;
  - `CraneModel.java` – интерфейс для класса `Crane`;

- `ru.ifmo.unicrane.co.gui` – объекты управления графическим интерфейсом:
  - `Screen.java` – объект управления `Gui`;
  - `MonitorPanel.java` – вспомогательный класс визуализации;
- `ru.ifmo.unicrane.ep` – поставщики событий:
  - `ControlEventProvider.java` – поставщик событий `Gui`;
  - `UniCraneTimer.java` – поставщик событий от системного таймера;

6.1. Интерпретационный подход

При интерпретационном подходе программа напрямую использует файл с `xml`-описанием автоматов, интерпретируя его средствами *UniMod*, совместно с кодом входных и выходных воздействий, написанным вручную. При этом программе требуется библиотеки средства *UniMod*, что препятствует распространению программы с использованием этого подхода. Чтобы уменьшить число библиотек, от которых зависит программа, применяется компилятивный подход.

6.2. Компилятивный подход

При компилятивном подходе xml-файл, описывающий автоматы, преобразуется в *Java*-код, что позволяет (совместно с кодом входных и выходных воздействий, написанным вручную) запускать программу без использования интерпретатора *UniMod*.

Выводы

Автоматный подход при создании программного обеспечения, реальных систем и их моделей помогает существенно облегчить процесс проектирования, отладки и модификации программного кода. Явное выделение состояний делает логику программы более простой и прозрачной для понимания, что в совокупности с протоколированием работы автоматов позволяет разработчику успешно следить за поведением программы, как в период разработки, так и во время сопровождения программного продукта.

Литература

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
2. *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения “реактивных” систем //Программирование. 2001. № 5. http://is.ifmo.ru/works/switch/1/

Приложения. Исходные коды программы

1. Поставщики событий

1.1. ControlEventProvider.java

```
1   package ru.ifmo.unicrane.ep;
2
3   import com.evelopers.common.exception.CommonException;
4   import com.evelopers.unimod.core.stateworks.Event;
5   import com.evelopers.unimod.runtime.EventProvider;
6   import com.evelopers.unimod.runtime.ModelEngine;
7   import com.evelopers.unimod.runtime.context.Parameter;
8   import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
9
10  public class ControlEventProvider implements EventProvider {
11
12      /**
13       * @unimod.event.descr Нажата кнопка "Вверх"
14       */
15      public static final String E201 = "e201";
16      /**
17       * @unimod.event.descr Нажата кнопка "Вниз"
18       */
19      public static final String E202 = "e202";
20      /**
21       * @unimod.event.descr Нажата кнопка "Влево"
22       */
23      public static final String E203 = "e203";
24      /**
25       * @unimod.event.descr Нажата кнопка "Вправо"
26       */
27      public static final String E204 = "e204";
28      /**
29       * @unimod.event.descr Нажата кнопка "Захватить"
30       */
31      public static final String E205 = "e205";
32      /**
33       * @unimod.event.descr Нажата кнопка "Стоп"
34       */
35      public static final String E206 = "e206";
36      /**
37       * @unimod.event.descr Нажата кнопка "Отгрузить"
38       */
39      public static final String E207 = "e207";
40      /**
41       * @unimod.event.descr Нажата кнопка "Выключить"
42       */
43      public static final String E209 = "e209";
44      /**
45       * @unimod.event.descr Нажата кнопка "Включить"
46       */
47      public static final String E208 = "e208";
48      /**
49       * @unimod.event.descr Нажата кнопка "Выход"
50       */
51      public static final String E210 = "e210";
52
53      /**
54       * @unimod.event.descr Нажата кнопка "Начать/Закончить калибровку"
55       */
56      public static final String E211 = "e211";
57
58      private static ModelEngine engine;
59
60
61      public void init(ModelEngine engine) throws CommonException {
62          if (ControlEventProvider.engine != null)
63              return;
64
65          ControlEventProvider.engine = engine;
66      }
67
```

```
68      public static void fireEvent(String eventName, Parameter parameter) {
69          if (engine != null) {
70              if (parameter == null) {
71                  engine.getEventManager().handle(new Event(eventName),
72                          StateMachineContextImpl.create());
73              } else {
74                  engine.getEventManager().handle(
75                          new Event(eventName, new Parameter[] { parameter }),
76                          StateMachineContextImpl.create());
77              }
78          }
79      }
80
81      public void dispose() {
82          // TODO Auto-generated method stub
83
84      }
85   }
```

1.2. UniCraneTimer.java

```
1    package ru.ifmo.unicrane.ep;
2
3    import com.evelopers.unimod.core.stateworks.Event;
4    import com.evelopers.unimod.runtime.EventProvider;
5    import com.evelopers.unimod.runtime.ModelEngine;
6    import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
7
8    import java.util.TimerTask;
9
10   /**
11    * This is rewroted class of UniMod event provider, which just decreases
12    * period between ticks
13    */
14   public class UniCraneTimer implements EventProvider {
15       /**
16        * @unimod.event.descr Next tick happened
17        */
18       public static final String E101 = "e101";
19
20       private java.util.Timer t;
21       private ModelEngine engine;
22
23       public void init(ModelEngine engine) {
24           this.engine = engine;
25
26           t = new java.util.Timer(false);
27           t.schedule(
28                   new TimerTask() {
29                       public void run() {
30                           UniCraneTimer.this.engine.getEventManager().handle(
31                                   new Event(E101),
StateMachineContextImpl.create());
32                       }
33                   }, 100, 500);
34
35       }
36
37       public void dispose() {
38           t.cancel();
39       }
40   }
```

2. Объекты управления

2.1. Crane.java

```
1    package ru.ifmo.unicrane.co;
2
3    import com.evelopers.unimod.runtime.ControlledObject;
4    import com.evelopers.unimod.runtime.context.StateMachineContext;
```

```
5    import ru.ifmo.unicrane.co.gui.MonitorPanel;
6
7    public class Crane implements CraneModel, ControlledObject {
8        public static int craneCoord;
9
10       public static int clutchCoord;
11
12       public static boolean clutchIsEmpty;
13
14       public static int stepVal;
15
16       public final int EXTREME_UP = 0;
17
18       public final int EXTREME_DOWN = 1;
19
20       public final int EXTREME_LEFT = 2;
21
22       public final int EXTREME_RIGHT = 3;
23
24       public final int COORD_UP = -24;
25
26       public final int COORD_DOWN = 24;
27
28       public final int COORD_LEFT = -20;
29
30       public final int COORD_RIGHT = 40;
31
32       public Crane() {
33           clutchCoord = 0;
34           craneCoord = 2;
35           stepVal = 4;
36           clutchIsEmpty = true;
37           MonitorPanel.craneInitialised = true;
38       }
39
40
41
42       public boolean checkPosition(int state) {
43           if ((state == this.EXTREME_RIGHT) && (clutchCoord / this.COORD_RIGHT >=
1))
44               return true;
45           else if ((state == this.EXTREME_LEFT) && (clutchCoord / this.COORD_LEFT
>= 1))
46               return true;
47           else if ((state == this.EXTREME_UP) && (craneCoord / this.COORD_UP >= 1))
48               return true;
49           else if ((state == this.EXTREME_DOWN) && (craneCoord / this.COORD_DOWN >=
1))
50               return true;
51           else
52               return false;
53       }
54
55       public void moveCraneUp() {
56           craneCoord = craneCoord - stepVal;
57       }
58
59       public void moveCraneDown() {
60           craneCoord = craneCoord + stepVal;
61       }
62
63       public void moveClutchLeft() {
64           clutchCoord = clutchCoord - stepVal;
65
66       }
67
68       public void moveClutchRight() {
69           clutchCoord = clutchCoord + stepVal;
70       }
71
72       public void changeStepValue() {
73           if (stepVal == 4)
74               stepVal = 1;
75           else stepVal = 4;
76
77       }
```

```
78
79      public boolean isClutchEmpty() {
80          return clutchIsEmpty;
81      }
82
83      public boolean isItCorrectToCatch() {
84          if ((clutchCoord >= 24) && (clutchCoord <= 36)
85                  && (craneCoord >= -16) && (craneCoord <= 16)
86                  && (Math.abs(craneCoord) % 8 == 0)&& (Math.abs(clutchCoord) % 4
== 0))
87              return true;
88
89          return false;
90      }
91
92      public boolean isItCorrectToPut() {
93          if ((clutchCoord == -20) && (craneCoord >= -16) &&
94                  (craneCoord <= 32) && ((craneCoord % 8) == 0))
95              return true;
96
97          return false;
98      }
99
100     public void catchContainer() {
101         if (MonitorPanel.placedContainers[craneCoord / 8 + 2][clutchCoord/4 - 6]
== true) {
102             clutchIsEmpty = false;
103             MonitorPanel.placedContainers[craneCoord / 8 + 2][clutchCoord/4 - 6]
= false;
104         }
105     }
106
107     public void putContainer() {
108         clutchIsEmpty = true;
109         MonitorPanel.unloadedContainers[craneCoord / 8 + 2] = true;
110     }
111
112     // ------------------------------------------------------------
113     /**
114      * @unimod.action.descr Захват в крайнем левом положении
115      */
116     public boolean x1(StateMachineContext context) {
117         return this.checkPosition(this.EXTREME_LEFT);
118     }
119
120     /**
121      * @unimod.action.descr Переместить кран на шаг вверх
122      */
123     public void z1(StateMachineContext context) {
124         this.moveCraneUp();
125     }
126
127     /**
128      * @unimod.action.descr Переместить кран на шаг вниз
129      */
130     public void z2(StateMachineContext context) {
131         this.moveCraneDown();
132     }
133
134     /**
135      * @unimod.action.descr Переместить захват на шаг влево
136      */
137     public void z3(StateMachineContext context) {
138         this.moveClutchLeft();
139     }
140
141     /**
142      * @unimod.action.descr Переместить захват на шаг вправо
143      */
144     public void z4(StateMachineContext context) {
145         this.moveClutchRight();
146     }
147
148     /**
149      * @unimod.action.descr Захват в крайнем правом положении
150      */
```

```
151      public boolean x2(StateMachineContext context) {
152          return this.checkPosition(this.EXTREME_RIGHT);
153      }
154
155      /**
156       * @unimod.action.descr Захватить контейнер
157       */
158      public void z5(StateMachineContext context) {
159          this.catchContainer();
160      }
161
162      /**
163       * @unimod.action.descr Захват пуст
164       */
165      public boolean x3(StateMachineContext context) {
166          return this.isClutchEmpty();
167      }
168
169      /**
170       * @unimod.action.descr Разрешено захватить контейнер
171       */
172      public boolean x4(StateMachineContext context) {
173          return this.isItCorrectToCatch();
174      }
175
176      /**
177       * @unimod.action.descr Разрешено отгрузить контейнер
178       */
179      public boolean x5(StateMachineContext context) {
180          return this.isItCorrectToPut();
181      }
182
183      /**
184       * @unimod.action.descr Отгрузить контейнер
185       */
186      public void z6(StateMachineContext context) {
187          this.putContainer();
188      }
189
190      /**
191       * @unimod.action.descr Кран в крайнем верхнем положении
192       */
193      public boolean x6(StateMachineContext context) {
194          return this.checkPosition(this.EXTREME_UP);
195      }
196
197      /**
198       * @unimod.action.descr Кран в крайнем нижнем положении
199       */
200      public boolean x7(StateMachineContext context) {
201          return this.checkPosition(this.EXTREME_DOWN);
202      }
203
204      /**
205       * @unimod.action.descr Изменить величину шага
206       */
207      public void z7(StateMachineContext context) {
208          this.changeStepValue();
209      }
210
211      /**
212       * @unimod.action.descr Кран в калибровочном состоянии
213       */
214      public boolean x8(StateMachineContext context) {
215          if (Crane.stepVal == 1) return true;
216          return false;
217      }
218
219
220  }
```

2.2. Screen.java

```
1    package ru.ifmo.unicrane.co.gui;
2
3    import java.awt.BorderLayout;
4    import java.awt.Container;
5    import java.awt.GridLayout;
6    import java.awt.event.ActionEvent;
7
8    import javax.swing.AbstractAction;
9    import javax.swing.Action;
10   import javax.swing.BorderFactory;
11   import javax.swing.BoxLayout;
12   import javax.swing.JButton;
13   import javax.swing.JFrame;
14   import javax.swing.JPanel;
15   import javax.swing.border.CompoundBorder;
16
17   import ru.ifmo.unicrane.co.Crane;
18   import ru.ifmo.unicrane.ep.ControlEventProvider;
19
20   import com.evelopers.unimod.runtime.ControlledObject;
21   import com.evelopers.unimod.runtime.context.StateMachineContext;
22
23   public class Screen extends JFrame implements ControlledObject {
24
25       private static final long serialVersionUID = 1L;
26
27       public final Action engineOnAction;
28
29       public final Action engineOffAction;
30
31       public final Action moveUpAction;
32
33       public final Action moveDownAction;
34
35       public final Action moveLeftAction;
36
37       public final Action moveRightAction;
38
39       public final Action stopAction;
40
41       public final Action catchAction;
42
43       public final Action putAction;
44
45       public final Action calibrateAction;
46
47       public final Action exitAction;
48
49       public final JButton engineOnActionButton;
50
51       public final JButton engineOffActionButton;
52
53       public final JButton moveUpActionButton;
54
55       public final JButton moveDownActionButton;
56
57       public final JButton moveLeftActionButton;
58
59       public final JButton moveRightActionButton;
60
61       public final JButton stopActionButton;
62
63       public final JButton catchActionButton;
64
65       public final JButton putActionButton;
66
67       public final JButton calibrateActionButton;
68
69       public final JButton exitActionButton;
70
71       public Screen() {
72           super("UniCrane demonstration");
73           this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
74           new Crane();
```

```
75
76          engineOnAction = new EngineOnAction();
77          engineOffAction = new EngineOffAction();
78          moveUpAction = new MoveUpAction();
79          moveDownAction = new MoveDownAction();
80          moveLeftAction = new MoveLeftAction();
81          moveRightAction = new MoveRightAction();
82          stopAction = new StopAction();
83          catchAction = new CatchAction();
84          putAction = new PutAction();
85          exitAction = new ExitAction();
86          calibrateAction = new CalibrateAction();
87
88          engineOnActionButton = new JButton(engineOnAction);
89          engineOffActionButton = new JButton(engineOffAction);
90          moveUpActionButton = new JButton(moveUpAction);
91          moveDownActionButton = new JButton(moveDownAction);
92          moveLeftActionButton = new JButton(moveLeftAction);
93          moveRightActionButton = new JButton(moveRightAction);
94          stopActionButton = new JButton(stopAction);
95          catchActionButton = new JButton(catchAction);
96          putActionButton = new JButton(putAction);
97          calibrateActionButton = new JButton(calibrateAction);
98          exitActionButton = new JButton(exitAction);
99
100         JPanel controlPanel = new JPanel();
101         controlPanel.setLayout(new BoxLayout(controlPanel, BoxLayout.Y_AXIS));
102
103         Container topC = new Container();
104         topC.setLayout(new GridLayout(4, 3));
105         topC.add(engineOnActionButton);
106         topC.add(moveUpActionButton);
107         topC.add(engineOffActionButton);
108         topC.add(moveLeftActionButton);
109         topC.add(stopActionButton);
110         topC.add(moveRightActionButton);
111         topC.add(catchActionButton);
112         topC.add(moveDownActionButton);
113         topC.add(putActionButton);
114         topC.add(exitActionButton);
115         topC.add(calibrateActionButton);
116
117         controlPanel.add(topC);
118
119
120         controlPanel.setBorder(new
CompoundBorder(BorderFactory.createTitledBorder("Органы управления"),
121                 BorderFactory.createEmptyBorder(0, 70, 0, 70)));
122
123         this.add(new MonitorPanel(), BorderLayout.NORTH);
124         this.add(controlPanel, BorderLayout.SOUTH);
125         this.pack();
126
127         this.setVisible(true);
128     }
129
130
131     /*
132      * Controls actions
133      */
134
135     private class EngineOnAction extends AbstractAction {
136         private static final long serialVersionUID = 1L;
137
138         public EngineOnAction() {
139             putValue(Action.NAME, "Включить");
140             putValue(Action.SHORT_DESCRIPTION, "Включить двигатель");
141             putValue(Action.MNEMONIC_KEY, new Integer('o'));
142             setEnabled(true);
143         }
144
145         public void actionPerformed(ActionEvent e) {
146             ControlEventProvider.fireEvent(ControlEventProvider.E208, null);
147         }
148     }
149
```

```
150     private class EngineOffAction extends AbstractAction {
151         private static final long serialVersionUID = 1L;
152 
153         public EngineOffAction() {
154             putValue(Action.NAME, "Выключить");
155             putValue(Action.SHORT_DESCRIPTION, "Выключить двигатель");
156             putValue(Action.MNEMONIC_KEY, new Integer('f'));
157             setEnabled(false);
158         }
159 
160         public void actionPerformed(ActionEvent e) {
161             ControlEventProvider.fireEvent(ControlEventProvider.E206, null);
162             ControlEventProvider.fireEvent(ControlEventProvider.E209, null);
163         }
164     }
165 
166     private class MoveUpAction extends AbstractAction {
167         private static final long serialVersionUID = 1L;
168 
169         public MoveUpAction() {
170             putValue(Action.NAME, "Вверх");
171             putValue(Action.SHORT_DESCRIPTION, "Переместиться вверх");
172             putValue(Action.MNEMONIC_KEY, new Integer('u'));
173             setEnabled(false);
174         }
175 
176         public void actionPerformed(ActionEvent e) {
177             ControlEventProvider.fireEvent(ControlEventProvider.E201, null);
178         }
179     }
180 
181     private class MoveDownAction extends AbstractAction {
182         private static final long serialVersionUID = 1L;
183 
184         public MoveDownAction() {
185             putValue(Action.NAME, "Вниз");
186             putValue(Action.SHORT_DESCRIPTION, "Переместиться вниз");
187             putValue(Action.MNEMONIC_KEY, new Integer('d'));
188             setEnabled(false);
189         }
190 
191         public void actionPerformed(ActionEvent e) {
192             ControlEventProvider.fireEvent(ControlEventProvider.E202, null);
193         }
194     }
195 
196     private class MoveLeftAction extends AbstractAction {
197         private static final long serialVersionUID = 1L;
198 
199         public MoveLeftAction() {
200             putValue(Action.NAME, "Влево");
201             putValue(Action.SHORT_DESCRIPTION, "Переместиться влево");
202             putValue(Action.MNEMONIC_KEY, new Integer('l'));
203             setEnabled(false);
204         }
205 
206         public void actionPerformed(ActionEvent e) {
207             ControlEventProvider.fireEvent(ControlEventProvider.E203, null);
208         }
209     }
210 
211     private class MoveRightAction extends AbstractAction {
212         private static final long serialVersionUID = 1L;
213 
214         public MoveRightAction() {
215             putValue(Action.NAME, "Вправо");
216             putValue(Action.SHORT_DESCRIPTION, "Переместиться вправо");
217             putValue(Action.MNEMONIC_KEY, new Integer('r'));
218             setEnabled(false);
219         }
220 
221         public void actionPerformed(ActionEvent e) {
222             ControlEventProvider.fireEvent(ControlEventProvider.E204, null);
223         }
224     }
225 
```

```
226     private class StopAction extends AbstractAction {
227         private static final long serialVersionUID = 1L;
228 
229         public StopAction() {
230             putValue(Action.NAME, "Стоп");
231             putValue(Action.SHORT_DESCRIPTION, "Остановить кран");
232             putValue(Action.MNEMONIC_KEY, new Integer('s'));
233             setEnabled(false);
234         }
235 
236         public void actionPerformed(ActionEvent e) {
237             ControlEventProvider.fireEvent(ControlEventProvider.E206, null);
238         }
239     }
240 
241     private class CatchAction extends AbstractAction {
242         private static final long serialVersionUID = 1L;
243 
244         public CatchAction() {
245             putValue(Action.NAME, "Захватить");
246             putValue(Action.SHORT_DESCRIPTION, "Захватить контейнер");
247             putValue(Action.MNEMONIC_KEY, new Integer('c'));
248             setEnabled(false);
249         }
250 
251         public void actionPerformed(ActionEvent e) {
252             ControlEventProvider.fireEvent(ControlEventProvider.E205, null);
253         }
254     }
255 
256     private class PutAction extends AbstractAction {
257         private static final long serialVersionUID = 1L;
258 
259         public PutAction() {
260             putValue(Action.NAME, "Отгрузить");
261             putValue(Action.SHORT_DESCRIPTION, "Отгрузить контейнер");
262             putValue(Action.MNEMONIC_KEY, new Integer('p'));
263             setEnabled(false);
264         }
265 
266         public void actionPerformed(ActionEvent e) {
267             ControlEventProvider.fireEvent(ControlEventProvider.E207, null);
268         }
269     }
270 
271     private class CalibrateAction extends AbstractAction {
272         private static final long serialVersionUID = 1L;
273 
274         public CalibrateAction() {
275             putValue(Action.NAME, "Начать калибровку");
276             putValue(Action.SHORT_DESCRIPTION, "Calibrate the clutch");
277             putValue(Action.MNEMONIC_KEY, new Integer('a'));
278             setEnabled(true);
279         }
280 
281         public void actionPerformed(ActionEvent e) {
282             ControlEventProvider.fireEvent(ControlEventProvider.E211, null);
283         }
284     }
285 
286     private class ExitAction extends AbstractAction {
287         private static final long serialVersionUID = 1L;
288 
289         public ExitAction() {
290             putValue(Action.NAME, "Выход");
291             putValue(Action.SHORT_DESCRIPTION, "Exit program");
292             putValue(Action.MNEMONIC_KEY, new Integer('e'));
293             setEnabled(true);
294         }
295 
296         public void actionPerformed(ActionEvent e) {
297             ControlEventProvider.fireEvent(ControlEventProvider.E210, null);
298         }
299     }
300 
301     // --------------------------------------------------------------------
```

```
302     /**
303      * @unimod.action.descr Активизировать кнопку "Включить"
304      */
305     public void z1(StateMachineContext context) {
306         this.engineOnAction.setEnabled(true);
307     }
308 
309     /**
310      * @unimod.action.descr Деактивизировать кнопку "Включить"
311      */
312     public void z2(StateMachineContext context) {
313         this.engineOnAction.setEnabled(false);
314     }
315 
316     /**
317      * @unimod.action.descr Активизировать кнопку "Выключить"
318      */
319     public void z3(StateMachineContext context) {
320         this.engineOffAction.setEnabled(true);
321     }
322 
323     /**
324      * @unimod.action.descr Деактивизировать кнопку "Выключить"
325      */
326     public void z4(StateMachineContext context) {
327         this.engineOffAction.setEnabled(false);
328     }
329 
330     /**
331      * @unimod.action.descr Активизировать кнопки управления
332      */
333     public void z5(StateMachineContext context) {
334         this.moveUpAction.setEnabled(true);
335         this.moveDownAction.setEnabled(true);
336         this.moveLeftAction.setEnabled(true);
337         this.moveRightAction.setEnabled(true);
338         this.putAction.setEnabled(true);
339         this.catchAction.setEnabled(true);
340         this.stopAction.setEnabled(true);
341         this.calibrateAction.setEnabled(true);
342 
343     }
344 
345     /**
346      * @unimod.action.descr Деактивизировать кнопки управления
347      */
348     public void z6(StateMachineContext context) {
349         this.moveUpAction.setEnabled(false);
350         this.moveDownAction.setEnabled(false);
351         this.moveLeftAction.setEnabled(false);
352         this.moveRightAction.setEnabled(false);
353         this.putAction.setEnabled(false);
354         this.catchAction.setEnabled(false);
355         this.stopAction.setEnabled(false);
356         this.calibrateAction.setEnabled(false);
357     }
358 
359     /**
360      * @unimod.action.descr Закрыть окно
361      */
362     public void z10(StateMachineContext context) {
363         this.dispose();
364     }
365 
366     /**
367      * @unimod.action.descr Обновить окно
368      */
369     public void z7(StateMachineContext context) {
370         this.repaint();
371     }
372 
373     /**
374      * @unimod.action.descr Активизировать кнопки "Захватить" и "Отгрузить"
375      */
376     public void z8(StateMachineContext context) {
377         this.putAction.setEnabled(true);
```

```
378          this.catchAction.setEnabled(true);
379          this.calibrateAction.putValue(Action.NAME,"Начать калибровку");
380      }
381
382      /**
383       * @unimod.action.descr Деактивизировать кнопки "Захватить" и "Отгрузить"
384       */
385      public void z9(StateMachineContext context) {
386          this.putAction.setEnabled(false);
387          this.catchAction.setEnabled(false);
388          this.calibrateAction.putValue(Action.NAME,"Закончить калибровку");
389      }
390
391  }
```

3. Интерпретационный подход. XML-описание автоматов (A1.xml)

```
1    <?xml version="1.0" encoding="UTF-8" standalone="no"?><!DOCTYPE model PUBLIC "-
//eVelopers Corp.//DTD State machine model V1.0//EN"
"http://www.evelopers.com/dtd/unimod/statemachine.dtd">
2    <model name="Model1">
3      <controlledObject class="ru.ifmo.unicrane.co.Crane" name="o1"/>
4      <controlledObject class="ru.ifmo.unicrane.co.gui.Screen" name="o2"/>
5      <eventProvider class="ru.ifmo.unicrane.ep.UniCraneTimer" name="p1">
6        <association clientRole="p1" targetRef="A1"/>
7      </eventProvider>
8      <eventProvider class="ru.ifmo.unicrane.ep.ControlEventProvider" name="p2">
9        <association clientRole="p2" targetRef="A1"/>
10     </eventProvider>
11     <rootStateMachine>
12       <stateMachineRef name="A1"/>
13     </rootStateMachine>
14     <stateMachine name="A1">
15       <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
16       <association clientRole="A1" supplierRole="A2" targetRef="A2"/>
17       <association clientRole="A1" supplierRole="o2" targetRef="o2"/>
18       <state name="Top" type="NORMAL">
19         <state name="1.Кран выключен" type="NORMAL">
20           <outputAction ident="o2.z1"/>
21           <outputAction ident="o2.z6"/>
22           <outputAction ident="o2.z4"/>
23         </state>
24         <state name="2.Кран включен" type="NORMAL">
25           <stateMachineRef name="A2"/>
26           <outputAction ident="o2.z2"/>
27           <outputAction ident="o2.z3"/>
28           <outputAction ident="o2.z5"/>
29         </state>
30         <state name="s1" type="INITIAL"/>
31         <state name="s2" type="FINAL"/>
32       </state>
33       <transition event="e208" sourceRef="1.Кран выключен" targetRef="2.Кран
включен"/>
34       <transition event="e210" sourceRef="1.Кран выключен" targetRef="s2">
35         <outputAction ident="o2.z10"/>
36       </transition>
37       <transition event="e209" sourceRef="2.Кран включен" targetRef="1.Кран
выключен"/>
38       <transition event="e210" sourceRef="2.Кран включен" targetRef="s2">
39         <outputAction ident="o2.z10"/>
40       </transition>
41       <transition sourceRef="s1" targetRef="1.Кран выключен"/>
42     </stateMachine>
43     <stateMachine name="A2">
44       <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
45       <association clientRole="A2" supplierRole="o1" targetRef="o1"/>
46       <association clientRole="A2" supplierRole="o2" targetRef="o2"/>
47       <state name="Top" type="NORMAL">
48         <state name="s1" type="INITIAL"/>
49         <state name="Кран остановлен" type="NORMAL"/>
```

```
50          <state name="Кран перемещается вверх" type="NORMAL"/>
51          <state name="Захват перемещается влево" type="NORMAL"/>
52          <state name="Кран перемещается вниз" type="NORMAL"/>
53          <state name="Захват перемещается вправо" type="NORMAL"/>
54          <state name="Захватить контейнер" type="NORMAL">
55            <outputAction ident="o1.z5"/>
56          </state>
57          <state name="Отгрузить контейнер" type="NORMAL">
58            <outputAction ident="o1.z6"/>
59          </state>
60          <state name="Начать калибровку" type="NORMAL">
61            <outputAction ident="o1.z7"/>
62            <outputAction ident="o2.z9"/>
63          </state>
64          <state name="Закончить калибровку" type="NORMAL">
65            <outputAction ident="o1.z7"/>
66            <outputAction ident="o2.z8"/>
67          </state>
68        </state>
69        <transition sourceRef="s1" targetRef="Кран остановлен"/>
70        <transition event="e205" guard="!o1.x3 || !o1.x4" sourceRef="Кран остановлен"
targetRef="Кран остановлен"/>
71        <transition event="e207" guard="o1.x3 || !o1.x5" sourceRef="Кран остановлен"
targetRef="Кран остановлен"/>
72        <transition event="e201" sourceRef="Кран остановлен" targetRef="Кран
перемещается вверх"/>
73        <transition event="e203" sourceRef="Кран остановлен" targetRef="Захват
перемещается влево"/>
74        <transition event="e202" sourceRef="Кран остановлен" targetRef="Кран
перемещается вниз"/>
75        <transition event="e204" sourceRef="Кран остановлен" targetRef="Захват
перемещается вправо"/>
76        <transition event="e205" guard="o1.x3 &amp;amp;&amp;amp; o1.x4" sourceRef="Кран
остановлен" targetRef="Захватить контейнер"/>
77        <transition event="e207" guard="!o1.x3 &amp;amp;&amp;amp; o1.x5" sourceRef="Кран
остановлен" targetRef="Отгрузить контейнер"/>
78        <transition event="e211" guard="!o1.x8" sourceRef="Кран остановлен"
targetRef="Начать калибровку"/>
79        <transition event="e211" guard="o1.x8" sourceRef="Кран остановлен"
targetRef="Закончить калибровку"/>
80        <transition event="e101" guard="o1.x6" sourceRef="Кран перемещается вверх"
targetRef="Кран остановлен"/>
81        <transition event="e206" sourceRef="Кран перемещается вверх" targetRef="Кран
остановлен"/>
82        <transition event="e101" guard="!o1.x6" sourceRef="Кран перемещается вверх"
targetRef="Кран перемещается вверх">
83          <outputAction ident="o1.z1"/>
84          <outputAction ident="o2.z7"/>
85        </transition>
86        <transition event="e101" guard="o1.x1" sourceRef="Захват перемещается влево"
targetRef="Кран остановлен"/>
87        <transition event="e206" sourceRef="Захват перемещается влево"
targetRef="Кран остановлен"/>
88        <transition event="e101" guard="!o1.x1" sourceRef="Захват перемещается влево"
targetRef="Захват перемещается влево">
89          <outputAction ident="o1.z3"/>
90          <outputAction ident="o2.z7"/>
91        </transition>
92        <transition event="e101" guard="o1.x7" sourceRef="Кран перемещается вниз"
targetRef="Кран остановлен"/>
93        <transition event="e206" sourceRef="Кран перемещается вниз" targetRef="Кран
остановлен"/>
94        <transition event="e101" guard="!o1.x7" sourceRef="Кран перемещается вниз"
targetRef="Кран перемещается вниз">
95          <outputAction ident="o1.z2"/>
96          <outputAction ident="o2.z7"/>
97        </transition>
98        <transition event="e101" guard="o1.x2" sourceRef="Захват перемещается вправо"
targetRef="Кран остановлен"/>
99        <transition event="e206" sourceRef="Захват перемещается вправо"
targetRef="Кран остановлен"/>
100       <transition event="e101" guard="!o1.x2" sourceRef="Захват перемещается
вправо" targetRef="Захват перемещается вправо">
101         <outputAction ident="o1.z4"/>
102         <outputAction ident="o2.z7"/>
103       </transition>
```

```
104       <transition event="*" sourceRef="Захватить контейнер" targetRef="Кран
остановлен"/>
105       <transition event="*" sourceRef="Отгрузить контейнер" targetRef="Кран
остановлен"/>
106       <transition event="*" sourceRef="Начать калибровку" targetRef="Кран
остановлен"/>
107       <transition event="*" sourceRef="Закончить калибровку" targetRef="Кран
остановлен"/>
108     </stateMachine>
109 </model>
```

4. Компилятивный подход. Сгенерированный класс логики автоматов (`Model1EventProcessor.java`)

```
1     /**
2      * This file was generated from model [Model1] on [Mon Dec 18 09:59:45 MSK 2006].
3      * Do not change content of this file.
4      */
5
6     import java.io.IOException;
7     import java.util.*;
8
9     import org.apache.commons.lang.BooleanUtils;
10    import org.apache.commons.lang.math.NumberUtils;
11    import org.apache.commons.lang.StringUtils;
12    import org.apache.commons.logging.Log;
13    import org.apache.commons.logging.LogFactory;
14
15    import com.evelopers.common.exception.*;
16    import com.evelopers.unimod.core.stateworks.*;
17    import com.evelopers.unimod.debug.app.AppDebugger;
18    import com.evelopers.unimod.debug.protocol.JavaSpecificMessageCoder;
19    import com.evelopers.unimod.runtime.*;
20    import com.evelopers.unimod.runtime.context.*;
21    import com.evelopers.unimod.runtime.logger.SimpleLogger;
22
23
24    public class Model1EventProcessor extends AbstractEventProcessor {
25
26        private ModelStructure modelStructure;
27
28        private static final int A1 = 1;
29        private static final int A2 = 2;
30
31        private int decodeStateMachine(String sm) {
32
33            if ("A1".equals(sm)) {
34                return A1;
35            } else if ("A2".equals(sm)) {
36                return A2;
37            }
38
39            return -1;
40        }
41
42        private A1EventProcessor _A1;
43        private A2EventProcessor _A2;
44
45        public Model1EventProcessor() {
46            modelStructure = new Model1ModelStructure();
47
48            _A1 = new A1EventProcessor();
49            _A2 = new A2EventProcessor();
50        }
51
52        public static void run(int debuggerPort, boolean debuggerSuspend) throws
53                InterruptedException, EventProcessorException, CommonException,
54                IOException {
55
```

```
56            /* Create runtime engine */
57            ModelEngine engine = createModelEngine(true);
58
59            /* Setup logger */
60            final Log log = LogFactory.getLog(Model1EventProcessor.class);
61            engine.getEventProcessor().addEventProcessorListener(new
SimpleLogger(log));
62
63            /* Setup exception handler */
64            engine.getEventProcessor().addExceptionHandler(new ExceptionHandler() {
65                public void handleException(StateMachineContext context,
SystemException e) {
66                    log.fatal(e.getChainedMessage(), e.getRootException());
67                }
68            });
69
70            if (debuggerPort > 0) {
71                AppDebugger d = new AppDebugger(
72                        debuggerPort, debuggerSuspend,
73                        new JavaSpecificMessageCoder(), engine);
74                d.start();
75            }
76            engine.start();
77        }
78
79        public static void main(String[] args) throws Exception {
80            int debuggerPort =
NumberUtils.stringToInt(System.getProperty("debugger.port"), -1);
81            boolean debuggerSuspend =
BooleanUtils.toBoolean(System.getProperty("debugger.suspend"));
82            Model1EventProcessor.run(debuggerPort, debuggerSuspend);
83        }
84
85        public static ModelEngine createModelEngine(boolean useEventQueue) throws
CommonException {
86            ObjectsManager objectsManager = new ObjectsManager();
87            return ModelEngine.createStandAlone(
88                    useEventQueue ? (EventManager) new QueuedHandler() :
(EventManager) new StrictHandler(),
89                    new Model1EventProcessor(),
90                    objectsManager.getControlledObjectsManager(),
91                    objectsManager.getEventProvidersManager());
92        }
93
94        public static class ObjectsManager {
95            private ru.ifmo.unicrane.co.Crane o1 = null;
96            private ru.ifmo.unicrane.co.gui.Screen o2 = null;
97            private ru.ifmo.unicrane.ep.UniCraneTimer p1 = null;
98            private ru.ifmo.unicrane.ep.ControlEventProvider p2 = null;
99
100           private ControlledObjectsManager controlledObjectsManager = new
ControlledObjectsManagerImpl();
101           private EventProvidersManager eventProvidersManager = new
EventProvidersManagerImpl();
102
103           public ControlledObjectsManager getControlledObjectsManager() {
104               return controlledObjectsManager;
105           }
106
107           public EventProvidersManager getEventProvidersManager() {
108               return eventProvidersManager;
109           }
110
111           private class ControlledObjectsManagerImpl implements
ControlledObjectsManager {
112               public void init(ModelEngine engine) throws CommonException {
113               }
114
115               public void dispose() {
116               }
117
118               public ControlledObject getControlledObject(String coName) {
119                   if (StringUtils.equals(coName, "o1")) {
120                       if (o1 == null) {
121                           o1 = new ru.ifmo.unicrane.co.Crane();
122                       }
```

```
123                     return o1;
124                 }
125                 if (StringUtils.equals(coName, "o2")) {
126                     if (o2 == null) {
127                         o2 = new ru.ifmo.unicrane.co.gui.Screen();
128                     }
129                     return o2;
130                 }
131                 throw new IllegalArgumentException("Controlled object with name
[" + coName + "] wasn't found");
132             }
133         }
134 
135         private class EventProvidersManagerImpl implements EventProvidersManager
{
136             private List nonameEventProviders = new ArrayList();
137 
138             public void init(ModelEngine engine) throws CommonException {
139                 EventProvider ep;
140                 ep = getEventProvider("p1");
141                 ep.init(engine);
142                 ep = getEventProvider("p2");
143                 ep.init(engine);
144             }
145 
146             public void dispose() {
147                 EventProvider ep;
148                 ep = getEventProvider("p1");
149                 ep.dispose();
150                 ep = getEventProvider("p2");
151                 ep.dispose();
152                 for (Iterator i = nonameEventProviders.iterator(); i.hasNext();)
{
153                     ep = (EventProvider) i.next();
154                     ep.dispose();
155                 }
156             }
157 
158             public EventProvider getEventProvider(String epName) {
159                 if (StringUtils.equals(epName, "p1")) {
160                     if (p1 == null) {
161                         p1 = new ru.ifmo.unicrane.ep.UniCraneTimer();
162                     }
163                     return p1;
164                 }
165                 if (StringUtils.equals(epName, "p2")) {
166                     if (p2 == null) {
167                         p2 = new ru.ifmo.unicrane.ep.ControlEventProvider();
168                     }
169                     return p2;
170                 }
171                 throw new IllegalArgumentException("Event provider with name [" +
epName + "] wasn't found");
172             }
173         }
174     }
175 
176     public ModelStructure getModelStructure() {
177         return modelStructure;
178     }
179 
180     public void setControlledObjectsMap(ControlledObjectsMap
controlledObjectsMap) {
181         super.setControlledObjectsMap(controlledObjectsMap);
182 
183         _A1.init(controlledObjectsMap);
184         _A2.init(controlledObjectsMap);
185     }
186 
187     protected StateMachineConfig process(
188             Event event, StateMachineContext context,
189             StateMachinePath path, StateMachineConfig config) throws
SystemException {
190 
191         // get state machine from path
192         int sm = decodeStateMachine(path.getStateMachine());
```

```
193
194          try {
195              switch (sm) {
196                  case A1:
197                      return _A1.process(event, context, path, config);
198                  case A2:
199                      return _A2.process(event, context, path, config);
200                  default:
201                      throw new EventProcessorException("Unknown state machine [" +
path.getStateMachine() + "]");
202              }
203          } catch (Exception e) {
204              if (e instanceof SystemException) {
205                  throw(SystemException) e;
206              } else {
207                  throw new SystemException(e);
208              }
209          }
210      }
211
212      protected StateMachineConfig transiteToStableState(
213              StateMachineContext context,
214              StateMachinePath path, StateMachineConfig config) throws
SystemException {
215
216          // get state machine from path
217          int sm = decodeStateMachine(path.getStateMachine());
218
219          try {
220              switch (sm) {
221                  case A1:
222                      return _A1.transiteToStableState(context, path, config);
223                  case A2:
224                      return _A2.transiteToStableState(context, path, config);
225                  default:
226                      throw new EventProcessorException("Unknown state machine [" +
path.getStateMachine() + "]");
227              }
228          } catch (Exception e) {
229              if (e instanceof SystemException) {
230                  throw(SystemException) e;
231              } else {
232                  throw new SystemException(e);
233              }
234          }
235      }
236
237
238      private class Model1ModelStructure implements ModelStructure {
239          private Map configManagers = new HashMap();
240
241          private Model1ModelStructure() {
242              configManagers.put("A1", new
com.evelopers.unimod.runtime.config.DistinguishConfigManager());
243              configManagers.put("A2", new
com.evelopers.unimod.runtime.config.DistinguishConfigManager());
244          }
245
246          public StateMachinePath getRootPath()
247                  throws EventProcessorException {
248              return new StateMachinePath("A1");
249          }
250
251          public StateMachineConfigManager getConfigManager(String stateMachine)
252                  throws EventProcessorException {
253              return (StateMachineConfigManager) configManagers.get(stateMachine);
254          }
255
256          public StateMachineConfig getTopConfig(String stateMachine)
257                  throws EventProcessorException {
258              int sm = decodeStateMachine(stateMachine);
259
260              switch (sm) {
261                  case A1:
262                      return new StateMachineConfig("Top");
263                  case A2:
```

```
264                     return new StateMachineConfig("Top");
265                 default:
266                     throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
267             }
268         }
269 
270         public boolean isFinal(String stateMachine, StateMachineConfig config)
271                 throws EventProcessorException {
272             /* Get state machine from path */
273             int sm = decodeStateMachine(stateMachine);
274             int state;
275             switch (sm) {
276                 case A1:
277                     state = _A1.decodeState(config.getActiveState());
278                     switch (state) {
279                         case A1EventProcessor.s2:
280                             return true;
281                         default:
282                             return false;
283                     }
284                 case A2:
285                     state = _A2.decodeState(config.getActiveState());
286                     switch (state) {
287                         default:
288                             return false;
289                     }
290                 default:
291                     throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
292             }
293         }
294     }
295 
296 
297     private class A1EventProcessor {
298 
299         // states
300         private static final int Top = 1;
301         private static final int _1_Кран_выключен = 2;
302         private static final int _2_Кран_включен = 3;
303         private static final int s1 = 4;
304         private static final int s2 = 5;
305 
306         private int decodeState(String state) {
307 
308             if ("Top".equals(state)) {
309                 return Top;
310             } else if ("1.Кран выключен".equals(state)) {
311                 return _1_Кран_выключен;
312             } else if ("2.Кран включен".equals(state)) {
313                 return _2_Кран_включен;
314             } else if ("s1".equals(state)) {
315                 return s1;
316             } else if ("s2".equals(state)) {
317                 return s2;
318             }
319 
320             return -1;
321         }
322 
323         // events
324         private static final int e209 = 1;
325         private static final int e208 = 2;
326         private static final int e210 = 3;
327 
328         private int decodeEvent(String event) {
329 
330             if ("e209".equals(event)) {
331                 return e209;
332             } else if ("e208".equals(event)) {
333                 return e208;
334             } else if ("e210".equals(event)) {
335                 return e210;
336             }
337 
```

```
338             return -1;
339         }
340
341         private ru.ifmo.unicrane.co.gui.Screen o2;
342
343         private void init(ControlledObjectsMap controlledObjectsMap) {
344             o2 = (ru.ifmo.unicrane.co.gui.Screen)
controlledObjectsMap.getControlledObject("o2");
345         }
346
347         private StateMachineConfig process(Event event, StateMachineContext
context, StateMachinePath path, StateMachineConfig config) throws Exception {
348             config = lookForTransition(event, context, path, config);
349
350             config = transiteToStableState(context, path, config);
351
352             // execute included state machines
353             executeSubmachines(event, context, path, config);
354
355             return config;
356         }
357
358         private void executeSubmachines(Event event, StateMachineContext context,
StateMachinePath path, StateMachineConfig config) throws Exception {
359             int state = decodeState(config.getActiveState());
360
361             while (true) {
362                 switch (state) {
363                     case _1_Кран_выключен:
364
365                         return;
366                     case _2_Кран_включен:
367                         // 2.Кран включен includes A2
368
369                         fireBeforeSubmachineExecution(context, event, path,
"2.Кран включен", "A2");
370
371                         Model1EventProcessor.this.process(event, context, new
StateMachinePath(path,
372                                 "2.Кран включен", "A2"));
373
374                         fireAfterSubmachineExecution(context, event, path,
"2.Кран включен", "A2");
375
376                         return;
377                     case s1:
378
379                         return;
380                     case s2:
381
382                         return;
383                     default:
384                         throw new EventProcessorException("State with name [" +
config.getActiveState() + "] is unknown for state machine [A1]");
385                 }
386             }
387         }
388
389         private StateMachineConfig transiteToStableState(StateMachineContext
context, StateMachinePath path, StateMachineConfig config) throws Exception {
390
391             int s = decodeState(config.getActiveState());
392             Event event;
393
394             switch (s) {
395                 case Top:
396
397
398                     fireComeToState(context, path, "s1");
399
400                     // s1->1.Кран выключен [true]/
401                     event = Event.NO_EVENT;
402                     fireTransitionFound(context, path, "s1", event, "s1#1.Кран
выключен##true");
403
404
```

```
405                     fireComeToState(context, path, "1.Кран выключен");
406
407                     // 1.Кран выключен [o2.z1, o2.z6, o2.z4]
408                     fireBeforeOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z1");
409
410                     o2.z1(context);
411
412                     fireAfterOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z1");
413                     fireBeforeOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z6");
414
415                     o2.z6(context);
416
417                     fireAfterOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z6");
418                     fireBeforeOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z4");
419
420                     o2.z4(context);
421
422                     fireAfterOutputActionExecution(context, path, "s1#1.Кран
выключен##true", "o2.z4");
423
424                     return new StateMachineConfig("1.Кран выключен");
425             }
426
427             return config;
428         }
429
430         private StateMachineConfig lookForTransition(Event event,
StateMachineContext context, StateMachinePath path, StateMachineConfig config) throws
Exception {
431
432
433             BitSet calculatedInputActions = new BitSet(0);
434
435             int s = decodeState(config.getActiveState());
436             int e = decodeEvent(event.getName());
437
438             while (true) {
439                 switch (s) {
440                     case _1_Кран_выключен:
441
442
443                         switch (e) {
444                             case e208:
445
446                                 // 1.Кран выключен->2.Кран включен e208[true]/
447
448                                 fireTransitionCandidate(context, path, "1.Кран
выключен", event, "1.Кран выключен#2.Кран включен#e208#true");
449
450
451                                 fireTransitionFound(context, path, "1.Кран
выключен", event, "1.Кран выключен#2.Кран включен#e208#true");
452
453
454                                 fireComeToState(context, path, "2.Кран включен");
455
456                                 // 2.Кран включен [o2.z2, o2.z3, o2.z5]
457                                 fireBeforeOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z2");
458
459                                 o2.z2(context);
460
461                                 fireAfterOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z2");
462                                 fireBeforeOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z3");
463
464                                 o2.z3(context);
465
466                                 fireAfterOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z3");
```

```
467                                 fireBeforeOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z5");
468
469                                 o2.z5(context);
470
471                                 fireAfterOutputActionExecution(context, path,
"1.Кран выключен#2.Кран включен#e208#true", "o2.z5");
472                                 return new StateMachineConfig("2.Кран включен");
473
474
475                             case e210:
476
477                                 // 1.Кран выключен->s2 e210[true]/o2.z10
478
479                                 fireTransitionCandidate(context, path, "1.Кран
выключен", event, "1.Кран выключен#s2#e210#true");
480
481
482                                 fireTransitionFound(context, path, "1.Кран
выключен", event, "1.Кран выключен#s2#e210#true");
483
484                                 fireBeforeOutputActionExecution(context, path,
"1.Кран выключен#s2#e210#true", "o2.z10");
485
486                                 o2.z10(context);
487
488                                 fireAfterOutputActionExecution(context, path,
"1.Кран выключен#s2#e210#true", "o2.z10");
489
490                                 fireComeToState(context, path, "s2");
491
492                                 // s2 []
493                                 return new StateMachineConfig("s2");
494
495
496                             default:
497
498                                 // transition not found
499                                 return config;
500                         }
501
502                     case _2_Кран_включен:
503
504
505                         switch (e) {
506                             case e209:
507
508                                 // 2.Кран включен->1.Кран выключен e209[true]/
509
510                                 fireTransitionCandidate(context, path, "2.Кран
включен", event, "2.Кран включен#1.Кран выключен#e209#true");
511
512
513                                 fireTransitionFound(context, path, "2.Кран
включен", event, "2.Кран включен#1.Кран выключен#e209#true");
514
515
516                                 fireComeToState(context, path, "1.Кран
выключен");
517
518                                 // 1.Кран выключен [o2.z1, o2.z6, o2.z4]
519                                 fireBeforeOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z1");
520
521                                 o2.z1(context);
522
523                                 fireAfterOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z1");
524                                 fireBeforeOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z6");
525
526                                 o2.z6(context);
527
528                                 fireAfterOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z6");
```

```
529                                 fireBeforeOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z4");
530
531                                 o2.z4(context);
532
533                                 fireAfterOutputActionExecution(context, path,
"2.Кран включен#1.Кран выключен#e209#true", "o2.z4");
534                                 return new StateMachineConfig("1.Кран выключен");
535
536
537                             case e210:
538
539                                 // 2.Кран включен->s2 e210[true]/o2.z10
540
541                                 fireTransitionCandidate(context, path, "2.Кран
включен", event, "2.Кран включен#s2#e210#true");
542
543
544                                 fireTransitionFound(context, path, "2.Кран
включен", event, "2.Кран включен#s2#e210#true");
545
546                                 fireBeforeOutputActionExecution(context, path,
"2.Кран включен#s2#e210#true", "o2.z10");
547
548                                 o2.z10(context);
549
550                                 fireAfterOutputActionExecution(context, path,
"2.Кран включен#s2#e210#true", "o2.z10");
551
552                                 fireComeToState(context, path, "s2");
553
554                                 // s2 []
555                                 return new StateMachineConfig("s2");
556
557
558                             default:
559
560                                 // transition not found
561                                 return config;
562                         }
563
564                     default:
565                         throw new EventProcessorException("Incorrect stable state
[" + config.getActiveState() + "] in state machine [A1]");
566                 }
567             }
568         }
569
570
571     }
572
573
574     private class A2EventProcessor {
575
576         // states
577         private static final int Top = 1;
578         private static final int s1 = 2;
579         private static final int Кран_остановлен = 3;
580         private static final int Кран_перемещается_вверх = 4;
581         private static final int Захват_перемещается_влево = 5;
582         private static final int Кран_перемещается_вниз = 6;
583         private static final int Захват_перемещается_вправо = 7;
584         private static final int Захватить_контейнер = 8;
585         private static final int Отгрузить_контейнер = 9;
586         private static final int Начать_калибровку = 10;
587         private static final int Закончить_калибровку = 11;
588
589         private int decodeState(String state) {
590
591             if ("Top".equals(state)) {
592                 return Top;
593             } else if ("s1".equals(state)) {
594                 return s1;
595             } else if ("Кран остановлен".equals(state)) {
596                 return Кран_остановлен;
597             } else if ("Кран перемещается вверх".equals(state)) {
```

```
598                 return Кран_перемещается_вверх;
599             } else if ("Захват перемещается влево".equals(state)) {
600                 return Захват_перемещается_влево;
601             } else if ("Кран перемещается вниз".equals(state)) {
602                 return Кран_перемещается_вниз;
603             } else if ("Захват перемещается вправо".equals(state)) {
604                 return Захват_перемещается_вправо;
605             } else if ("Захватить контейнер".equals(state)) {
606                 return Захватить_контейнер;
607             } else if ("Отгрузить контейнер".equals(state)) {
608                 return Отгрузить_контейнер;
609             } else if ("Начать калибровку".equals(state)) {
610                 return Начать_калибровку;
611             } else if ("Закончить калибровку".equals(state)) {
612                 return Закончить_калибровку;
613             }
614 
615             return -1;
616         }
617 
618         // events
619         private static final int e207 = 1;
620         private static final int e205 = 2;
621         private static final int e206 = 3;
622         private static final int e101 = 4;
623         private static final int e203 = 5;
624         private static final int e204 = 6;
625         private static final int e202 = 7;
626         private static final int e201 = 8;
627         private static final int e211 = 9;
628 
629         private int decodeEvent(String event) {
630 
631             if ("e207".equals(event)) {
632                 return e207;
633             } else if ("e205".equals(event)) {
634                 return e205;
635             } else if ("e206".equals(event)) {
636                 return e206;
637             } else if ("e101".equals(event)) {
638                 return e101;
639             } else if ("e203".equals(event)) {
640                 return e203;
641             } else if ("e204".equals(event)) {
642                 return e204;
643             } else if ("e202".equals(event)) {
644                 return e202;
645             } else if ("e201".equals(event)) {
646                 return e201;
647             } else if ("e211".equals(event)) {
648                 return e211;
649             }
650 
651             return -1;
652         }
653 
654         private ru.ifmo.unicrane.co.Crane o1;
655         private ru.ifmo.unicrane.co.gui.Screen o2;
656 
657         private void init(ControlledObjectsMap controlledObjectsMap) {
658             o1 = (ru.ifmo.unicrane.co.Crane)
controlledObjectsMap.getControlledObject("o1");
659             o2 = (ru.ifmo.unicrane.co.gui.Screen)
controlledObjectsMap.getControlledObject("o2");
660         }
661 
662         private StateMachineConfig process(Event event, StateMachineContext
context, StateMachinePath path, StateMachineConfig config) throws Exception {
663             config = lookForTransition(event, context, path, config);
664 
665             config = transiteToStableState(context, path, config);
666 
667             // execute included state machines
668             executeSubmachines(event, context, path, config);
669 
670             return config;
```

```
671          }
672
673          private void executeSubmachines(Event event, StateMachineContext context,
StateMachinePath path, StateMachineConfig config) throws Exception {
674              int state = decodeState(config.getActiveState());
675
676              while (true) {
677                  switch (state) {
678                      case s1:
679
680                          return;
681                      case Кран_остановлен:
682
683                          return;
684                      case Кран_перемещается_вверх:
685
686                          return;
687                      case Захват_перемещается_влево:
688
689                          return;
690                      case Кран_перемещается_вниз:
691
692                          return;
693                      case Захват_перемещается_вправо:
694
695                          return;
696                      case Захватить_контейнер:
697
698                          return;
699                      case Отгрузить_контейнер:
700
701                          return;
702                      case Начать_калибровку:
703
704                          return;
705                      case Закончить_калибровку:
706
707                          return;
708                      default:
709                          throw new EventProcessorException("State with name [" +
config.getActiveState() + "] is unknown for state machine [A2]");
710                  }
711              }
712          }
713
714          private StateMachineConfig transiteToStableState(StateMachineContext
context, StateMachinePath path, StateMachineConfig config) throws Exception {
715
716              int s = decodeState(config.getActiveState());
717              Event event;
718
719              switch (s) {
720                  case Top:
721
722
723                      fireComeToState(context, path, "s1");
724
725                      // s1->Кран остановлен [true]/
726                      event = Event.NO_EVENT;
727                      fireTransitionFound(context, path, "s1", event, "s1#Кран
остановлен##true");
728
729
730                      fireComeToState(context, path, "Кран остановлен");
731
732                      // Кран остановлен []
733
734                      return new StateMachineConfig("Кран остановлен");
735              }
736
737              return config;
738          }
739
740          private StateMachineConfig lookForTransition(Event event,
StateMachineContext context, StateMachinePath path, StateMachineConfig config) throws
Exception {
```

```
741
742
743             boolean
744
745                     o1_x8 = false,
746
747                     o1_x6 = false,
748
749                     o1_x7 = false,
750
751                     o1_x4 = false,
752
753                     o1_x5 = false,
754
755                     o1_x2 = false,
756
757                     o1_x3 = false,
758
759                     o1_x1 = false;
760
761
762             BitSet calculatedInputActions = new BitSet(8);
763
764             int s = decodeState(config.getActiveState());
765             int e = decodeEvent(event.getName());
766
767             while (true) {
768                 switch (s) {
769                     case Кран_остановлен:
770
771
772                         switch (e) {
773                             case e207:
774
775                                 // Кран остановлен->Кран остановлен e207[o1.x3 ||
!o1.x5]/
776
777                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5");
778
779                                 if
(!isInputActionCalculated(calculatedInputActions, _o1_x5)) {
780
781                                     fireBeforeInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5", "o1.x5");
782
783                                     o1_x5 = o1.x5(context);
784
785                                     fireAfterInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5", "o1.x5", new Boolean(o1_x5));
786                                 }
787                                 if
(!isInputActionCalculated(calculatedInputActions, _o1_x3)) {
788
789                                     fireBeforeInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5", "o1.x3");
790
791                                     o1_x3 = o1.x3(context);
792
793                                     fireAfterInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5", "o1.x3", new Boolean(o1_x3));
794                                 }
795
796
797                                 if (o1_x3 || !o1_x5) {
798
799                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Кран остановлен#e207#o1.x3 || !o1.x5");
800
801
802                                     fireComeToState(context, path, "Кран
остановлен");
803
804                                     // Кран остановлен []
805                                     return new StateMachineConfig("Кран
остановлен");
```

```
806
807                                 }
808                                 // Кран остановлен->Отгрузить контейнер
e207[!o1.x3 && o1.x5]/
809
810                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Отгрузить контейнер#e207#!o1.x3 && o1.x5");
811
812
813                                 if (!o1_x3 && o1_x5) {
814
815                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Отгрузить контейнер#e207#!o1.x3 && o1.x5");
816
817
818                                     fireComeToState(context, path, "Отгрузить
контейнер");
819
820                                     // Отгрузить контейнер [o1.z6]
821                                     fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Отгрузить контейнер#e207#!o1.x3 && o1.x5", "o1.z6");
822
823                                     o1.z6(context);
824
825                                     fireAfterOutputActionExecution(context, path,
"Кран остановлен#Отгрузить контейнер#e207#!o1.x3 && o1.x5", "o1.z6");
826                                     return new StateMachineConfig("Отгрузить
контейнер");
827
828                                 }
829
830                                 // transition not found
831                                 return config;
832                             case e205:
833
834                                 // Кран остановлен->Кран остановлен e205[!o1.x3
|| !o1.x4]/
835
836                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4");
837
838                                 if
(!isInputActionCalculated(calculatedInputActions, _o1_x4)) {
839
840                                     fireBeforeInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4", "o1.x4");
841
842                                     o1_x4 = o1.x4(context);
843
844                                     fireAfterInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4", "o1.x4", new Boolean(o1_x4));
845                                 }
846                                 if
(!isInputActionCalculated(calculatedInputActions, _o1_x3)) {
847
848                                     fireBeforeInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4", "o1.x3");
849
850                                     o1_x3 = o1.x3(context);
851
852                                     fireAfterInputActionExecution(context, path,
"Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4", "o1.x3", new Boolean(o1_x3));
853                                 }
854
855
856                                 if (!o1_x3 || !o1_x4) {
857
858                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Кран остановлен#e205#!o1.x3 || !o1.x4");
859
860
861                                     fireComeToState(context, path, "Кран
остановлен");
862
863                                     // Кран остановлен []
```

```
864                                     return new StateMachineConfig("Кран
остановлен");
865
866                                 }
867                                 // Кран остановлен->Захватить контейнер
e205[o1.x3 && o1.x4]/
868
869                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Захватить контейнер#e205#o1.x3 && o1.x4");
870
871
872                                 if (o1_x3 && o1_x4) {
873
874                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Захватить контейнер#e205#o1.x3 && o1.x4");
875
876
877                                     fireComeToState(context, path, "Захватить
контейнер");
878
879                                     // Захватить контейнер [o1.z5]
880                                     fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Захватить контейнер#e205#o1.x3 && o1.x4", "o1.z5");
881
882                                     o1.z5(context);
883
884                                     fireAfterOutputActionExecution(context, path,
"Кран остановлен#Захватить контейнер#e205#o1.x3 && o1.x4", "o1.z5");
885                                     return new StateMachineConfig("Захватить
контейнер");
886
887                                 }
888
889                                 // transition not found
890                                 return config;
891                             case e203:
892
893                                 // Кран остановлен->Захват перемещается влево
e203[true]/
894
895                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Захват перемещается влево#e203#true");
896
897
898                                 fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Захват перемещается влево#e203#true");
899
900
901                                 fireComeToState(context, path, "Захват
перемещается влево");
902
903                                 // Захват перемещается влево []
904                                 return new StateMachineConfig("Захват
перемещается влево");
905
906
907                             case e204:
908
909                                 // Кран остановлен->Захват перемещается вправо
e204[true]/
910
911                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Захват перемещается вправо#e204#true");
912
913
914                                 fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Захват перемещается вправо#e204#true");
915
916
917                                 fireComeToState(context, path, "Захват
перемещается вправо");
918
919                                 // Захват перемещается вправо []
920                                 return new StateMachineConfig("Захват
перемещается вправо");
921
```

```
922
923                             case e202:
924
925                                 // Кран остановлен->Кран перемещается вниз
e202[true]/
926
927                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Кран перемещается вниз#e202#true");
928
929
930                                 fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Кран перемещается вниз#e202#true");
931
932
933                                 fireComeToState(context, path, "Кран перемещается
вниз");
934
935                                 // Кран перемещается вниз []
936                                 return new StateMachineConfig("Кран перемещается
вниз");
937
938
939                             case e201:
940
941                                 // Кран остановлен->Кран перемещается вверх
e201[true]/
942
943                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Кран перемещается вверх#e201#true");
944
945
946                                 fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Кран перемещается вверх#e201#true");
947
948
949                                 fireComeToState(context, path, "Кран перемещается
вверх");
950
951                                 // Кран перемещается вверх []
952                                 return new StateMachineConfig("Кран перемещается
вверх");
953
954
955                             case e211:
956
957                                 // Кран остановлен->Начать калибровку
e211[!o1.x8]/
958
959                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Начать калибровку#e211#!o1.x8");
960
961                                 if
(!isInputActionCalculated(calculatedInputActions, _o1_x8)) {
962
963                                     fireBeforeInputActionExecution(context, path,
"Кран остановлен#Начать калибровку#e211#!o1.x8", "o1.x8");
964
965                                     o1_x8 = o1.x8(context);
966
967                                     fireAfterInputActionExecution(context, path,
"Кран остановлен#Начать калибровку#e211#!o1.x8", "o1.x8", new Boolean(o1_x8));
968                                 }
969
970
971                                 if (!o1_x8) {
972
973                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Начать калибровку#e211#!o1.x8");
974
975
976                                     fireComeToState(context, path, "Начать
калибровку");
977
978                                     // Начать калибровку [o1.z7, o2.z9]
979                                     fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Начать калибровку#e211#!o1.x8", "o1.z7");
```

```
980
981                                     o1.z7(context);
982
983                                     fireAfterOutputActionExecution(context, path,
"Кран остановлен#Начать калибровку#e211#!o1.x8", "o1.z7");
984                                     fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Начать калибровку#e211#!o1.x8", "o2.z9");
985
986                                     o2.z9(context);
987
988                                     fireAfterOutputActionExecution(context, path,
"Кран остановлен#Начать калибровку#e211#!o1.x8", "o2.z9");
989                                     return new StateMachineConfig("Начать
калибровку");
990
991                                 }
992                                 // Кран остановлен->Закончить калибровку
e211[o1.x8]/
993
994                                 fireTransitionCandidate(context, path, "Кран
остановлен", event, "Кран остановлен#Закончить калибровку#e211#o1.x8");
995
996
997                                 if (o1_x8) {
998
999                                     fireTransitionFound(context, path, "Кран
остановлен", event, "Кран остановлен#Закончить калибровку#e211#o1.x8");
1000
1001
1002                                    fireComeToState(context, path, "Закончить
калибровку");
1003
1004                                    // Закончить калибровку [o1.z7, o2.z8]
1005                                    fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Закончить калибровку#e211#o1.x8", "o1.z7");
1006
1007                                    o1.z7(context);
1008
1009                                    fireAfterOutputActionExecution(context, path,
"Кран остановлен#Закончить калибровку#e211#o1.x8", "o1.z7");
1010                                    fireBeforeOutputActionExecution(context,
path, "Кран остановлен#Закончить калибровку#e211#o1.x8", "o2.z8");
1011
1012                                    o2.z8(context);
1013
1014                                    fireAfterOutputActionExecution(context, path,
"Кран остановлен#Закончить калибровку#e211#o1.x8", "o2.z8");
1015                                    return new StateMachineConfig("Закончить
калибровку");
1016
1017                                }
1018
1019                                // transition not found
1020                                return config;
1021                            default:
1022
1023                                // transition not found
1024                                return config;
1025                        }
1026
1027                    case Кран_перемещается_вверх:
1028
1029
1030                        switch (e) {
1031                            case e206:
1032
1033                                // Кран перемещается вверх->Кран остановлен
e206[true]/
1034
1035                                fireTransitionCandidate(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран остановлен#e206#true");
1036
1037
1038                                fireTransitionFound(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран остановлен#e206#true");
1039
```

```
1040
1041                                fireComeToState(context, path, "Кран
остановлен");
1042
1043                                // Кран остановлен []
1044                                return new StateMachineConfig("Кран остановлен");
1045
1046
1047                            case e101:
1048
1049                                // Кран перемещается вверх->Кран остановлен
e101[o1.x6]/
1050
1051                                fireTransitionCandidate(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран остановлен#e101#o1.x6");
1052
1053                                if
(!isInputActionCalculated(calculatedInputActions, _o1_x6)) {
1054
1055                                    fireBeforeInputActionExecution(context, path,
"Кран перемещается вверх#Кран остановлен#e101#o1.x6", "o1.x6");
1056
1057                                    o1_x6 = o1.x6(context);
1058
1059                                    fireAfterInputActionExecution(context, path,
"Кран перемещается вверх#Кран остановлен#e101#o1.x6", "o1.x6", new Boolean(o1_x6));
1060                                }
1061
1062
1063                                if (o1_x6) {
1064
1065                                    fireTransitionFound(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран остановлен#e101#o1.x6");
1066
1067
1068                                    fireComeToState(context, path, "Кран
остановлен");
1069
1070                                    // Кран остановлен []
1071                                    return new StateMachineConfig("Кран
остановлен");
1072
1073                                }
1074                                // Кран перемещается вверх->Кран перемещается
вверх e101[!o1.x6]/o1.z1,o2.z7
1075
1076                                fireTransitionCandidate(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран перемещается
вверх#e101#!o1.x6");
1077
1078
1079                                if (!o1_x6) {
1080
1081                                    fireTransitionFound(context, path, "Кран
перемещается вверх", event, "Кран перемещается вверх#Кран перемещается
вверх#e101#!o1.x6");
1082
1083                                    fireBeforeOutputActionExecution(context,
path, "Кран перемещается вверх#Кран перемещается вверх#e101#!o1.x6", "o1.z1");
1084
1085                                    o1.z1(context);
1086
1087                                    fireAfterOutputActionExecution(context, path,
"Кран перемещается вверх#Кран перемещается вверх#e101#!o1.x6", "o1.z1");
1088                                    fireBeforeOutputActionExecution(context,
path, "Кран перемещается вверх#Кран перемещается вверх#e101#!o1.x6", "o2.z7");
1089
1090                                    o2.z7(context);
1091
1092                                    fireAfterOutputActionExecution(context, path,
"Кран перемещается вверх#Кран перемещается вверх#e101#!o1.x6", "o2.z7");
1093
1094                                    fireComeToState(context, path, "Кран
перемещается вверх");
1095
1096                                    // Кран перемещается вверх []
```

```
1097                                    return new StateMachineConfig("Кран
перемещается вверх");
1098
1099                                }
1100
1101                                // transition not found
1102                                return config;
1103                            default:
1104
1105                                // transition not found
1106                                return config;
1107                        }
1108
1109                    case Захват_перемещается_влево:
1110
1111
1112                        switch (e) {
1113                            case e206:
1114
1115                                // Захват перемещается влево->Кран остановлен
e206[true]/
1116
1117                                fireTransitionCandidate(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Кран остановлен#e206#true");
1118
1119
1120                                fireTransitionFound(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Кран остановлен#e206#true");
1121
1122
1123                                fireComeToState(context, path, "Кран
остановлен");
1124
1125                                // Кран остановлен []
1126                                return new StateMachineConfig("Кран остановлен");
1127
1128
1129                            case e101:
1130
1131                                // Захват перемещается влево->Кран остановлен
e101[o1.x1]/
1132
1133                                fireTransitionCandidate(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Кран остановлен#e101#o1.x1");
1134
1135                                if
(!isInputActionCalculated(calculatedInputActions, _o1_x1)) {
1136
1137                                    fireBeforeInputActionExecution(context, path,
"Захват перемещается влево#Кран остановлен#e101#o1.x1", "o1.x1");
1138
1139                                    o1_x1 = o1.x1(context);
1140
1141                                    fireAfterInputActionExecution(context, path,
"Захват перемещается влево#Кран остановлен#e101#o1.x1", "o1.x1", new Boolean(o1_x1));
1142                                }
1143
1144
1145                                if (o1_x1) {
1146
1147                                    fireTransitionFound(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Кран остановлен#e101#o1.x1");
1148
1149
1150                                    fireComeToState(context, path, "Кран
остановлен");
1151
1152                                    // Кран остановлен []
1153                                    return new StateMachineConfig("Кран
остановлен");
1154
1155                                }
1156                                // Захват перемещается влево->Захват перемещается
влево e101[!o1.x1]/o1.z3,o2.z7
1157
```

```
1158                                fireTransitionCandidate(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Захват перемещается
влево#e101#!o1.x1");
1159
1160
1161                                if (!o1_x1) {
1162
1163                                    fireTransitionFound(context, path, "Захват
перемещается влево", event, "Захват перемещается влево#Захват перемещается
влево#e101#!o1.x1");
1164
1165                                    fireBeforeOutputActionExecution(context,
path, "Захват перемещается влево#Захват перемещается влево#e101#!o1.x1", "o1.z3");
1166
1167                                    o1.z3(context);
1168
1169                                    fireAfterOutputActionExecution(context, path,
"Захват перемещается влево#Захват перемещается влево#e101#!o1.x1", "o1.z3");
1170                                    fireBeforeOutputActionExecution(context,
path, "Захват перемещается влево#Захват перемещается влево#e101#!o1.x1", "o2.z7");
1171
1172                                    o2.z7(context);
1173
1174                                    fireAfterOutputActionExecution(context, path,
"Захват перемещается влево#Захват перемещается влево#e101#!o1.x1", "o2.z7");
1175
1176                                    fireComeToState(context, path, "Захват
перемещается влево");
1177
1178                                    // Захват перемещается влево []
1179                                    return new StateMachineConfig("Захват
перемещается влево");
1180
1181                                }
1182
1183                                // transition not found
1184                                return config;
1185                            default:
1186
1187                                // transition not found
1188                                return config;
1189                        }
1190
1191                    case Кран_перемещается_вниз:
1192
1193
1194                        switch (e) {
1195                            case e206:
1196
1197                                // Кран перемещается вниз->Кран остановлен
e206[true]/
1198
1199                                fireTransitionCandidate(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран остановлен#e206#true");
1200
1201
1202                                fireTransitionFound(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран остановлен#e206#true");
1203
1204
1205                                fireComeToState(context, path, "Кран
остановлен");
1206
1207                                // Кран остановлен []
1208                                return new StateMachineConfig("Кран остановлен");
1209
1210
1211                            case e101:
1212
1213                                // Кран перемещается вниз->Кран остановлен
e101[o1.x7]/
1214
1215                                fireTransitionCandidate(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран остановлен#e101#o1.x7");
1216
```

```
1217                                if
(!isInputActionCalculated(calculatedInputActions, _o1_x7)) {
1218
1219                                    fireBeforeInputActionExecution(context, path,
"Кран перемещается вниз#Кран остановлен#e101#o1.x7", "o1.x7");
1220
1221                                    o1_x7 = o1.x7(context);
1222
1223                                    fireAfterInputActionExecution(context, path,
"Кран перемещается вниз#Кран остановлен#e101#o1.x7", "o1.x7", new Boolean(o1_x7));
1224                                }
1225
1226
1227                                if (o1_x7) {
1228
1229                                    fireTransitionFound(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран остановлен#e101#o1.x7");
1230
1231
1232                                    fireComeToState(context, path, "Кран
остановлен");
1233
1234                                    // Кран остановлен []
1235                                    return new StateMachineConfig("Кран
остановлен");
1236
1237                                }
1238                                // Кран перемещается вниз->Кран перемещается вниз
e101[!o1.x7]/o1.z2,o2.z7
1239
1240                                fireTransitionCandidate(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран перемещается
вниз#e101#!o1.x7");
1241
1242
1243                                if (!o1_x7) {
1244
1245                                    fireTransitionFound(context, path, "Кран
перемещается вниз", event, "Кран перемещается вниз#Кран перемещается
вниз#e101#!o1.x7");
1246
1247                                    fireBeforeOutputActionExecution(context,
path, "Кран перемещается вниз#Кран перемещается вниз#e101#!o1.x7", "o1.z2");
1248
1249                                    o1.z2(context);
1250
1251                                    fireAfterOutputActionExecution(context, path,
"Кран перемещается вниз#Кран перемещается вниз#e101#!o1.x7", "o1.z2");
1252                                    fireBeforeOutputActionExecution(context,
path, "Кран перемещается вниз#Кран перемещается вниз#e101#!o1.x7", "o2.z7");
1253
1254                                    o2.z7(context);
1255
1256                                    fireAfterOutputActionExecution(context, path,
"Кран перемещается вниз#Кран перемещается вниз#e101#!o1.x7", "o2.z7");
1257
1258                                    fireComeToState(context, path, "Кран
перемещается вниз");
1259
1260                                    // Кран перемещается вниз []
1261                                    return new StateMachineConfig("Кран
перемещается вниз");
1262
1263                                }
1264
1265                                // transition not found
1266                                return config;
1267                            default:
1268
1269                                // transition not found
1270                                return config;
1271                        }
1272
1273                    case Захват_перемещается_вправо:
1274
1275
```

```
1276                        switch (e) {
1277                            case e206:
1278
1279                                // Захват перемещается вправо->Кран остановлен
e206[true]/
1280
1281                                fireTransitionCandidate(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Кран остановлен#e206#true");
1282
1283
1284                                fireTransitionFound(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Кран остановлен#e206#true");
1285
1286
1287                                fireComeToState(context, path, "Кран
остановлен");
1288
1289                                // Кран остановлен []
1290                                return new StateMachineConfig("Кран остановлен");
1291
1292
1293                            case e101:
1294
1295                                // Захват перемещается вправо->Кран остановлен
e101[o1.x2]/
1296
1297                                fireTransitionCandidate(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Кран остановлен#e101#o1.x2");
1298
1299                                if
(!isInputActionCalculated(calculatedInputActions, _o1_x2)) {
1300
1301                                    fireBeforeInputActionExecution(context, path,
"Захват перемещается вправо#Кран остановлен#e101#o1.x2", "o1.x2");
1302
1303                                    o1_x2 = o1.x2(context);
1304
1305                                    fireAfterInputActionExecution(context, path,
"Захват перемещается вправо#Кран остановлен#e101#o1.x2", "o1.x2", new Boolean(o1_x2));
1306                                }
1307
1308
1309                                if (o1_x2) {
1310
1311                                    fireTransitionFound(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Кран остановлен#e101#o1.x2");
1312
1313
1314                                    fireComeToState(context, path, "Кран
остановлен");
1315
1316                                    // Кран остановлен []
1317                                    return new StateMachineConfig("Кран
остановлен");
1318
1319                                }
1320                                // Захват перемещается вправо->Захват
перемещается вправо e101[!o1.x2]/o1.z4,o2.z7
1321
1322                                fireTransitionCandidate(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Захват перемещается
вправо#e101#!o1.x2");
1323
1324
1325                                if (!o1_x2) {
1326
1327                                    fireTransitionFound(context, path, "Захват
перемещается вправо", event, "Захват перемещается вправо#Захват перемещается
вправо#e101#!o1.x2");
1328
1329                                    fireBeforeOutputActionExecution(context,
path, "Захват перемещается вправо#Захват перемещается вправо#e101#!o1.x2", "o1.z4");
1330
1331                                    o1.z4(context);
1332
```

```
1333                                    fireAfterOutputActionExecution(context, path,
"Захват перемещается вправо#Захват перемещается вправо#e101#!o1.x2", "o1.z4");
1334                                    fireBeforeOutputActionExecution(context,
path, "Захват перемещается вправо#Захват перемещается вправо#e101#!o1.x2", "o2.z7");
1335
1336                                    o2.z7(context);
1337
1338                                    fireAfterOutputActionExecution(context, path,
"Захват перемещается вправо#Захват перемещается вправо#e101#!o1.x2", "o2.z7");
1339
1340                                    fireComeToState(context, path, "Захват
перемещается вправо");
1341
1342                                    // Захват перемещается вправо []
1343                                    return new StateMachineConfig("Захват
перемещается вправо");
1344
1345                                }
1346
1347                                // transition not found
1348                                return config;
1349                            default:
1350
1351                                // transition not found
1352                                return config;
1353                        }
1354
1355                    case Захватить_контейнер:
1356
1357
1358                        switch (e) {
1359                            default:
1360
1361                                // Захватить контейнер->Кран остановлен *[true]/
1362
1363                                fireTransitionCandidate(context, path, "Захватить
контейнер", event, "Захватить контейнер#Кран остановлен#*#true");
1364
1365
1366                                fireTransitionFound(context, path, "Захватить
контейнер", event, "Захватить контейнер#Кран остановлен#*#true");
1367
1368
1369                                fireComeToState(context, path, "Кран
остановлен");
1370
1371                                // Кран остановлен []
1372                                return new StateMachineConfig("Кран остановлен");
1373
1374
1375                        }
1376
1377                    case Отгрузить_контейнер:
1378
1379
1380                        switch (e) {
1381                            default:
1382
1383                                // Отгрузить контейнер->Кран остановлен *[true]/
1384
1385                                fireTransitionCandidate(context, path, "Отгрузить
контейнер", event, "Отгрузить контейнер#Кран остановлен#*#true");
1386
1387
1388                                fireTransitionFound(context, path, "Отгрузить
контейнер", event, "Отгрузить контейнер#Кран остановлен#*#true");
1389
1390
1391                                fireComeToState(context, path, "Кран
остановлен");
1392
1393                                // Кран остановлен []
1394                                return new StateMachineConfig("Кран остановлен");
1395
1396
1397                        }
```

```
1398
1399                     case Начать_калибровку:
1400
1401
1402                         switch (e) {
1403                             default:
1404
1405                                 // Начать калибровку->Кран остановлен *[true]/
1406
1407                                 fireTransitionCandidate(context, path, "Начать
калибровку", event, "Начать калибровку#Кран остановлен#*#true");
1408
1409
1410                                 fireTransitionFound(context, path, "Начать
калибровку", event, "Начать калибровку#Кран остановлен#*#true");
1411
1412
1413                                 fireComeToState(context, path, "Кран
остановлен");
1414
1415                                 // Кран остановлен []
1416                                 return new StateMachineConfig("Кран остановлен");
1417
1418
1419                         }
1420
1421                     case Закончить_калибровку:
1422
1423
1424                         switch (e) {
1425                             default:
1426
1427                                 // Закончить калибровку->Кран остановлен *[true]/
1428
1429                                 fireTransitionCandidate(context, path, "Закончить
калибровку", event, "Закончить калибровку#Кран остановлен#*#true");
1430
1431
1432                                 fireTransitionFound(context, path, "Закончить
калибровку", event, "Закончить калибровку#Кран остановлен#*#true");
1433
1434
1435                                 fireComeToState(context, path, "Кран
остановлен");
1436
1437                                 // Кран остановлен []
1438                                 return new StateMachineConfig("Кран остановлен");
1439
1440
1441                         }
1442
1443                     default:
1444                         throw new EventProcessorException("Incorrect stable state
[" + config.getActiveState() + "] in state machine [A2]");
1445                 }
1446             }
1447         }
1448
1449         //o1.x8
1450         private static final int _o1_x8 = 0;
1451         //o1.x6
1452         private static final int _o1_x6 = 1;
1453         //o1.x7
1454         private static final int _o1_x7 = 2;
1455         //o1.x4
1456         private static final int _o1_x4 = 3;
1457         //o1.x5
1458         private static final int _o1_x5 = 4;
1459         //o1.x2
1460         private static final int _o1_x2 = 5;
1461         //o1.x3
1462         private static final int _o1_x3 = 6;
1463         //o1.x1
1464         private static final int _o1_x1 = 7;
1465
1466     }
```

```
1467
1468     private static boolean isInputActionCalculated(BitSet calculatedInputActions,
int k) {
1469         boolean b = calculatedInputActions.get(k);
1470
1471         if (!b) {
1472             calculatedInputActions.set(k);
1473         }
1474
1475         return b;
1476     }
1477
1478 }
```