Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет «Информационных технологий и программирования»
Кафедра «Компьютерные технологии»

Г. М. Рыбаков, А. А. Шалыто

Моделирование работы системы интеллектуального здания

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

Санкт-Петербург
2007

Введение

Термин «интеллектуальное здание» (intelligent building - англ.; intelligent – «разумный, понятливый», в сочетании со словом building – «гибкий, приспосабливаемый») в первоначальном смысле означает «здание, готовое к изменениям» или «приспосабливаемое здание: здание, способное приспосабливаться к изменениям окружающей среды. Другими словами, это здание, инженерные системы которого способны обеспечить адаптацию к возможным изменениям в будущем. Здание, в котором различные системы объединены в интегрированный комплекс и правильно организованы уже на этапе проектирования (с учетом возможных будущих изменений). Однозначного определения «интеллектуального здания» нет, но в понимании большинства – это автоматизированная техническая система, которая:

- «чувствует», что происходит в здании и за его пределами;
- «реагирует», обеспечивая максимально безопасное и комфортабельное пребывание в здании, сводит до минимума потребление энергоресурсов;
- «взаимодействует» с людьми посредством применения простых и легкодоступных средств общения.

В сравнении с набором автономных систем комплексная система имеет следующие преимущества использования:

- происходит существенная экономия на кабельных сетях и сетевом оборудовании;
- снижается энергопотребление и повышается надежность системы;
- повышается оперативность управления объектами;
- снижаются трудозатраты эксплуатационных и диспетчерских служб;
- обеспечивается взаимодействие между системами;
- уменьшается вероятность возникновения страховых случаев;
- «открытость» комплексной системы обеспечивает возможность его наращивания и использования оборудования разных производителей.

Интеллектуальное здание является продуктом современного развития систем автоматики в зданиях в направлении:

- комплексной оптимизации использования ресурсов;
- повышения гибкости конфигурирования и снижения общей стоимости владения;
- интеграции с широким спектром технологического и телекоммуникационного оборудования;
- упрощения взаимодействия с пользователем.

В состав «интеллектуального здания» входят следующие системы и комплексы:

- Комплекс систем жизнеобеспечения (*Life Safety*). Предназначен для организации управления системами жизнеобеспечения.
- Комплекс систем безопасности. Предназначен для защиты человеческой жизни и сохранности материальных ценностей и информации организации.
- Комплекс систем информатизации. Являются базисом, на котором строятся все компоненты информационно-вычислительных сетей интеллектуального здания.
- Система сбора и обработки информации.

1. Постановка задачи

Задача проекта – построить модель интеллектуальной системы для одного помещения.

При этом были выбраны следующие подсистемы:

- пожарной безопасности;
- вентиляции;
- управления электропитанием;
- доступа (модуль дверной).

С прикладной точки зрения комплексная система интеллектуального здания представляет собой совокупность устройств (датчики, контроллеры, рабочие станции и т.д.) и среду обмена информацией между ними (сеть). С другой стороны, можно рассматривать комплексную систему, как совокупность подсистем (пожарной безопасности, контроля электрического питания и т.д.) – на более высоком логическом уровне.

Предложенная автором модель совмещает элементы этих двух подходов, поддерживая взаимодействие устройств и подсистем. Это позволяет подробно изучать работу некоторых сложных устройств по отдельности и рассматривать взаимодействие нескольких простых устройств как единой системы. Каждый из элементов (устройство или подсистема) в ходе эксплуатации может находиться в некоторых выделенных состояниях с определенными правилами перехода между ними. Правила задаются состояниями внешней среды и/или взаимодействием компонентов системы друг с другом. Благодаря использованию SWITCH-технологии анализ состояний и переходов позволяет восстановить или спрогнозировать правильное поведение системы. Это приводит к существенному уменьшению количества ошибок, как в процессе разработки, так и при эксплуатации системы.

Инструментальные средства: *Eclipse 3.2*, *Java 5.0*, *Unimod 1.3.38*.

2. Проектирование системы

2.1. Обмен сообщениями

Концептуально система базируется на некоторой общей шине – сети для обмена сообщениями, к которой подключены все устройства (рис.1).

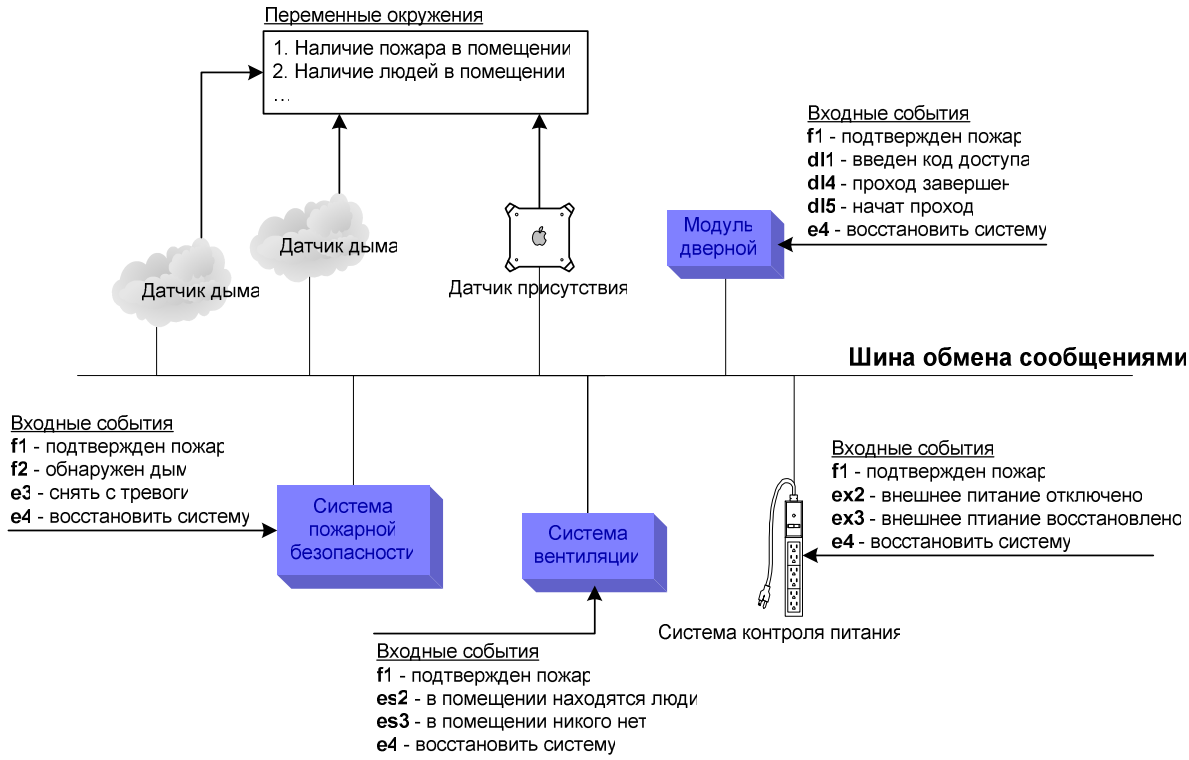

Рис. 1. Концептуальная модель системы

Устройство – подсистема или отдельный элемент, подключенный к сети и поддерживающий общий протокол обмена сообщениями.

Сообщение (входное сообщение) – некоторый объем информации (в программной реализации – объект), который устройства могут передавать всем остальным устройствам по общей шине обмена данными.

Эти устройства реализованы *параллельно в разных потоках*. Это позволяет максимально приблизить ситуацию к реальности. Каждое устройство содержит свою безопасную (с точки зрения `multithreading`) *очередь сообщений* и с некоторым интервалом времени проверяет наличие очередного сообщения в очереди. Роль шины в модели выполняет singleton-класс `Messager`, позволяющий устройствам подписаться на получение сообщений и отправлять сообщения всем подписчикам:

```
public class Messager {
      public void addReciever(IReciever r) {…} // Подписаться на сообщения
      public static void send(Message msg) {…} // Отправить сообщение всем
      ...
}
```

Для этого каждое устройство реализует интерфейс `IReciever`, описанный внутри `Messager`:

```
public interface IReciever {
      public void recieve(Message msg);
}
```

На рис.2 приведена диаграмма классов системы.

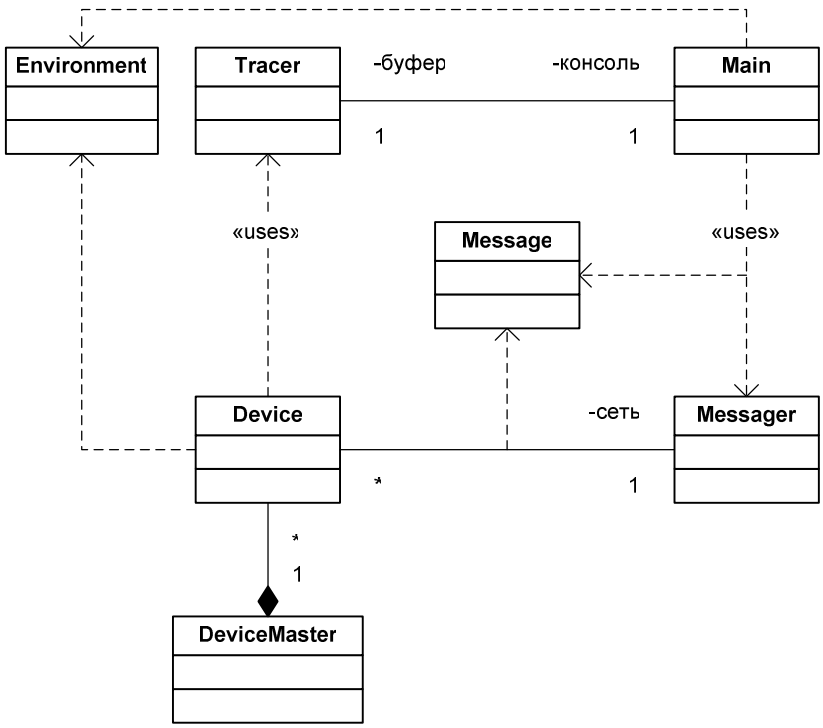

Рис.2. Статическая диаграмма классов системы

При отправке сообщения класс `Messager` помещает новое сообщение в очередь каждого устройства с помощью метода `receive(…)`. Сообщение представляет собой объект класса `Message`, содержащий помимо уникального идентификатора сообщения, идентификатор устройства-источника, идентификатор устройства-приемника и типа сообщения, а также набор параметров, посредством которых устройства могут обмениваться информацией:

```
public abstract class Message {
      public String event;          // Тип сообщения
      public long source;           // id источника
      public long destination;      // id назначения
      public long param1;           // Целый параметр 1
      public long param2;           // Целый параметр 2
      public String param3;         // Строковый параметр
      public abstract long id();    // Возвращает уникальный идентификатор
      public String toString() {…};
}
```

Все устройства при создании регистрируются в едином банке устройств `DeviceMaster`. Это позволяет одним устройствам находить другие по имени, что упрощает программирование. Класс `Environment` представляет собой набор переменных окружения, определяющих состояние помещения. Класс `Tracer` – утилитарный класс, перенаправляющий информационный вывод с устройств на консоль класса `Main`.

Модель устройства

Приведем общий принцип работы устройства в модели. Его базовая функциональность описана в классе `Device`. В зависимости от сложности устройства в его работе может использоваться или не использоваться внутренний автомат [1], созданный с помощью инструментального средства *Unimod*. Описание автомата содержится во внешнем файле в xml-формате, который автоматически компилируется при создании объекта.

Событие (*входное событие*) – порождается входным сообщением (или другими внутренними средствами устройства) и подается на обработку автомату устройства.

Обычно в модели автомата класс устройства выступает в роли, как источника событий, так и управляемого объекта (рис. 3).

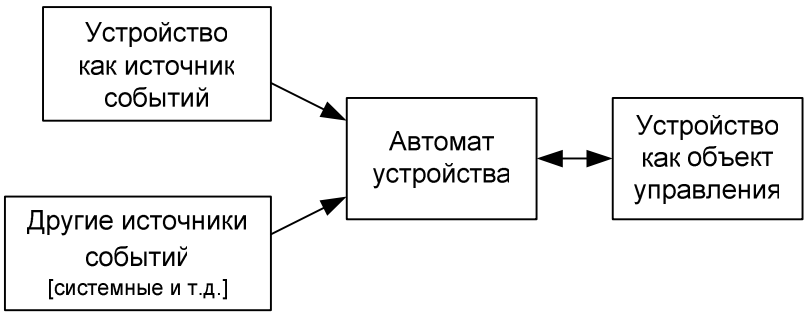

Рис.3. Диаграмма связей для автомата устройства

Помимо класса устройства могут использоваться и любые другие источники событий (системный, общий или класс любого другого устройства, например, подсистемы пожарной безопасности).

Рассмотрим общую схему обработки входного сообщения (проверка выполняется постоянно с заданным интервалом) (рис. 4).

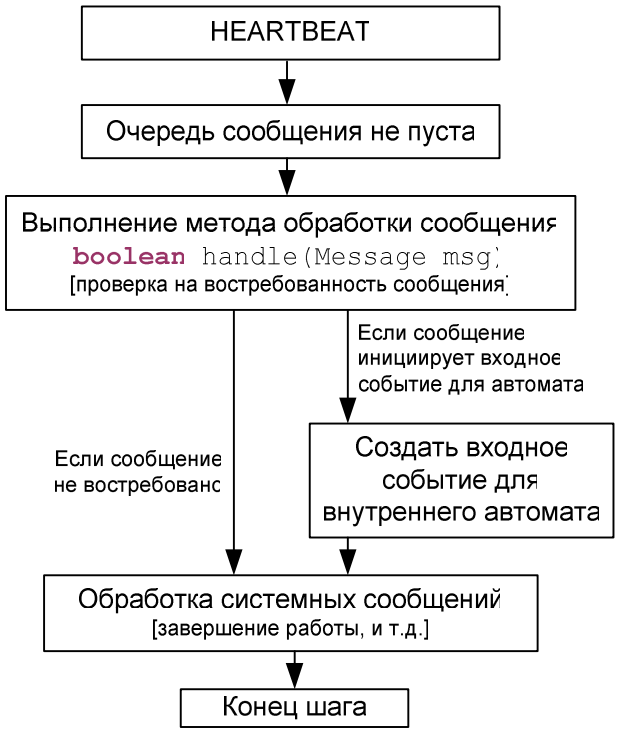

Рис. 4. Схема процесса обработки входного сообщения

На схеме HEARTBEAT – этап отправки сообщения типа *HEARTBEAT* – «сердцебиение» (`SystemEventProvider.E_HEARTBEAT`). Постоянная отправка таких сообщений позволяет заинтересованным участникам системы отслеживать работоспособность устройств (например, если в течение некоторого времени не получено сообщение типа *HEARTBEAT*

от какого-либо устройства, то следовательно устройство находится в неисправном состоянии или с ним потеряна связь).

Далее, если в очереди есть сообщение – оно извлекается и обрабатывается устройством. На первом этапе обработки входного сообщения вызывается метод `handle` (рис. 4) устройства. Его могут переопределять дочерние классы устройств, что позволяет им по-своему обработать полученное сообщение: сохранить переданную информацию, принять решение о его востребованности конкретным типом устройства и т.д. После этого, если необходимо (решение принимается методом `handle`), по сообщению создается новое входное событие для автомата. Входные события также могут быть созданы любыми другими внутренними средствами устройства (например, внутренним таймером, как в датчике присутствия). Извне входные события могут быть инициированы только отправкой сообщения.

Для датчиков разработан специальный класс `Sensor`, унаследованный от класса `Device`, который предоставляет готовый к использованию внутренний таймер (рис. 5). Таймер работает с заданным интервалом времени. Это позволяет с некоторой периодичностью проверять состояние помещения (переменные окружения и т.д.).

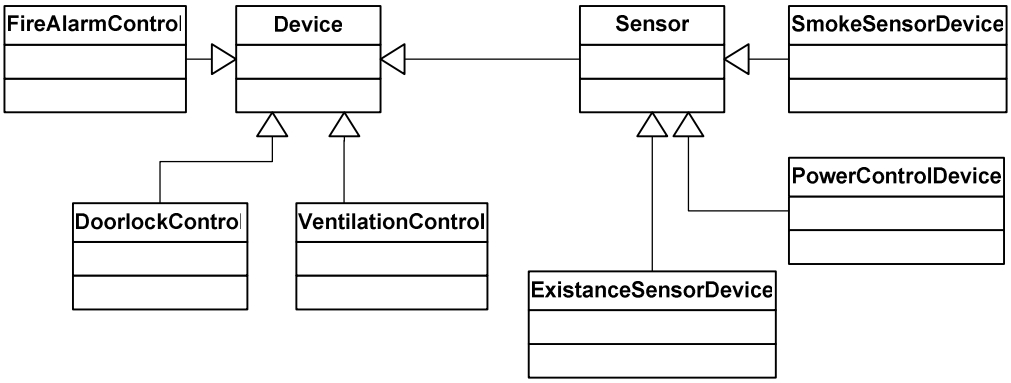

Рис.5. Отношение наследования между устройствами

Конфигурация системы

Конфигурация системы (список устройств и их параметров) описывается во внешнем файле в *xml*-формате.

Каждое устройство описывается блоком тегов `<device></device>`, с параметрами `name` (уникальное имя устройства), `class` (класс устройства) и `xml` (путь к *xml*-файлу с описанием автомата устройства). Параметр `xml` может быть опущен, если в работе соответствующего устройства не используется внутренний автомат. *Xml*-файл с описанием автомата устройства создается с помощью инструментального средства *Unimod*.

Внутри блока устройства можно указывать его параметры в формате `<param name="имя параметра" value="значение параметра"/>`. При интерпретации конфигурационного файла с помощью технологии *Java Reflections* создаются объекты соответствующих классов, и у них выставляются соответствующие параметры. Имя параметра должно соответствовать `public` полю объекта устройства. В зависимости от типа поля будет выполнена попытка приведения к нему указанной строки значения параметра.

Приведем отрывок конфигурационного файла:

```
<?xml version="1.0" encoding="Windows-1251"?>

<ihouse xmlns = "http://www.ifmo.ru/ihouse">
    <device
      name  = "Power Controller"
      class = "ru.ifmo.ihouse.powercontrol.PowerControlDevice"
      xml   = "xml/powercontrol.xml"
    >
    </device>

    <!-- Пожарная безопасность -->
    <device
      name  = "SmokeSensor1"
      class = "ru.ifmo.ihouse.smokesensor.SmokeSensorDevice"
    >
    </device>
    <device
      name  = "SmokeSensor2"
      class = "ru.ifmo.ihouse.smokesensor.SmokeSensorDevice"
    >
    </device>
    <device
      name  = "FireAlarm1"
      class = "ru.ifmo.ihouse.firealarm.FireAlarmControl"
      xml   = "xml/firealarm.xml"
    >
      <param name="smokeSensors" value="SmokeSensor1,SmokeSensor2"/>
    </device>
    ...
</ihouse>
```

Модель помещения

Состояние помещения описывается набором переменных окружения (табл. 1), которые определяют наличие внешнего электропитания, наличие людей в помещении или пожарной обстановки. Переменные описаны в утилитарном статическом классе `Environment`:

```
public class Environment {
      public static Boolean isPersonsInside = false;
      public static Boolean isFireStarted   = false;
      public static Boolean isExSupply      = true;
}
```

Таблица 1. Переменные окружения

Переменная окружения	**Описание**
isPersonsInside	True, если в помещении кто-то есть. False – в обратном случае (по-умолчанию False).
isFireStarted	True, если в помещении начался пожар. False – в обратном случае (по-умолчанию False).
isExSupply	True, если внешнее питание включено. False – в обратном случае (по-умолчанию True).

Значение переменных можно изменить с помощью консоли управления. Ниже приведен пример установки переменной окружения «пожар»:

```
...
17:13:47 [TRACE] (4) 'FireAlarm1' reported: ПОЖАР1 (возможен пожар)
17:13:47 [TRACE] (4) 'FireAlarm1' reported: получен тревожный сигнал от датчика дыма
```

```
17:13:47 [TRACE] (2) 'SmokeSensor1' reported: сработал датчик дыма
17:13:47 [TRACE] (3) 'SmokeSensor2' reported: сработал датчик дыма
> setenv isFireStarted true - УСТАНОВЛЕНО // Начался пожар
17:13:31 [TRACE] (7) 'DoorLock' reported: Ожидание кода.
17:13:31 [TRACE] (6) 'VentilationControl' reported: переход в режим энергосбережения
17:13:31 [TRACE] (1) 'Power Controller' reported: переход в режим внешнего питания
...
```

Консоль управления

Для управления работой системы предусмотрена простая текстовая консоль для ввода команд. С ее помощью можно отправлять сообщения в систему и выставлять значения переменных окружения. Приведем отрывок лога работы системы:

```
11:45:15 [TRACE] (7) 'DoorLock' reported: Ожидание кода.
11:45:15 [TRACE] (4) 'FireAlarm1' reported: Система пожарной тревоги перезагружена.
11:45:15 [TRACE] (6) 'VentilationControl' reported: переход в режим энергосбережения
11:45:15 [TRACE] (7) 'DoorLock' reported: Тревога снята. Дверь на охране.
11:45:15 [TRACE] (1) 'Power Controller' reported: переход в режим внешнего питания
> send e4 - ОТПРАВЛЕНО : ID=75 e=e4 d=0 s=0 p1=0 p2=0 p3=null
11:45:07 [TRACE] (6) 'VentilationControl' reported: переход в экстренный режим
пожарной тревоги. Вентиляция остановлена, начата заборка дыма.
11:45:07 [TRACE] (7) 'DoorLock' reported: Пожар подтвержден. Дверь разблокирована.
11:45:07 [TRACE] (1) 'Power Controller' reported: переход в режим АВАРИЙНОГО
низковольтового питания
11:45:07 [TRACE] (4) 'FireAlarm1' reported: отправлена тревога в службу пожарной
безопасности. Ожидание ответа.
11:45:07 [ALERT] (4) 'FireAlarm1' reported: ПОЖАР2 (подтверждение пожара)
11:45:07 [TRACE] (4) 'FireAlarm1' reported: получен тревожный сигнал от датчика дыма
11:45:07 [TRACE] (4) 'FireAlarm1' reported: ПОЖАР1 (возможен пожар)
11:45:07 [TRACE] (4) 'FireAlarm1' reported: получен тревожный сигнал от датчика дыма
11:45:06 [TRACE] (2) 'SmokeSensor1' reported: сработал датчик дыма
11:45:06 [TRACE] (3) 'SmokeSensor2' reported: сработал датчик дыма
> setenv isFireStarted true - УСТАНОВЛЕНО
11:44:52 [TRACE] (6) 'VentilationControl' reported: переход в режим энергосбережения
11:44:52 [TRACE] (7) 'DoorLock' reported: Ожидание кода.
11:44:52 [TRACE] (1) 'Power Controller' reported: переход в режим внешнего питания
```

Синим цветом выделены команды введенные пользователем. Черным – обычные системные уведомления от устройств. Красным – тревожные сообщения. Формат уведомления от устройства:

```
<время> [TRACE|ALERT] (<id устройства>) '<имя устройства>' reported: текст
```

Формат команд управления

Отправка сообщения:

send <тип сообщения> *<устройство назначения> <устройство источник> <параметр 1> <параметр 2> <параметр 3 (строка)>*

Элементы, выделенные курсивом, можно опустить с конца, например:

```
send dl1 0 0 111
```

Отправка входного кода дверному модулю. Код передается в первом целочисленном параметре, поэтому необходимо указать источник и назначение (могут быть любыми), а второй и третий параметры можно опустить.

Установка переменной окружения:

setenv <название переменной> <значение>

Например, следующая строка создаст в помещении пожар:

```
setenv isFireStarted true
```

Проектирование подсистем

Датчик дыма

Описание

Простейший датчик (класс `SmokeSensorDevice`), сигнализирующий об обнаружении дыма в помещении. Для определения наличия дыма используется переменная окружения наличия пожара в помещении `Environment.isFireStarted`. Если значение переменной равно TRUE, датчик один раз отправляет сообщение `FireAlarmControl.F2` – «обнаружен дым» (табл. 2). Автомат в этом устройстве не используется, датчик унаследован от класса сенсор и проверяет состояние по внутреннему таймеру.

Таблица 2. Выходные сообщения датчика дыма

Выходные сообщения	Описания
`FireAlarmControl.F2`	В помещении обнаружен дым

Датчик присутствия

Описание

Датчик присутствия (класс `ExistanceSensorDevice`) позволяет контролировать наличие людей в помещении, что позволяет некоторым устройствам (например, системе вентиляции) переходить в энергосберегающий режим (когда никого нет) и обратно. В основе устройства датчика лежит простейший автомат, который по внутреннему событию `ExistanceSensorDevice.ES1` (состояние присутствия изменилось) в зависимости от наличия людей в помещении отправляет в сеть соответствующее сообщение (табл. 3).

Таблица 3. Выходные сообщения датчика присутствия

Выходные сообщения	Описание
`ExistanceSensorDevice.ES2`	В помещении кто-то есть
`ExistanceSensorDevice.ES3`	В помещении никого нет

Схема связей

Схема связей автомата приведена на рис. 6.

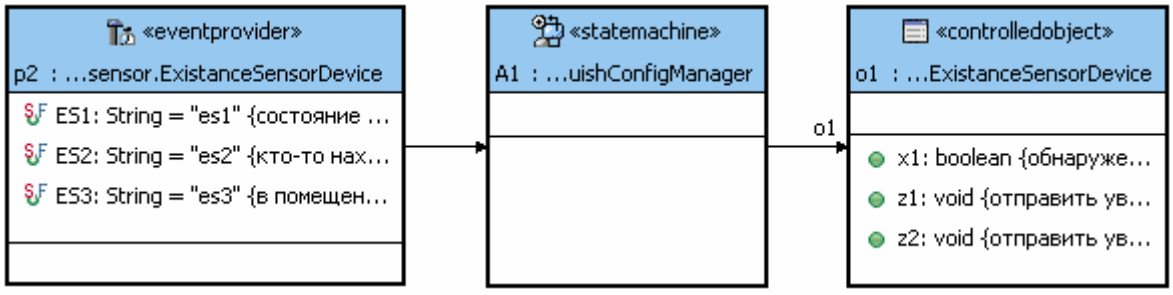

Рис. 6. Схема связей автомата датчика присутствия

Ниже приведены входные сообщения датчика присутствия (табл. 4).

Таблица 4. Входные сообщения датчика присутствия

Источник событий ExistanceSensorDevice	
Входные события	**Описание**
ExistanceSensorDevice.*ES1*	Состояние присутствия в помещении изменилось
ExistanceSensorDevice.*ES2*	В помещении кто-то есть
ExistanceSensorDevice.*ES3*	В помещении никого нет

В табл. 5 приведены входные воздействия датчика присутствия.

Таблица 5. Входные воздействия датчика присутствия

Объект управления ExistanceSensorDevice	
Воздействие	**Описание**
ExistanceSensorDevice.x1	Булевое входное воздействие присутствия людей в помещении
ExistanceSensorDevice.z1	Отправить сообщение о наличии людей в помещении
ExistanceSensorDevice.z2	Отправить сообщение об отсутствии людей в помещении

Описание автомата

Автомат датчика присутствия (рис. 7) всегда находится в одном состоянии проверки присутствия людей в помещении. По событию **es1** (состояние присутствия в помещении изменилось), в зависимости от того, появился кто-то или помещение опустело, соответствующие информационные сообщения отправляются в сеть.

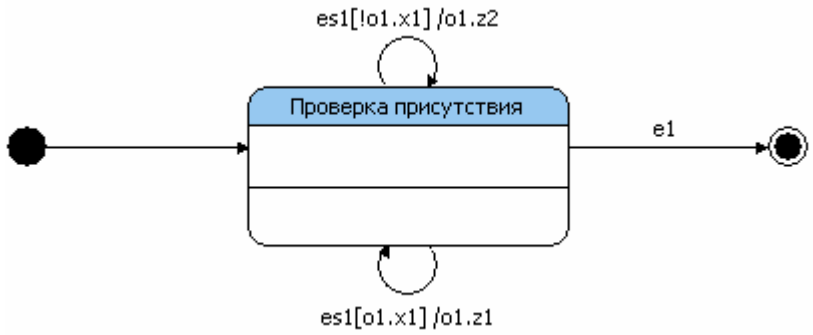

Рис 7. Диаграмма переходов для автомата датчика присутствия

Подсистема пожарной безопасности

Описание

Подсистема пожарной безопасности (класс `FireAlarmControl`) – сложная структура, контролирующая пожарную обстановку в помещении и способная самостоятельно принять меры к тушению пожара в случае его появления. По Российским стандартам для автоматического подтверждения пожара достаточно двух утвердительных сигналов от датчиков дыма.

Различают два состояния пожарной тревоги: *ПОЖАР-1* и *ПОЖАР-2*. Первому состоянию соответствует ситуация возможного пожара (сработал только один датчик). В состоянии *ПОЖАР-2* (подтверждение пожара) система автоматически примет меры по тушению. В первую очередь, будет отправлен сигнал подтверждения пожара в систему (табл. 6). После этого будет отправлен запрос во внешнюю службу пожарной безопасности (в ближайшую пожарную часть или на пульт охраны). Если ответ о приеме вызова не будет получен, система начнет автоматическое тушение пожара.

Таблица 6. Выходные сообщения подсистемы пожарной безопасности

Выходные сообщения	**Описания**
FireAlarmControl.*F1*	В помещении подтвержден пожар (ПОЖАР-1)
FireAlarmControl.*F3*	В возможен пожар (ПОЖАР-2)

В табл. 7 приведены входные сообщения подсистемы пожарной безопасности.

Таблица 7. Входные сообщения подсистемы пожарной безопасности

Входные сообщения	**Описания**
SystemEventProvider.*e3*	Сообщение «Снять с охраны»
SystemEventProvider.*e4*	Сообщение «Перезагрузить/Восстановить систему»
FireAlarmControl.*F1*	В помещении подтвержден пожар (ПОЖАР-1); например, второй датчик дыма не сработал, но пришло подтверждения от оператора
FireAlarmControl.*F2*	В помещении обнаружен дым

Схема связей

Схема связей автомата подсистемы пожарной безопасности приведена на рис. 8.

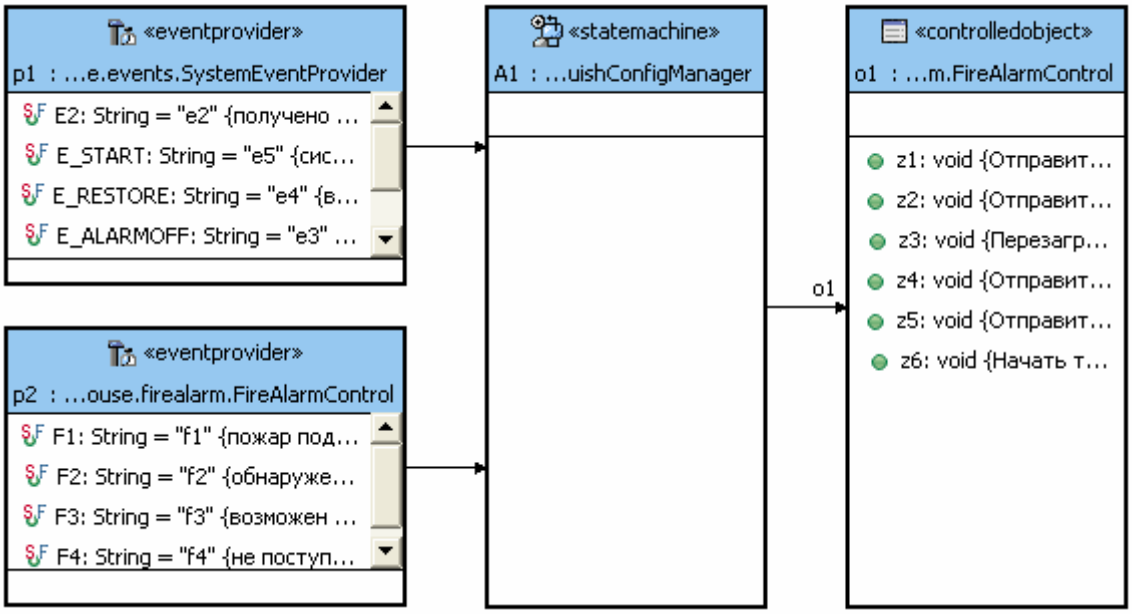

Рис. 8. Схема связей автомата подсистемы пожарной безопасности

В табл. 8 приведены входные сообщения подсистемы пожарной безопасности.

Таблица 8. Входные сообщения подсистемы пожарной безопасности

Источник событий FireAlarmControl	
Входные события	**Описание**
FireAlarmControl.*F1*	Пожар подтвержден (ПОЖАР-2)
FireAlarmControl.*F2*	В помещении обнаружен дым
FireAlarmControl.*F3*	В помещении возможен пожар (ПОЖАР-1)
FireAlarmControl.*F4*	Не поступил ответ от внешней пожарной службы

Входные воздействия подсистемы пожарной безопасности приведены в табл. 9.

Таблица 9. Входные воздействия подсистемы пожарной безопасности

Объект управления FireAlarmControl	
Воздействие	**Описание**
FireAlarmControl.z1	Отправить сообщение о возможном пожаре (ПОЖАР-1)
FireAlarmControl.z2	Отправить сообщение о пожаре (ПОЖАР-2)
FireAlarmControl.z3	Перезагрузить систему пожарной безопасности
FireAlarmControl.z4	Сообщить о сигнале от датчика дыма
FireAlarmControl.z5	Отправить запрос во внешнюю службу пожарной безопасности
FireAlarmControl.z6	Начать тушение пожара (включить систему пожаротушения)

Описание автомата

В начальный момент времени автомат находится в состоянии проверки пожарного состояния в помещении. Как только поступает первый сигнал от датчика дыма, автомат переходит в состояние *ПОЖАР-1* (пожар возможен) и отправляет соответствующее уведомление (FireAlarmControl.*F3* – в помещении возможен пожар). При срабатывании второго датчика дыма или при подтверждении пожара оператором, система переходит в тревожное состояние *ПОЖАР-2* и отправляет запрос во внешнюю службу пожарной безопасности. Если в течение определенного времени ответ не приходит система приступает к самостоятельному тушению пожара. При этом из любого состояния по приходу соответствующего сообщения снятия тревоги система может перейти в нетревожное начальное состояние (рис. 9).

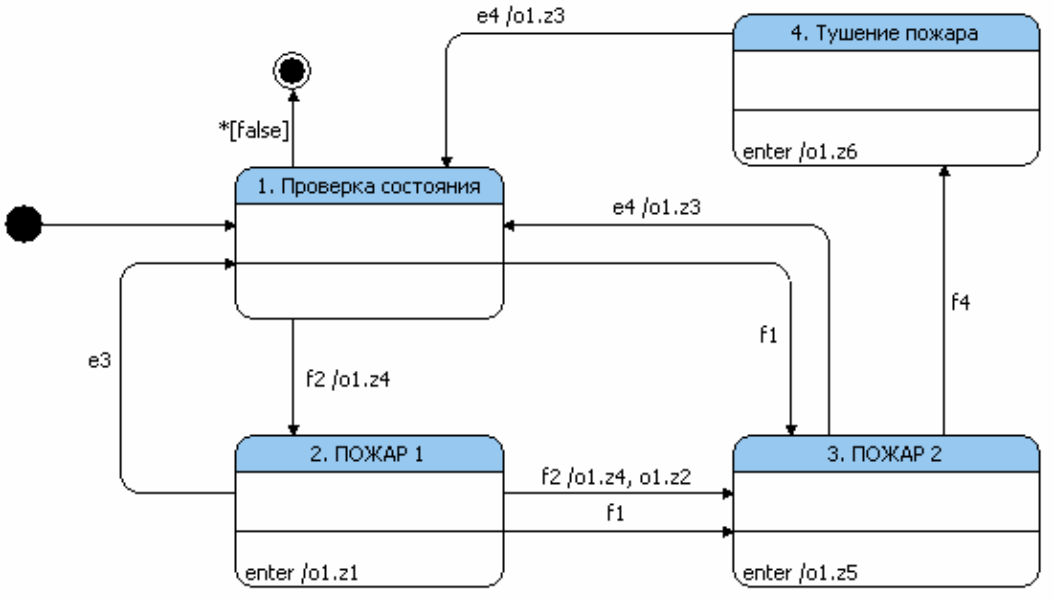

Рис. 9. Входные сообщения подсистемы пожарной безопасности

Подсистема контроля питания

Описание

Подсистема контроля электропитания (класс `PowerControlDevice`) отвечает за работу в режимах наличия и отсутствия внешнего питания и за перевод в аварийный пожарный низковольтовый режим питания в случае подтверждения пожарной тревоги и обратно (табл. 10). Проверка наличия внешнего питания выполняется аналогично датчикам, класс подсистемы также наследуется от класса Sensor.

Таблица 10. Входные сообщения подсистемы контроля питания

Входные сообщения	**Описания**
SystemEventProvider.*e3*	Сообщение «Снять с охраны»
SystemEventProvider.*e4*	Сообщение «Перезагрузить/Восстановить систему»
FireAlarmControl.*F1*	В помещении подтвержден пожар (ПОЖАР-1); например, второй датчик дыма не сработал, но пришло подтверждения от оператора
FireAlarmControl.*F2*	В помещении обнаружен дым

При изменении состояния питания в сеть отправляются сообщения P1 (нет внешнего питания) или P2 (внешнее питание восстановлено) (табл. 11).

Таблица 11. Выходные сообщения подсистемы пожарной безопасности

Выходные сообщения	**Описания**
PowerControlDevice.*P1*	Внешнее питание отключено
PowerControlDevice.*P2*	Внешнее питание включено

Схема связей

Схема связей автомата подсистемы контроля питания приведена на рис. 10.

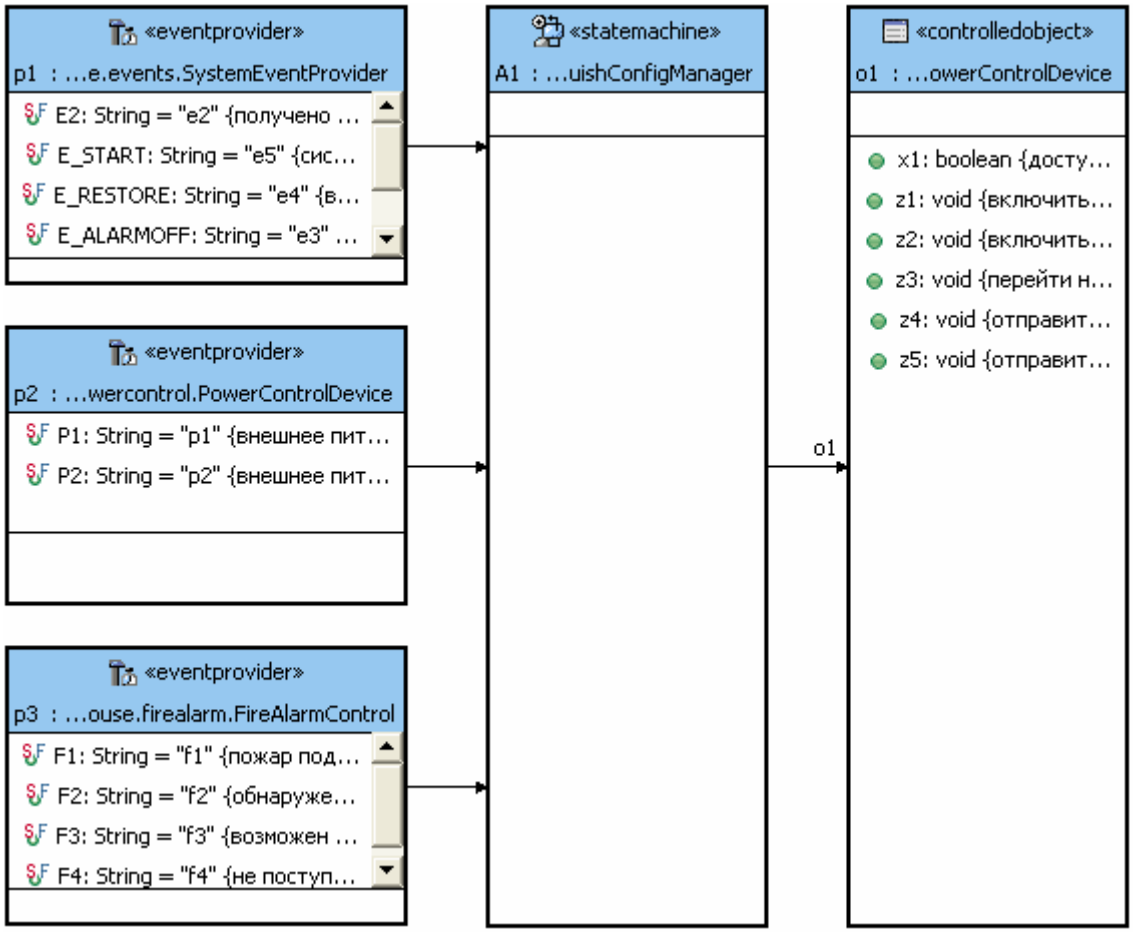

Рис. 10. Схема связей для автомата подсистемы контроля питания

В табл. 12 приведены входные события подсистемы контроля питания.

Таблица 12. Входные события подсистемы пожарной контроля питания

Источник событий PowerControlDevice	
Входные события	**Описание**
PowerControlDevice.*P1*	Внешнее питание недоступно
PowerControlDevice.*P2*	Внешнее питание восстановлено

Ниже приведены входные воздействия подсистемы контроля питания (табл. 13).

Таблица 13. Входные воздействия подсистемы пожарной безопасности

Объект управления PowerControlDevice	
Воздействие	**Описание**
PowerControlDevice.x1	Булевое входное воздействие наличие внешнего питания
PowerControlDevice.z1	Включить аварийное низковольтовое питание
PowerControlDevice.z2	Включить запасное питание
PowerControlDevice.z3	Перейти на внешнее питание
PowerControlDevice.z4	Проинформировать о доступном внешнем питании
PowerControlDevice.z5	Проинформировать о недоступном внешнем питании

Описание автомата

Как следует из диаграммы (рис. 11), автомат может находиться в одном из трех рабочих состояний, переход между которыми зависит от состояния внешнего питания и пожарной обстановки.

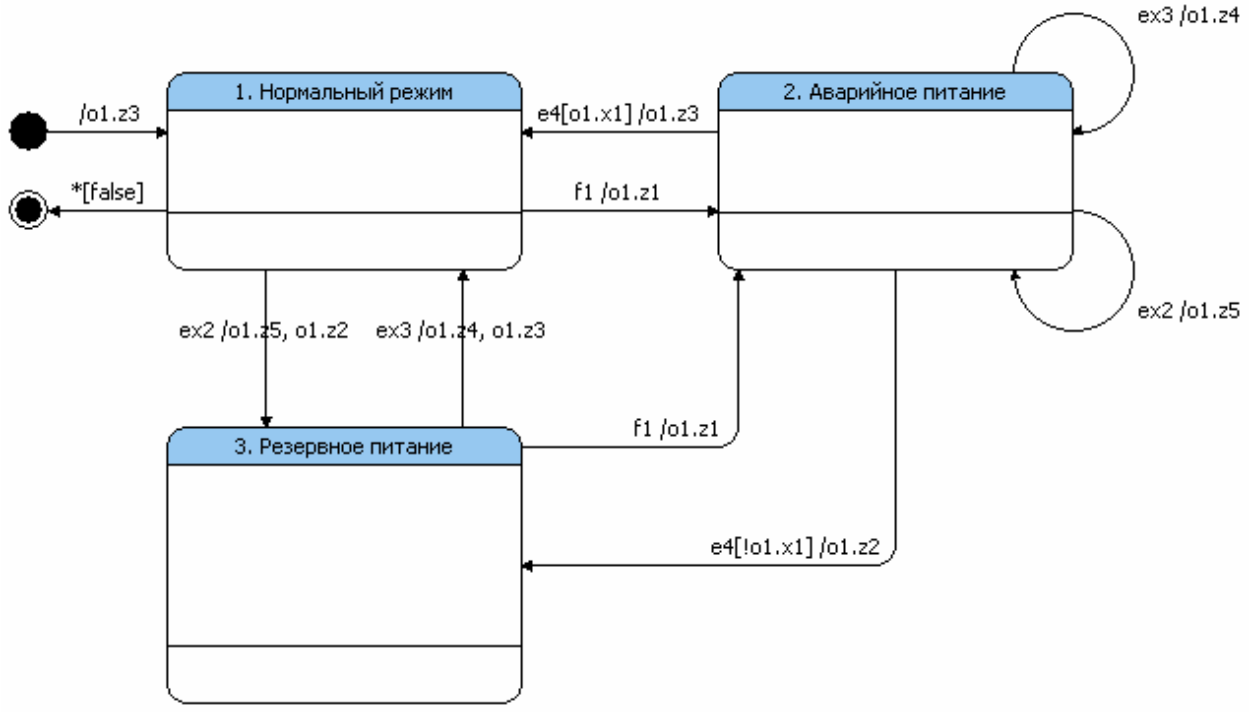

Рис. 11. Диаграмма переходов для автомата подсистемы контроля питания

Подсистема вентиляции

Описание

Данная подсистема (класс `VentilationControl`) отвечает за управления вентиляцией помещения. Основной задачей подсистемы является эффективный расход электроэнергии в зависимости от нахождения людей в помещении и корректная работа в случае пожарной тревоги. Активная вентиляция помещения в большинстве случае необходима только, если кто-то в нем находится. Как только помещение становится пустым, система переходит в энергосберегающий режим и находится в нем до тех пор, пока датчик присутствия не сообщит о наличие людей в помещении. В случае пожарной тревоги (табл. 14) система переходит в аварийный режим, прекращая подачу свежего воздуха в помещение (для предотвращения активного горения) и удаляя дыма из помещения, для предотвращения отравления людей при эвакуации.

Таблица 14. Входные сообщения подсистемы вентиляции

Входные сообщения	**Описания**
`ExistanceSensorDevice.`*`ES2`*	`В помещении кто-то есть`
`ExistanceSensorDevice.`*`ES3`*	`В помещении никого нет`
`FireAlarmControl.`*`F1`*	`В помещении подтвержден пожар (ПОЖАР-2)`
`SystemEventProvider.`*`e4`*	`Сообщение «Перезагрузить/Восстановить систему»`

Схема связей

Схема связей автомата подсистемы вентиляции приведена на рис. 12.

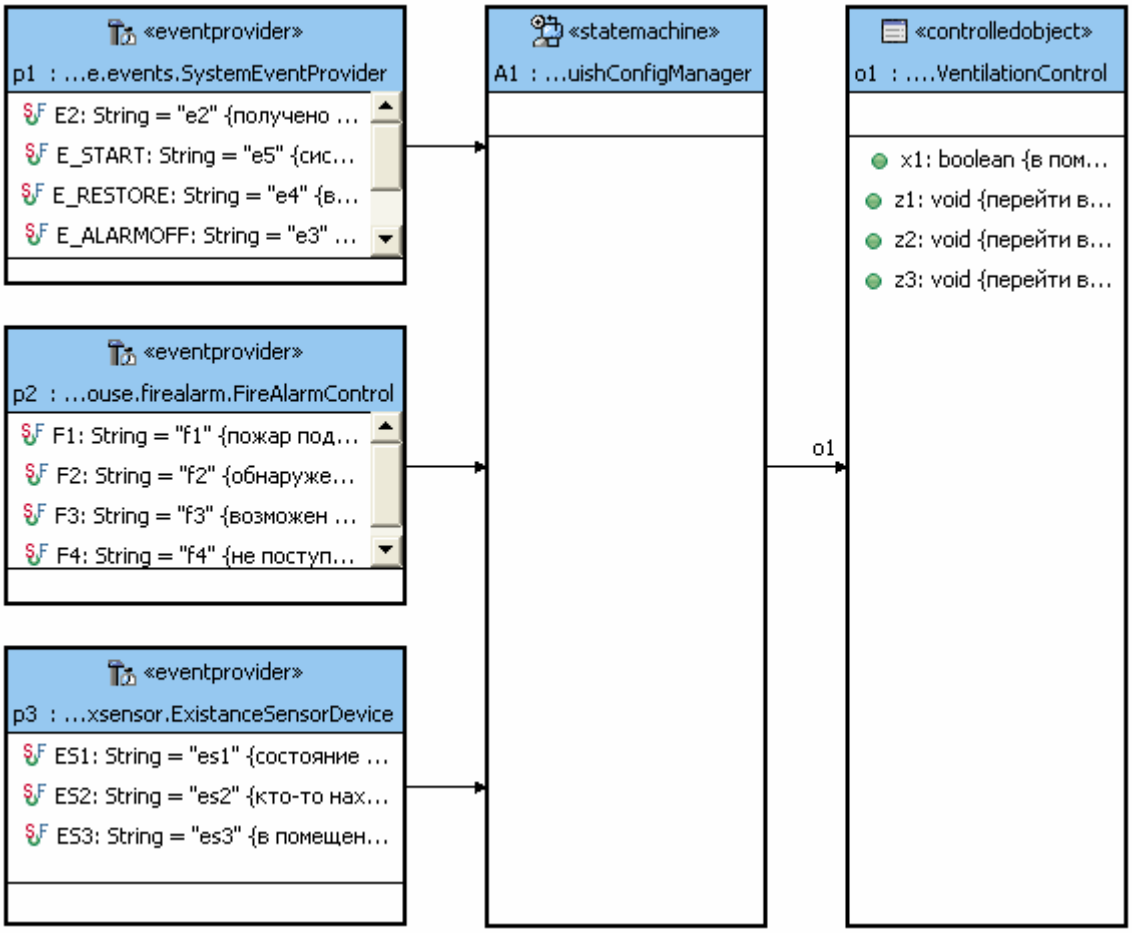

Рис. 12. Схема связей для автомата подсистемы вентиляции

Ниже приведены входные воздействия подсистемы вентиляции (табл. 15).

Таблица 15. Входные воздействия для подсистемы вентиляции

Объект управления VentilationControl	
Воздействие	**Описание**
VentilationControl.x1	В помещении есть люди
VentilationControl.z1	Перейти в нормальный режим вентилирования
VentilationControl.z2	Перейти в энергосберегающий режим
VentilationControl.z3	Перейти в аварийный пожарный режим

Описание автомата

Описание автомата подсистемы вентиляции приведено на рис. 13.

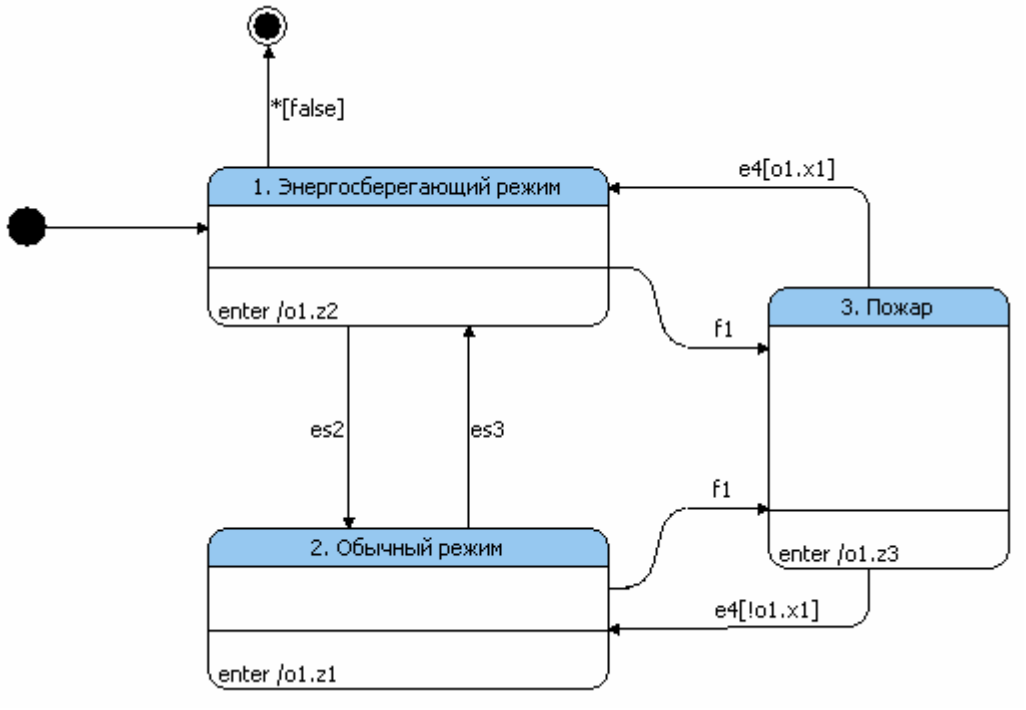

Рис. 13. Диаграмма переходов автомата подсистемы вентиляции

Модуль дверной

Описание

Дверной модуль (класс `DoorlockControl`) – это подсистема пропуска в помещение, состоящая из электронного замка и устройства доступа (клавиатура или считыватель магнитных карт). В предложенной модели пользователь может вводить код доступа, и, если код принят, войти в помещение. Если код был принят, но проход не был совершен в течение определенного времени, то дверь снова блокируется. В случае пожарной тревоги (табл. 16) замок автоматически разблокируется для обеспечения беспрепятственной эвакуации людей из помещения.

Таблица 16. Входные сообщения подсистемы дверного модуля

Входные сообщения	**Описания**
`SystemEventProvider.`*`e3`*	Сообщение «Снять с охраны»
`SystemEventProvider.`*`e4`*	Сообщение «Перезагрузить/Восстановить систему»
`FireAlarmControl.`*`F1`*	В помещении подтвержден пожар

Схема связей

Схема связей автомата подсистемы вентиляции приведена на рис. 14.

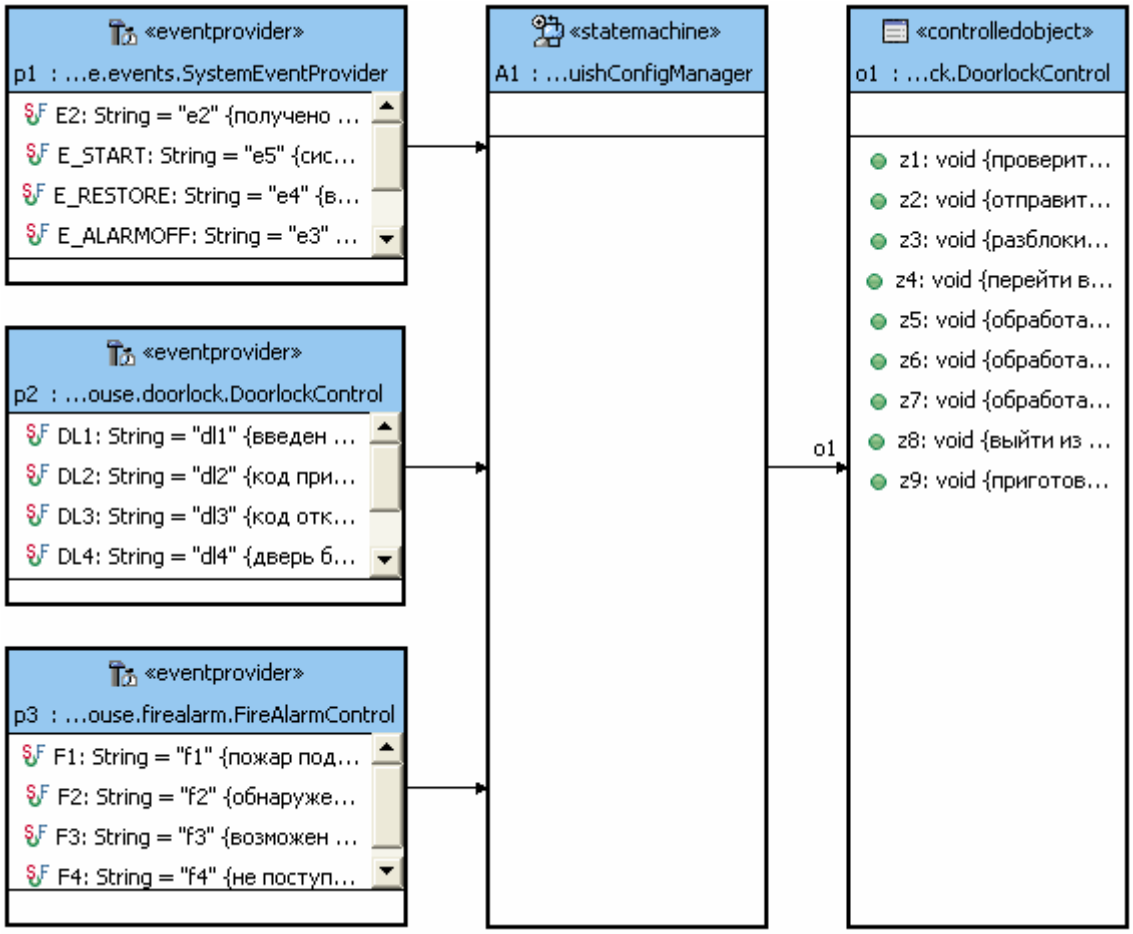

Рис. 14. Схема связей для автомата подсистемы дверного модуля

В табл. 17 приведены входные события подсистемы дверного модуля.

Таблица 17. Входные события подсистемы дверного модуля

Источник событий DoorlockControl	
Входные события	**Описание**
DoorlockControl.*dl1*	Введен код доступа
DoorlockControl.*dl2*	Код доступа принят
DoorlockControl.*dl3*	Код доступа отклонен
DoorlockControl.*dl4*	Дверь была закрыта
DoorlockControl.*dl5*	Дверь была открыта

Ниже приведены входные воздействия подсистемы дверного модуля (табл. 18).

Таблица 18. Входные воздействия подсистемы дверного модуля

Объект управления DoorlockControl	
Воздействие	**Описание**
DoorlockControl.z1	Проверить введенный код
DoorlockControl.z2	Сообщить охране о неверном коде
DoorlockControl.z3	Пожар: разблокировать все двери
DoorlockControl.z4	Перейти в состояние «ожидается проход»
DoorlockControl.z5	Обработать открытие двери
DoorlockControl.z6	Обработать истечение времени ожидания прохода
DoorlockControl.z7	Обработать закрытие двери
DoorlockControl.z8	Выйти из аварийного пожарного режима
DoorlockControl.z9	Приготовиться к ожиданию кода

Описание автомата

Как следует из схемы автомата (рис. 15), в основе работы модуля лежит основной цикл «ожидание кода – обработка кода – управление замком». В начале система находится в состоянии ожидания кода. Как только код был введен, он отправляется на проверку. Если код был отклонен, то отправляется уведомление охране, и система возвращается в исходное состояние. Если код был принят дверь разблокируется и система переходит в состояние ожидания прохода. После этого, если в течение заданного интервала проход не был начат (дверь не была открыта), то дверь снова блокируется и система возвращается в исходное состояние. Если же дверь была открыта, то после ее закрытия замок блокируется, и система снова будет готова к приему кода. В случае пожара необходимо разблокировать все двери для обеспечения беспрепятственной эвакуации людей из помещения, потому в состояние аварийной разблокировки система может перейти из любого состояния, как только придет сообщение подтверждения пожарной тревоги.

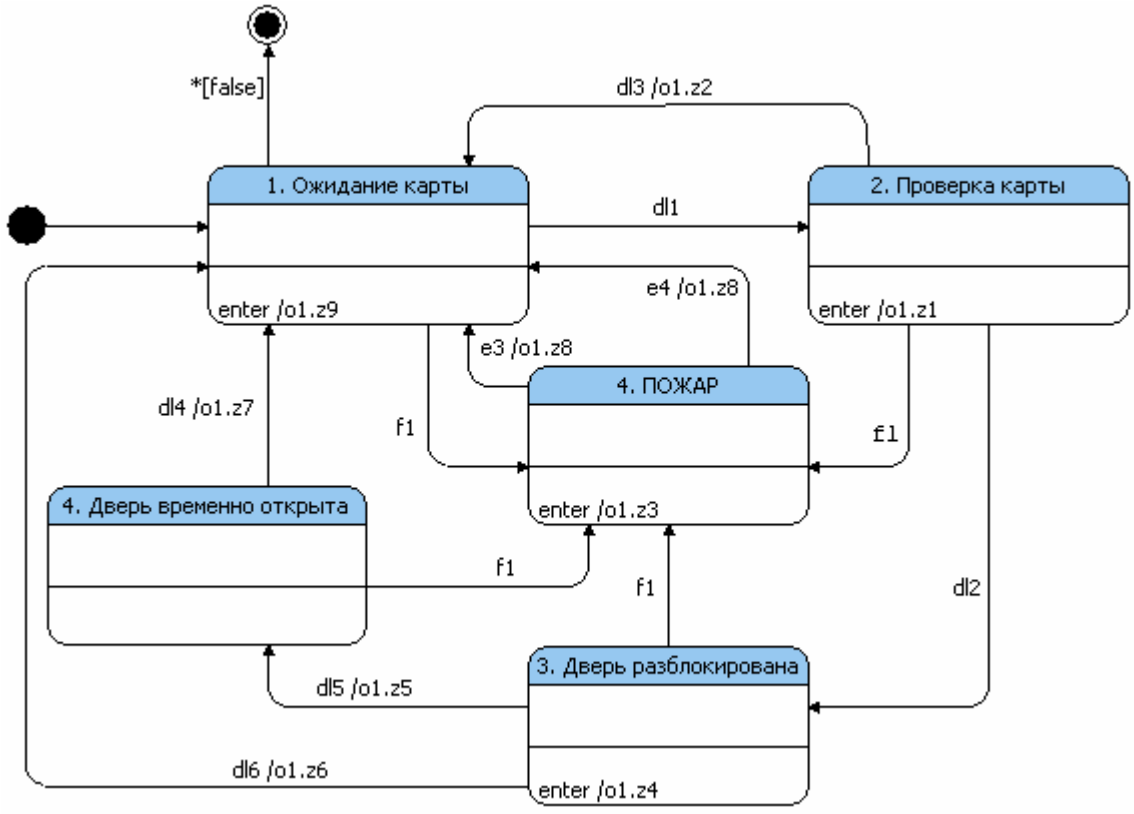

Рис. 15. Диаграмма переходов автомата подсистемы дверного модуля

Реализация

Реализация программы выполнена с помощью плагина *UniMod* к среде разработки *Java*-приложений *Eclipse*. Указанный плагин позволяет построить схему связей, а также автоматы устройств. После этого вручную реализуются объекты управления, источники событий и интерфейс программы на языке *Java*, а также классы, необходимые для обеспечения обмена информацией между устройствами. Существует два подхода для создания приложения: *интерпретационный* и *компилятивный*.

3.1. Интерпретационный подход

При реализации программы на основе интерпретационного подхода автоматы не преобразуются в *java*-код, а интерпретируются — сами запускаются, как исходный код. При этом классы, описывающие источники событий и объекты управления, компилируются в *java* байт-код, а интерпретатор по *XML*-описанию автомата имитирует его работу, выводя в консоль протокол работы. В статье [2] акцентируются внимание на том, что триединое использование автоматов (при спецификации, программировании и протоколировании) является важнейшей особенностью *Switch*-технологии. По мнению авторов статьи: «протоколы позволяют наблюдать за ходом выполнения программы и демонстрируют тот факт, что автоматы являются не «картинками», а реально действующими сущностями». Недостатками интерпретационного подхода являются низкое быстродействие и необходимость подключения многих библиотек.

3.2. Компиляционный подход

Альтернативой интерпретационному подходу является компиляционный подход. По *XML*-описанию автомата строится изоморфный ему *java*-код, который потом будет скомпилирован в байт-код. Это позволяет (совместно с кодом входных и выходных воздействий) запускать программу без использования интерпретатора *UniMod*, уменьшая, таким образом, количество необходимых библиотек, а, следовательно, объем используемой памяти. Компиляционный подход целесообразно применять для устройств с ограниченными ресурсами, таких как программируемые контроллеры или мобильные телефоны [3]. Структурные схемы интерпретационного и компиляционного подходов приведены в статье [4].

3.3. Запуск приложения

Для запуска приложения распакуйте архив с программой и выполните командный файл **run.cmd**.

На рис. 16 приведен пример работающей программы.

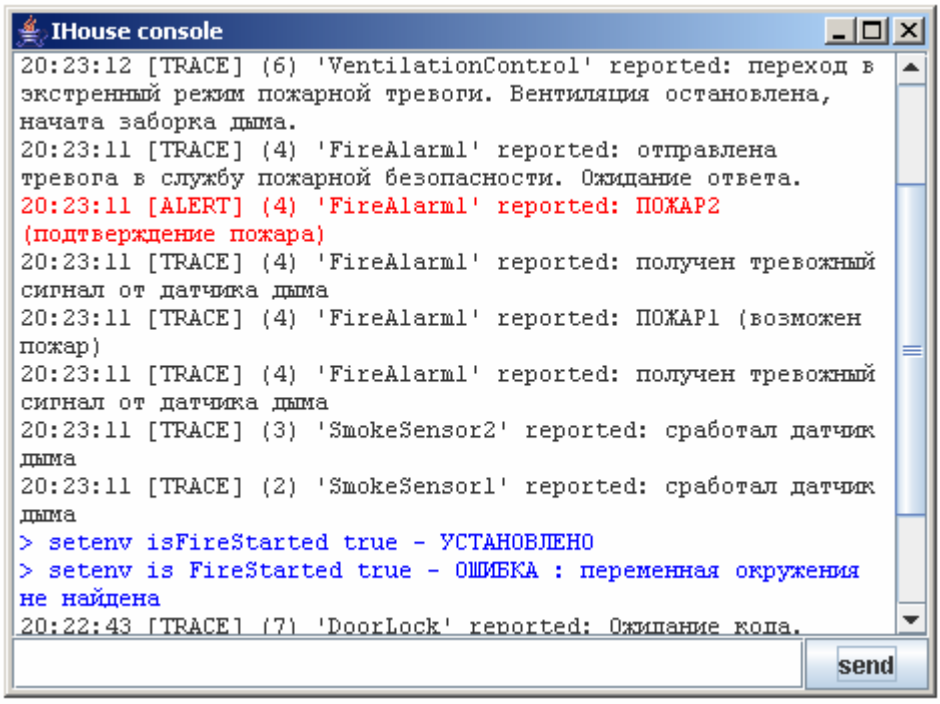

Рис. 16. Схема связей для автомата подсистемы дверного модуля

Заключение

Выбор автоматного подхода оказался полностью оправданным. Модели таких выбранных устройств, как подсистема пожарной безопасности и дверного модуля, достаточно сложны и содержат большое количество внутренних взаимосвязей и зависимостей. Использование инструментального средства *Unimod* позволило многократно сократить количество ошибок при проектировании и реализации внутренней логики устройств, что привело к уменьшению общего времени и трудоемкости разработки.

Источники

1. *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/
2. *Шалыто А. А., Туккель Н. И.* Программирование с явным выделением состояний //Мир ПК. 2001. № 8, 9. http://is.ifmo.ru/works/mirk/
3. *Шалыто А. А.* Технология автоматного программирования //Мир ПК. 2003. № 10. http://is.ifmo.ru/works/tech_aut_prog/
4. *Гуров В. С., Мазин М. А., Шалыто А. А. UniMod* — инструментальное средство для автоматного программирования //Научно-технический вестник. Вып. 30. Фундаментальные и прикладные исследования информационных систем и технологий. 2006. http://is.ifmo.ru/works/_instrsr.pdf

Приложения

Отрывок лога работы программы

Следующий отрывок лога системы соответствует началу пожара в помещении:

```
18:21:01 [TRACE] (1) 'Power Controller' reported: переход в режим внешнего
питания
18:21:01 [TRACE] (6) 'VentilationControl' reported: переход в режим
энергосбережения
18:21:01 [TRACE] (7) 'DoorLock' reported: Ожидание кода.
> setenv isFireStarted true - УСТАНОВЛЕНО
18:21:11 [TRACE] (2) 'SmokeSensor1' reported: сработал датчик дыма
18:21:11 [TRACE] (3) 'SmokeSensor2' reported: сработал датчик дыма
18:21:11 [TRACE] (4) 'FireAlarm1' reported: получен тревожный сигнал от
датчика дыма
18:21:11 [TRACE] (4) 'FireAlarm1' reported: ПОЖАР1 (возможен пожар)
18:21:11 [TRACE] (4) 'FireAlarm1' reported: получен тревожный сигнал от
датчика дыма
18:21:11 [ALERT] (4) 'FireAlarm1' reported: ПОЖАР2 (подтверждение пожара)
18:21:11 [TRACE] (4) 'FireAlarm1' reported: отправлена тревога в службу
пожарной безопасности. Ожидание ответа.
18:21:12 [TRACE] (6) 'VentilationControl' reported: переход в экстренный
режим пожарной тревоги. Вентиляция остановлена, начата заборка дыма.
18:21:12 [TRACE] (7) 'DoorLock' reported: Пожар подтвержден. Дверь
разблокирована.
18:21:12 [TRACE] (1) 'Power Controller' reported: переход в режим АВАРИЙНОГО
низковольтового питания
```

Подробный лог системы по состояниям

```
18:21:00,968 INFO  [Run] Start event [e5] processing. In state [/A1:Top]
18:21:00,968 INFO  [Run] Transition to go found [s1#1. Нормальный
режим##true]
18:21:00,968 INFO  [Run] Start output action [o1.z3] execution
18:21:00,984 INFO  [Run] Start event [e5] processing. In state [/A1:Top]
18:21:00,984 INFO  [Run] Transition to go found [s1#1. Проверка
состояния##true]
18:21:00,984 DEBUG [Run] Try transition [1. Проверка состояния#s3#*#false]
18:21:00,984 ERROR [Run] No transition defined for event [*] from state
18:21:00,984 INFO  [Run] Finish event [e5] processing. In state [/A1:1.
Проверка состояния]
18:21:01,000 INFO  [Run] Start event [e5] processing. In state [/A1:Top]
18:21:01,000 INFO  [Run] Transition to go found [s1#Проверка
присутствия##true]
18:21:01,000 DEBUG [Run] Try transition [Проверка присутствия#s3#*#false]
18:21:01,000 ERROR [Run] No transition defined for event [*] from state
18:21:01,000 INFO  [Run] Finish event [e5] processing. In state [/A1:Проверка
присутствия]
18:21:01,015 INFO  [Run] Finish output action [o1.z3] execution
18:21:01,015 DEBUG [Run] Try transition [1. Нормальный режим#s3#*#false]
18:21:01,015 ERROR [Run] No transition defined for event [*] from state
18:21:01,015 INFO  [Run] Finish event [e5] processing. In state [/A1:1.
Нормальный режим]
18:21:01,031 INFO  [Run] Start event [e5] processing. In state [/A1:Top]
18:21:01,031 INFO  [Run] Transition to go found [s1#1. Энергосберегающий
режим##true]
18:21:01,031 INFO  [Run] Start on-enter action [o1.z2] execution
18:21:01,031 INFO  [Run] Finish on-enter action [o1.z2] execution
```

```
18:21:01,031 DEBUG [Run] Try transition [1. Энергосберегающий
режим#s3#*#false]
18:21:01,031 ERROR [Run] No transition defined for event [*] from state
18:21:01,031 INFO  [Run] Finish event [e5] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:01,078 INFO  [Run] Start event [e5] processing. In state [/A1:Top]
18:21:01,078 INFO  [Run] Transition to go found [s1#1. Ожидание карты##true]
18:21:01,078 INFO  [Run] Start on-enter action [o1.z9] execution
18:21:01,078 INFO  [Run] Finish on-enter action [o1.z9] execution
18:21:01,078 DEBUG [Run] Try transition [1. Ожидание карты#s3#*#false]
18:21:01,078 ERROR [Run] No transition defined for event [*] from state
18:21:01,078 INFO  [Run] Finish event [e5] processing. In state [/A1:1.
Ожидание карты]
18:21:11,484 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Проверка состояния]
18:21:11,484 DEBUG [Run] Try transition [1. Проверка состояния#2. ПОЖАР
1#f2#true]
18:21:11,484 INFO  [Run] Transition to go found [1. Проверка состояния#2.
ПОЖАР 1#f2#true]
18:21:11,484 INFO  [Run] Start output action [o1.z4] execution
18:21:11,484 INFO  [Run] Finish output action [o1.z4] execution
18:21:11,484 INFO  [Run] Start on-enter action [o1.z1] execution
18:21:11,500 INFO  [Run] Finish on-enter action [o1.z1] execution
18:21:11,500 INFO  [Run] Finish event [f2] processing. In state [/A1:2. ПОЖАР
1]
18:21:11,500 INFO  [Run] Start event [f2] processing. In state [/A1:Проверка
присутствия]
18:21:11,500 DEBUG [Run] Try transition [Проверка присутствия#s3#*#false]
18:21:11,500 ERROR [Run] No transition defined for event [*] from state
18:21:11,500 INFO  [Run] Finish event [f2] processing. In state [/A1:Проверка
присутствия]
18:21:11,531 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:11,531 DEBUG [Run] Try transition [1. Энергосберегающий
режим#s3#*#false]
18:21:11,531 ERROR [Run] No transition defined for event [*] from state
18:21:11,531 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:11,578 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Ожидание карты]
18:21:11,578 DEBUG [Run] Try transition [1. Ожидание карты#s3#*#false]
18:21:11,578 ERROR [Run] No transition defined for event [*] from state
18:21:11,593 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Ожидание карты]
18:21:11,953 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Нормальный режим]
18:21:11,953 DEBUG [Run] Try transition [1. Нормальный режим#s3#*#false]
18:21:11,953 ERROR [Run] No transition defined for event [*] from state
18:21:11,953 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Нормальный режим]
18:21:11,984 INFO  [Run] Start event [f2] processing. In state [/A1:2. ПОЖАР
1]
18:21:11,984 DEBUG [Run] Try transition [2. ПОЖАР 1#3. ПОЖАР 2#f2#true]
18:21:11,984 INFO  [Run] Transition to go found [2. ПОЖАР 1#3. ПОЖАР
2#f2#true]
18:21:11,984 INFO  [Run] Start output action [o1.z4] execution
18:21:11,984 INFO  [Run] Finish output action [o1.z4] execution
18:21:11,984 INFO  [Run] Start output action [o1.z2] execution
18:21:11,984 INFO  [Run] Finish output action [o1.z2] execution
18:21:11,984 INFO  [Run] Start on-enter action [o1.z5] execution
18:21:11,984 INFO  [Run] Finish on-enter action [o1.z5] execution
18:21:11,984 INFO  [Run] Finish event [f2] processing. In state [/A1:3. ПОЖАР
2]
```

```
18:21:12,000 INFO  [Run] Start event [f2] processing. In state [/A1:Проверка
присутствия]
18:21:12,000 DEBUG [Run] Try transition [Проверка присутствия#s3#*#false]
18:21:12,000 ERROR [Run] No transition defined for event [*] from state
18:21:12,000 INFO  [Run] Finish event [f2] processing. In state [/A1:Проверка
присутствия]
18:21:12,031 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:12,031 DEBUG [Run] Try transition [1. Энергосберегающий
режим#s3#*#false]
18:21:12,031 ERROR [Run] No transition defined for event [*] from state
18:21:12,031 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:12,078 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Ожидание карты]
18:21:12,078 DEBUG [Run] Try transition [1. Ожидание карты#s3#*#false]
18:21:12,078 ERROR [Run] No transition defined for event [*] from state
18:21:12,078 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Ожидание карты]
18:21:12,453 INFO  [Run] Start event [f2] processing. In state [/A1:1.
Нормальный режим]
18:21:12,453 DEBUG [Run] Try transition [1. Нормальный режим#s3#*#false]
18:21:12,453 ERROR [Run] No transition defined for event [*] from state
18:21:12,453 INFO  [Run] Finish event [f2] processing. In state [/A1:1.
Нормальный режим]
18:21:12,484 INFO  [Run] Start event [f1] processing. In state [/A1:3. ПОЖАР
2]
18:21:12,484 INFO  [Run] Finish event [f1] processing. In state [/A1:3. ПОЖАР
2]
18:21:12,500 INFO  [Run] Start event [f1] processing. In state [/A1:Проверка
присутствия]
18:21:12,500 DEBUG [Run] Try transition [Проверка присутствия#s3#*#false]
18:21:12,500 ERROR [Run] No transition defined for event [*] from state
18:21:12,500 INFO  [Run] Finish event [f1] processing. In state [/A1:Проверка
присутствия]
18:21:12,531 INFO  [Run] Start event [f1] processing. In state [/A1:1.
Энергосберегающий режим]
18:21:12,531 DEBUG [Run] Try transition [1. Энергосберегающий режим#3.
Пожар#f1#true]
18:21:12,531 INFO  [Run] Transition to go found [1. Энергосберегающий
режим#3. Пожар#f1#true]
18:21:12,531 INFO  [Run] Start on-enter action [o1.z3] execution
18:21:12,531 INFO  [Run] Finish on-enter action [o1.z3] execution
18:21:12,531 INFO  [Run] Finish event [f1] processing. In state [/A1:3.
Пожар]
18:21:12,578 INFO  [Run] Start event [f1] processing. In state [/A1:1.
Ожидание карты]
18:21:12,578 DEBUG [Run] Try transition [1. Ожидание карты#4. ПОЖАР#f1#true]
18:21:12,578 INFO  [Run] Transition to go found [1. Ожидание карты#4.
ПОЖАР#f1#true]
18:21:12,578 INFO  [Run] Start on-enter action [o1.z3] execution
18:21:12,593 INFO  [Run] Finish on-enter action [o1.z3] execution
18:21:12,593 INFO  [Run] Finish event [f1] processing. In state [/A1:4.
ПОЖАР]
18:21:12,953 INFO  [Run] Start event [f1] processing. In state [/A1:1.
Нормальный режим]
18:21:12,953 DEBUG [Run] Try transition [1. Нормальный режим#2. Аварийное
питание#f1#true]
18:21:12,953 INFO  [Run] Transition to go found [1. Нормальный режим#2.
Аварийное питание#f1#true]
18:21:12,953 INFO  [Run] Start output action [o1.z1] execution
18:21:12,953 INFO  [Run] Finish output action [o1.z1] execution
18:21:12,953 INFO  [Run] Finish event [f1] processing. In state [/A1:2.
Аварийное питание]
```