Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет «Информационные технологии и программирование»
Кафедра «Компьютерные технологии»

И. И. Колыхматов, О. О. Рыбак, А. А. Шалыто

Моделирование устройства для продажи газированной воды на инструментальном средстве UniMod

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

Санкт-Петербург
2006

Оглавление

Введение

> Если бы строители строили здания также,
> как программисты пишут программы, первый
> залетевший дятел разрушил бы цивилизацию.
> *Второй закон Вейнберга*

SWITCH-технология [1, 2] акцентирует внимание разработчиков на проектировании программ. При этом программы предлагается строить так же, как автоматизируются объекты управления в промышленности. При таком подходе процесс реализации нового проекта начинается не с запуска среды разработки и написания кода `public static void main(String[] args)`, а с изучения предметной области, выделения объектов управления и поставщиков событий. Затем проектируется один или несколько автоматов, для каждого из которых создается схема связей и граф переходов [3]. Такой подход позволяет на ранних стадиях проектирования выявить и устранить множество возникающих неясностей в постановке задачи, а также предусмотреть весьма не очевидные детали поведения системы. На сайте `http://is.ifmo.ru` размещено большое количество материалов, посвященных автоматному программированию, а также проектов, построенных по *SWITCH*-технологии.

Даже хорошие идеи бесполезны, если вы ни с кем не поделитесь ими. Поэтому «Движение за открытую проектную документацию» [4] провозглашает публикацию проектной документации, по которой любой человек, знакомый с основами *SWITCH*-технологии, сможет понять и модифицировать программу.

Проект *UniMod* (`http://unimod.sourceforge.net`) с открытым исходным кодом содержит набор инструментов, позволяющих визуально проектировать и реализовывать программы. При этом первоначально строится схема связей, состоящая, как и в *SWITCH*-технологии, из источников событий, системы управления и объектов управления, в которых реализованы вызываемые из автоматов выходные воздействия и опрашиваемые автоматами входные переменные. В рассматриваемом проекте система управления — это система взаимосвязанных автоматов.

Визуальное проектирование в рамках проекта *UniMod* выполняется в бесплатной и многофункциональной среде разработки *Eclipse* (`http://eclipse.org`) для языка программирования *Java*. Для этого используется встраиваемый модуль, разработанный в компании *eVelopers Corporation* (`http://www.evelopers.com`). Классы, реализующие функциональность поставщиков событий и объектов управления, пишутся на языке *Java*. Существуют две возможности реализации программы.

В первом случае по *UML*-диаграммам создается *XML*-описание проекта, которое затем поступает на вход интерпретатора (еще одного компонента *UniMod*), а во втором — по диаграммам автоматически генерируется код готовой программы на языке *Java*. Это позволяет утверждать, что инструментальное средство *UniMod* позволяет реализовывать программы в целом, а не только автоматы.

Таким образом, жизненный цикл разработанного приложения при использовании инструментального средства *UniMod* выглядит следующим образом. Сна-

чала поставщики событий формируют события, которые обрабатываются одним из автоматов в соответствии с графом переходов. Автомат при поступлении события может проверить различные логические условия и в результате выбрать необходимый переход в новое состояние. С переходом может быть ассоциирован некий набор выходных воздействий на объекты управления. Эти воздействия выполняются при выборе данного перехода. В общем случае действия могут выполняться не только на переходах. При поступлении определенного события автомат может перейти в конечное состояние, завершив тем самым работу приложения.

В данной работе представлен проект моделирования работы устройства для продажи газированной воды с использованием инструментария *UniMod*.

1. Описание основных узлов устройства

Устройство для продажи газированных напитков (вендинговый автомат, торговый автомат) не требует вмешательства человека, кроме как для изъятия выручки и закладки товаров.

Опишем составляющие торгового устройства и предполагаемую функциональность, которая будет в дальнейшем реализована с использованием пакета *UniMod*. Основные узлы устройства для продажи газированной воды:

- дисплей;
- управляющая панель с кнопками для выбора напитков;
- монетный механизм;
- устройство для приготовления напитка, включающее:
  - дозатор;
  - смеситель;
  - сатуратор (аппарат для насыщения жидкости диоксидом углерода);
  - транспортирующее устройство;
- механизм выдачи напитка;
- холодильная установка в случае торговли прохладительными напитками.

Опущенные покупателем монеты (платежные средства) поступают в монетный механизм, где определяется их достоинство и проверяется их подлинность с помощью контрольных устройств. Монетный механизм принимает монеты номиналом один, два и пять рублей. Непригодные монеты отбраковываются. На дисплее указывается принятый аванс и стоимость напитков.

Устройство должно отключаться при отсутствии двуокиси углерода, стаканчиков одноразового использования, при открывании двери. При отсутствии одного из напитков покупатель извещается об этом потухшим встроенным индикатором кнопки выбора напитка.

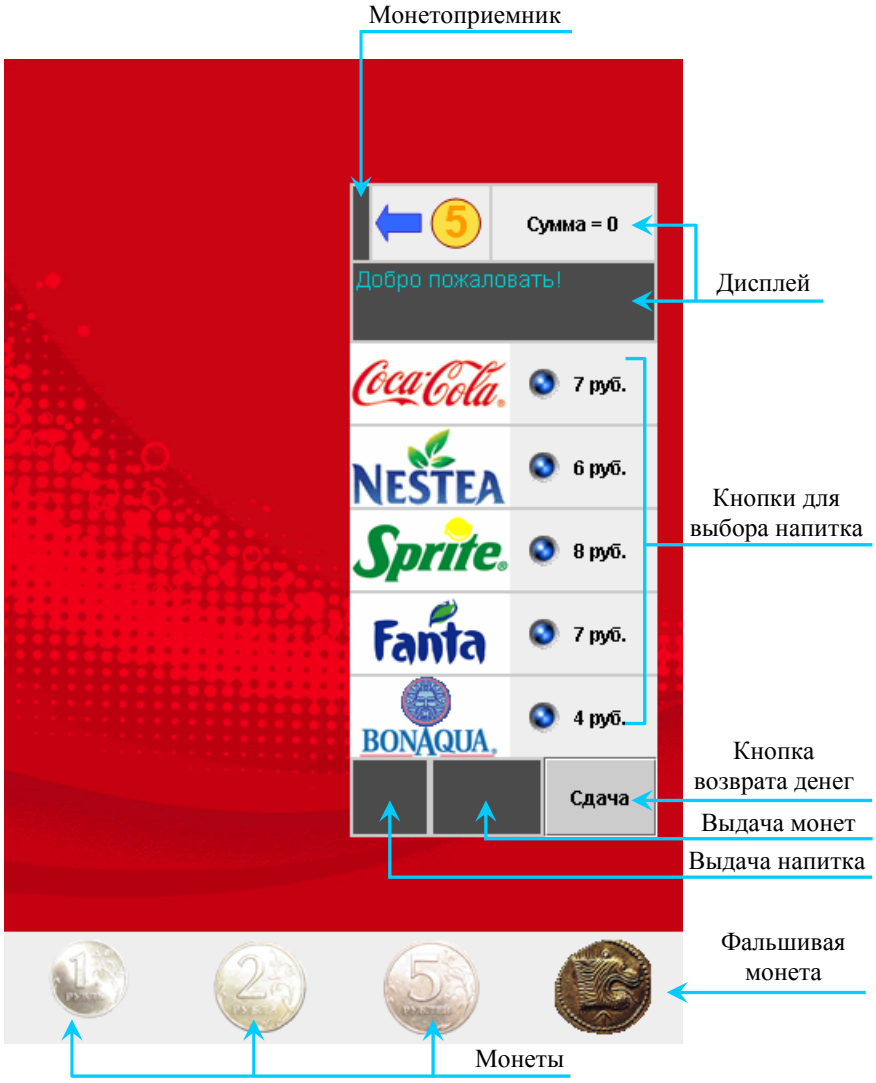

Рис. 1. Внешний вид приложения

Интерфейс приложения (рис. 1) представляет собой модель устройства для продажи газированной воды. Снизу располагаются изображения монет различного номинала, одна из которых фальшивая. Чтобы опустить монету, необходимо щелкнуть по ней левой кнопкой мыши и перенести ее в монетоприемник. Для выбора напитка имеются кнопки со встроенными индикаторами наличия напитка. Сообщения устройства показываются на дисплее.

2. Описание поведения

Приведем основные сценарии работы клиента с торговым устройством.

1. *Клиент опустил монету.* Монетный механизм выполняет проверку поступившей монеты на подлинность. В случае обнаружения фальшивой монеты соответствующее сообщение выводится на дисплей, монета выдается клиенту. Если монета подлинная, но величина внесенного аванса уже достаточна

для приобретения любого напитка, то монета также выдается клиенту. Наконец, при отсутствии препятствий к зачислению поступивших денежных средств, номинал поступившей монеты добавляется к сумме, которая отображается на дисплее.

2. *Клиент выбрал напиток.* Если сумма нулевая, то ничего не происходит. Если же сумма ненулевая и выбранный напиток закончился, то клиент уведомляется об этом сообщением на дисплее. Если сумма недостаточна для приобретения выбранного напитка, то сообщение о нехватке средств выводится на дисплей. Если денег достаточно и напиток имеется в наличии, то выдается стаканчик, затем порция напитка насыщается углекислым газом и наливается в стаканчик. Если после выдачи напитка этот тип напитка закончился, то гасится встроенный индикатор кнопки выбора напитка. На дисплей выводится благодарность.

3. *Клиент решил забрать деньги.* Если сумма ненулевая, то деньги выдаются клиенту полностью, а значение суммы обнуляется. Торговое устройство готово к обслуживанию следующего клиента.

4. *Закончилась двуокись углерода или стаканчики одноразового использования.* В этом случае устройство выключается, предварительно вернув деньги клиенту и известив о причине выключения.

3. Реализация с использованием одного автомата

3.1. Схема связей автомата

Управление устройством для продажи газированной воды выполняет автомат A1. В отличие от *SWITCH*-технологии [1] схема связей при использовании пакета *UniMod* является не картинкой, а диаграммой классов *UML*, изображенной нетрадиционным путем (она ориентирована не сверху вниз, а слева направо).

На схеме связей автомата A1 (рис. 2) каждому событию соответствует краткое обозначение в нотации e#, где вместо символа # стоит число. Все входные переменные (x1, x2,. . .) и выходные воздействия (z0, z1,. . .) объектов управления являются методами, которые реализуются вручную на языке *Java*.

Кодирование названий входных переменных, выходных воздействий и событий краткими обозначениями в указанной нотации обеспечивает возможность компактного построения автоматов при сложном поведении системы. Обратите внимание, что рядом с краткими обозначениями на схеме связей приводятся комментарии, поясняющие назначение той или иной входной переменной, функциональность каждого выходного воздействия. Всплывающие подсказки и удобный редактор условий переходов обеспечивают простоту построения графов переходов автоматов. Подробнее с достоинствами и недостатками такого способа задания имен можно ознакомиться в работе [5], в которой дана оценка перспективности визуального подхода к проектированию, сформулированы требования,

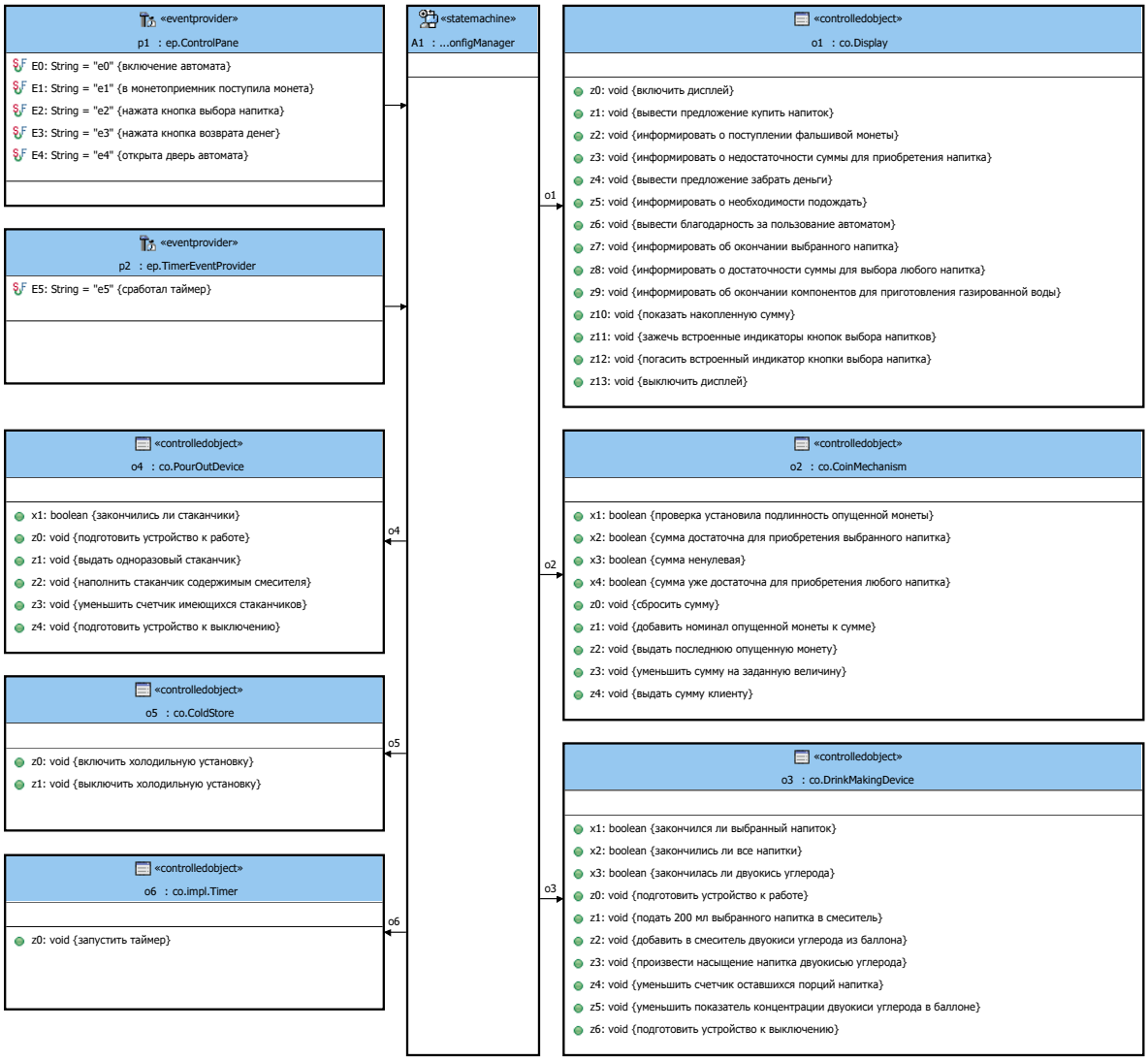

Рис. 2. Схема связей автомата A1

предъявляемые к инструментальным средствам для визуального конструирования программ, и выполнен анализ возможностей существующих инструментов.

Каждый компонент системы изображен в виде прямоугольника, в верхней части которого приведено краткое обозначение, а также название *Java*-класса, реализующего функциональность данного объекта. Отметим, что на схеме (рис. 2) не указан *Java*-класс для автомата A1, поскольку для него автоматически генерируется *XML*-описание, которое в дальнейшем интерпретируется.

3.2. События и передаваемые параметры

Рассмотрим сначала механизм создания и обработки событий. Каждому источнику событий ставится в соответствие *Java*-класс, который содержит метод

`init()`, вызываемый при запуске приложения и инициализации данного источника событий. В рассматриваемом случае в методе `init()` формируется графическая модель устройства для продажи газированной воды (рис. 1). Далее пользователь как-то взаимодействует с интерфейсом приложения, например, переносит изображение монеты в монетоприемник на основе механизма *drag-and-drop*. С помощью вызова методов из библиотеки времени выполнения *UniMod* происходит добавление нового события в очередь событий. Кроме того, в контексте события могут быть переданы произвольные данные в качестве параметров. Так, например, генерируется событие `e1` («В монетоприемник поступила монета»), вместе с которым как параметр передается номинал опущенной монеты. Впоследствии событие будет обработано в соответствии с графом переходов автомата.

Для реализации входных и выходных воздействий иногда необходима дополнительная информация. Например, когда пользователь нажимает кнопку выбора напитка, генерируется событие `e2`, в контексте которого передается наименование выбранного напитка. Затем для определения требуемого перехода при поступлении события `e2` будут вычисляться логические условия. При вычислении входной переменной `x1` устройства для приготовления напитка (в методе `o3.x1()`) будет использовано значение параметра из контекста события `e2`.

Приведем описание данных, передаваемых как параметры при создании событий (табл. 1).

Таблица 1. События и соответствующие им параметры

Событие	Описание события	Параметр	Значение параметра
`e1`	В монетоприемник поступила монета	`COIN_TYPE`	Номинал монеты
`e2`	Нажата кнопка выбора напитка	`REQUESTED_DRINK`	Наименование напитка, который выбрал клиент

3.3. Граф переходов автомата

Граф переходов (рис. 3) задает логику работы автомата `A1`.

Начальное состояние обозначено черным кругом, а конечное — двойным кругом с закрашенной серединой. Состояния обозначены прямоугольниками со скругленными краями, а переходы — стрелками, рядом с которыми написано условие перехода. Общая форма записи перехода:

`e#` [логическое выражение] `/` список выходных воздействий,

где вместо символа `#` стоит цифра.

Логическое выражение может состоять из:

- входных переменных в нотации `o#.x#` (здесь, как и выше, символ `#` обозначает любую цифру);

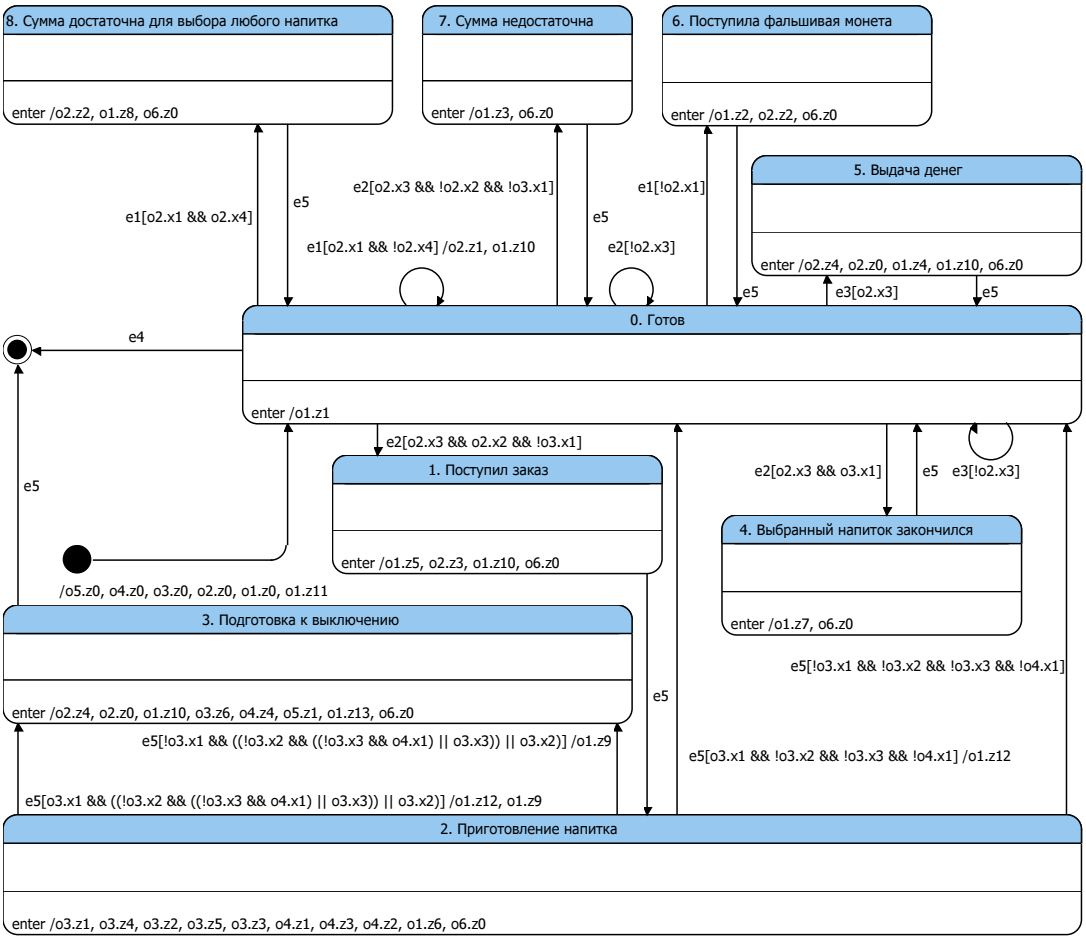

Рис. 3. Граф переходов автомата A1

- логических операций && (и), || (или), ! (не);
- операций сравнения > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), != (не равно);
- скобок ( ).

Допустим, что автомат находится в каком-либо состоянии. Если поступило событие, то для всех переходов по этому событию проверяется логическое условие (если оно присутствует). Если найден переход, для которого логическое условие истинно, то сначала выполняются все выходные воздействия, указанные на переходе после символа /, а затем — выходные воздействия, предусмотренные для исполнения в момент входа в состояние (список таких воздействий приводится в нижней части прямоугольника-состояния после слов «enter /»).

3.4. Полнота и непротиворечивость логических условий

Исследуя граф переходов и схему связей (рис. 2), можно понять, что происходит в ответ на то или иное событие. При проектировании графа переходов

с помощью инструментального средства *UniMod* выполняется автоматическая проверка условий переходов на полноту и непротиворечивость. Прокомментируем в этой связи один неочевидный момент. Из состояния «2. Приготовление напитка» (рис. 3) есть четыре перехода по событию e5 в состояния «0. Готов» и «3. Подготовка к выключению»:

- e5[o3.x1 && ((!o3.x2 && ((!o3.x3 && o4.x1) || o3.x3)) || o3.x2)] /o1.z12, o1.z9;
- e5[!o3.x1 && ((!o3.x2 && ((!o3.x3 && o4.x1) || o3.x3)) || o3.x2)] /o1.z9;
- e5[o3.x1 && !o3.x2 && !o3.x3 && !o4.x1] /o1.z12;
- e5[!o3.x1 && !o3.x2 && !o3.x3 && !o4.x1].

Объясним, почему система этих логических условий полна и не содержит противоречий. Введем для краткости записи обозначения для входных переменных, встречающихся в каждом из четырех условий: a = o3.x1, b = o3.x2, c = o3.x3 и d = o4.x1. Схема связей подсказывает следующее:

- переменная a истинна тогда, когда закончился выбранный напиток;
- переменная b истинна в случае окончания всех напитков;
- переменная c истинна, если закончилась двуокись углерода;
- переменная d истинна в том случае, если закончились стаканчики.

Разберемся, что следует предпринять, когда выполнен какой-нибудь набор значений переменных $\{a, b, c, d\}$. Для этого обратимся к таблице (табл. 2), в ко-

Таблица 2. Комбинации входных переменных

a	b	c	d	Действие
0	0	0	0	(I) Перейти в состояние готовности
0	0	0	1	
0	0	1	0	
0	0	1	1	(II) Выключить устройство,
0	1	0	0	поскольку компоненты для приготовления
0	1	0	1	газированной воды закончились
0	1	1	0	
0	1	1	1	
1	0	0	0	(III) Погасить индикатор наличия напитка, состояние готовности
1	0	0	1	
1	0	1	0	(IV) Погасить индикатор наличия напитка,
1	0	1	1	потом выключить устройство,
1	1	0	0	поскольку компоненты для приготовления
1	1	0	1	газированной воды закончились
1	1	1	0	
1	1	1	1	

торой представлены все комбинации этих переменных.

В зависимости от того, какие компоненты закончились, можно предпринять четыре различных варианта действий. Рассмотрим группу из семи комбинаций (табл. 2), отмеченную римской цифрой II. Логическое условие, истинное для любой комбинации из набора {0001, 0010, 0011, 0100, 0101, 0110, 0111}, с использованием конъюнкции ($\wedge$) и дизъюнкции ($\vee$) запишем в виде:

$$(\overline{a} \wedge \overline{b} \wedge \overline{c} \wedge d) \vee (\overline{a} \wedge \overline{b} \wedge c \wedge \overline{d}) \vee (\overline{a} \wedge \overline{b} \wedge c \wedge d) \vee (\overline{a} \wedge b \wedge \overline{c} \wedge \overline{d}) \vee (\overline{a} \wedge b \wedge \overline{c} \wedge d) \vee$$
$$\vee (\overline{a} \wedge b \wedge c \wedge \overline{d}) \vee (\overline{a} \wedge b \wedge c \wedge d).$$

Преобразуем это условие: оставим сначала первую скобку без изменений, а оставшиеся шесть скобок сгруппируем попарно и выполним упрощение, затем применим логические правила:

$$(\overline{a} \wedge \overline{b} \wedge \overline{c} \wedge d) \vee (\overline{a} \wedge \overline{b} \wedge c \wedge \overline{d}) \vee (\overline{a} \wedge \overline{b} \wedge c \wedge d) \vee (\overline{a} \wedge b \wedge \overline{c} \wedge \overline{d}) \vee (\overline{a} \wedge b \wedge \overline{c} \wedge d) \vee$$
$$\vee (\overline{a} \wedge b \wedge c \wedge \overline{d}) \vee (\overline{a} \wedge b \wedge c \wedge d) = (\overline{a} \wedge \overline{b} \wedge \overline{c} \wedge d) \vee (\overline{a} \wedge \overline{b} \wedge c) \vee$$
$$\vee (\overline{a} \wedge b \wedge \overline{c}) \vee (\overline{a} \wedge b \wedge c) = (\overline{a} \wedge \overline{b} \wedge \overline{c} \wedge d) \vee (\overline{a} \wedge \overline{b} \wedge c) \vee (\overline{a} \wedge b) =$$
$$= \overline{a} \wedge ((\overline{b} \wedge \overline{c} \wedge d) \vee (\overline{b} \wedge c) \vee b) = \overline{a} \wedge ((\overline{b} \wedge ((\overline{c} \wedge d) \vee c)) \vee b).$$

Такое же логическое условие в нотации *UML* записано на одном из переходов:

```
e5[!o3.x1 && ((!o3.x2 && ((!o3.x3 && o4.x1) || o3.x3)) || o3.x2)] /o1.z9.
```

Покажем теперь, что один из четырех переходов всегда будет выбран при любом наборе входных переменных. Для этого воспользуемся наглядным аппаратом разрешающих диаграмм (рис. 4).

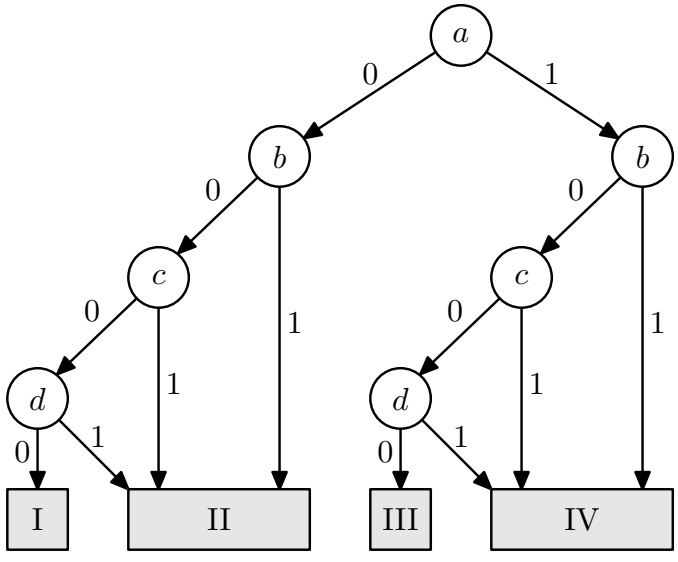

Рис. 4. Разрешающая диаграмма

Булеву функцию $F\colon \{0,1\}^k \to \{0,1\}$ от k переменных можно задать с помощью двоичной разрешающей диаграммы, для которой верно:

- разрешающая диаграмма — ориентированный ациклический граф;
- каждая вершина помечена какой-нибудь переменной;
- одно исходящее ребро помечено единицей, другое — нулем;
- в листьях (терминалах) записаны значения функции.

Двоичные разрешающие диаграммы применяются, в частности, для символьной верификации моделей программ [6, 7]. Кроме того, диаграмма называется упорядоченной, если переменные упорядочены и на каждом пути встречаются именно в этом порядке.

Вообще говоря, описываемая диаграмма (рис. 4) не является классической двоичной разрешающей диаграммой, поскольку терминальные вершины отмечены не нулем или единицей, а римскими цифрами, обозначающими набор действий для выполнения (табл. 2). Тем не менее, если по относительно громоздким логическим условиям, приведенным в начале данного раздела, и таблице комбинаций входных переменных (табл. 2) трудно разглядеть, что система условий полна и не содержит противоречий, то зная, как были получены логические условия, и используя разрешающую диаграмму, можно убедиться в справедливости этого утверждения для системы условий.

Для большей наглядности перепишем формулу для набора действий II с использованием коммутативного закона для дизъюнкции:

$$\overline{a} \wedge ((\overline{b} \wedge ((\overline{c} \wedge d) \vee c)) \vee b) = \overline{a} \wedge (b \vee (\overline{b} \wedge (c \vee (\overline{c} \wedge d)))).$$

Теперь посмотрим, как на разрешающей диаграмме (рис. 4) можно попасть из вершины с меткой a в терминальную вершину, отмеченную римской цифрой II. Для этого из a нужно пойти по дуге, на которой написан 0 ($\overline{a}$). Описывая пути, в скобках будем приводить подформулу, соответствующую данному пути. Если из вершины b пойти по дуге, отмеченной 1, то сразу попадем в вершину II ($\overline{a} \wedge b$). Из вершины b в вершину II также ведут пути $b \overset{0}{\to} c \overset{1}{\to}$ II (соответствующая подформула — $\overline{a} \wedge (\overline{b} \wedge c)$) и $b \overset{0}{\to} c \overset{0}{\to} d \overset{1}{\to}$ II ($\overline{a} \wedge (\overline{b} \wedge (\overline{c} \wedge d))$).

Итак, самая «трудная» система логических условий на рассматриваемом графе «побеждена»! Выполнение подобной проверки для всех остальных логических условий, на наш взгляд, уже не вызовет затруднений.

4. Реализация с двумя автоматами

В данном разделе приводится вариант реализации торгового устройства с двумя автоматами: для логики (автомат A1) и для интерфейса (автомат A2). Идея отделения функциональности от интерфейса характерна для объектно-ориентированного подхода.

4.1. Схема связей

Схема связей (рис. 5) теперь показывает еще и взаимосвязь автоматов A1 и A2 друг с другом.

Рис. 5. Схема связей

Автомат A1 управляет внутренними системами устройства для продажи газированной воды:

- монетным механизмом;
- устройством для приготовления напитка;
- механизмом выдачи напитка;
- холодильной установкой.

За взаимодействие с клиентом отвечает автомат A2, который непосредственно управляет таймером и дисплеем, осуществляя вывод сообщений на экран и информации о накопленной сумме. Отметим, что автомат A2 опрашивает входные переменные внутренних компонентов торгового устройства для адекватного реагирования при поступлении события.

4.2. Граф переходов автомата A1

Сравнивая граф переходов автомата A1 (рис. 6) с первоначальным графом переходов (рис. 3), отметим, что исчезли громоздкие логические условия, обсуждению корректности которых посвящен разд. 3.4. Правда, это произошло ценой появления автомата A2.

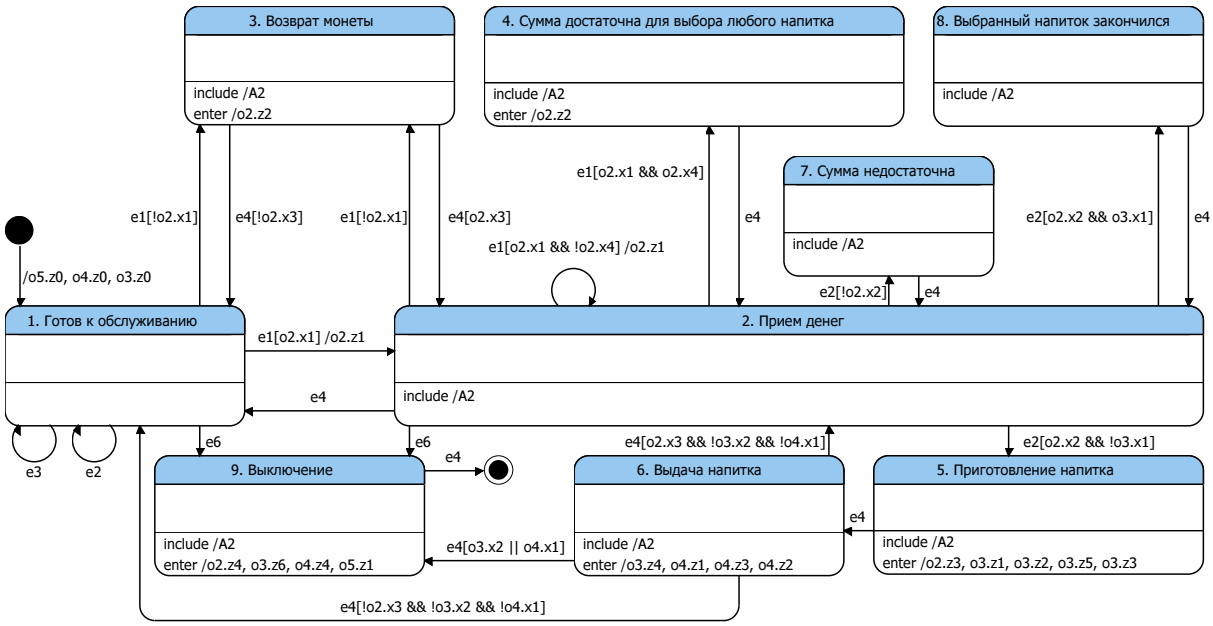

Рис. 6. Граф переходов автомата A1

Фраза «include /A2» в нижней части вершины свидетельствует о том, что в нее вложен автомат A2. В связи с этим изменяются и правила исполнения данной *UML*-диаграммы. После анализа логических условий и выбора перехода в новое состояние:

- сначала выполняется список выходных воздействий, ассоциированных с переходом;

- затем выполняются выходные воздействия, предусмотренные для исполнения в момент входа в состояние;
- управление передается вложенному автомату A2.

4.3. Граф переходов автомата A2

Автомат A2 (рис. 7) вызывает выходные воздействия объекта управления o1 («Дисплей») для показа сообщений, управления состоянием индикаторов наличия напитков, отображения текущего значения суммы.

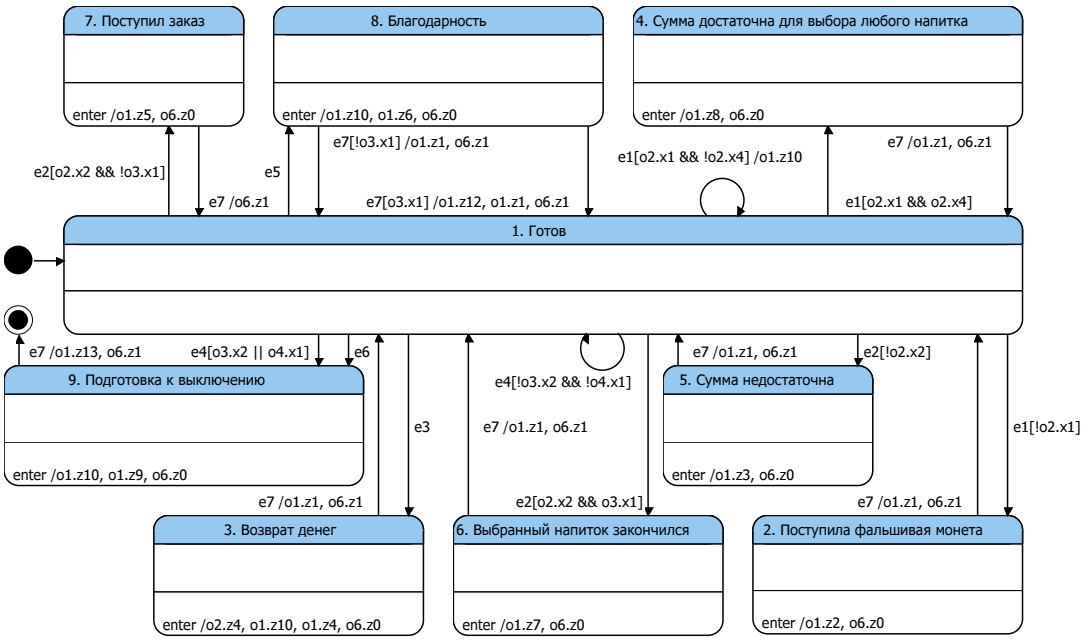

Рис. 7. Граф переходов автомата A2

5. Третий вариант реализации

Идея разграничения функциональности и интерфейса, воплощенная во втором варианте реализации, привела к созданию двух автоматов: A1 и A2 для управления внутренними системами устройства и взаимодействия с пользователем соответственно. Заметим, что суммарное число состояний в двух автоматах возросло по сравнению с первоначальным вариантом. Это произошло из-за того, что часто требуется одновременно выполнить выходные воздействия на объекты управления и отобразить какое-нибудь сообщение на дисплее.

Кроме того, предыдущие два варианта реализации имели следующее ограничение. Если в момент приготовления напитка пользователь опустит монету или нажмет кнопку «Сдача», то эти действия будут проигнорированы. Это вызвано отсутствием переходов по событиям e1 и e3 из состояния «5. Приготовление напитка» (рис. 6).

5.1. Схема связей

Рассматриваемый в данном разделе вариант реализации лишен описанного выше ограничения. Дублирование состояний в варианте с двумя автоматами (разд. 4) во многом было вызвано необходимостью отображения сообщений на дисплее устройства в течение некоторого промежутка времени. За это отвечал автомат A2 (рис. 7). Отметим, что для показа девяти различных сообщений (рис. 5) выполнялась схожая последовательность действий, в чем можно убедиться по графу переходов автомата A2 (рис. 7) и исходному коду выходных воздействий.

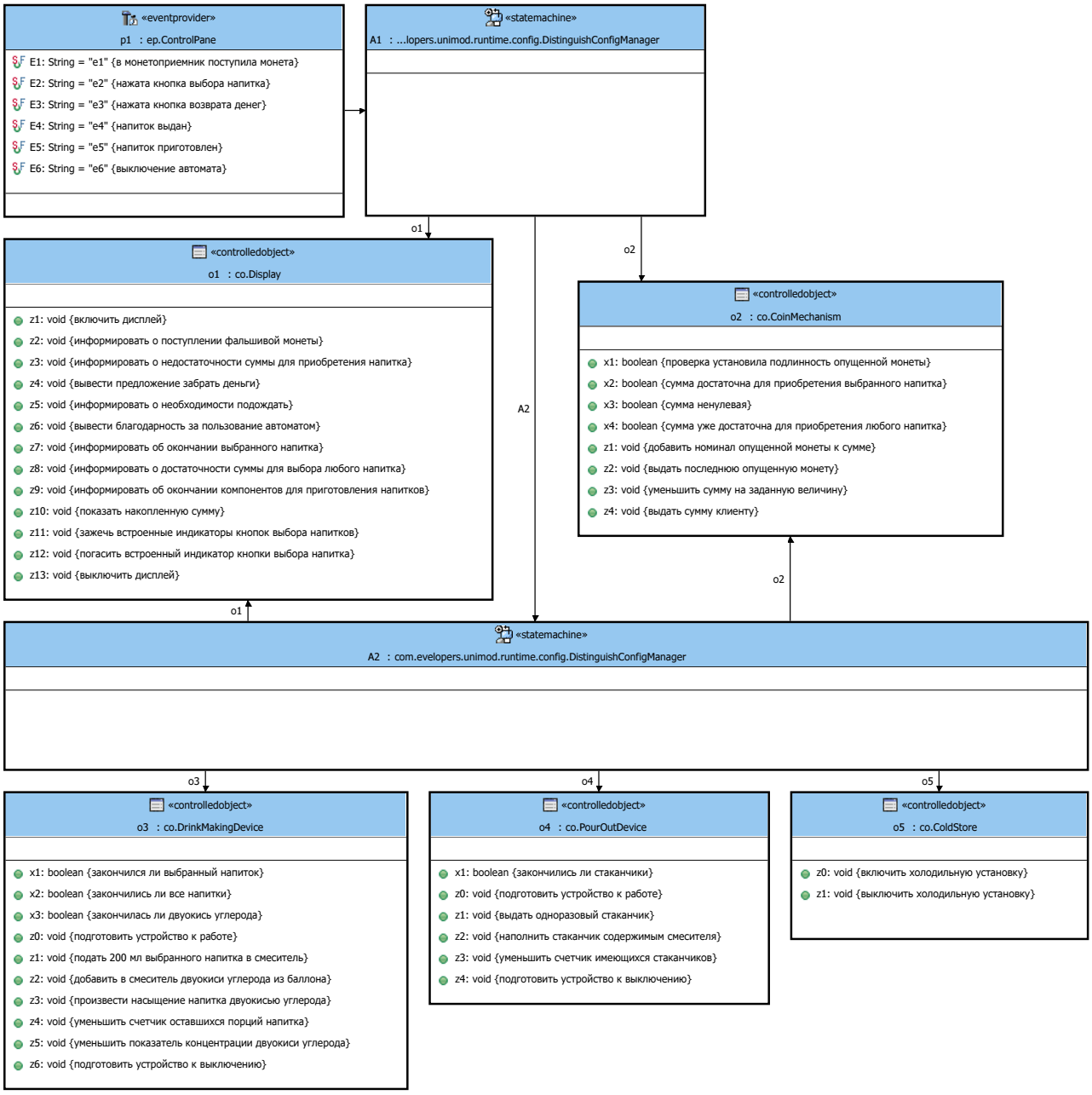

Рис. 8. Схема связей третьего варианта реализации

Поэтому для упрощения системы автоматов было принято решение изменить функциональность объектов управления. Дисплей в данной реализации позволяет показывать сообщение в течение заданного интервала времени. Благодаря такому изменению функциональности дисплея, потребность в таймере отпала. Итак, объект управления об (таймер) отсутствует на схеме связей (рис. 8) и более не используется.

5.2. Граф переходов автомата A1

Для преодоления ограничения, связанного с игнорированием событий в процессе приготовления и выдачи напитка, основной цикл обслуживания (приготовление и выдача напитка) теперь осуществляет автомат A2, вложенный в состояние «1. Готов к обслуживанию» автомата A1 (рис. 9). Таким образом, автомат A1 способен реагировать на поступающие события независимо от работы автомата A2.

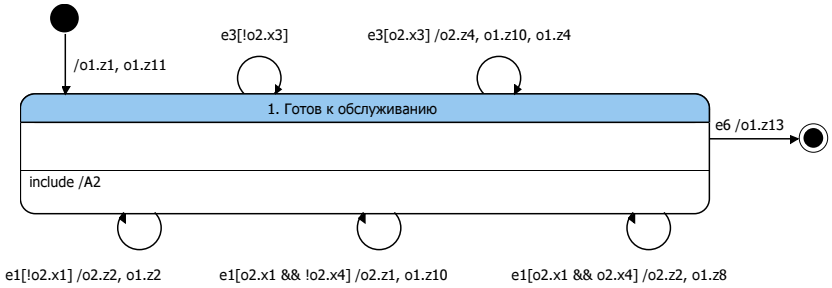

Рис. 9. Граф переходов автомата A1 для третьего варианта реализации

5.3. Граф переходов автомата A2

Граф переходов автомата A2 (рис. 10) заметно упростился.

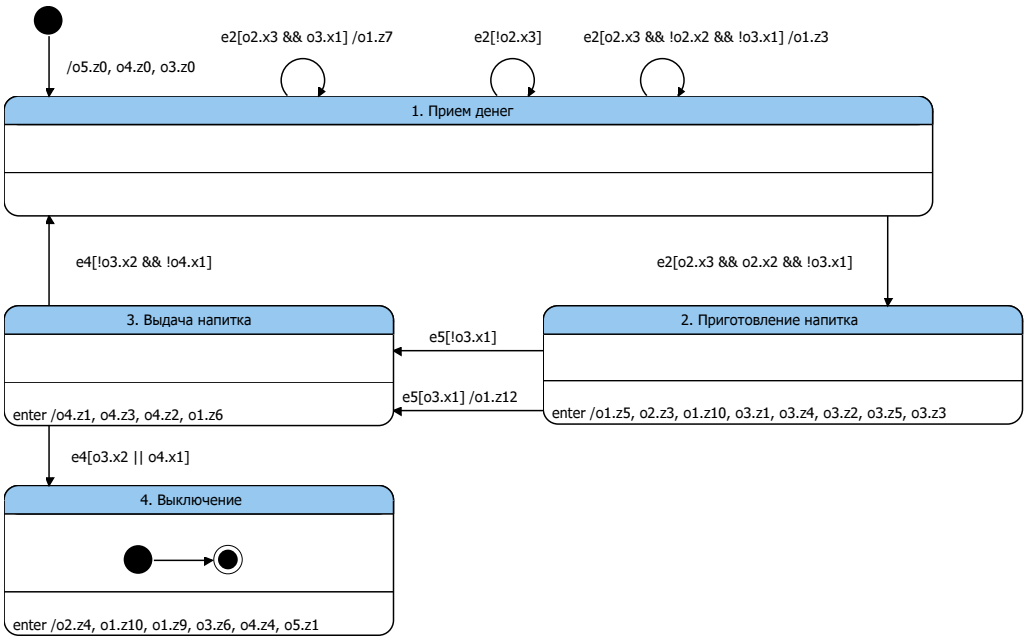

Рис. 10. Граф переходов автомата A2 для третьего варианта реализации

«Сложное» состояние «4. Выключение» (рис. 10) означает, что после перехода в данное состояние сначала выполнится набор выходных воздействий на объекты управления, а затем автомат A2 перейдет в финальное состояние.

Автомат A2 вложен в основное рабочее состояние автомата A1. Следовательно, все возникающие события переадресовываются также автомату A2. Заметим, что обработка события e2 («Нажата кнопка выбора напитка») выполняется только автоматом A2, в то время как у автомата A1 вообще нет переходов по событию e2.

6. Два подхода к проектированию

Одна из сильных сторон пакета *UniMod* — это возможность визуального конструирования программ [5]. В отличие от распространенного подхода, когда вспомогательные картинки и *UML*-диаграммы рисуются с надеждой на улучшение документации и продуктивности труда, разработанные с помощью инструментального средства *UniMod* диаграммы и вручную написанные классы в целом формируют работающее приложение.

6.1. Интерпретационный подход

При разработке программы в среде *Eclipse* применяется интерпретационный подход, позволяющий эффективно выполнять тестирование и отладку. Для получения готового приложения в рамках данного подхода *UML*-диаграммы экспортируются в *XML*-описание (Приложение 1).

В рамках интерпретационного подхода (рис. 11) во время исполнения программы в памяти создаются экземпляры *Java*-классов источников событий и объектов управления.

В процессе обработки события интерпретатор по текущему состоянию получает набор переходов в новое состояние, вычисляет логические условия переходов. Если результатом вычисления какого-либо логического условия является истина, то происходит следующее:

- интерпретатор с использованием механизма *Reflection* языка *Java* вызывает выходные воздействия, ассоциированные с выбранным переходом;
- затем производится переход в новое состояние, и выполняются выходные воздействия, предусмотренные для исполнения в момент входа в новое состояние;
- управление передается вложенным автоматам.

Рис. 11. Этапы разработки и выполнения приложения на основе интерпретационного подхода

В процессе работы программы ведется протокол, в который заносятся:

- поступающие события;
- состояние, в котором находится каждый автомат;
- значения входных переменных;
- переходы;
- выходные воздействия.

Интерпретационный подход весьма удобен в процессе разработки приложения, однако он обладает следующими недостатками:

- для запуска программы, кроме скомпилированных *Java*-классов, дополнительно необходимы *XML*-описание и интерпретатор *UniMod*;

- по производительности интерпретационный подход выполнения программы проигрывает компиляционному.

Все это привело к созданию компиляционного подхода. В заключение раздела отметим, что приложения 2 и 3 содержат исходные коды третьего варианта реализации.

6.2. Компиляционный подход

Данный подход позволяет транслировать *UML*-диаграммы в код на языке *Java* (рис. 12). Трансляция *XML*-описания, полученного по диаграммам, в код на языке *Java* осуществляется с помощью довольно гибкого инструмента *Velocity* (`http://jakarta.apache.org/velocity`). С автоматической генерацией кода по *UML*-диаграммам с использованием шаблонов *Velocity* можно ознакомиться в работе [8].

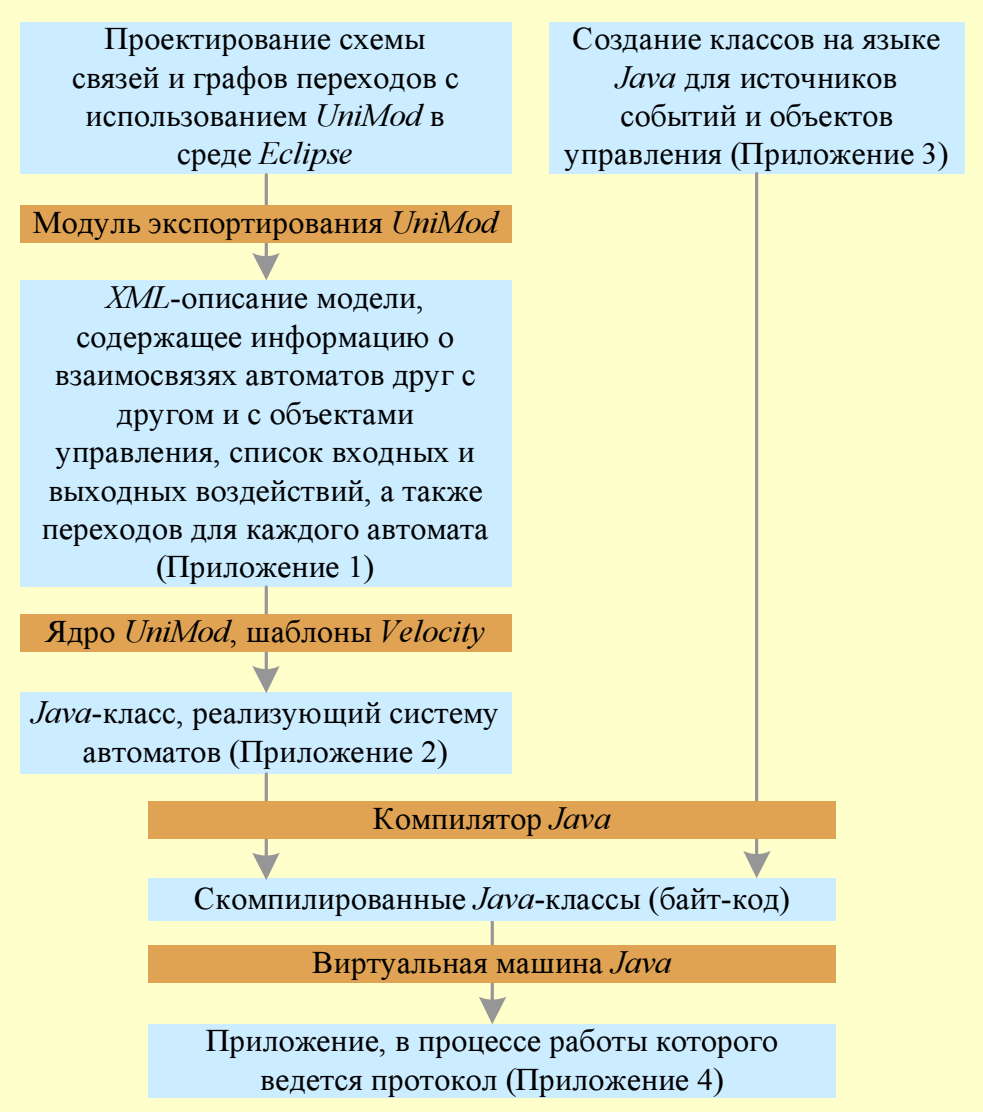

Рис. 12. Этапы разработки и выполнения приложения на основе компиляционного подхода

По *XML*-описанию автоматически генерируется *Java*-класс, который реализует систему автоматов (Приложение 2). Затем этот *Java*-класс и классы, реализующие функциональность источников событий и объектов управления (Приложение 3), компилируются. Полученное таким способом приложение уже не зависит ни от *XML*-описания, ни от интерпретатора *UniMod*. Для запуска программы потребуется только библиотека времени выполнения *UniMod* в скомпилированном виде.

Отметим, что *UML*-диаграммам однозначно соответствует автоматически генерируемый код на языке *Java*. При этом приложение, разработанное в рамках интерпретационного подхода, будет функционально эквивалентно его аналогу, построенному с использованием трансляции диаграмм в код на языке *Java*.

По *XML*-описанию автоматически был сгенерирован класс на языке *Java* (Приложение 2), который содержит более двух тысяч строк кода для второго варианта реализации (рис. 13). Таким образом, код для основы приложения — системы взаимодействующих автоматов — создается автоматически. Это позволяет проектировать диаграммы на более высоком уровне абстракции, нежели чем при традиционном подходе.

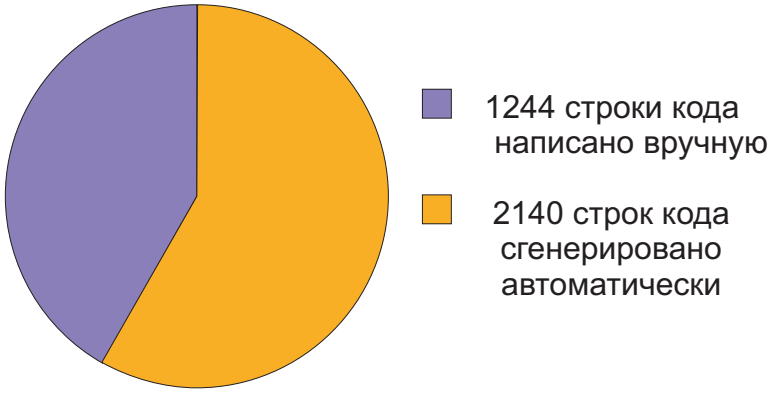

Рис. 13. Сравнение количества строк кода на языке *Java*, написанных вручную и сгенерированных по *UML*-диаграммам

6.3. Структура *XML*-описания

Этап построения *XML*-описания является общим для интерпретационного и компиляционного подходов.

Рассмотрим структуру *XML*-описания (рис. 14). В него входит информация как со схемы связей, так и с графов переходов.

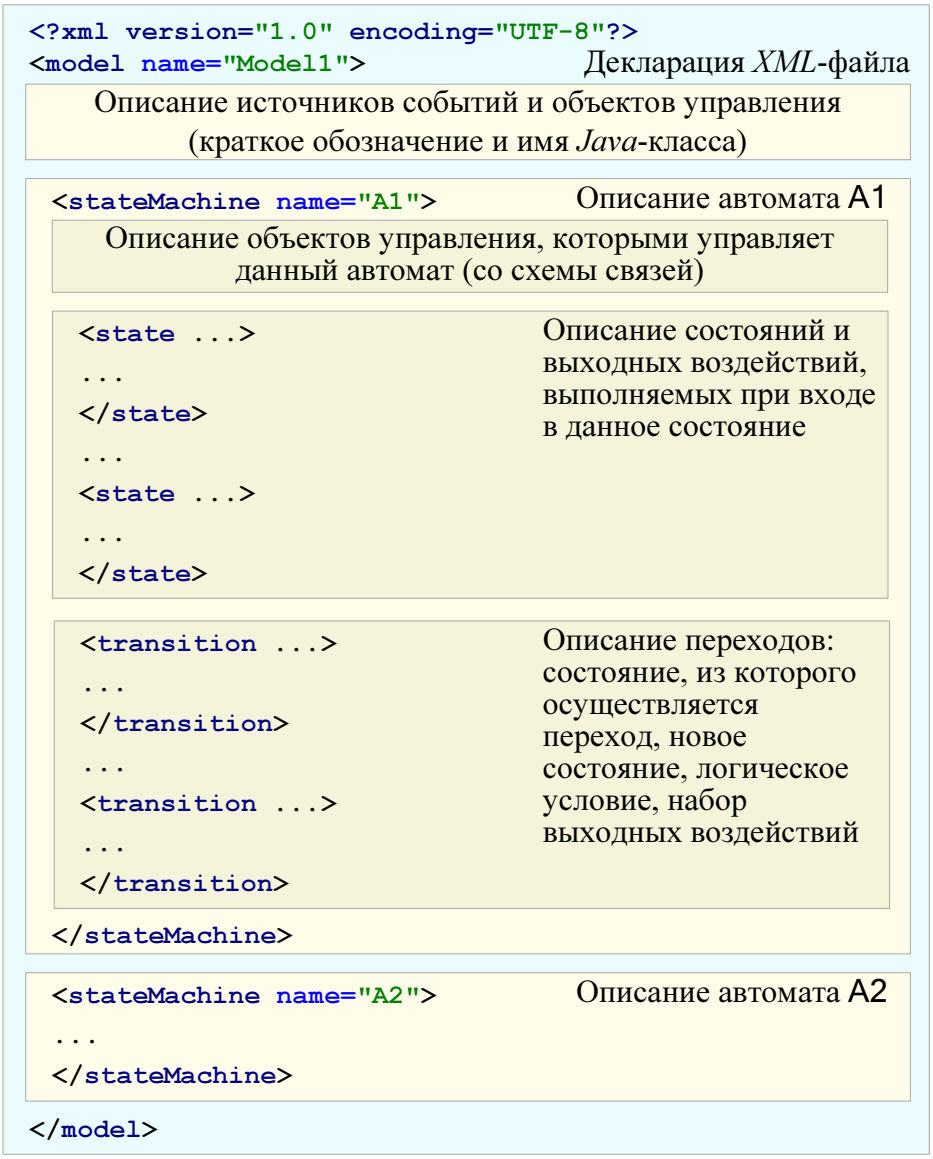

Рис. 14. Структура *XML*-описания

Заключение

SWITCH-технология динамично развивается. Недавно появился мощный набор инструментов *UniMod*, который позволяет более доступно и наглядно применить автоматный подход для проектирования достаточно сложных систем. При разработке программ с помощью инструментального средства *UniMod* схемы связей и графы переходов создаются с использованием *UML*-нотации диаграмм классов и диаграмм состояний соответственно. При этом диаграммы автоматически проверяются на наличие ошибок: выявляются недостижимые состояния, противоречия в логических условиях на переходах. Отличительной чертой визуального конструирования программ с помощью пакета *UniMod* является то, что:

- проект четко структурирован;
- связи между компонентами выделены явно.

Данный проект торгового устройства наряду со многими другими, опубликованными на сайте http://is.ifmo.ru, демонстрирует хорошую применимость автоматного подхода и *SWITCH*-технологии для управления «реактивными» системами [3], а также в тех предметных областях, где можно выделить несколько объектов, сложным образом взаимодействующих друг с другом.

Модель устройства для продажи газированной воды, представленную в данной работе, можно усовершенствовать в нескольких направлениях.

1. Можно расширить разновидность принимаемых платежных средств — наряду с монетами можно осуществить поддержку банкнот как средства оплаты напитков.
2. Спроектированное торговое устройство является автономным. При построении модели предполагалось, что запаса монет всегда достаточно, чтобы выдать сдачу клиенту. Если же запас монет ограничен, то может возникнуть ситуация, когда не удастся выдать сдачу из-за отсутствия монет номиналом один рубль, например. Можно сделать связь торгового устройства со службой поддержки, чтобы оно могло сообщить о недостатке монет какого-либо номинала для сдачи или о скором окончании того или иного компонента для приготовления газированной воды.
3. Обычно торговые устройства защищены от взлома и изъятия выручки корпусом из листа железа и сейфовым замком. Можно включить в комплектацию устройства сигнализацию и предпринимать определенные действия при нападении и взломе.
4. Можно реализовать такое устройство для продажи напитков, которое будет комплектоваться баллонами с водой и сиропами и будет торговать тонизирующими напитками.

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.* Танки и автоматы // BYTE/Россия. 2003. № 2, с. 69–73. http://is.ifmo.ru/works/tanks_new/.

3. *Шалыто А. А., Туккель Н. И.* SWITCH-технология — автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. 2001. № 5, с. 45–62. http://is.ifmo.ru/works/switch/1/.

4. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9, с. 52–56. http://is.ifmo.ru/works/open_doc/.

5. *Новиков Ф. А.* Визуальное конструирование программ // Информационно-управляющие системы. 2005. № 6, с. 9–22. http://is.ifmo.ru/works/visualcons/.

6. *Bryant R. E.* Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. ACM Computing Surveys. 1992. No. 3, pp. 293–318.

7. *Кларк Э. М., Грамберг О., Пелед Д.* Верификация моделей программ: Model checking. М.: МЦНМО, 2002.

8. *Sturm T., Voss J., Boger M.* Generating Code from UML with Velocity Templates. Lecture Notes in Computer Science. 2002. Vol. 2460, pp. 150–161.

Приложение 1.

XML-описание для третьего варианта реализации

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE model PUBLIC
   "-//eVelopers Corp.//DTD State machine model V1.0//EN"
   "http://www.evelopers.com/dtd/unimod/statemachine.dtd">
<model name="Model1">
   <controlledObject class="co.Display" name="o1"/>
   <controlledObject class="co.CoinMechanism" name="o2"/>
   <controlledObject class="co.DrinkMakingDevice" name="o3"/>
   <controlledObject class="co.PourOutDevice" name="o4"/>
   <controlledObject class="co.ColdStore" name="o5"/>
   <eventProvider class="ep.ControlPane" name="p1">
      <association clientRole="p1" targetRef="A1"/>
   </eventProvider>
   <rootStateMachine>
      <stateMachineRef name="A1"/>
   </rootStateMachine>
   <stateMachine name="A1">
      <configStore class=
         "com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
      <association clientRole="A1" supplierRole="o2" targetRef="o2"/>
      <association clientRole="A1" supplierRole="A2" targetRef="A2"/>
      <association clientRole="A1" supplierRole="o1" targetRef="o1"/>
      <state name="Top" type="NORMAL">
         <state name="2. Финальное состояние" type="FINAL"/>
         <state name="0. Начальное состояние" type="INITIAL"/>
         <state name="1. Готов к обслуживанию" type="NORMAL">
            <stateMachineRef name="A2"/>
         </state>
      </state>
      <transition sourceRef="0. Начальное состояние"
                  targetRef="1. Готов к обслуживанию">
         <outputAction ident="o1.z1"/>
         <outputAction ident="o1.z11"/>
```

```
    </transition>
    <transition event="e6" sourceRef="1. Готов к обслуживанию"
                targetRef="2. Финальное состояние">
       <outputAction ident="o1.z13"/>
    </transition>
    <transition event="e1" guard="!o2.x1"
                sourceRef="1. Готов к обслуживанию"
                targetRef="1. Готов к обслуживанию">
       <outputAction ident="o2.z2"/>
       <outputAction ident="o1.z2"/>
    </transition>
    <transition event="e1" guard="o2.x1 &amp;amp;&amp;amp; !o2.x4"
                sourceRef="1. Готов к обслуживанию"
                targetRef="1. Готов к обслуживанию">
       <outputAction ident="o2.z1"/>
       <outputAction ident="o1.z10"/>
    </transition>
    <transition event="e1" guard="o2.x1 &amp;amp;&amp;amp; o2.x4"
                sourceRef="1. Готов к обслуживанию"
                targetRef="1. Готов к обслуживанию">
       <outputAction ident="o2.z2"/>
       <outputAction ident="o1.z8"/>
    </transition>
    <transition event="e3" guard="!o2.x3"
                sourceRef="1. Готов к обслуживанию"
                targetRef="1. Готов к обслуживанию"/>
    <transition event="e3" guard="o2.x3"
                sourceRef="1. Готов к обслуживанию"
                targetRef="1. Готов к обслуживанию">
       <outputAction ident="o2.z4"/>
       <outputAction ident="o1.z10"/>
       <outputAction ident="o1.z4"/>
    </transition>
 </stateMachine>
 <stateMachine name="A2">
    <configStore class=
       "com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
    <association clientRole="A2" supplierRole="o3" targetRef="o3"/>
    <association clientRole="A2" supplierRole="o5" targetRef="o5"/>
    <association clientRole="A2" supplierRole="o4" targetRef="o4"/>
    <association clientRole="A2" supplierRole="o1" targetRef="o1"/>
    <association clientRole="A2" supplierRole="o2" targetRef="o2"/>
    <state name="Top" type="NORMAL">
       <state name="1. Прием денег" type="NORMAL"/>
```

```
   <state name="4. Выключение" type="NORMAL">
      <state name="5. Начальное состояние" type="INITIAL"/>
      <state name="6. Финальное состояние" type="FINAL"/>
      <outputAction ident="o2.z4"/>
      <outputAction ident="o1.z10"/>
      <outputAction ident="o1.z9"/>
      <outputAction ident="o3.z6"/>
      <outputAction ident="o4.z4"/>
      <outputAction ident="o5.z1"/>
   </state>
   <state name="0. Начальное состояние" type="INITIAL"/>
   <state name="3. Выдача напитка" type="NORMAL">
      <outputAction ident="o4.z1"/>
      <outputAction ident="o4.z3"/>
      <outputAction ident="o4.z2"/>
      <outputAction ident="o1.z6"/>
   </state>
   <state name="2. Приготовление напитка" type="NORMAL">
      <outputAction ident="o1.z5"/>
      <outputAction ident="o2.z3"/>
      <outputAction ident="o1.z10"/>
      <outputAction ident="o3.z1"/>
      <outputAction ident="o3.z4"/>
      <outputAction ident="o3.z2"/>
      <outputAction ident="o3.z5"/>
      <outputAction ident="o3.z3"/>
   </state>
</state>
<transition event="e2"
            guard="o2.x3 &amp;&amp; o2.x2 &amp;&amp; !o3.x1"
            sourceRef="1. Прием денег"
            targetRef="2. Приготовление напитка"/>
<transition event="e2" guard="o2.x3 &amp;&amp; o3.x1"
            sourceRef="1. Прием денег"
            targetRef="1. Прием денег">
   <outputAction ident="o1.z7"/>
</transition>
<transition event="e2" guard="!o2.x3" sourceRef="1. Прием денег"
            targetRef="1. Прием денег"/>
<transition event="e2"
            guard="o2.x3 &amp;&amp; !o2.x2 &amp;&amp; !o3.x1"
            sourceRef="1. Прием денег"
            targetRef="1. Прием денег">
   <outputAction ident="o1.z3"/>
```

```
      </transition>
      <transition sourceRef="5. Начальное состояние"
                  targetRef="6. Финальное состояние"/>
      <transition sourceRef="0. Начальное состояние"
                  targetRef="1. Прием денег">
         <outputAction ident="o5.z0"/>
         <outputAction ident="o4.z0"/>
         <outputAction ident="o3.z0"/>
      </transition>
      <transition event="e4" guard="o3.x2 || o4.x1"
                  sourceRef="3. Выдача напитка"
                  targetRef="4. Выключение"/>
      <transition event="e4" guard="!o3.x2 &amp;amp;&amp;amp; !o4.x1"
                  sourceRef="3. Выдача напитка"
                  targetRef="1. Прием денег"/>
      <transition event="e5" guard="!o3.x1"
                  sourceRef="2. Приготовление напитка"
                  targetRef="3. Выдача напитка"/>
      <transition event="e5" guard="o3.x1"
                  sourceRef="2. Приготовление напитка"
                  targetRef="3. Выдача напитка">
         <outputAction ident="o1.z12"/>
      </transition>
   </stateMachine>
</model>
```

Приложение 2.

Сгенерированный по *XML*-описанию код на *Java*

Файл `Model1EventProcessor.java`

```
import java.util.*;

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

/**
 * Класс, сгенерированный по XML-описанию.
 */
public class Model1EventProcessor extends AbstractEventProcessor {
    // Ссылка на служебный класс, предоставляющий информацию о модели
    private ModelStructure modelStructure;

    // Коды автоматов A1 и A2
    private static final int A1 = 1;
    private static final int A2 = 2;

    // Служебный метод, который по строковому описанию автомата
    // возвращает его код
    private int decodeStateMachine(String sm) {
        if ("A1".equals(sm)) {
            return A1;
        } else if ("A2".equals(sm)) {
            return A2;
        }
        return -1;
    }

    // Ссылки на автоматы A1 и A2
```

```
private A1EventProcessor _A1;
private A2EventProcessor _A2;

// Конструктор сгенерированного класса
public Model1EventProcessor() {
    modelStructure = new Model1ModelStructure();
    _A1 = new A1EventProcessor();
    _A2 = new A2EventProcessor();
}

// Возвращает ссылку на служебный класс,
// предоставляющий информацию о модели
public ModelStructure getModelStructure() {
    return modelStructure;
}

// Информирует об объектах управления
public void setControlledObjectsMap(
    ControlledObjectsMap controlledObjectsMap) {
    super.setControlledObjectsMap(controlledObjectsMap);
    _A1.init(controlledObjectsMap);
    _A2.init(controlledObjectsMap);
}

// Метод для обработки событий, в котором событие делегируется
// одному из автоматов для дальнейшей обработки
protected StateMachineConfig process(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws SystemException {

    // get state machine from path
    int sm = decodeStateMachine(path.getStateMachine());
    try {
        switch (sm) {
        case A1:
            return _A1.process(event, context, path, config);
        case A2:
            return _A2.process(event, context, path, config);
        default:
            throw new EventProcessorException(
                "Unknown state machine ["
                    + path.getStateMachine() + "]");
        }
    } catch (Exception e) {
```

```
            if (e instanceof SystemException) {
                throw (SystemException) e;
            } else {
                throw new SystemException(e);
            }
        }
    }

    // Осуществляет инициализацию автоматов
    protected StateMachineConfig transiteToStableState(
        StateMachineContext context, StateMachinePath path,
        StateMachineConfig config) throws SystemException {

        // get state machine from path
        int sm = decodeStateMachine(path.getStateMachine());
        try {
            switch (sm) {
            case A1:
                return _A1.transiteToStableState(context, path,
                    config);
            case A2:
                return _A2.transiteToStableState(context, path,
                    config);
            default:
                throw new EventProcessorException(
                    "Unknown state machine ["
                        + path.getStateMachine() + "]");
            }
        } catch (Exception e) {
            if (e instanceof SystemException) {
                throw (SystemException) e;
            } else {
                throw new SystemException(e);
            }
        }
    }

    // Служебный класс, предоставляющий информацию о модели
    private class Model1ModelStructure implements ModelStructure {

        private Map configManagers = new HashMap();

        private Model1ModelStructure() {
            configManagers.put("A1",
```

```
            new com.evelopers.unimod.runtime.config
                .DistinguishConfigManager());
        configManagers.put("A2",
            new com.evelopers.unimod.runtime.config
                .DistinguishConfigManager());
    }

    public StateMachinePath getRootPath()
        throws EventProcessorException {
        return new StateMachinePath("A1");
    }

    public StateMachineConfigManager getConfigManager(
        String stateMachine) throws EventProcessorException {
        return (StateMachineConfigManager) configManagers.get(
            stateMachine);
    }

    public StateMachineConfig getTopConfig(String stateMachine)
        throws EventProcessorException {
        int sm = decodeStateMachine(stateMachine);
        switch (sm) {
        case A1:
            return new StateMachineConfig("Top");
        case A2:
            return new StateMachineConfig("Top");
        default:
            throw new EventProcessorException(
                "Unknown state machine [" + stateMachine + "]");
        }
    }

    // Проверяет, достиг ли автомат финального состояния
    public boolean isFinal(String stateMachine,
        StateMachineConfig config) throws EventProcessorException {
        // get state machine from path
        int sm = decodeStateMachine(stateMachine);
        int state;
        switch (sm) {
        case A1:
            state = _A1.decodeState(config.getActiveState());
            switch (state) {
            case A1EventProcessor._2__Финальное_состояние:
                return true;
```

```
            default:
                return false;
            }
        case A2:
            state = _A2.decodeState(config.getActiveState());
            switch (state) {
            case A2EventProcessor._6__Финальное_состояние:
                return true;
            default:
                return false;
            }
        default:
            throw new EventProcessorException(
                "Unknown state machine [" + stateMachine + "]");
        }
    }
}

/**
 * Класс, реализующий автомат A1.
 */
private class A1EventProcessor {
    // Целочисленные коды состояний (в состоянии Top автомат
    // находится до инициализации)
    private static final int Top = 1;
    private static final int _2__Финальное_состояние = 2;
    private static final int _0__Начальное_состояние = 3;
    private static final int _1__Готов_к_обслуживанию = 4;

    // Служебный метод, который по строковому описанию состояния
    // возвращает его код
    private int decodeState(String state) {
        if ("Top".equals(state)) {
            return Top;
        } else if ("2. Финальное состояние".equals(state)) {
            return _2__Финальное_состояние;
        } else if ("0. Начальное состояние".equals(state)) {
            return _0__Начальное_состояние;
        } else if ("1. Готов к обслуживанию".equals(state)) {
            return _1__Готов_к_обслуживанию;
        }
        return -1;
    }
```

```
// Коды событий
private static final int e1 = 1;
private static final int e3 = 2;
private static final int e6 = 3;

// Служебный метод, который по строковому описанию события
// возвращает его код
private int decodeEvent(String event) {
    if ("e1".equals(event)) {
        return e1;
    } else if ("e3".equals(event)) {
        return e3;
    } else if ("e6".equals(event)) {
        return e6;
    }
    return -1;
}

// Объекты управления автомата A1
private co.CoinMechanism o2;
private co.Display o1;

// Инициализация объектов управления автомата A1
private void init(ControlledObjectsMap controlledObjectsMap) {
    o2 = (co.CoinMechanism)
        controlledObjectsMap.getControlledObject("o2");
    o1 = (co.Display)
        controlledObjectsMap.getControlledObject("o1");
}

// Метод для обработки событий: сначала выполняется выбор
// перехода, затем вызываются вложенные автоматы
private StateMachineConfig process(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    config = lookForTransition(event, context, path, config);
    config = transiteToStableState(context, path, config);

    // Вызов вложенных автоматов
    executeSubmachines(event, context, path, config);
    return config;
}

// Выполняет вызов вложенных автоматов
```

```
private void executeSubmachines(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    int state = decodeState(config.getActiveState());
    while (true) {
        switch (state) {
        case _2__Финальное_состояние:
            return;
        case _0__Начальное_состояние:
            return;
        case _1__Готов_к_обслуживанию:
            // 1. Готов к обслуживанию includes A2
            fireBeforeSubmachineExecution(context, event, path,
                "1. Готов к обслуживанию", "A2");
            Model1EventProcessor.this.process(event, context,
                new StateMachinePath(path,
                    "1. Готов к обслуживанию", "A2"));
            fireAfterSubmachineExecution(context, event, path,
                "1. Готов к обслуживанию", "A2");
            return;
        default:
            throw new EventProcessorException(
                "State with name [" + config.getActiveState()
                    + "] is unknown for state machine [A1]");
        }
    }
}

// Осуществляет инициализацию автомата A1
private StateMachineConfig transiteToStableState(
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    int s = decodeState(config.getActiveState());
    Event event;
    switch (s) {
    case Top:
        fireComeToState(context, path,
            "0. Начальное состояние");

        // 0. Начальное состояние->
        // 1. Готов к обслуживанию [true]/o1.z1,o1.z11
        event = Event.NO_EVENT;
        fireTransitionFound(context, path,
            "0. Начальное состояние", event,
```

```
                "0. Начальное состояние#1. Готов к обслуживанию##true");
            fireBeforeOutputActionExecution(context, path,
                "0. Начальное состояние#1. Готов к обслуживанию##true",
                "o1.z1");
            o1.z1(context);
            fireAfterOutputActionExecution(context, path,
                "0. Начальное состояние#1. Готов к обслуживанию##true",
                "o1.z1");
            fireBeforeOutputActionExecution(context, path,
                "0. Начальное состояние#1. Готов к обслуживанию##true",
                "o1.z11");
            o1.z11(context);
            fireAfterOutputActionExecution(context, path,
                "0. Начальное состояние#1. Готов к обслуживанию##true",
                "o1.z11");
            fireComeToState(context, path,
                "1. Готов к обслуживанию");

            // 1. Готов к обслуживанию []
            return new StateMachineConfig(
                "1. Готов к обслуживанию");
        }
        return config;
    }

    // Выполняет поиск перехода (для этого логические условия
    // проверяются на истинность). Когда переход найден,
    // выполняются выходные воздействия,
    // ассоциированные с данным переходом.
    private StateMachineConfig lookForTransition(Event event,
        StateMachineContext context, StateMachinePath path,
        StateMachineConfig config) throws Exception {
        boolean o2_x3 = false,
            o2_x1 = false,
            o2_x4 = false;
        BitSet calculatedInputActions = new BitSet(3);
        int s = decodeState(config.getActiveState());
        int e = decodeEvent(event.getName());
        while (true) {
            switch (s) {
            case _1__Готов_к_обслуживанию:
                switch (e) {
                case e1:
                    // 1. Готов к обслуживанию->
```

```
// 1. Готов к обслуживанию e1[!o2.x1]/o2.z2,o1.z2
fireTransitionCandidate(context, path,
    "1. Готов к обслуживанию", event,
    "1. Готов к обслуживанию#" +
        "1. Готов к обслуживанию#e1#!o2.x1");
if (!isInputActionCalculated(
    calculatedInputActions, _o2_x1)) {
    fireBeforeInputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1",
        "o2.x1");
    o2_x1 = o2.x1(context);
    fireAfterInputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1",
        "o2.x1", new Boolean(o2_x1));
}
if (!o2_x1) {
    fireTransitionFound(context, path,
        "1. Готов к обслуживанию", event,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1");
    fireBeforeOutputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1",
        "o2.z2");
    o2.z2(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1",
        "o2.z2");
    fireBeforeOutputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#!o2.x1",
        "o1.z2");
    o1.z2(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Готов к обслуживанию#" +
```

```
                "1. Готов к обслуживанию#e1#!o2.x1",
            "o1.z2");
        fireComeToState(context, path,
            "1. Готов к обслуживанию");

        // 1. Готов к обслуживанию []
        return new StateMachineConfig(
            "1. Готов к обслуживанию");
    }
    // 1. Готов к обслуживанию-> 1. Готов к обслуживанию
    // e1[o2.x1 && !o2.x4]/o2.z1,o1.z10
    fireTransitionCandidate(context, path,
        "1. Готов к обслуживанию", event,
        "1. Готов к обслуживанию#1. Готов к обслуживанию"
            + "#e1#o2.x1 && !o2.x4");
    if (!isInputActionCalculated(
        calculatedInputActions, _o2_x4)) {
        fireBeforeInputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o2.x4");
        o2_x4 = o2.x4(context);
        fireAfterInputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o2.x4", new Boolean(o2_x4));
    }
    if (o2_x1 && !o2_x4) {
        fireTransitionFound(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o2.z1");
        o2.z1(context);
```

```
        fireAfterOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o2.z1");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o1.z10");
        o1.z10(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && !o2.x4",
            "o1.z10");
        fireComeToState(context, path,
            "1. Готов к обслуживанию");

        // 1. Готов к обслуживанию []
        return new StateMachineConfig(
            "1. Готов к обслуживанию");
    }
    // 1. Готов к обслуживанию->1. Готов к обслуживанию
    // e1[o2.x1 && o2.x4]/o2.z2,o1.z8
    fireTransitionCandidate(context, path,
        "1. Готов к обслуживанию", event,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e1#o2.x1 && o2.x4");
    if (o2_x1 && o2_x4) {
        fireTransitionFound(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && o2.x4");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#" +
                "e1#o2.x1 && o2.x4",
            "o2.z2");
```

```
                o2.z2(context);
                fireAfterOutputActionExecution(context,
                    path,
                    "1. Готов к обслуживанию#" +
                        "1. Готов к обслуживанию#" +
                        "e1#o2.x1 && o2.x4",
                    "o2.z2");
                fireBeforeOutputActionExecution(context,
                    path,
                    "1. Готов к обслуживанию#" +
                        "1. Готов к обслуживанию#" +
                        "e1#o2.x1 && o2.x4",
                    "o1.z8");
                o1.z8(context);
                fireAfterOutputActionExecution(context,
                    path,
                    "1. Готов к обслуживанию#" +
                        "1. Готов к обслуживанию#" +
                        "e1#o2.x1 && o2.x4",
                    "o1.z8");
                fireComeToState(context, path,
                    "1. Готов к обслуживанию");

                // 1. Готов к обслуживанию []
                return new StateMachineConfig(
                    "1. Готов к обслуживанию");
            }

            // transition not found
            return config;
        case e3:
            // 1. Готов к обслуживанию->
            // 1. Готов к обслуживанию e3[!o2.x3]/
            fireTransitionCandidate(context, path,
                "1. Готов к обслуживанию", event,
                "1. Готов к обслуживанию#" +
                    "1. Готов к обслуживанию#e3#!o2.x3");
            if (!isInputActionCalculated(
                calculatedInputActions, _o2_x3)) {
                fireBeforeInputActionExecution(context,
                    path,
                    "1. Готов к обслуживанию#" +
                        "1. Готов к обслуживанию#e3#!o2.x3",
                    "o2.x3");
```

```
        o2_x3 = o2.x3(context);
        fireAfterInputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#e3#!o2.x3",
            "o2.x3", new Boolean(o2_x3));
    }
    if (!o2_x3) {
        fireTransitionFound(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#e3#!o2.x3");
        fireComeToState(context, path,
            "1. Готов к обслуживанию");

        // 1. Готов к обслуживанию []
        return new StateMachineConfig(
            "1. Готов к обслуживанию");
    }
    // 1. Готов к обслуживанию->1. Готов к обслуживанию
    // e3[o2.x3]/o2.z4,o1.z10,o1.z4
    fireTransitionCandidate(context, path,
        "1. Готов к обслуживанию", event,
        "1. Готов к обслуживанию#" +
            "1. Готов к обслуживанию#e3#o2.x3");
    if (o2_x3) {
        fireTransitionFound(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#e3#o2.x3");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#e3#o2.x3",
            "o2.z4");
        o2.z4(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
                "1. Готов к обслуживанию#e3#o2.x3",
            "o2.z4");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Готов к обслуживанию#" +
```

```
                    "1. Готов к обслуживанию#e3#o2.x3",
                "o1.z10");
            o1.z10(context);
            fireAfterOutputActionExecution(context,
                path,
                "1. Готов к обслуживанию#" +
                    "1. Готов к обслуживанию#e3#o2.x3",
                "o1.z10");
            fireBeforeOutputActionExecution(context,
                path,
                "1. Готов к обслуживанию#" +
                    "1. Готов к обслуживанию#e3#o2.x3",
                "o1.z4");
            o1.z4(context);
            fireAfterOutputActionExecution(context,
                path,
                "1. Готов к обслуживанию#" +
                    "1. Готов к обслуживанию#e3#o2.x3",
                "o1.z4");
            fireComeToState(context, path,
                "1. Готов к обслуживанию");

            // 1. Готов к обслуживанию []
            return new StateMachineConfig(
                "1. Готов к обслуживанию");
        }

        // transition not found
        return config;
    case e6:
        // 1. Готов к обслуживанию->
        // 2. Финальное состояние e6[true]/o1.z13
        fireTransitionCandidate(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "2. Финальное состояние#e6#true");
        fireTransitionFound(context, path,
            "1. Готов к обслуживанию", event,
            "1. Готов к обслуживанию#" +
                "2. Финальное состояние#e6#true");
        fireBeforeOutputActionExecution(context, path,
            "1. Готов к обслуживанию#" +
                "2. Финальное состояние#e6#true",
            "o1.z13");
```

```
                    o1.z13(context);
                    fireAfterOutputActionExecution(context, path,
                        "1. Готов к обслуживанию#" +
                            "2. Финальное состояние#e6#true",
                        "o1.z13");
                    fireComeToState(context, path,
                        "2. Финальное состояние");

                    // 2. Финальное состояние []
                    return new StateMachineConfig(
                        "2. Финальное состояние");
                default:

                    // transition not found
                    return config;
                }
            default:
                throw new EventProcessorException(
                    "Incorrect stable state [" + config.getActiveState()
                        + "] in state machine [A1]");
            }
        }
    }

    // Коды входных переменных
    //o2.x3
    private static final int _o2_x3 = 0;
    //o2.x1
    private static final int _o2_x1 = 1;
    //o2.x4
    private static final int _o2_x4 = 2;
}

/**
 * Класс, реализующий автомат A2.
 */
private class A2EventProcessor {
    // Целочисленные коды состояний (в состоянии Top автомат
    // находится до инициализации)
    private static final int Top = 1;
    private static final int _1__Прием_денег = 2;
    private static final int _4__Выключение = 3;
    private static final int _5__Начальное_состояние = 4;
    private static final int _6__Финальное_состояние = 5;
```

```
private static final int _0__Начальное_состояние = 6;
private static final int _3__Выдача_напитка = 7;
private static final int _2__Приготовление_напитка = 8;

// Служебный метод, который по строковому описанию состояния
// возвращает его код
private int decodeState(String state) {
    if ("Top".equals(state)) {
        return Top;
    } else if ("1. Прием денег".equals(state)) {
        return _1__Прием_денег;
    } else if ("4. Выключение".equals(state)) {
        return _4__Выключение;
    } else if ("5. Начальное состояние".equals(state)) {
        return _5__Начальное_состояние;
    } else if ("6. Финальное состояние".equals(state)) {
        return _6__Финальное_состояние;
    } else if ("0. Начальное состояние".equals(state)) {
        return _0__Начальное_состояние;
    } else if ("3. Выдача напитка".equals(state)) {
        return _3__Выдача_напитка;
    } else if ("2. Приготовление напитка".equals(state)) {
        return _2__Приготовление_напитка;
    }
    return -1;
}

// Коды событий
private static final int e4 = 1;
private static final int e2 = 2;
private static final int e5 = 3;

// Служебный метод, который по строковому описанию события
// возвращает его код
private int decodeEvent(String event) {
    if ("e4".equals(event)) {
        return e4;
    } else if ("e2".equals(event)) {
        return e2;
    } else if ("e5".equals(event)) {
        return e5;
    }
    return -1;
}
```

```
// Объекты управления автомата A2
private co.DrinkMakingDevice o3;
private co.ColdStore o5;
private co.PourOutDevice o4;
private co.Display o1;
private co.CoinMechanism o2;

// Инициализация объектов управления автомата A2
private void init(ControlledObjectsMap controlledObjectsMap) {
    o3 = (co.DrinkMakingDevice)
        controlledObjectsMap.getControlledObject("o3");
    o5 = (co.ColdStore)
        controlledObjectsMap.getControlledObject("o5");
    o4 = (co.PourOutDevice)
        controlledObjectsMap.getControlledObject("o4");
    o1 = (co.Display)
        controlledObjectsMap.getControlledObject("o1");
    o2 = (co.CoinMechanism)
        controlledObjectsMap.getControlledObject("o2");
}

// Метод для обработки событий: сначала выполняется выбор
// перехода, затем вызываются вложенные автоматы
private StateMachineConfig process(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    config = lookForTransition(event, context, path, config);
    config = transiteToStableState(context, path, config);

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

// Выполняет вызов вложенных автоматов
private void executeSubmachines(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    int state = decodeState(config.getActiveState());
    while (true) {
        switch (state) {
        case _1__Прием_денег:
            return;
```

```
            case _4__Выключение:
                return;
            case _5__Начальное_состояние:
                state = _4__Выключение;
                break;
            case _6__Финальное_состояние:
                state = _4__Выключение;
                break;
            case _0__Начальное_состояние:
                return;
            case _3__Выдача_напитка:
                return;
            case _2__Приготовление_напитка:
                return;
            default:
                throw new EventProcessorException(
                    "State with name [" + config.getActiveState()
                        + "] is unknown for state machine [A2]");
            }
        }
    }

    // Осуществляет инициализацию автомата A2
    private StateMachineConfig transiteToStableState(
        StateMachineContext context, StateMachinePath path,
        StateMachineConfig config) throws Exception {
        int s = decodeState(config.getActiveState());
        Event event;
        switch (s) {
        case Top:
            fireComeToState(context, path,
                "0. Начальное состояние");

            // 0. Начальное состояние->
            // 1. Прием денег [true]/o5.z0,o4.z0,o3.z0
            event = Event.NO_EVENT;
            fireTransitionFound(context, path,
                "0. Начальное состояние", event,
                "0. Начальное состояние#1. Прием денег##true");
            fireBeforeOutputActionExecution(context, path,
                "0. Начальное состояние#1. Прием денег##true",
                "o5.z0");
            o5.z0(context);
            fireAfterOutputActionExecution(context, path,
```

```
                "0. Начальное состояние#1. Прием денег##true",
                "o5.z0");
            fireBeforeOutputActionExecution(context, path,
                "0. Начальное состояние#1. Прием денег##true",
                "o4.z0");
            o4.z0(context);
            fireAfterOutputActionExecution(context, path,
                "0. Начальное состояние#1. Прием денег##true",
                "o4.z0");
            fireBeforeOutputActionExecution(context, path,
                "0. Начальное состояние#1. Прием денег##true",
                "o3.z0");
            o3.z0(context);
            fireAfterOutputActionExecution(context, path,
                "0. Начальное состояние#1. Прием денег##true",
                "o3.z0");
            fireComeToState(context, path, "1. Прием денег");

            // 1. Прием денег []
            return new StateMachineConfig("1. Прием денег");
        case _4__Выключение:
            fireCompositeTargetState(context, path,
                "4. Выключение");
            fireComeToState(context, path,
                "5. Начальное состояние");

            // 5. Начальное состояние->6. Финальное состояние [true]/
            event = Event.NO_EVENT;
            fireTransitionFound(context, path,
                "5. Начальное состояние", event,
                "5. Начальное состояние#6. Финальное состояние##true");
            fireComeToState(context, path,
                "6. Финальное состояние");

            // 6. Финальное состояние []
            return new StateMachineConfig(
                "6. Финальное состояние");
        }
        return config;
    }

    // Выполняет поиск перехода (для этого логические условия
    // проверяются на истинность). Когда переход найден,
    // выполняются выходные воздействия,
```

```
// ассоциированные с данным переходом.
private StateMachineConfig lookForTransition(Event event,
    StateMachineContext context, StateMachinePath path,
    StateMachineConfig config) throws Exception {
    boolean o2_x3 = false,
        o4_x1 = false,
        o3_x2 = false,
        o3_x1 = false,
        o2_x2 = false;
    BitSet calculatedInputActions = new BitSet(5);
    int s = decodeState(config.getActiveState());
    int e = decodeEvent(event.getName());
    while (true) {
        switch (s) {
        case _1__Прием_денег:
            switch (e) {
            case e2:
                // 1. Прием денег->2. Приготовление напитка
                // e2[o2.x3 && o2.x2 && !o3.x1]/
                fireTransitionCandidate(context, path,
                    "1. Прием денег", event,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1");
                if (!isInputActionCalculated(
                    calculatedInputActions, _o2_x3)) {
                    fireBeforeInputActionExecution(context,
                        path,
                        "1. Прием денег#2. Приготовление напитка#" +
                            "e2#o2.x3 && o2.x2 && !o3.x1",
                        "o2.x3");
                    o2_x3 = o2.x3(context);
                    fireAfterInputActionExecution(context,
                        path,
                        "1. Прием денег#2. Приготовление напитка#" +
                            "e2#o2.x3 && o2.x2 && !o3.x1",
                        "o2.x3", new Boolean(o2_x3));
                }
                if (!isInputActionCalculated(
                    calculatedInputActions, _o3_x1)) {
                    fireBeforeInputActionExecution(context,
                        path,
                        "1. Прием денег#2. Приготовление напитка#" +
                            "e2#o2.x3 && o2.x2 && !o3.x1",
                        "o3.x1");
```

```
                o3_x1 = o3.x1(context);
                fireAfterInputActionExecution(context,
                    path,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1",
                    "o3.x1", new Boolean(o3_x1));
            }
            if (!isInputActionCalculated(
                calculatedInputActions, _o2_x2)) {
                fireBeforeInputActionExecution(context,
                    path,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1",
                    "o2.x2");
                o2_x2 = o2.x2(context);
                fireAfterInputActionExecution(context,
                    path,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1",
                    "o2.x2", new Boolean(o2_x2));
            }
            if (o2_x3 && o2_x2 && !o3_x1) {
                fireTransitionFound(context, path,
                    "1. Прием денег", event,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1");
                fireComeToState(context, path,
                    "2. Приготовление напитка");

                // 2. Приготовление напитка [o1.z5, o2.z3,
                // o1.z10, o3.z1, o3.z4, o3.z2, o3.z5, o3.z3]
                fireBeforeOutputActionExecution(context,
                    path,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1",
                    "o1.z5");
                o1.z5(context);
                fireAfterOutputActionExecution(context,
                    path,
                    "1. Прием денег#2. Приготовление напитка#" +
                        "e2#o2.x3 && o2.x2 && !o3.x1",
                    "o1.z5");
                fireBeforeOutputActionExecution(context,
                    path,
```

```
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o2.z3");
    o2.z3(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o2.z3");
    fireBeforeOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o1.z10");
    o1.z10(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o1.z10");
    fireBeforeOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o3.z1");
    o3.z1(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o3.z1");
    fireBeforeOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o3.z4");
    o3.z4(context);
    fireAfterOutputActionExecution(context,
        path,
        "1. Прием денег#2. Приготовление напитка#" +
            "e2#o2.x3 && o2.x2 && !o3.x1",
        "o3.z4");
    fireBeforeOutputActionExecution(context,
        path,
```

```
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z2");
        o3.z2(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z2");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z5");
        o3.z5(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z5");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z3");
        o3.z3(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Прием денег#2. Приготовление напитка#" +
                "e2#o2.x3 && o2.x2 && !o3.x1",
            "o3.z3");
        return new StateMachineConfig(
            "2. Приготовление напитка");
    }
    // 1. Прием денег->
    // 1. Прием денег e2[o2.x3 && o3.x1]/o1.z7
    fireTransitionCandidate(context, path,
        "1. Прием денег", event,
        "1. Прием денег#1. Прием денег#" +
            "e2#o2.x3 && o3.x1");
    if (o2_x3 && o3_x1) {
        fireTransitionFound(context, path,
            "1. Прием денег", event,
            "1. Прием денег#1. Прием денег#" +
```

```
                "e2#o2.x3 && o3.x1");
        fireBeforeOutputActionExecution(context,
            path,
            "1. Прием денег#1. Прием денег#" +
                "e2#o2.x3 && o3.x1",
            "o1.z7");
        o1.z7(context);
        fireAfterOutputActionExecution(context,
            path,
            "1. Прием денег#1. Прием денег#" +
                "e2#o2.x3 && o3.x1",
            "o1.z7");
        fireComeToState(context, path,
            "1. Прием денег");

        // 1. Прием денег []
        return new StateMachineConfig(
            "1. Прием денег");
    }
    // 1. Прием денег->1. Прием денег e2[!o2.x3]/
    fireTransitionCandidate(context, path,
        "1. Прием денег", event,
        "1. Прием денег#1. Прием денег#e2#!o2.x3");
    if (!o2_x3) {
        fireTransitionFound(context, path,
            "1. Прием денег", event,
            "1. Прием денег#1. Прием денег#e2#!o2.x3");
        fireComeToState(context, path,
            "1. Прием денег");

        // 1. Прием денег []
        return new StateMachineConfig(
            "1. Прием денег");
    }
    // 1. Прием денег->1. Прием денег
    // e2[o2.x3 && !o2.x2 && !o3.x1]/o1.z3
    fireTransitionCandidate(context, path,
        "1. Прием денег", event,
        "1. Прием денег#1. Прием денег#" +
            "e2#o2.x3 && !o2.x2 && !o3.x1");
    if (o2_x3 && !o2_x2 && !o3_x1) {
        fireTransitionFound(context, path,
            "1. Прием денег", event,
            "1. Прием денег#1. Прием денег#" +
```

```
                        "e2#o2.x3 && !o2.x2 && !o3.x1");
                fireBeforeOutputActionExecution(context,
                    path,
                    "1. Прием денег#1. Прием денег#" +
                        "e2#o2.x3 && !o2.x2 && !o3.x1",
                    "o1.z3");
                o1.z3(context);
                fireAfterOutputActionExecution(context,
                    path,
                    "1. Прием денег#1. Прием денег#" +
                        "e2#o2.x3 && !o2.x2 && !o3.x1",
                    "o1.z3");
                fireComeToState(context, path,
                    "1. Прием денег");

                // 1. Прием денег []
                return new StateMachineConfig(
                    "1. Прием денег");
            }

            // transition not found
            return config;
        default:

            // transition not found
            return config;
        }
    case _4__Выключение:
        fireTransitionsOfSuperstate(context, path,
            "4. Выключение", event);
        switch (e) {
        default:

            // transition not found
            return config;
        }
    case _3__Выдача_напитка:
        switch (e) {
        case e4:

            // 3. Выдача напитка->
            // 4. Выключение e4[o3.x2 || o4.x1]/
            fireTransitionCandidate(context, path,
                "3. Выдача напитка", event,
```

```
            "3. Выдача напитка#4. Выключение#" +
                "e4#o3.x2 || o4.x1");
        if (!isInputActionCalculated(
            calculatedInputActions, _o4_x1)) {
            fireBeforeInputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o4.x1");
            o4_x1 = o4.x1(context);
            fireAfterInputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o4.x1", new Boolean(o4_x1));
        }
        if (!isInputActionCalculated(
            calculatedInputActions, _o3_x2)) {
            fireBeforeInputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o3.x2");
            o3_x2 = o3.x2(context);
            fireAfterInputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o3.x2", new Boolean(o3_x2));
        }
        if (o3_x2 || o4_x1) {
            fireTransitionFound(context, path,
                "3. Выдача напитка", event,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1");
            fireComeToState(context, path,
                "4. Выключение");

            // 4. Выключение [o2.z4, o1.z10, o1.z9,
            // o3.z6, o4.z4, o5.z1]
            fireBeforeOutputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
```

```
    "o2.z4");
o2.z4(context);
fireAfterOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o2.z4");
fireBeforeOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o1.z10");
o1.z10(context);
fireAfterOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o1.z10");
fireBeforeOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o1.z9");
o1.z9(context);
fireAfterOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o1.z9");
fireBeforeOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o3.z6");
o3.z6(context);
fireAfterOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
    "o3.z6");
fireBeforeOutputActionExecution(context,
    path,
    "3. Выдача напитка#4. Выключение#" +
        "e4#o3.x2 || o4.x1",
```

```
                "o4.z4");
            o4.z4(context);
            fireAfterOutputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o4.z4");
            fireBeforeOutputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o5.z1");
            o5.z1(context);
            fireAfterOutputActionExecution(context,
                path,
                "3. Выдача напитка#4. Выключение#" +
                    "e4#o3.x2 || o4.x1",
                "o5.z1");
            return new StateMachineConfig(
                "4. Выключение");
        }
        // 3. Выдача напитка->
        // 1. Прием денег e4[!o3.x2 && !o4.x1]/
        fireTransitionCandidate(context, path,
            "3. Выдача напитка", event,
            "3. Выдача напитка#1. Прием денег#" +
                "e4#!o3.x2 && !o4.x1");
        if (!o3_x2 && !o4_x1) {
            fireTransitionFound(context, path,
                "3. Выдача напитка", event,
                "3. Выдача напитка#1. Прием денег#" +
                    "e4#!o3.x2 && !o4.x1");
            fireComeToState(context, path,
                "1. Прием денег");

            // 1. Прием денег []
            return new StateMachineConfig(
                "1. Прием денег");
        }

        // transition not found
        return config;
    default:
```

```
            // transition not found
            return config;
        }
    case _2__Приготовление_напитка:
        switch (e) {
        case e5:

            // 2. Приготовление напитка->
            // 3. Выдача напитка e5[!o3.x1]/
            fireTransitionCandidate(context, path,
                "2. Приготовление напитка", event,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#!o3.x1");
            if (!isInputActionCalculated(
                calculatedInputActions, _o3_x1)) {
                fireBeforeInputActionExecution(context,
                    path,
                    "2. Приготовление напитка#" +
                        "3. Выдача напитка#e5#!o3.x1",
                    "o3.x1");
                o3_x1 = o3.x1(context);
                fireAfterInputActionExecution(context,
                    path,
                    "2. Приготовление напитка#" +
                        "3. Выдача напитка#e5#!o3.x1",
                    "o3.x1", new Boolean(o3_x1));
            }
            if (!o3_x1) {
                fireTransitionFound(context, path,
                    "2. Приготовление напитка", event,
                    "2. Приготовление напитка#" +
                        "3. Выдача напитка#e5#!o3.x1");
                fireComeToState(context, path,
                    "3. Выдача напитка");

                // 3. Выдача напитка [o4.z1, o4.z3, o4.z2, o1.z6]
                fireBeforeOutputActionExecution(context,
                    path,
                    "2. Приготовление напитка#" +
                        "3. Выдача напитка#e5#!o3.x1",
                    "o4.z1");
                o4.z1(context);
                fireAfterOutputActionExecution(context,
                    path,
```

```
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o4.z1");
        fireBeforeOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o4.z3");
        o4.z3(context);
        fireAfterOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o4.z3");
        fireBeforeOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o4.z2");
        o4.z2(context);
        fireAfterOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o4.z2");
        fireBeforeOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o1.z6");
        o1.z6(context);
        fireAfterOutputActionExecution(context,
            path,
            "2. Приготовление напитка#" +
                "3. Выдача напитка#e5#!o3.x1",
            "o1.z6");
        return new StateMachineConfig(
            "3. Выдача напитка");
    }
    // 2. Приготовление напитка->
    // 3. Выдача напитка e5[o3.x1]/o1.z12
    fireTransitionCandidate(context, path,
        "2. Приготовление напитка", event,
        "2. Приготовление напитка#" +
```

```
                "3. Выдача напитка#e5#o3.x1");
        if (o3_x1) {
            fireTransitionFound(context, path,
                "2. Приготовление напитка", event,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1");
            fireBeforeOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o1.z12");
            o1.z12(context);
            fireAfterOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o1.z12");
            fireComeToState(context, path,
                "3. Выдача напитка");

            // 3. Выдача напитка [o4.z1, o4.z3, o4.z2, o1.z6]
            fireBeforeOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o4.z1");
            o4.z1(context);
            fireAfterOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o4.z1");
            fireBeforeOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o4.z3");
            o4.z3(context);
            fireAfterOutputActionExecution(context,
                path,
                "2. Приготовление напитка#" +
                    "3. Выдача напитка#e5#o3.x1",
                "o4.z3");
            fireBeforeOutputActionExecution(context,
```

```
                        path,
                        "2. Приготовление напитка#" +
                            "3. Выдача напитка#e5#o3.x1",
                        "o4.z2");
                    o4.z2(context);
                    fireAfterOutputActionExecution(context,
                        path,
                        "2. Приготовление напитка#" +
                            "3. Выдача напитка#e5#o3.x1",
                        "o4.z2");
                    fireBeforeOutputActionExecution(context,
                        path,
                        "2. Приготовление напитка#" +
                            "3. Выдача напитка#e5#o3.x1",
                        "o1.z6");
                    o1.z6(context);
                    fireAfterOutputActionExecution(context,
                        path,
                        "2. Приготовление напитка#" +
                            "3. Выдача напитка#e5#o3.x1",
                        "o1.z6");
                    return new StateMachineConfig(
                        "3. Выдача напитка");
                }

                // transition not found
                return config;
            default:

                // transition not found
                return config;
            }
        default:
            throw new EventProcessorException(
                "Incorrect stable state [" + config.getActiveState()
                    + "] in state machine [A2]");
        }
    }
}

// Коды входных переменных
//o2.x3
private static final int _o2_x3 = 0;
//o4.x1
```

```
        private static final int _o4_x1 = 1;
        //o3.x2
        private static final int _o3_x2 = 2;
        //o3.x1
        private static final int _o3_x1 = 3;
        //o2.x2
        private static final int _o2_x2 = 4;
    }

    // Служебный метод для выяснения того, вычислено ли уже значение
    // заданной входной переменной
    private static boolean isInputActionCalculated(
        BitSet calculatedInputActions, int k) {
        boolean b = calculatedInputActions.get(k);
        if (!b) {
            calculatedInputActions.set(k);
        }
        return b;
    }
}
```

Файл Test.java

```
import co.*;
import com.evelopers.unimod.log.Logger;
import com.evelopers.unimod.runtime.*;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import com.evelopers.unimod.runtime.logger.SimpleLogger;
import ep.ControlPane;

/**
 * Запускает приложение. Сначала создаются экземпляры объектов
 * управления, источников событий, затем инициализируются автоматы.
 */
public class Test {
    /**
     * Метод, вызываемый при запуске приложения.
     *
     * @param args аргументы командной строки.
     */
    public static void main(String[] args) {
        Test a = new Test();
        a.run();
```

```
}

/**
 * Когда главный автомат перейдет в финальное состояние,
 * данный класс сигнализирует об этом.
 */
private class ExitListener
        extends AbstractEventProcessorListener {
    private boolean done;

    public void stateMachineCameToFinalState(
            StateMachineContext context, StateMachinePath path,
            StateMachineConfig config) {
        if (path.isRoot()) done = true;
    }
}

/**
 * Создает экземпляры объектов управления, источников событий,
 * затем инициализирует автоматы.
 */
void run() {
    ModelEngine me;
    try {
        ControlledObjectsManager b
            = new ControlledObjectsManager() {
            // Инициализация объектов управления.
            Display o1 = new Display();
            CoinMechanism o2 = new CoinMechanism();
            DrinkMakingDevice o3 = new DrinkMakingDevice();
            PourOutDevice o4 = new PourOutDevice();
            ColdStore o5 = new ColdStore();

            public void init(ModelEngine engine) {
            }

            public void dispose() {
            }

            public ControlledObject getControlledObject(
                    String coName) {
                if (coName.equals("o1")) return o1;
                if (coName.equals("o2")) return o2;
                if (coName.equals("o3")) return o3;
```

```
                if (coName.equals("o4")) return o4;
                if (coName.equals("o5")) return o5;
                return null;
            }
        };
        me = ModelEngine.createStandAlone(new QueuedHandler(),
                new Model1EventProcessor(), b,
                new EventProvidersManager() {
                    // Инициализация источников событий.
                    ControlPane p1 = new ControlPane();

                    public void init(ModelEngine engine) {
                        p1.init(engine);
                    }

                    public void dispose() {
                        p1.dispose();
                    }

                    public EventProvider getEventProvider(
                        String epName) {
                        if (epName.equals("p1")) return p1;
                        return null;
                    }
                });
        me.getEventProcessor().addEventProcessorListener(
                new SimpleLogger(Logger.CONSOLE));
        ExitListener l1 = new ExitListener();
        me.getEventProcessor().addEventProcessorListener(l1);
        me.start();

        while (!l1.done) {
            Thread.sleep(10);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Приложение 3.

Исходный код на *Java* для третьего варианта реализации

Файл `CoinMechanism.java`

```
package co;

import co.ui.ControlPaneView;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Parameters;

/**
 * Хранит информацию о накопленной сумме, а также выполняет
 * проверку подлинности опущенных монет.
 */
public class CoinMechanism implements ControlledObject {
    private int prepayment;

    /**
     * @unimod.action.descr добавить номинал опущенной монеты к сумме
     */
    public void z1(StateMachineContext context) {
        int coinType = ((Integer) context.getEventContext()
            .getParameter(Parameters.COIN_TYPE)).intValue();
        context.getEventContext().setParameter(
                Parameters.PREPAYMENT, new Integer(prepayment));
        prepayment += coinType;
        context.getApplicationContext().setParameter(
                Parameters.PREPAYMENT, new Integer(prepayment));
        System.out.println(Parameters.getMessage(
            "Logger.CoinMechanism.z1", coinType));
    }

    /**
```

```
 * @unimod.action.descr выдать последнюю опущенную монету
 */
public void z2(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.CoinMechanism.z2"));
    ControlPaneView.getVm().drawCoins(1);
}

/**
 * @unimod.action.descr уменьшить сумму на заданную величину
 */
public void z3(StateMachineContext context) {
    int drinkPrice = Integer.parseInt(Parameters.getMessage(
        "ControlPaneView.Drinks.Price."
        + context.getEventContext().getParameter(
            Parameters.REQUESTED_DRINK)));
    context.getEventContext().setParameter(
            Parameters.PREPAYMENT, new Integer(prepayment));
    prepayment -= drinkPrice;
    context.getApplicationContext().setParameter(
            Parameters.PREPAYMENT, new Integer(prepayment));
    System.out.println(Parameters.getMessage(
        "Logger.CoinMechanism.z3", drinkPrice));
}

/**
 * @unimod.action.descr выдать сумму клиенту
 */
public void z4(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.CoinMechanism.z4"));
    int change = prepayment;
    int coins = (change / 5) + ((change % 5) / 2)
            + ((change % 5) % 2);
    context.getEventContext().setParameter(Parameters.PREPAYMENT,
            new Integer(prepayment));
    prepayment = 0;
    context.getApplicationContext().setParameter(
            Parameters.PREPAYMENT, new Integer(prepayment));
    ControlPaneView.getVm().drawCoins(coins);
}

/**
 * @unimod.action.descr проверка установила
```

```
 * подлинность опущенной монеты
 */
public boolean x1(StateMachineContext context) {
    int coinType = ((Integer) context.getEventContext()
        .getParameter(Parameters.COIN_TYPE)).intValue();
    if (coinType == Parameters.FALSE_COIN) {
        System.out.println(Parameters.getMessage(
            "Logger.CoinMechanism.x1.false"));
        return false;
    } else {
        System.out.println(Parameters.getMessage(
            "Logger.CoinMechanism.x1.true"));
        return true;
    }
}

/**
 * @unimod.action.descr сумма достаточна
 * для приобретения выбранного напитка
 */
public boolean x2(StateMachineContext context) {
    int drinkPrice = Integer.parseInt(Parameters.getMessage(
        "ControlPaneView.Drinks.Price."
        + context.getEventContext().getParameter(
            Parameters.REQUESTED_DRINK)));
    Integer iSum = (Integer) context.getEventContext()
        .getParameter(Parameters.PREPAYMENT);
    int sum;
    if (iSum == null) sum = prepayment;
    else sum = iSum.intValue();
    if (drinkPrice > sum) {
        System.out.println(Parameters.getMessage(
            "Logger.CoinMechanism.x2.false"));
        return false;
    } else {
        System.out.println(Parameters.getMessage(
            "Logger.CoinMechanism.x2.true"));
        return true;
    }
}

/**
 * @unimod.action.descr сумма ненулевая
 */
```

```
    public boolean x3(StateMachineContext context) {
        Integer iSum = (Integer) context.getEventContext()
            .getParameter(Parameters.PREPAYMENT);
        int sum;
        if (iSum == null) sum = prepayment;
        else sum = iSum.intValue();
        if (sum == 0) {
            System.out.println(Parameters.getMessage(
                "Logger.CoinMechanism.x3.false"));
            return false;
        } else {
            System.out.println(Parameters.getMessage(
                "Logger.CoinMechanism.x3.true"));
            return true;
        }
    }

    /**
     * @unimod.action.descr сумма уже достаточна для
     * приобретения любого напитка
     */
    public boolean x4(StateMachineContext context) {
        Integer iSum = (Integer) context.getEventContext()
            .getParameter(Parameters.PREPAYMENT);
        int sum;
        if (iSum == null) sum = prepayment;
        else sum = iSum.intValue();
        int maxPrice = 0;
        for (int j = 0; j < Parameters.NUM_DRINKS; j++) {
            maxPrice = Math.max(maxPrice,
                Integer.parseInt(Parameters.getMessage(
                    "ControlPaneView.Drinks.Price." + j)));
        }
        if (sum < maxPrice) {
            System.out.println(Parameters.getMessage(
                "Logger.CoinMechanism.x4.false"));
            return false;
        } else {
            System.out.println(Parameters.getMessage(
                "Logger.CoinMechanism.x4.true"));
            return true;
        }
    }
}
```

Файл ColdStore.java

```
package co;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Parameters;

/**
 * Управляет холодильной установкой.
 */
public class ColdStore implements ControlledObject {
    /**
     * @unimod.action.descr включить холодильную установку
     */
    public void z0(StateMachineContext context) {
        System.out.println(Parameters.getMessage(
            "Logger.ColdStore.z0"));
    }

    /**
     * @unimod.action.descr выключить холодильную установку
     */
    public void z1(StateMachineContext context) {
        System.out.println(Parameters.getMessage(
             "Logger.ColdStore.z1"));
    }
}
```

Файл Display.java

```
package co;

import co.ui.ControlPaneView;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Parameters;

import java.util.*;
import java.util.Timer;

/**
```

```
 * Отвечает за показ поступающих сообщений на дисплее.
 */
public class Display implements ControlledObject {
    private static String defaultMessage = Parameters.getMessage(
        "Message.BuyDrink");
    private Timer timer;

    private void showMessage(String msg) {
        int delay = Integer.parseInt(
            Parameters.getParameter("timer-delay"));
        if (timer != null) {
            timer.cancel();
        }
        ControlPaneView.getVm().drawMessage(msg);
        timer = new Timer(true);
        timer.schedule(new TimerTask() {
            public void run() {
                ControlPaneView.getVm().drawMessage(defaultMessage);
                ControlPaneView.getVm().drawCoins(0);
                ControlPaneView.getVm().drawCan(false);
            }
        }, delay);
    }

    /**
     * @unimod.action.descr включить дисплей
     */
    public void z1(StateMachineContext context) {
        System.out.println(Parameters.getMessage("Logger.Display.z0"));
    }

    /**
     * @unimod.action.descr информировать о поступлении
     * фальшивой монеты
     */
    public void z2(StateMachineContext context) {
        showMessage(Parameters.getMessage("Message.FalseCoin"));
        System.out.println(Parameters.getMessage("Logger.Display.z2"));
    }

    /**
     * @unimod.action.descr информировать о недостаточности
     * суммы для приобретения напитка
     */
```

```
public void z3(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.NeedMoreMoney"));
    System.out.println(Parameters.getMessage("Logger.Display.z3"));
}

/**
 * @unimod.action.descr вывести предложение забрать деньги
 */
public void z4(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.TakeMoney"));
    System.out.println(Parameters.getMessage("Logger.Display.z4"));
}

/**
 * @unimod.action.descr информировать о необходимости подождать
 */
public void z5(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.PleaseWait"));
    timer.cancel();
    System.out.println(Parameters.getMessage("Logger.Display.z5"));
}

/**
 * @unimod.action.descr вывести благодарность
 * за пользование автоматом
 */
public void z6(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.Thanks"));
    System.out.println(Parameters.getMessage("Logger.Display.z6"));
}

/**
 * @unimod.action.descr информировать об окончании
 * выбранного напитка
 */
public void z7(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.NoDrinkAvailable"));
    System.out.println(Parameters.getMessage("Logger.Display.z7"));
}

/**
 * @unimod.action.descr информировать о достаточности суммы
 * для выбора любого напитка
 */
```

```
public void z8(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.EnoughMoney"));
    System.out.println(Parameters.getMessage("Logger.Display.z8"));
}

/**
 * @unimod.action.descr информировать об окончании
 * компонентов для приготовления напитков
 */
public void z9(StateMachineContext context) {
    showMessage(Parameters.getMessage("Message.TurnOff"));
    System.out.println(Parameters.getMessage("Logger.Display.z9"));
}

/**
 * @unimod.action.descr показать накопленную сумму
 */
public void z10(StateMachineContext context) {
    int currPrepayment = ((Integer) context.getApplicationContext()
        .getParameter(Parameters.PREPAYMENT)).intValue();
    ControlPaneView.getVm().drawPrepayment(currPrepayment);
    System.out.println(
        Parameters.getMessage("Logger.Display.z10"));
}

/**
 * @unimod.action.descr зажечь встроенные индикаторы
 * кнопок выбора напитков
 */
public void z11(StateMachineContext context) {
    System.out.println(
        Parameters.getMessage("Logger.Display.z11"));
}

/**
 * @unimod.action.descr погасить встроенный индикатор
 * кнопки выбора напитка
 */
public void z12(StateMachineContext context) {
    int drink = ((Integer) context.getApplicationContext()
        .getParameter(Parameters.REQUESTED_DRINK)).intValue();
    ControlPaneView.getVm().drawTurnedOffIndicator(drink);
    System.out.println(Parameters.getMessage("Logger.Display.z12",
        Parameters.getMessage("ControlPaneView.Drinks." + drink)));
```

```
    }

    /**
     * @unimod.action.descr выключить дисплей
     */
    public void z13(StateMachineContext context) {
        System.out.println(
            Parameters.getMessage("Logger.Display.z13"));
    }
}
```

Файл DrinkMakingDevice.java

```
package co;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Parameters;
import co.ui.ControlPaneView;

import java.util.TimerTask;
import java.util.Timer;

/**
 * Обеспечивает приготовление заданного напитка. Сначала с
 * использованием дозатора порция напитка подается в смеситель,
 * затем сатуратор производит насыщение напитка двуокисью
 * углерода. Далее газированная вода посредством транспортирующего
 * устройства поступает в механизм выдачи напитка.
 */
public class DrinkMakingDevice implements ControlledObject {
    private int[] drinksQuantity;
    private double CO2Volume;

    /**
     * @unimod.action.descr подготовить устройство к работе
     */
    public void z0(StateMachineContext context) {
        int initialQuantity = Integer.parseInt(Parameters
            .getParameter("initial-drink-quantity"));
        drinksQuantity = new int[]{initialQuantity,
                                   initialQuantity, initialQuantity,
                                   initialQuantity, initialQuantity};
```

```
    CO2Volume = Double.parseDouble(Parameters.getParameter(
        "initial-co2-volume"));
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z0"));
}

/**
 * @unimod.action.descr подать 200 мл выбранного
 * напитка в смеситель
 */
public void z1(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z1"));
}

/**
 * @unimod.action.descr добавить в смеситель
 * двуокиси углерода из баллона
 */
public void z2(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z2"));
}

/**
 * @unimod.action.descr произвести насыщение
 * напитка двуокисью углерода
 */
public void z3(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z3"));
    final Integer drinkNumber = (Integer) context.getEventContext()
        .getParameter(Parameters.REQUESTED_DRINK);
    int delay = Integer.parseInt(Parameters.getParameter(
        "timer-delay"));
    (new Timer(true)).schedule(new TimerTask() {
        public void run() {
            ControlPaneView.getVm().fireEvent(ep.ControlPane.E5,
                Parameters.REQUESTED_DRINK, drinkNumber);
        }
    }, delay);
}

/**
```

```
 * @unimod.action.descr уменьшить счетчик
 * оставшихся порций напитка
 */
public void z4(StateMachineContext context) {
    int drinkNumber = ((Integer) context.getApplicationContext()
        .getParameter(Parameters.REQUESTED_DRINK)).intValue();
    drinksQuantity[drinkNumber]-;
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z4",
        Parameters.getMessage("ControlPaneView.Drinks."
            + drinkNumber)));
}

/**
 * @unimod.action.descr уменьшить показатель
 * концентрации двуокиси углерода
 */
public void z5(StateMachineContext context) {
    CO2Volume -= 1.6;
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z5"));
}

/**
 * @unimod.action.descr подготовить устройство к выключению
 */
public void z6(StateMachineContext context) {
    System.out.println(Parameters.getMessage(
        "Logger.DrinkMakingDevice.z6"));

    int delay = Integer.parseInt(
        Parameters.getParameter("timer-delay"));
    if (delay > 200) delay -= 200;
    Timer timer = new Timer(true);
    timer.schedule(new TimerTask() {
        public void run() {
            ControlPaneView.getVm().fireEvent(ep.ControlPane.E6);
        }
    }, delay);
}

/**
 * @unimod.action.descr закончился ли выбранный напиток
 */
```

```
public boolean x1(StateMachineContext context) {
    int f = ((Integer) context.getApplicationContext()
        .getParameter(Parameters.REQUESTED_DRINK)).intValue();
    if (drinksQuantity[f] > 0) {
        System.out.println(Parameters.getMessage(
            "Logger.DrinkMakingDevice.x1.false",
            Parameters.getMessage("ControlPaneView.Drinks." + f)));
        return false;
    } else {
        System.out.println(Parameters.getMessage(
            "Logger.DrinkMakingDevice.x1.true",
            Parameters.getMessage("ControlPaneView.Drinks." + f)));
        return true;
    }
}

/**
 * @unimod.action.descr закончились ли все напитки
 */
public boolean x2(StateMachineContext context) {
    int drinksAvail = 0;
    for (int i = 0; i < Parameters.NUM_DRINKS; i++) {
        drinksAvail += drinksQuantity[i];
    }
    if (drinksAvail == 0) {
        System.out.println(Parameters.getMessage(
            "Logger.DrinkMakingDevice.x2.true"));
        return true;
    } else {
        System.out.println(Parameters.getMessage(
            "Logger.DrinkMakingDevice.x2.false"));
        return false;
    }
}

/**
 * @unimod.action.descr закончилась ли двуокись углерода
 */
public boolean x3(StateMachineContext context) {
    if (CO2Volume <= 0) {
        System.out.println(Parameters.getMessage(
            "Logger.DrinkMakingDevice.x3.true"));
        return true;
    } else {
```

```
            System.out.println(Parameters.getMessage(
                "Logger.DrinkMakingDevice.x3.false"));
            return false;
        }
    }
}
```

Файл PourOutDevice.java

```
package co;

import co.ui.ControlPaneView;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Parameters;

/**
 * Осуществляет выдачу стаканчика одноразового использования
 * и разлив газированной воды.
 */
public class PourOutDevice implements ControlledObject {
    private int canCounter;

    /**
     * @unimod.action.descr подготовить устройство к работе
     */
    public void z0(StateMachineContext context) {
        canCounter = Integer.parseInt(Parameters.getParameter(
            "initial-amount-of-cans"));
        System.out.println(Parameters.getMessage(
            "Logger.PourOutDevice.z0"));
    }

    /**
     * @unimod.action.descr выдать одноразовый стаканчик
     */
    public void z1(StateMachineContext context) {
        System.out.println(Parameters.getMessage(
            "Logger.PourOutDevice.z1"));
        ControlPaneView.getVm().drawCan(true);
    }

    /**
```

```
     * @unimod.action.descr наполнить стаканчик содержимым смесителя
     */
    public void z2(StateMachineContext context) {
        System.out.println(Parameters.getMessage(
            "Logger.PourOutDevice.z2"));
        ControlPaneView.getVm().fireEvent(ep.ControlPane.E4);
    }

    /**
     * @unimod.action.descr уменьшить счетчик имеющихся стаканчиков
     */
    public void z3(StateMachineContext context) {
        canCounter-;
        System.out.println(Parameters.getMessage(
            "Logger.PourOutDevice.z3"));
    }

    /**
     * @unimod.action.descr подготовить устройство к выключению
     */
    public void z4(StateMachineContext context) {
        System.out.println(Parameters.getMessage(
            "Logger.PourOutDevice.z4"));
    }

    /**
     * @unimod.action.descr закончились ли стаканчики
     */
    public boolean x1(StateMachineContext context) {
        if (canCounter == 0) {
            System.out.println(Parameters.getMessage(
                "Logger.PourOutDevice.x1.true"));
            return true;
        } else {
            System.out.println(Parameters.getMessage(
                "Logger.PourOutDevice.x1.false"));
            return false;
        }
    }
}
```

Файл ControlPaneView.java

```
package co.ui;

import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.Parameter;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import common.Parameters;

import javax.swing.*;
import javax.swing.border.LineBorder;
import javax.swing.border.MatteBorder;
import javax.swing.text.html.HTMLEditorKit;
import java.awt.*;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.StringSelection;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.UnsupportedFlavorException;
import java.awt.dnd.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;

/**
 * Реализует управляющую панель торгового автомата с кнопками
 * и отвечает за взаимодействие с клиентом.
 */
public class ControlPaneView implements VendingMachine,
                                        ActionListener {
    private JFrame frame;
    private JLabel canPicture;
    private JLabel payBackPicture;
    private Image bgimage = null;
    private JRadioButton[] radio;
    private JLabel lblPrepayment;
    private JEditorPane display;
    private JButton moneyBack;

    private int returnedCoinsCount;
```

```
private static final Color NICE_DARK_GRAY = new Color(75, 75, 75);
private static final Color NICE_GRAY = new Color(204, 204, 204);
private static final Color NICE_WHITE = new Color(238, 238, 238);

private ModelEngine engine = null;
private static VendingMachine vm = new ControlPaneView();

/**
 * Создает новое окно приложения.
 */
private ControlPaneView() {
    frame = new JFrame();
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createUI();
        }
    });
}

/**
 * Возвращает панель, содержащую элементы управления.
 *
 * @return панель, содержащая элементы управления.
 */
public Container getContentPane() {
    return frame.getContentPane();
}

/**
 * Устанавливает "движок" модели, чтобы впоследствии использовать его
 * для постановки нового события в очередь.
 *
 * @param engine оповещается о происходящих событиях.
 */
public void setEngine(ModelEngine engine) {
    this.engine = engine;
}

/**
 * Возвращает текущую реализацию интерфейса торгового устройства.
 *
 * @return реализация интерфейса торгового устройства.
 */
```

```
public static VendingMachine getVm() {
    return vm;
}

/**
 * Ставит новое событие в очередь на обработку.
 *
 * @param event событие.
 */
public void fireEvent(String event) {
    engine.getEventManager().handle(
            new Event(event), StateMachineContextImpl.create());
}

/**
 * Ставит новое событие в очередь на обработку.
 *
 * @param event  событие.
 * @param pName  название параметра.
 * @param pValue значение параметра.
 */
public void fireEvent(String event, String pName, Object pValue) {
    StateMachineContext context = StateMachineContextImpl.create();
    context.getApplicationContext().setParameter(pName, pValue);
    engine.getEventManager().handle(new Event(event,
            new Parameter[]{new Parameter(pName, pValue)}),
            context);
}

/**
 * Показывает или скрывает окно приложения.
 *
 * @param a если истинно, показывает окно;
 *          иначе скрывает.
 */
public void setVisible(final boolean a) {
    if (Parameters.runningMode == Parameters.RUN_MODE_FRAME) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                frame.setVisible(a);
            }
        });
    }
}
```

```
/**
 * Освобождает системные ресурсы, занимаемые окном приложения.
 */
public void dispose() {
    frame.dispose();
}

/**
 * Создает пользовательский интерфейс приложения.
 */
private void createUI() {
    final int WIDTH = 430;
    final int HEIGHT = 640;
    final int COIN_HEIGHT = 70;
    final int COIN_WIDTH = 1 + WIDTH / Parameters.NUM_COINS;

    frame.setTitle(Parameters.getMessage(
        "ControlPaneView.Caption"));
    frame.setResizable(false);
    frame.setDefaultCloseOperation(
        WindowConstants.DO_NOTHING_ON_CLOSE);
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            fireEvent(ep.ControlPane.E6);
        }
    });
    frame.setSize(WIDTH, HEIGHT);
    Dimension ss = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((ss.width - WIDTH) » 1,
        (ss.height - HEIGHT) » 1);
    Container contentPane = frame.getContentPane();
    contentPane.setLayout(null);

    for (int i = 0; i < Parameters.NUM_COINS; i++) {
        ImageIcon picCoin = Parameters.getImageIcon(
            "coin-picture-" + i);
        JPanel coinDragPanel = new JPanel();
        coinDragPanel.setBounds(i * COIN_WIDTH, 545,
            COIN_WIDTH, COIN_HEIGHT);
        coinDragPanel.setBackground(ControlPaneView.NICE_WHITE);
        coinDragPanel.add((new MoneyDragProvider(
                Integer.parseInt(Parameters.getMessage(
                    "ControlPaneView.Coins.FaceValue." + i)),
```

```
            picCoin,
            Parameters.getMessage(
                "ControlPaneView.Coins.ToolTip." + i)))
        .getTransferable());
    contentPane.add(coinDragPanel);
}

bgimage = Parameters.getImageIcon(
    "background-picture").getImage();
JPanel content = new JPanel(null) {
    private static final long serialVersionUID
        = -193398762874121747lL;

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        if (bgimage != null) {
            int imwidth = bgimage.getWidth(null);
            int imheight = bgimage.getHeight(null);
            if ((imwidth > 0) && (imheight > 0)) {
                g.drawImage(bgimage, 0, 0, null);
            }
        }
    }
};

content.setBounds(0, 0, WIDTH, HEIGHT - COIN_HEIGHT);
contentPane.add(content);

final int POS_X = 216;
final int POS_Y = 78;
final int PANEL_WIDTH = 192;
final int PANEL_HEIGHT = 50;
final int DRINK_PIC_WIDTH = 100;

JComponent coinDropPanel = (new MoneyDropProvider())
    .createDropPanel();
coinDropPanel.setBounds(POS_X, POS_Y, 14, PANEL_HEIGHT);
content.add(coinDropPanel);

JLabel coinHinter = new JLabel(Parameters.getImageIcon(
    "drop-coin-here-hint"));
coinHinter.setToolTipText(Parameters.getMessage(
    "CoinHinter.ToolTipText"));
```

```
coinHinter.setBackground(NICE_WHITE);
coinHinter.setOpaque(true);
coinHinter.setBorder(
        new MatteBorder(2, 0, 2, 2, NICE_GRAY));
coinHinter.setBounds(POS_X + 14, POS_Y, 75, PANEL_HEIGHT);
content.add(coinHinter);

lblPrepayment = new JLabel(Parameters.getMessage(
    "Display.Sum", 0));
lblPrepayment.setHorizontalAlignment(SwingConstants.CENTER);
lblPrepayment.setBackground(NICE_WHITE);
lblPrepayment.setOpaque(true);
lblPrepayment.setBorder(
        new MatteBorder(2, 0, 2, 2, NICE_GRAY));
lblPrepayment.setBounds(POS_X + 89, POS_Y, 103, PANEL_HEIGHT);
content.add(lblPrepayment);

JLabel box = new JLabel();
box.setBorder(new MatteBorder(0, 2, 2, 2, NICE_GRAY));
box.setBounds(POS_X, POS_Y + PANEL_HEIGHT,
    PANEL_WIDTH, PANEL_HEIGHT);
content.add(box);

display = new JEditorPane();
display.setBackground(NICE_DARK_GRAY);
display.setEditable(false);
display.setEditorKit(new HTMLEditorKit());
display.setText(Parameters.getMessage("Display.Message",
    Parameters.getMessage("Message.BuyDrink")));
display.setMargin(new Insets(0, 2, 0, 0));
display.setBounds(POS_X + 2, POS_Y + PANEL_HEIGHT,
    PANEL_WIDTH - 4, PANEL_HEIGHT - 2);
content.add(display);

JLabel[] lblPrices = new JLabel[Parameters.NUM_DRINKS];
radio = new JRadioButton[Parameters.NUM_DRINKS];
for (int i = 0; i < Parameters.NUM_DRINKS; i++) {
    int curr_y = POS_Y + (PANEL_HEIGHT + 2) * i
        + 2 * PANEL_HEIGHT;
    JLabel drinkPic = new JLabel(Parameters.getImageIcon(
        "drink-picture-" + i));
    drinkPic.setToolTipText(Parameters.getMessage(
        "ControlPaneView.Drinks." + i));
    drinkPic.setBounds(POS_X, curr_y,
```

```
            DRINK_PIC_WIDTH, PANEL_HEIGHT);
        content.add(drinkPic);

        radio[i] = new JRadioButton(Parameters.getImageIcon(
            "indicator-initial"));
        radio[i].setActionCommand(Integer.toString(i));
        radio[i].setBackground(NICE_WHITE);
        radio[i].addActionListener(this);
        radio[i].setPressedIcon(Parameters.getImageIcon(
            "indicator-pressed"));
        radio[i].setMargin(new Insets(0, 10, 0, 0));
        radio[i].setBounds(POS_X + DRINK_PIC_WIDTH, curr_y,
            40, PANEL_HEIGHT);
        content.add(radio[i]);

        lblPrices[i] = new JLabel(Parameters.getMessage(
            "PriceLabel.Caption",
                Parameters.getMessage(
                    "ControlPaneView.Drinks.Price." + i)));
        lblPrices[i].setBounds(POS_X + DRINK_PIC_WIDTH + 40,
            curr_y, PANEL_WIDTH - DRINK_PIC_WIDTH - 40,
            PANEL_HEIGHT);
        lblPrices[i].setBackground(NICE_WHITE);
        lblPrices[i].setOpaque(true);
        content.add(lblPrices[i]);

        if (i < Parameters.NUM_DRINKS - 1) {
            JLabel splitter = new JLabel();
            splitter.setBackground(NICE_GRAY);
            splitter.setOpaque(true);
            splitter.setBounds(POS_X, curr_y + PANEL_HEIGHT,
                PANEL_WIDTH, 2);
            content.add(splitter);
        }
    }

    int curr_y = POS_Y + PANEL_HEIGHT
        * (2 + Parameters.NUM_DRINKS) + 8;
    canPicture = new JLabel(Parameters.getImageIcon(
        "drink-box-no-drink"));
    canPicture.setBounds(POS_X, curr_y, 50, 50);
    content.add(canPicture);

    payBackPicture = new JLabel(Parameters.getImageIcon(
```

```
            "money-back-0"));
        payBackPicture.setBounds(POS_X + 50, curr_y, 71, 50);
        content.add(payBackPicture);

        moneyBack = new JButton(Parameters.getMessage(
            "MoneyBackButton.Caption"));
        moneyBack.addActionListener(this);
        moneyBack.setBounds(POS_X + 121, curr_y, 71, 50);
        content.add(moneyBack);
    }

    /**
     * Вызывается для обработки нажатий кнопок и переключателей.
     *
     * @param e событие.
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == moneyBack) {
            fireEvent(ep.ControlPane.E3);
            return;
        }
        JRadioButton radioButton = ((JRadioButton) e.getSource());
        radioButton.setSelected(false);
        fireEvent(ep.ControlPane.E2, Parameters.REQUESTED_DRINK,
                new Integer(radioButton.getActionCommand()));
    }

    /**
     * Рисует монеты.
     *
     * @param coins число монет.
     */
    public void drawCoins(int coins) {
        int d = (coins > 6 ? 6 : coins);
        if (coins != 0) {
            d += returnedCoinsCount;
            if (d > 6) {
                d = 6;
            }
        }
        final int c = d;
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                returnedCoinsCount = c;
```

```
            payBackPicture.setIcon(Parameters.getImageIcon(
                "money-back-" + c));
        }
    });
}

/**
 * Выводит сообщение на дисплее.
 *
 * @param message сообщение, которое следует показать.
 */
public void drawMessage(final String message) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            display.setText(Parameters.getMessage(
                "Display.Message", message));
        }
    });
}

/**
 * Показывает или скрывает изображение стаканчика.
 *
 * @param showCan если истинно, показывает изображение стаканчика;
 *                иначе скрывает его.
 */
public void drawCan(final boolean showCan) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            if (showCan) {
                canPicture.setIcon(Parameters.getImageIcon(
                    "drink-box-drink"));
            } else {
                canPicture.setIcon(Parameters.getImageIcon(
                    "drink-box-no-drink"));
            }
        }
    });
}

/**
 * Показывает текущую сумму.
 *
 * @param prepayment сумма.
```

```
 */
public void drawPrepayment(final int prepayment) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            lblPrepayment.setText(Parameters.getMessage(
                "Display.Sum", prepayment));
        }
    });
}

/**
 * Рисует погашенный встроенный индикатор кнопки выбора напитка.
 *
 * @param indicatorID номер индикатора.
 */
public void drawTurnedOffIndicator(final int indicatorID) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            radio[indicatorID].setIcon(Parameters.getImageIcon(
                "indicator-off"));
        }
    });
}

/**
 * Изображение монетки, которое можно захватить
 * и перетащить с помощью мыши.
 */
private static class MoneyDragProvider implements
                                        DragGestureListener {
    private final JComponent transferable;
    private int value = 0;

    /**
     * Создает "перемещаемое" изображение монетки.
     *
     * @param v       номинал монеты.
     * @param image   картинка.
     * @param toolTip текст всплывающей подсказки.
     */
    public MoneyDragProvider(int v, ImageIcon image,
        String toolTip) {
        value = v;
        transferable = new JLabel(image);
```

```
        transferable.setToolTipText(toolTip);

        DragSource dragSource = DragSource.getDefaultDragSource();
        dragSource.createDefaultDragGestureRecognizer(
                transferable,
                DnDConstants.ACTION_COPY_OR_MOVE,
                this);
    }

    /**
     * Возвращает "перемещаемое" изображение монетки.
     *
     * @return "перемещаемое" изображение монетки.
     */
    public JComponent getTransferable() {
        return transferable;
    }

    /**
     * Распознает "жест": захват монеты.
     *
     * @param e событие.
     */
    public void dragGestureRecognized(DragGestureEvent e) {
        e.startDrag(Parameters.COIN_CURSOR,
                new StringSelection(Integer.toString(value)),
                null);
    }
}

/**
 * Панель монетоприемника.
 */
private class MoneyDropProvider extends DropTargetAdapter {
    /**
     * Создает панель, на которую может быть перетащен объект
     * с помощью механизма drag-and-drop.
     *
     * @return панель, на которую может быть перетащен объект.
     */
    public JPanel createDropPanel() {
        JPanel panel = new JPanel();
        panel.setToolTipText(Parameters.getMessage(
            "MoneyDropProvider.ToolTipText"));
```

```
            new DropTarget(panel,
                    DnDConstants.ACTION_COPY_OR_MOVE,
                    this);
            panel.setBackground(NICE_DARK_GRAY);
            panel.setBorder(new LineBorder(NICE_GRAY, 2));
            return panel;
        }

        /**
         * Вызывается для обработки события: объект перетащен
         * с помощью механизма drag-and-drop и опущен на панель.
         *
         * @param e событие.
         */
        public void drop(DropTargetDropEvent e) {
            try {
                DataFlavor stringFlavor = DataFlavor.stringFlavor;
                Transferable tr = e.getTransferable();
                if (e.isDataFlavorSupported(stringFlavor)) {
                    String data = (String) tr.getTransferData(
                        stringFlavor);
                    e.acceptDrop(DnDConstants.ACTION_COPY_OR_MOVE);
                    e.dropComplete(true);
                    fireEvent(ep.ControlPane.E1, Parameters.COIN_TYPE,
                            new Integer(data));
                } else {
                    e.rejectDrop();
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            } catch (UnsupportedFlavorException ufe) {
                ufe.printStackTrace();
            }
        }
    }
}
```

Файл VendingMachine.java

```
package co.ui;

import com.evelopers.unimod.runtime.ModelEngine;
```

```
import java.awt.*;

/**
 * Описывает методы устройства для продажи газированной воды.
 */
public interface VendingMachine {
    /**
     * Освобождает системные ресурсы, занимаемые окном приложения.
     */
    public void dispose();

    /**
     * Рисует монеты.
     *
     * @param coins число монет.
     */
    public void drawCoins(int coins);

    /**
     * Выводит сообщение на дисплее.
     *
     * @param message сообщение, которое следует показать.
     */
    public void drawMessage(final String message);

    /**
     * Показывает или скрывает изображение стаканчика.
     *
     * @param showCan если истинно, показывает изображение стаканчика;
     *                иначе скрывает его.
     */
    public void drawCan(final boolean showCan);

    /**
     * Показывает текущую сумму.
     *
     * @param prepayment сумма.
     */
    public void drawPrepayment(final int prepayment);

    /**
     * Рисует погашенный встроенный индикатор кнопки выбора напитка.
     *
     * @param indicatorID номер индикатора.
```

```
     */
    public void drawTurnedOffIndicator(final int indicatorID);

    /**
     * Ставит новое событие в очередь на обработку.
     *
     * @param event событие.
     */
    public void fireEvent(String event);

    /**
     * Ставит новое событие в очередь на обработку.
     *
     * @param event  событие.
     * @param pName  название параметра.
     * @param pValue значение параметра.
     */
    public void fireEvent(String event, String pName, Object pValue);

    /**
     * Возвращает панель, содержащую элементы управления.
     *
     * @return панель, содержащая элементы управления.
     */
    public Container getContentPane();

    /**
     * Устанавливает "движок" модели, чтобы впоследствии использовать его
     * для постановки нового события в очередь.
     *
     * @param engine оповещается о происходящих событиях.
     */
    public void setEngine(ModelEngine engine);

    /**
     * Показывает или скрывает окно приложения.
     *
     * @param a если истинно, показывает окно;
     *          иначе скрывает.
     */
    public void setVisible(final boolean a);
}
```

Файл Parameters.java

```
package common;

import javax.swing.*;
import java.awt.*;
import java.text.MessageFormat;
import java.util.*;

/**
 * Вспомогательный класс, содержащий параметры и средства для
 * чтения локализованных сообщений из properties-файлов.
 */
public class Parameters {
    private static ResourceBundle vmConfig = loadBundle("vm-config");
    private static ResourceBundle vmBundle = loadBundle(
        vmConfig.getString("locale"));
    private static Map pictureCache = new HashMap();

    /**
     * Курсор, который используется при перетаскивании монеты (drag-and-drop).
     */
    public final static Cursor COIN_CURSOR
            = Toolkit.getDefaultToolkit().createCustomCursor(
            Parameters.getImageIcon("coin-drag").getImage(),
            new Point(16, 16), "");

    /**
     * Число различных напитков.
     */
    public final static int NUM_DRINKS = 5;

    /**
     * Число типов монет.
     */
    public final static int NUM_COINS = 4;

    /**
     * Номинал фальшивой монеты.
     */
    public final static int FALSE_COIN = -1;

    /**
```

```
 * Обозначение параметра "выбранный напиток".
 */
public static final String REQUESTED_DRINK = "REQUESTED_DRINK";

/**
 * Обозначение параметра "тип опущенной монеты".
 */
public static final String COIN_TYPE = "COIN_TYPE";

/**
 * Обозначение параметра "текущая сумма".
 */
public static final String PREPAYMENT = "PREPAYMENT";

/**
 * Обозначение того, что приложение запущено
 * как самостоятельное приложение.
 */
public static final int RUN_MODE_FRAME = 0;

/**
 * Обозначение того, что приложение запущено как аплет.
 */
public static final int RUN_MODE_APPLET = 1;

/**
 * Определяет тип работы приложения: как аплет или как
 * самостоятельное приложение.
 */
public static int runningMode = RUN_MODE_FRAME;

private static ResourceBundle loadBundle(String name) {
    ResourceBundle bundle = null;
    try {
        bundle = ResourceBundle.getBundle(name);
    } catch (MissingResourceException e) {
        try {
            bundle = ResourceBundle.getBundle(name, Locale.US);
        } catch (MissingResourceException ex) {
            System.out.println("Configuration file \"" + name
                + ".properties\" not found");
            System.exit(1);
        }
    }
```

```
    return bundle;
}

/**
 * Получает локализованное сообщение.
 *
 * @param key ключ в *.properties-файле.
 * @return локализованное сообщение.
 */
public static String getMessage(String key) {
    return vmBundle.getString(key);
}

/**
 * Получает локализованное сообщение и форматирует его,
 * осуществляя подстановку целочисленного параметра.
 *
 * @param key ключ в *.properties-файле.
 * @param arg текст подстановки.
 * @return локализованное отформатированное сообщение.
 */
public static String getMessage(String key, int arg) {
    return MessageFormat.format(vmBundle.getString(key),
        new Object[]{new Integer(arg)});
}

/**
 * Получает локализованное сообщение и форматирует его,
 * осуществляя подстановку.
 *
 * @param key ключ в *.properties-файле.
 * @param arg текст подстановки.
 * @return локализованное отформатированное сообщение.
 */
public static String getMessage(String key, Object arg) {
    return MessageFormat.format(vmBundle.getString(key),
        new Object[]{arg});
}

/**
 * Получает параметр конфигурации.
 *
 * @param key ключ в *.properties-файле.
 * @return значение параметра.
```

```
     */
    public static String getParameter(String key) {
        return vmConfig.getString(key);
    }

    /**
     * Загружает рисунок.
     *
     * @param key ключ в *.properties-файле (для получения имени файла).
     * @return загруженный рисунок.
     */
    public static ImageIcon getImageIcon(String key) {
        ImageIcon img = (ImageIcon) pictureCache.get(key);
        if (img == null) {
            String imgFileName = getParameter(key);
            img = new ImageIcon(
                Parameters.class.getResource(imgFileName));
            pictureCache.put(key, img);
        }
        return img;
    }
}
```

Файл ControlPane.java

```
package ep;

import co.ui.ControlPaneView;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;

/**
 * Генерирует события для устройства по продаже газированной воды.
 */
public class ControlPane implements EventProvider {
    /**
     * @unimod.event.descr в монетоприемник поступила монета
     */
    public static final String E1 = "e1";
    /**
     * @unimod.event.descr нажата кнопка выбора напитка
     */
    public static final String E2 = "e2";
```

```
    /**
     * @unimod.event.descr нажата кнопка возврата денег
     */
    public static final String E3 = "e3";
    /**
     * @unimod.event.descr напиток выдан
     */
    public static final String E4 = "e4";
    /**
     * @unimod.event.descr напиток приготовлен
     */
    public static final String E5 = "e5";
    /**
     * @unimod.event.descr выключение автомата
     */
    public static final String E6 = "e6";

    /**
     * Вызывается при запуске приложения.
     *
     * @param engine оповещается о происходящих событиях.
     */
    public void init(ModelEngine engine) {
        ControlPaneView.getVm().setEngine(engine);
        ControlPaneView.getVm().setVisible(true);
    }

    /**
     * Вызывается при завершении работы программы.
     */
    public void dispose() {
        ControlPaneView.getVm().setVisible(false);
        ControlPaneView.getVm().dispose();
    }
}
```

Приложение 4.

Пример протокола работы программы

Протокол приводится для следующей ситуации:

- сначала устройство для продажи газированной воды включается;
- затем клиент опускает две пятирублевые монеты, выбирает напиток «Coca-Cola» стоимостью семь рублей и забирает сдачу.

```
20:55:13,890 INFO  [Run] Start event [e1] processing. In state [/A1:Top]
20:55:13,906 INFO  [Run] Transition to go found
                         [0. Начальное состояние#1. Готов к обслуживанию##true]
20:55:13,906 INFO  [Run] Start output action [o1.z1] execution
Дисплей включен
20:55:13,906 INFO  [Run] Finish output action [o1.z1] execution
20:55:13,906 INFO  [Run] Start output action [o1.z11] execution
Встроенные индикаторы кнопок для выбора напитков зажжены
20:55:13,906 INFO  [Run] Finish output action [o1.z11] execution
20:55:13,906 DEBUG [Run] Try transition
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#!o2.x1]
20:55:14,062 INFO  [Run] Start input action [o2.x1] calculation
Проверка установила подлинность монеты
20:55:14,062 INFO  [Run] Finish input action [o2.x1] calculation. Its value is [true]
20:55:14,062 DEBUG [Run] Try transition
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#o2.x1 && !o2.x4]
20:55:14,062 INFO  [Run] Start input action [o2.x4] calculation
Сумма недостаточна для приобретения любого напитка
20:55:14,062 INFO  [Run] Finish input action [o2.x4] calculation. Its value is [false]
20:55:14,062 INFO  [Run] Transition to go found
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#o2.x1 && !o2.x4]
20:55:14,062 INFO  [Run] Start output action [o2.z1] execution
Сумма увеличена на 5
20:55:14,062 INFO  [Run] Finish output action [o2.z1] execution
20:55:14,062 INFO  [Run] Start output action [o1.z10] execution
Величина накопленной суммы показана на дисплее
20:55:14,062 INFO  [Run] Finish output action [o1.z10] execution
20:55:14,062 INFO  [Run] Start event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:Top]
20:55:14,062 INFO  [Run] Transition to go found
                   [0. Начальное состояние#1. Прием денег##true]
20:55:14,062 INFO  [Run] Start output action [o5.z0] execution
Холодильная установка включена
20:55:14,078 INFO  [Run] Finish output action [o5.z0] execution
20:55:14,078 INFO  [Run] Start output action [o4.z0] execution
```

```
Запуск механизма выдачи напитка
20:55:14,078 INFO  [Run] Finish output action [o4.z0] execution
20:55:14,078 INFO  [Run] Start output action [o3.z0] execution
Запуск устройства приготовления напитка
20:55:14,078 INFO  [Run] Finish output action [o3.z0] execution
20:55:14,078 INFO  [Run] Finish event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:14,078 INFO  [Run] Finish event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:15,921 INFO  [Run] Start event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:15,921 DEBUG [Run] Try transition
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#!o2.x1]
20:55:15,921 INFO  [Run] Start input action [o2.x1] calculation
Проверка установила подлинность монеты
20:55:15,921 INFO  [Run] Finish input action [o2.x1] calculation. Its value is [true]
20:55:15,921 DEBUG [Run] Try transition
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#o2.x1 && !o2.x4]
20:55:15,921 INFO  [Run] Start input action [o2.x4] calculation
Сумма недостаточна для приобретения любого напитка
20:55:15,937 INFO  [Run] Finish input action [o2.x4] calculation. Its value is [false]
20:55:15,937 INFO  [Run] Transition to go found
                   [1. Готов к обслуживанию#1. Готов к обслуживанию#e1#o2.x1 && !o2.x4]
20:55:15,953 INFO  [Run] Start output action [o2.z1] execution
Сумма увеличена на 5
20:55:15,953 INFO  [Run] Finish output action [o2.z1] execution
20:55:15,953 INFO  [Run] Start output action [o1.z10] execution
Величина накопленной суммы показана на дисплее
20:55:15,953 INFO  [Run] Finish output action [o1.z10] execution
20:55:15,953 INFO  [Run] Start event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:15,953 INFO  [Run] Finish event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:15,953 INFO  [Run] Finish event [e1] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:17,281 INFO  [Run] Start event [e2] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:17,281 INFO  [Run] Start event [e2] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:17,281 DEBUG [Run] Try transition
                   [1. Прием денег#2. Приготовление напитка#e2#o2.x3 && o2.x2 && !o3.x1]
20:55:17,296 INFO  [Run] Start input action [o2.x3] calculation
Сумма не равна 0
20:55:17,296 INFO  [Run] Finish input action [o2.x3] calculation. Its value is [true]
20:55:17,296 INFO  [Run] Start input action [o2.x2] calculation
Сумма достаточна для приобретения выбранного напитка
20:55:17,296 INFO  [Run] Finish input action [o2.x2] calculation. Its value is [true]
20:55:17,296 INFO  [Run] Start input action [o3.x1] calculation
Напиток "Coca-Cola" не закончился
20:55:17,296 INFO  [Run] Finish input action [o3.x1] calculation. Its value is [false]
20:55:17,296 INFO  [Run] Transition to go found
                   [1. Прием денег#2. Приготовление напитка#e2#o2.x3 && o2.x2 && !o3.x1]
20:55:17,296 INFO  [Run] Start on-enter action [o1.z5] execution
```

```
Сообщение о необходимости подождать показано на дисплее
20:55:17,328 INFO  [Run] Finish on-enter action [o1.z5] execution
20:55:17,328 INFO  [Run] Start on-enter action [o2.z3] execution
Сумма уменьшена на 7
20:55:17,328 INFO  [Run] Finish on-enter action [o2.z3] execution
20:55:17,328 INFO  [Run] Start on-enter action [o1.z10] execution
Величина накопленной суммы показана на дисплее
20:55:17,328 INFO  [Run] Finish on-enter action [o1.z10] execution
20:55:17,328 INFO  [Run] Start on-enter action [o3.z1] execution
200 мл выбранного напитка подано в смеситель
20:55:17,328 INFO  [Run] Finish on-enter action [o3.z1] execution
20:55:17,328 INFO  [Run] Start on-enter action [o3.z4] execution
Уменьшено число оставшихся порций напитка "Coca-Cola"
20:55:17,328 INFO  [Run] Finish on-enter action [o3.z4] execution
20:55:17,328 INFO  [Run] Start on-enter action [o3.z2] execution
Двуокись углерода из баллона добавлена в смеситель
20:55:17,328 INFO  [Run] Finish on-enter action [o3.z2] execution
20:55:17,328 INFO  [Run] Start on-enter action [o3.z5] execution
Показатель концентрации двуокиси углерода в баллоне стал меньше
20:55:17,328 INFO  [Run] Finish on-enter action [o3.z5] execution
20:55:17,328 INFO  [Run] Start on-enter action [o3.z3] execution
Произведено насыщение напитка двуокисью углерода
20:55:17,328 INFO  [Run] Finish on-enter action [o3.z3] execution
20:55:17,328 INFO  [Run] Finish event [e2] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:2. Приготовление напитка]
20:55:17,328 INFO  [Run] Finish event [e2] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:19,328 INFO  [Run] Start event [e5] processing.
                   In state [/A1:1. Готов к обслуживанию]
20:55:19,328 INFO  [Run] Start event [e5] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:2. Приготовление напитка]
20:55:19,328 DEBUG [Run] Try transition
                   [2. Приготовление напитка#3. Выдача напитка#e5#!o3.x1]
20:55:19,328 INFO  [Run] Start input action [o3.x1] calculation
Напиток "Coca-Cola" не закончился
20:55:19,328 INFO  [Run] Finish input action [o3.x1] calculation. Its value is [false]
20:55:19,328 INFO  [Run] Transition to go found
                   [2. Приготовление напитка#3. Выдача напитка#e5#!o3.x1]
20:55:19,328 INFO  [Run] Start on-enter action [o4.z1] execution
Выдан новый одноразовый стаканчик
20:55:19,359 INFO  [Run] Finish on-enter action [o4.z1] execution
20:55:19,359 INFO  [Run] Start on-enter action [o4.z3] execution
Уменьшен счетчик имеющихся стаканчиков
20:55:19,359 INFO  [Run] Finish on-enter action [o4.z3] execution
20:55:19,359 INFO  [Run] Start on-enter action [o4.z2] execution
Стаканчик наполнен содержимым смесителя
20:55:19,359 INFO  [Run] Finish on-enter action [o4.z2] execution
20:55:19,359 INFO  [Run] Start on-enter action [o1.z6] execution
Благодарность за пользование автоматом показана на дисплее
20:55:19,359 INFO  [Run] Finish on-enter action [o1.z6] execution
20:55:19,359 INFO  [Run] Finish event [e5] processing.
                   In state [/A1:1. Готов к обслуживанию/A2:3. Выдача напитка]
20:55:19,359 INFO  [Run] Finish event [e5] processing.
```

```
                    In state [/A1:1. Готов к обслуживанию]
20:55:19,359 INFO   [Run] Start event [e4] processing.
                    In state [/A1:1. Готов к обслуживанию]
20:55:19,359 INFO   [Run] Start event [e4] processing.
                    In state [/A1:1. Готов к обслуживанию/A2:3. Выдача напитка]
20:55:19,359 DEBUG [Run] Try transition
                    [3. Выдача напитка#4. Выключение#e4#o3.x2 || o4.x1]
20:55:19,359 INFO   [Run] Start input action [o3.x2] calculation
Все напитки еще не закончились
20:55:19,359 INFO   [Run] Finish input action [o3.x2] calculation. Its value is [false]
20:55:19,359 INFO   [Run] Start input action [o4.x1] calculation
Стаканчики не закончились
20:55:19,359 INFO   [Run] Finish input action [o4.x1] calculation. Its value is [false]
20:55:19,359 DEBUG [Run] Try transition
                    [3. Выдача напитка#1. Прием денег#e4#!o3.x2 && !o4.x1]
20:55:19,359 INFO   [Run] Transition to go found
                    [3. Выдача напитка#1. Прием денег#e4#!o3.x2 && !o4.x1]
20:55:19,359 INFO   [Run] Finish event [e4] processing.
                    In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:19,359 INFO   [Run] Finish event [e4] processing.
                    In state [/A1:1. Готов к обслуживанию]
20:55:22,578 INFO   [Run] Start event [e3] processing.
                    In state [/A1:1. Готов к обслуживанию]
20:55:22,578 DEBUG [Run] Try transition
                    [1. Готов к обслуживанию#1. Готов к обслуживанию#e3#!o2.x3]
20:55:22,578 INFO   [Run] Start input action [o2.x3] calculation
Сумма не равна 0
20:55:22,578 INFO   [Run] Finish input action [o2.x3] calculation. Its value is [true]
20:55:22,578 DEBUG [Run] Try transition
                    [1. Готов к обслуживанию#1. Готов к обслуживанию#e3#o2.x3]
20:55:22,578 INFO   [Run] Transition to go found
                    [1. Готов к обслуживанию#1. Готов к обслуживанию#e3#o2.x3]
20:55:22,578 INFO   [Run] Start output action [o2.z4] execution
Сумма выдана клиенту
20:55:22,578 INFO   [Run] Finish output action [o2.z4] execution
20:55:22,578 INFO   [Run] Start output action [o1.z10] execution
Величина накопленной суммы показана на дисплее
20:55:22,578 INFO   [Run] Finish output action [o1.z10] execution
20:55:22,578 INFO   [Run] Start output action [o1.z4] execution
Предложение забрать деньги показано на дисплее
20:55:22,578 INFO   [Run] Finish output action [o1.z4] execution
20:55:22,578 INFO   [Run] Start event [e3] processing.
                    In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:22,609 INFO   [Run] Finish event [e3] processing.
                    In state [/A1:1. Готов к обслуживанию/A2:1. Прием денег]
20:55:22,609 INFO   [Run] Finish event [e3] processing.
                    In state [/A1:1. Готов к обслуживанию]
```

Приложение 5.

Установка и запуск приложения

На сайте http://is.ifmo.ru в разделе «Unimod-проекты» размещены проектная документация, *Java*-аплет и исходные тексты на языке *Java* модели устройства для продажи газированной воды.

Аплет можно запустить прямо в браузере при наличии виртуальной машины *Java*. С сайта можно загрузить версию приложения, выполненную в рамках компиляционного подхода. Для запуска данного приложения также требуется только виртуальная машина *Java*, которую можно скачать с сайта *Sun Microsystems* (http://java.sun.com/j2se/1.5.0/download.jsp).

Исходные тексты программы предоставляются в двух вариантах:

- в виде файлов приложения, построенного по компиляционному подходу;
- в виде проекта для *Eclipse*.

Оба варианта содержат написанные вручную классы для объектов управления и источников событий. Отличие между двумя версиями исходных текстов состоит в том, что в одном случае в поставку включены *UML*-диаграммы и некоторые другие файлы для просмотра проекта в среде *Eclipse*, а в другом — вместо диаграмм включен автоматически сгенерированный класс на языке *Java*.

Для работы с файлами проекта в среде *Eclipse*, сначала необходимо установить данную среду разработки. Загрузить последнюю версию *Eclipse SDK* можно по ссылке http://www.eclipse.org/downloads. Кроме того, необходимы следующие плагины:

- *GEF* (http://download.eclipse.org/tools/gef/downloads);
- *UniMod* (http://unimod.sourceforge.net/download.html).

Краткие инструкции по интеграции этих плагинов в *Eclipse* приведены на странице http://unimod.sourceforge.net/eclipse-plugin.html. Рассматриваемый проект разрабатывался в среде *Eclipse* версии 3.1 с плагином *UniMod* версии 1.3.1.35 (релиз от 10.01.2006).

Для того чтобы открыть проект в среде разработки *Eclipse*, следует выбрать пункт меню «File/Import...» («Файл/Импорт...»), а в открывшемся диалоге мастера — «Existing Project into Workspace» («Существующий проект в рабочую область») из списка возможных действий и нажать кнопку «Next >» («Далее >»). В окне выбора папки с файлами проекта необходимо указать ту папку, в которую

был распакован скачанный с сайта http://is.ifmo.ru архив с исходными текстами. Теперь следует нажать кнопку «Finish» («Завершить импорт») и подождать несколько секунд, в течение которых *Eclipse* в фоновом режиме скомпилирует *Java*-классы.

Запустить приложение можно, например, так: открыть файл с диаграммами (`resources/diagrams/vending-machine.unimod`), щелкнуть правой кнопкой мыши на графе переходов автомата A1 и выбрать пункт контекстного меню «Launch A1!» («Запустить A1!»).