Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет “Информационных технологий и программирования"

В.А. Козлов, О.А. Комалёва

Моделирование работы банкомата

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

Санкт-Петербург
2006

Оглавление

Введение

В настоящее время широкое распространение получило программирование различных устройств, используемых в автомобилях, стиральных машинах, фотоаппаратах и т.д. Такие устройства используются очень часто, и поэтому их применение должно быть максимально безопасным. При этом программы, «зашитые» в эти устройства, должны работать корректно.

Одним из способов решения этой задачи является доказательство корректности работы программ. Это очень трудоемкая задача математической логики, и существующие сейчас приложения умеют доказывать корректность только для определенного класса программ. При этом существуют такие свойства программ, выполнение которых вообще нельзя доказать.

Другой способ решения этой проблемы – писать программы, корректные, по крайней мере, с инженерной точки зрения. Это достигается, если для каждой спецификации хотя бы поведение программы не реализовывать вручную. Поэтому, если по спецификации поведение программ реализуютеся формально, то так как все программы генерируются при помощи одного и того же преобразователя, то он многократно тестируется. Это приводит к повышению качества его программной реализации. Такая ситуация, например, имеет место для компиляторов широко известных языков программирования.

В данной работе предлагается формально реализовывать ядро программы по спецификации, состоящей из схемы связей и автоматов, входящих в эту схему. При этом разработчик основную часть времени занимается проектированием программного обеспечения – создает взаимосвязанную систему конечных автоматов, а работа по выполнению или реализации этой части программы выполняется самой средой разработки.

В проекте на основе автоматного подхода реализовано программное обеспечение для моделирования работы банкомата в среде разработки *Eclipse 3.1* с использованием инструментального средства *Unimod 1.3.1.35* [1, 2] и языка *Java 1.5*.

При этом отметим, что другой вариант автоматной реализации банкомата приведен в работе [3].

1. Постановка задачи

Банкомат – это устройство, автоматизирующее операции по выдаче и переводу денег, хранящихся в банке, лицу, которому они принадлежат. Идентификация каждого клиента происходит с помощью имеющейся у него кредитной карты банка и соответствующего карте секретного *pin*-кода. Поэтому человек может вне банка снимать деньги или оплачивать определенные услуги. Так как банк заинтересован в том, чтобы человек хранил в нем свои деньги, то он должен обеспечивать удобство работы своим клиентам. При этом, если банкомат будет не надежным, то страдать будет как банк, так и его клиенты, а это совершенно неприемлемая ситуация.

Теперь, можно сформулировать основные требования к устройству банкомата. Он должен:

- идентифицировать клиента;
- выполнять операции «Показать доступные средства» и «Снять определенную сумму денег»;
- уметь связываться с банком.

Клиентская программа запускается и предлагает **пользователю** выполнять различные операции с его личной картой.

Первое, что требуется от **пользователя** – вставить карту. В этом банкомате этот процесс эмулируется вводом номера карты. В реальном банкомате номер карты считывается с неё автоматически.

Далее **пользователь** вводит свой личный *pin*-код. Если на сервере не найдется запись о счете, с введенным номером и *pin*-кодом, то работа с этой картой прекращается. Если же *pin*-код и номер счета был введён правильно, то **пользователю** предлагается выполнить одну из следующих операций:

- «Забрать карту» – возврат карты. Все текущие операции отменяются, а карта возвращается на руки **пользователю**;
- «Баланс» – отображает текущий остаток на счете, выводя его на экран и предоставляя возможность распечатать на чеке;
- «Снятие денег» – производит операцию снятия денег с карты. Для этого **пользователь** должен ввести сумму, которую он хочет снять. Клиент пошлёт запрос на сервер о текущем балансе, и получит ответ. Если на карте есть достаточно денег, то операция на сервере завершится успешно, и банкомат выдаст требуемую сумму денег. Так же при нажатии на кнопку "Печать" будет напечатан чек по данной операции. Если обнаружится, что на карте недостаточно денег, то с карточки ничего не снимется, и клиент выводит соответствующее сообщение на экран.

После возврата карты, **пользователь** может вставить её опять либо уйти, нажав кнопку «Выход».

В данной реализации вся информация о счетах хранится в файле Bank.dat, в котором сначала записана информация о количестве счетов, а потом перечисляются блоками по три строки счета в формате: номер счета, *pin*-код и сумма на счете. Ниже приведен пример этого файла:

```
2    - количество счетов;
1234 - номер счета;
5678 - pin-код;
497  - сумма на счете;
1111 - номер счета;
8765 - pin-код;
10   - сумма на счете.
```

Также как в реальном банкомате операции общения с сервером обеспечиваются по сети. Подавтомат *AServer* удаленно вызывает нужные методы сервера.

При возникновении события «Ошибка при работе с сервером» клиент осуществляет возврат карты и переходит в начальное состояние

На рис.1, 2 приведен внешний вид модели банкомата в некоторых диалогах.

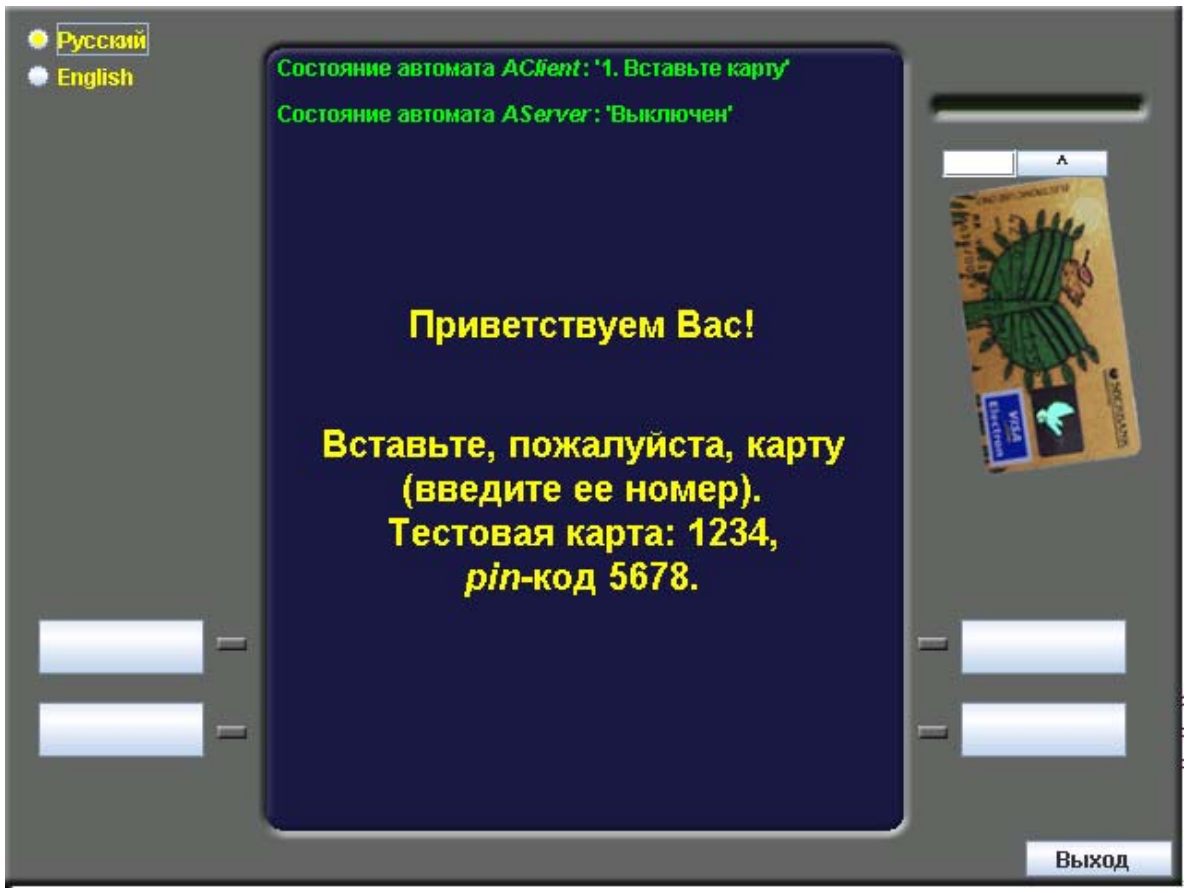

Рис. 1. Внешний вид банкомата. Начальный диалог

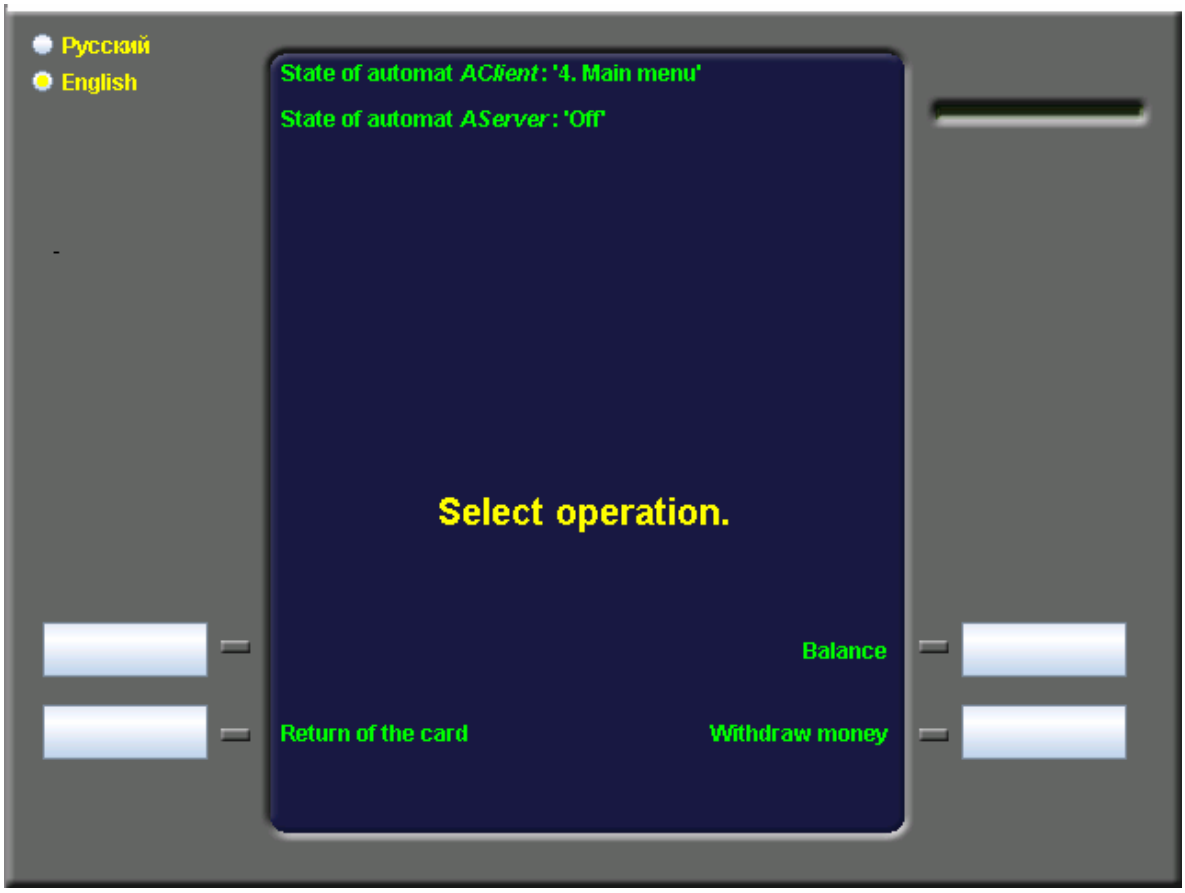

Рис. 2. Внешний вид банкомата. Диалог выбора операции над картой

В Приложении 1 приведены пример протокола работы программы.

2. Сценарии

2.1. Неформальный текст сценария

Рассмотрим банкомат, состоящий из дисплея, четырех кнопок для команд, кнопки выход и отверстия для вставления карточки.

Пользователь вводит номер карты и вставляет ее. После этого он вводит *pin*-код карты. Если *pin*-код не верен, то карта возвращается. В противном случае пользователь может выбрать одну из трех операций: просмотр баланса, снятие денег, возврат карты.

При запросе баланса, он выводится на экран. При этом можно выбрать команду «Печать» и напечатается чек с балансом. После этого можно забрать карту.

При выполнении операции снятия денег, пользователь вводит сумму, которую он собирается снять со счёта. Этот запрос посылается на сервер. При наличии требуемой суммы на счёте, банкомат выдаст деньги и предложит распечатать чек. После этого банкомат «отдаст» карту.

Кнопку «Выход» можно нажать только при вынутой карте.

2.2. Формализация сценария

Перейдем к формализации сценария. Первоначально введем элементы управления, а также выделим входные переменные, события и выходные действия.

Для управления операциями введем следующие кнопки:

- кнопка *«Выход»* – завершение работы с банкоматом;
- четыре **кнопки управления** по краям лицевой панели банкомата.

Кроме того, на панели могут появляться кнопки эмуляции действий:

- кнопка *«Вставить карту»* – эмулирует ввод карты пользователем;
- кнопка *«Забрать карту»* – эмулирует изъятие карты;
- кнопка *«Взять чек»* – эмулирует получение чека пользователем;
- кнопка *«Взять деньги»* – эмулирует получение денег из банкомата.

Выделим состояния, каждому из которых начначим номер, соответствующий вершине графа переходов. Переходы обозначим парой чисел *(i, j)*, где *i* – номер состояния, из которого осуществляется переход, а *j* – состояние, в которое осуществляется переход. Введем также обозначение событий *e*, инициирующих переходы, и обозначения выходных действий *z*. При этом статические изображения формируются в вершинах, а системные действия — на переходах.

Каждому состоянию с номером *n* соответствует действие *zn*, которое визуализирует его.

Вывод текста сообщения на экран дисплея назовем **сообщением**. Сообщение, на которое пользователь должен совершить какие-то действия, назовем ***просьбой***.

Запись *(cmd1, cmd2, cmd3, cmd3)* обозначает метки, описывающие функциональность четырех кнопок управления. Могут быть задействованы не все кнопки управления сразу. Кнопки, которые были задействованы, генерируют события *e1-e4* соответственно.

Функциональность кнопки действительна только в текущем состоянии. Это означает, что в каждом состоянии реакции на одни и те же события различны.

2.3. Формальный текст сценария

1. Вставьте карту. (*z1*) – сообщение приветствия и просьба ввести номер карты в текстовое поле.
Показать кнопку *«Вставить карту»*.
(*1, halt*) Нажали кнопку – событие *e0*. Клиент завершает свою работу.
(*1, 2*) Вставили карту (*e6*).

2. Ввод *pin*-кода. (*z2*) – просьба ввести *pin*-код в текстовом поле.
Кнопки управления: (*–, Отмена, –, Ok*).
(*2, 13*) Нажали кнопку «*Отмена»* (*e2*).
(*2, 3*) Нажали кнопку «*Ok»* (*e4*).

3. Авторизация. (*z3*) – сообщения о том, что клиент авторизируется на сервере.
(*3, 4*) Сервер принял *pin*-код (*e10*).
(3, 13) Сервер не принял *pin*-код (*e11*).

4. Главное меню. (*z4*) – просьба выбрать операцию с карты (баланс, снятие денег, возврат)
Кнопки управления: (*–, Отмена, Баланс, Снятие денег*).
(*4, 13*) Нажали кнопку «*Отмена»* (*e2*).
(*4, 5*) Нажали кнопку «*Баланс»* (*e3*).
(*4, 8*) Нажали кнопку «*Снятие денег»* (*e4*).

5. Запрос баланса. (*z5*) – сообщение о том, что клиент запрашивает баланс счета с сервера.
(*5, 6*) Баланс получен (*e12*).
(*5, 13*) Ошибка при получении баланса (*e15*).

6. Вывод баланса на экран. (*z6*) – сообщение о текущем балансе.
Кнопки управления: (*–, –, Печать, Ok*).
(*6, 7*) Нажали кнопку «*Печать»* (*e3*).
(*6, 13*) Нажали кнопку «*Ok»* (*e4*).

7. Печать чека 1. (*z7*) – печать чека с текущим балансом и просьба забрать чек.
Показать кнопку «*Взять чек»*.
(*7, 13*) Чек взят пользователем (*e8*).

8. Ввод суммы. (z8) – просьба ввести сумму, которая снимается.
Кнопки управления: (*–, Отмена, –, Ok*).
(*8, 13*) Нажали кнопку *«Отмена»* (*e2*).
(*8, 9*) Нажали кнопку *«Ok»* (*e4*).

9. Запрос денег. (*z9*) – сообщение о том, что клиент снимает деньги с карточки на сервере.
(*9, 10*) Снятие денег прошло удачно на сервере (*e13*).
(*9, 13*) Нет запрашиваемой суммы на карточке (*e14*).

10. Выдача денег. (*z10*) – выдача снятой суммы денег и просьба взять деньги.
Показать кнопку *«Взять деньги»*.
(*10, 11*) Деньги взяты пользователем (*e9*).

11. Печатать чек?. (*z11*) – просьба решить следует ли печатать чек.
Кнопки управления: (*–, Отмена, Печать, –*).
(*11, 13*) Нажали кнопку *«Отмена»* (*e2*).
(*11, 12*) Нажали кнопку *«Печать»* (*e3*).

12. Печать чека 2. (*z12*) – печать чека о снятии денег и просьба забрать чек.
    Показать кнопку *«Взять чек»*.
(*12, 13*) Чек взят пользователем (*e8*).

13. Возврат карты. (*z13*) – выдвинуть карту на половину и просьба забрать её.
    Показать кнопку *«Возврат карты»*.
(*13, 1*) Карта вытащена пользователем (*e7*).

3. Проектирование

Данный проект состоит из двух частей – клиентской и серверной. В клиентской части реализован пользовательский интерфейс (*AClient*), а также интерфейс отправки запросов на сервер (*AServer*). Серверная часть производит операции со счетами.

Роль сервера исполняет класс *Server*, написанный и запускающийся отдельно. Поведение клиента моделируется автоматом *AClient* и вложенным в него автоматом *AServer*.

3.1. Поставщики событий и объекты управления модели в целом

Поставщики событий:
HardwareEventProvider – системные события, генерируемые оборудованием;
HumanEventProvider – события, инициируемые пользователем;
ServerEventProvider – ответы на запросы, поступающие от сервера;
ClientEventProvider – запросы, поступающие на сервер.

Объекты управления:
FormPainter – визуализация работы;
ServerQuery – отправляет запросы на сервер;
ServerReply – отвечает на клиентские запросы.

3.2. Схема связей

На рис. 3. изображена схема связей автоматов *AClient* и *AServer*.

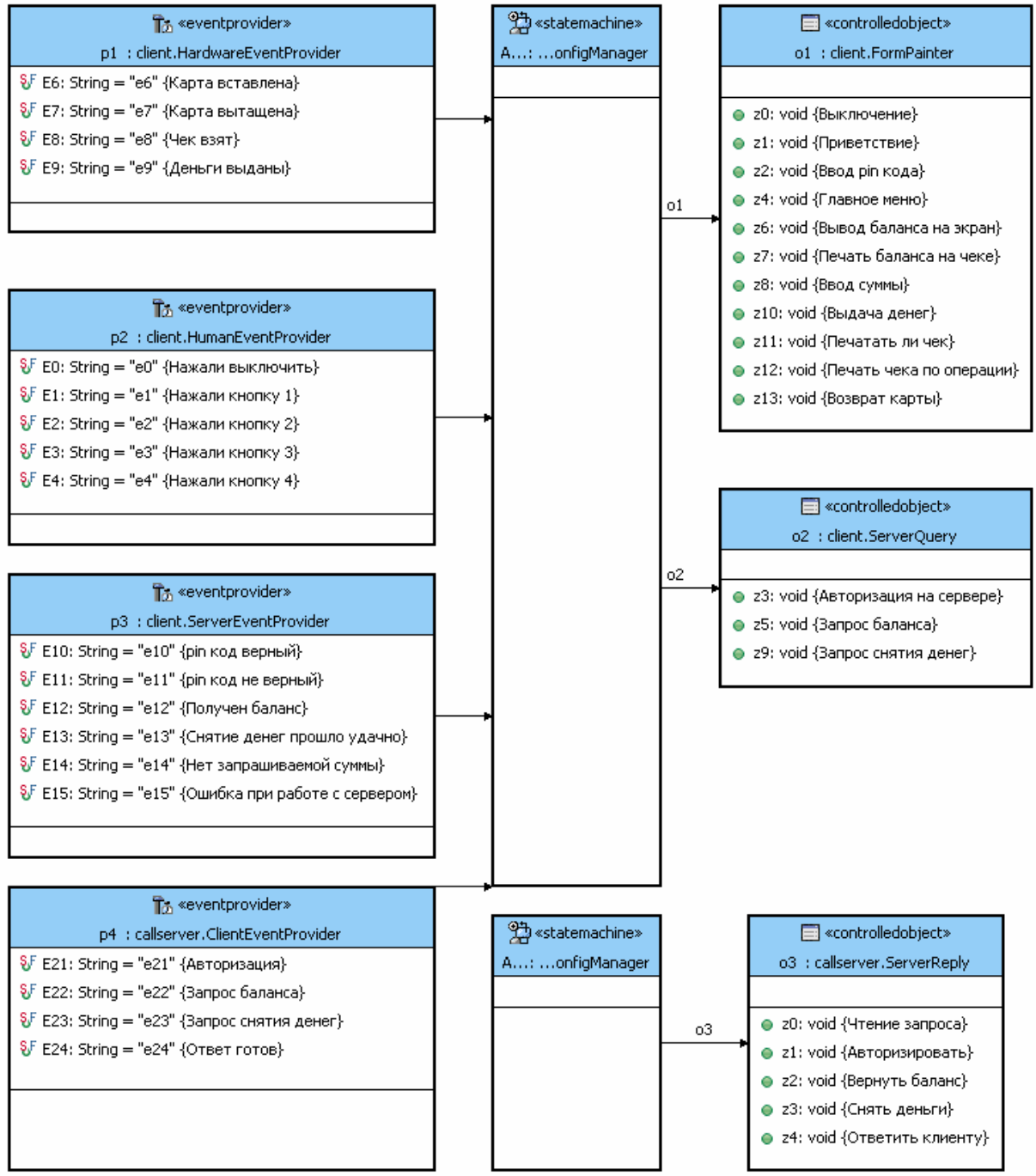

Рис. 3. Схема связей. Несмотря на отсутствие связи между автоматами, *AServer* вложен в *AClient*

3.3. Автоматы

3.3.1. Клиент

На рис. 4 приведен граф переходов автомата *AClient*.

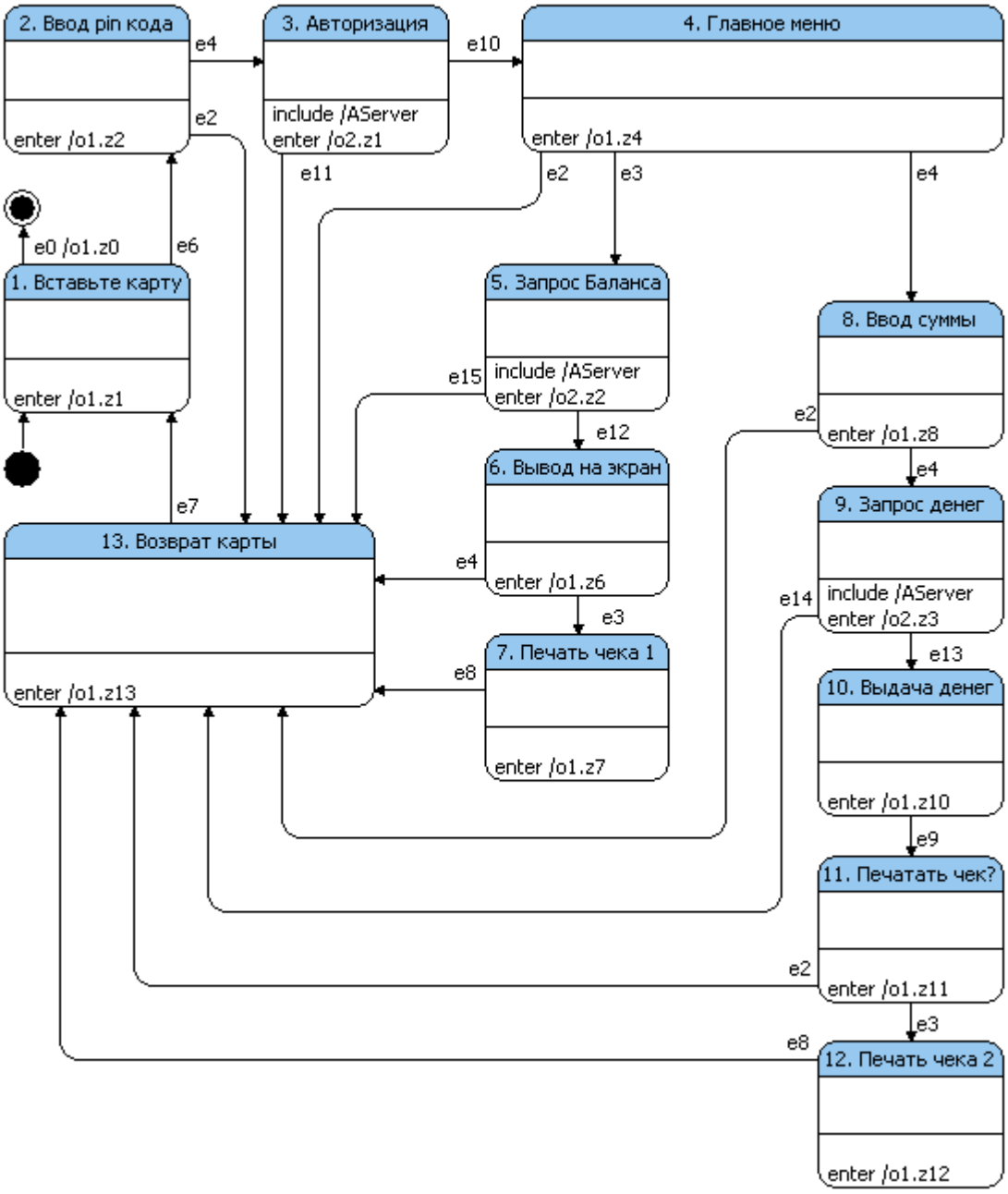

Рис. 4. Автомат *AClient*

3.3.2. Подавтомат клиента

На рис. 5 приведен граф переходов автомата *AServer*, посылающего запросы на сервер.

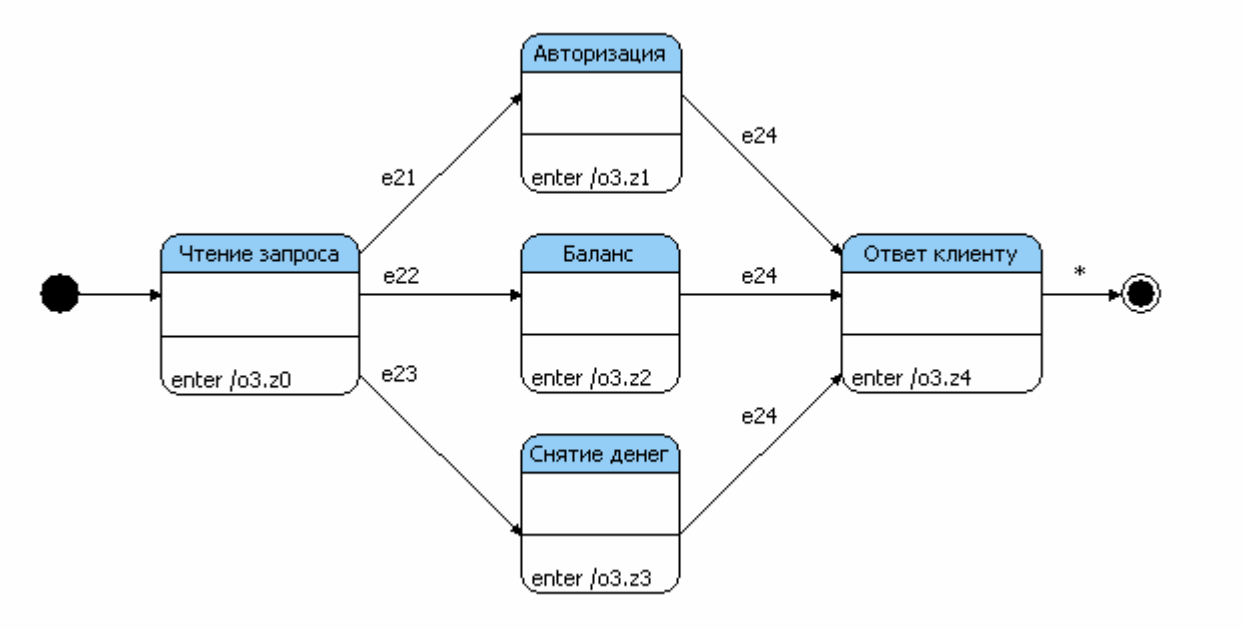

Рис. 5. Автомат *AServer*

3.4. Проектирование сервера

Сервер спроектирован традиционным путем и реализован как консольное приложение (Приложение 2.3.).

4. Реализация

4.1. Запуск программы

Запуск программы осуществляется в следующей последовательности – сначала запускается сервер, а затем клиент.

Для запуска сервера необходимо, во-первых, запустить программу *rmiregistry*. При этом переменная среды *CLASSPATH* должна включать в себя путь к jar файлу сервера, а в переменной окружения *PATH* должны быть установлены пути к *Java*. Во-вторых, запустить класс server.Server. Все эти действия выполняются файлом *runServer.bat*.

Клиент запускается файлом *runClient.bat*.

Обмен данными между клиентом и сервером осуществляется по технологии *RMI*.

Существуют два способа запуска клиентской части проекта вне среды разработки *Eclipse*: интерпретация и компиляция. В обоих подходах поставщики событий и объекты управления написаны вручную, их исходные коды приведены в Приложении 2.

4.2. Интерпретационный подход

При интерпретационном подходе *java*-класс *Run* из пакета *com.evelopers.unimod.adapter.standalone* интерпретирует *xml*-файлы (Приложение 3), в

которых описаны автоматы. При этом интерпретатору необходим *jar*-файл (архив) с проектом и библиотекой *Unimod*. Этот архив генерируется автоматически. Недостатком метода является то, что получившийся архив требует более 2 Мб памяти. При этом отметим, что все инструментальное средства занимает порядка 10 Мб памяти.

4.3. Компилятивный подход

При этом подходе программа запускается при помощи класса *BankomatRun*, который создает экземпляр класса *ModelEngine*, управляющий течением программы. Для создания обекта типа *ModelEngine* необходим сгенерированный средством *Unimod* класс *Model1EventProcessor*, а также реализация классов *ControlledObjectsManager* и *EventProvidersManager*. Приведем значение этих объектов:

Model1EventProcessor – описывает атоматы; является аналогом *xml*-файлов интерпретационного подхода;

ControlledObjectsManager – создает экземпляры объектов управления;

EventProvidersManager – создает экземпляры поставщиков событий и запускает их.

Классу *BankomatRun*, также как при интерпретации, требуется некоторые файлы библиотеки *Unimod*, но их значительно меньше. Поэтому объем, занимаемой проектом памяти, в этом случае составляет около 200 Кб. Выборка нужных файлов библиотеке осуществляется эмпирическим путем. С другой стороны, требуется писать вручную класс *BankomatRun*. Разработчики *Unimod* обещают исправить этот недостаток в одной из следующих версий своего продукта.

Исходный коды классов *BankomatRun* и *Model1EventProcessor* приведены в Приложении 4.

Заключение

Данная работа показывает, что инструментальное средство *UniMod* позволяет упростить процесс создание программы, по сравнению с традиционным подходом. При этом большая часть времени разработчика уходит на проектирование системы. Ввиду того, что основная часть кода генерируется автоматически, повышается надежность программы.

Источники

1. Сайт по автоматному программированию и мотивации к творчеству
2. UniMod at SourceForge
3. Моделирование банкомата

Приложение 1. Пример протокола работы программы

```
 [info] Start event [] processing. In state [/AClient:Top]
 [info] Transition to go found [s1#Вставьте карту##true]
 [info] Start on-enter action [o1.z2] execution
 [info] Finish on-enter action [o1.z2] execution
 [info] Finish event [] processing. In state [/AClient:Вставьте карту]
 [info] Start event [e6] processing. In state [/AClient:Вставьте карту]
[debug] Try transition [Вставьте карту#Ввод PIN-кода#e6#true]
 [info] Transition to go found [Вставьте карту#Ввод PIN-кода#e6#true]
 [info] Start on-enter action [o1.z3] execution
 [info] Finish on-enter action [o1.z3] execution
 [info] Finish event [e6] processing. In state [/AClient:Ввод PIN-кода]
 [info] Start event [e4] processing. In state [/AClient:Ввод PIN-кода]
[debug] Try transition [Ввод PIN-кода#Авторизация#e4#true]
 [info] Transition to go found [Ввод PIN-кода#Авторизация#e4#true]
 [info] Start on-enter action [o2.z1] execution
 [info] Finish on-enter action [o2.z1] execution
 [info] Start event [e4] processing. In state [/AClient:Авторизация/AServer:Top]
 [info] Transition to go found [s1#Чтение запроса##true]
 [info] Start on-enter action [o3.z0] execution
 [info] Finish on-enter action [o3.z0] execution
 [info] Finish event [e4] processing. In state [/AClient:Авторизация/AServer:Чтение
запроса]
 [info] Finish event [e4] processing. In state [/AClient:Авторизация]
 [info] Start event [e21] processing. In state [/AClient:Авторизация]
 [info] Start event [e21] processing. In state [/AClient:Авторизация/AServer:Чтение
запроса]
[debug] Try transition [Чтение запроса#Авторизация#e21#true]
 [info] Transition to go found [Чтение запроса#Авторизация#e21#true]
 [info] Start on-enter action [o3.z1] execution
N: '0'
pin: '0'
 [info] Finish on-enter action [o3.z1] execution
 [info] Finish event [e21] processing. In state
[/AClient:Авторизация/AServer:Авторизация]
 [info] Finish event [e21] processing. In state [/AClient:Авторизация]
 [info] Start event [e24] processing. In state [/AClient:Авторизация]
 [info] Start event [e24] processing. In state
[/AClient:Авторизация/AServer:Авторизация]
[debug] Try transition [Авторизация#Ответ клиенту#e24#true]
 [info] Transition to go found [Авторизация#Ответ клиенту#e24#true]
 [info] Start on-enter action [o3.z4] execution
 [info] Finish on-enter action [o3.z4] execution
 [info] Finish event [e24] processing. In state [/AClient:Авторизация/AServer:Ответ
клиенту]
 [info] Finish event [e24] processing. In state [/AClient:Авторизация]
 [info] Start event [] processing. In state [/AClient:Авторизация]
 [info] Start event [] processing. In state [/AClient:Авторизация/AServer:Ответ
клиенту]
[debug] Try transition [Ответ клиенту#s2#*#true]
 [info] Transition to go found [Ответ клиенту#s2#*#true]
 [info] State machine came to final state [/AClient:Авторизация/AServer:s2]
 [info] Finish event [] processing. In state [/AClient:Авторизация/AServer:s2]
 [info] Finish event [] processing. In state [/AClient:Авторизация]
 [info] Start event [e10] processing. In state [/AClient:Авторизация]
[debug] Try transition [Авторизация#Главное меню#e10#true]
 [info] Transition to go found [Авторизация#Главное меню#e10#true]
 [info] Start on-enter action [o1.z4] execution
 [info] Finish on-enter action [o1.z4] execution
 [info] Finish event [e10] processing. In state [/AClient:Главное меню]
```

```
 [info] Start event [e4] processing. In state [/AClient:Главное меню]
[debug] Try transition [Главное меню#Ввод суммы#e4#true]
 [info] Transition to go found [Главное меню#Ввод суммы#e4#true]
 [info] Start on-enter action [o1.z8] execution
 [info] Finish on-enter action [o1.z8] execution
 [info] Finish event [e4] processing. In state [/AClient:Ввод суммы]
 [info] Start event [e4] processing. In state [/AClient:Ввод суммы]
[debug] Try transition [Ввод суммы#Запрос денег#e4#true]
 [info] Transition to go found [Ввод суммы#Запрос денег#e4#true]
 [info] Start on-enter action [o2.z3] execution
 [info] Finish on-enter action [o2.z3] execution
 [info] Start event [e4] processing. In state [/AClient:Запрос денег/AServer:Top]
 [info] Transition to go found [s1#Чтение запроса##true]
 [info] Start on-enter action [o3.z0] execution
 [info] Finish on-enter action [o3.z0] execution
 [info] Finish event [e4] processing. In state [/AClient:Запрос денег/AServer:Чтение
запроса]
 [info] Finish event [e4] processing. In state [/AClient:Запрос денег]
 [info] Start event [e23] processing. In state [/AClient:Запрос денег]
 [info] Start event [e23] processing. In state [/AClient:Запрос денег/AServer:Чтение
запроса]
[debug] Try transition [Чтение запроса#Снять денег#e23#true]
 [info] Transition to go found [Чтение запроса#Снять денег#e23#true]
 [info] Start on-enter action [o3.z3] execution
 [info] Finish on-enter action [o3.z3] execution
 [info] Finish event [e23] processing. In state [/AClient:Запрос денег/AServer:Снять
денег]
 [info] Finish event [e23] processing. In state [/AClient:Запрос денег]
 [info] Start event [e24] processing. In state [/AClient:Запрос денег]
 [info] Start event [e24] processing. In state [/AClient:Запрос денег/AServer:Снять
денег]
[debug] Try transition [Снять денег#Ответ клиенту#e24#true]
 [info] Transition to go found [Снять денег#Ответ клиенту#e24#true]
 [info] Start on-enter action [o3.z4] execution
 [info] Finish on-enter action [o3.z4] execution
 [info] Finish event [e24] processing. In state [/AClient:Запрос денег/AServer:Ответ
клиенту]
 [info] Finish event [e24] processing. In state [/AClient:Запрос денег]
 [info] Start event [] processing. In state [/AClient:Запрос денег]
 [info] Start event [] processing. In state [/AClient:Запрос денег/AServer:Ответ
клиенту]
[debug] Try transition [Ответ клиенту#s2#*#true]
 [info] Transition to go found [Ответ клиенту#s2#*#true]
 [info] State machine came to final state [/AClient:Запрос денег/AServer:s2]
 [info] Finish event [] processing. In state [/AClient:Запрос денег/AServer:s2]
 [info] Finish event [] processing. In state [/AClient:Запрос денег]
 [info] Start event [e13] processing. In state [/AClient:Запрос денег]
[debug] Try transition [Запрос денег#Выдача денег#e13#true]
 [info] Transition to go found [Запрос денег#Выдача денег#e13#true]
 [info] Start on-enter action [o1.z9] execution
 [info] Finish on-enter action [o1.z9] execution
 [info] Finish event [e13] processing. In state [/AClient:Выдача денег]
 [info] Start event [e9] processing. In state [/AClient:Выдача денег]
[debug] Try transition [Выдача денег#Печатать ли чек#e9#true]
 [info] Transition to go found [Выдача денег#Печатать ли чек#e9#true]
 [info] Start on-enter action [o1.z10] execution
 [info] Finish on-enter action [o1.z10] execution
 [info] Finish event [e9] processing. In state [/AClient:Печатать ли чек]
 [info] Start event [e3] processing. In state [/AClient:Печатать ли чек]
[debug] Try transition [Печатать ли чек#Печать чека#e3#true]
 [info] Transition to go found [Печатать ли чек#Печать чека#e3#true]
 [info] Start on-enter action [o1.z7] execution
 [info] Finish on-enter action [o1.z7] execution
 [info] Finish event [e3] processing. In state [/AClient:Печать чека]
 [info] Start event [e8] processing. In state [/AClient:Печать чека]
[debug] Try transition [Печать чека#Возврат карты#e8#true]
```

```
 [info] Transition to go found [Печать чека#Возврат карты#e8#true]
 [info] Start on-enter action [o1.z1] execution
 [info] Finish on-enter action [o1.z1] execution
 [info] Finish event [e8] processing. In state [/AClient:Возврат карты]
 [info] Start event [e7] processing. In state [/AClient:Возврат карты]
[debug] Try transition [Возврат карты#Вставьте карту#e7#true]
 [info] Transition to go found [Возврат карты#Вставьте карту#e7#true]
 [info] Start on-enter action [o1.z2] execution
 [info] Finish on-enter action [o1.z2] execution
 [info] Finish event [e7] processing. In state [/AClient:Вставьте карту]
 [info] Start event [e0] processing. In state [/AClient:Вставьте карту]
[debug] Try transition [Вставьте карту#s2#e0#true]
 [info] Transition to go found [Вставьте карту#s2#e0#true]
 [info] Start output action [o1.z0] execution
 [info] Finish output action [o1.z0] execution
 [info] State machine came to final state [/AClient:s2]
 [info] Finish event [e0] processing. In state [/AClient:s2]
```

Приложение 2. Исходные коды программы

2.1. Поставщики событий

2.1.1. *HardwareEventProvider*

```
/**
*  Системные события, генерируемые оборудованием
*/
package client;


import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;

public class HardwareEventProvider implements EventProvider {
    /**
     * @unimod.event.descr Карта вставлена
     */
    public static final String E6 = "e6";
    /**
     * @unimod.event.descr Карта вытащена
     */
    public static final String E7 = "e7";
    /**
     * @unimod.event.descr Чек взят
     */
    public static final String E8 = "e8";
    /**
     * @unimod.event.descr Деньги выданы
     */

    public static final String E9 = "e9";
    public static ModelEngine engine;

    public void init(ModelEngine engine) throws CommonException {
        HardwareEventProvider.engine = engine;
        FormPainter.init(engine);

        FormPainter.form.notify("");
    }

    public void dispose() {

    }

    public static void setTimeout(final String event, int time_delay){
        java.util.Timer t = new java.util.Timer(false);
        t.schedule(
            new java.util.TimerTask() {
                public void run() {
                    engine.getEventManager().handle(new Event(event),
StateMachineContextImpl.create());
                }
            },
            time_delay
```

```
        );
    }
}
```

2.1.2. *HumanEventProvider*

```
/**
*  События, инициируемые пользователем
*/
package client;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;

public class HumanEventProvider implements EventProvider {

    /**
     * @unimod.event.descr Нажали выключить
     */
    public static final String E0 = "e0";
    /**
     * @unimod.event.descr Нажали кнопку 1
     */
    public static final String E1 = "e1";
    /**
     * @unimod.event.descr Нажали кнопку 2
     */
    public static final String E2 = "e2";
    /**
     * @unimod.event.descr Нажали кнопку 3
     */
    public static final String E3 = "e3";
    /**
     * @unimod.event.descr Нажали кнопку 4
     */
    public static final String E4 = "e4";

    public void init(ModelEngine engine) throws CommonException {
        // TODO Auto-generated method stub

    }

    public void dispose() {
        // TODO Auto-generated method stub

    }

}
```

2.1.3. *ServerEventProvider*

```
/**
*  Ответы на запросы, поступающие от сервера
*/
package client;


import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
```

```
public class ServerEventProvider implements EventProvider {

    /**
     * @unimod.event.descr pin код верный
     */
    public static final String E10 = "e10";
    /**
     * @unimod.event.descr pin код не верный
     */
    public static final String E11 = "e11";
    /**
     * @unimod.event.descr Получен баланс
     */
    public static final String E12 = "e12";
    /**
     * @unimod.event.descr Снятие денег прошло удачно
     */
    public static final String E13 = "e13";
    /**
     * @unimod.event.descr Нет запрашиваемой суммы
     */
    public static final String E14 = "e14";
    /**
     * @unimod.event.descr Ошибка при работе с сервером
     */
    public static final String E15 = "e15";

    public void init(ModelEngine engine) throws CommonException {
        // TODO Auto-generated method stub

    }

    public void dispose() {
        // TODO Auto-generated method stub

    }

}
```

2.1.4. *ClientEventProvider*

```
/**
*  Запросы, поступающие на сервер
*/
package callserver;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;

public class ClientEventProvider implements EventProvider {
    /**
     * @unimod.event.descr Авторизация
     */
    public static final String E21 = "e21";
    /**
```

```
     * @unimod.event.descr Запрос баланса
     */
    public static final String E22 = "e22";
    /**
     * @unimod.event.descr Запрос снятия денег
     */
    public static final String E23 = "e23";


    public static ModelEngine engine;
    /**
     * @unimod.event.descr Ответ готов
     */
    public static final String E24 = "e24";

    public static void notify(String event){
        engine.getEventManager().handle(new Event(event),
StateMachineContextImpl.create());
    }

    public void init(ModelEngine engine) throws CommonException {
        ClientEventProvider.engine = engine;


    }

    public void dispose() {
    }

}
```

2.2. Объекты управления

2.2.1. *FormPainter*

```
/**
* Визуализация работы
*/
package client;


import client.ServerQuery.ServerInfo;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContext;

public class FormPainter implements ControlledObject {
    public static TForm form;

    public static void init(ModelEngine engine){
        form = new TForm(engine);
    }

    /**
     * @unimod.action.descr Выключение
     */
    public void z0(StateMachineContext context) {
        form.dispose();
    }

    /**
     * @unimod.action.descr Приветствие
```

```
     */
    public void z1(StateMachineContext context) {
        form.panelMain.showState("welcome", "state1", "serverstateoff", "st1label",
"");
        form.panelMain.markButton("empty", "empty", "empty", "empty");
    }

    /**
     * @unimod.action.descr Ввод pin кода
     */
    public void z2(StateMachineContext context) {
        form.panelMain.edit_Pin.setText("");
        form.panelMain.showState("enter pin", "state2", "serverstateoff", "st2label",
"");
        form.panelMain.markButton("empty", "btncancel", "empty", "btnok");
    }

    /**
     * @unimod.action.descr Главное меню
     */
    public void z4(StateMachineContext context) {
        form.panelMain.showState("main menu", "state4", "serverstateoff", "st4label",
"");
        form.panelMain.markButton("empty", "btnbackcard", "btnbalance", "btnmoney");
    }

    /**
     * @unimod.action.descr Вывод баланса на экран
     */
    public void z6(StateMachineContext context) {
        form.panelMain.showState("balance", "state6", "serverstateoff", "st6label",
Integer.toString(ServerQuery.ServerInfo.balance));//todo
        form.panelMain.markButton("empty", "empty", "btnprint", "btnok");
    }

    /**
     * @unimod.action.descr Печать баланса на чеке
     */
    public void z7(StateMachineContext context) {
        form.panelMain.showState("print1", "state7", "serverstateoff", "st7label",
"");
        form.panelMain.markButton("empty", "empty", "empty", "empty");
    }

    /**
     * @unimod.action.descr Ввод суммы
     */
    public void z8(StateMachineContext context) {
        form.panelMain.showState("enter summ", "state8", "serverstateoff", "st8label",
"");
        form.panelMain.markButton("empty", "btncancel", "empty", "btnok");
    }

    /**
     * @unimod.action.descr Выдача денег
     */
    public void z10(StateMachineContext context) {
        form.panelMain.showState("money out", "state10", "serverstateoff",
"st10label", "");
        form.panelMain.markButton("empty", "empty", "empty", "empty");
    }

    /**
     * @unimod.action.descr Печатать ли чек
     */
    public void z11(StateMachineContext context) {
```

```
        form.panelMain.showState("", "state11", "serverstateoff", "st11label", "");
        form.panelMain.markButton("empty", "btncancel", "btnprint", "empty");
    }

    /**
     * @unimod.action.descr Печать чека по операции
     */
    public void z12(StateMachineContext context) {
        form.panelMain.showState("print2", "state12", "serverstateoff", "st12label",
"");
        form.panelMain.markButton("empty", "empty", "empty", "empty");
    }

    /**
     * @unimod.action.descr Возврат карты
     */
    public void z13(StateMachineContext context) {
        form.panelMain.showState("back card", "state13", "serverstateoff",
"st13label", "");
        form.panelMain.markButton("empty", "empty", "empty", "empty");
    }
}
```

2.2.2. *ServerQuery*

```
/**
* Отправляет запросы на сервер
*/

package client;


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

public class ServerQuery implements ControlledObject {

    /**
     * Вложенный класс, написанный вручную
     */

    public static class ServerInfo{
        public static int balance;
        public static char command;
        public static int cardN;
        public static int cardPIN;
        public static int money;
        public static void make(char com){
            command = com;
            try{
                cardN = Integer.parseInt(FormPainter.form.panelMain.edit_N.getText());
            }catch(Exception e){
                cardN = 0;
            }
            try{
                cardPIN =
Integer.parseInt(FormPainter.form.panelMain.edit_Pin.getText());
            }catch(Exception e){
                cardPIN = 0;
            }
            try{
                money =
Integer.parseInt(FormPainter.form.panelMain.edit_Summ.getText());
            }catch(Exception e){
```

```
                money = 0;
            }
        }
    }


    /**
     * @unimod.action.descr Авторизация на сервере
     */
    public void z3(StateMachineContext context) {
        FormPainter.form.panelMain.showState("", "state3", "serverstateon",
"st3label", "");
        FormPainter.form.panelMain.markButton("empty", "empty", "empty", "empty");
        ServerInfo.make('A');
    }

    /**
     * @unimod.action.descr Запрос баланса
     */
    public void z5(StateMachineContext context) {
        FormPainter.form.panelMain.showState("", "state5", "serverstateon",
"st5label", "");
        FormPainter.form.panelMain.markButton("empty", "empty", "empty", "empty");
        ServerInfo.make('B');
    }

    /**
     * @unimod.action.descr Запрос снятия денег
     */
    public void z9(StateMachineContext context) {
        FormPainter.form.panelMain.showState("", "state9", "serverstateon",
"st9label", "");
        FormPainter.form.panelMain.markButton("empty", "empty", "empty", "empty");
        ServerInfo.make('G');
    }

}
```

2.2.3. *ServerReply*

```
/**
*  Отвечает на клиентские запросы
*/
package callserver;

import java.rmi.*;
import java.net.*;


import client.ServerEventProvider;
import client.ServerQuery;
import server.Bank;

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

public class ServerReply implements ControlledObject {
    private char command;
    private int cardN;
    private int cardPIN;
    private int money;
    private String reply;
    Bank bank;
```

```
/**
 * @unimod.action.descr Чтение запроса
 */
public void z0(StateMachineContext context) {
    command = ServerQuery.ServerInfo.command;
    cardN = ServerQuery.ServerInfo.cardN;
    cardPIN = ServerQuery.ServerInfo.cardPIN;
    money = ServerQuery.ServerInfo.money;
    switch(command){
    case 'A': ClientEventProvider.notify(ClientEventProvider.E21); break;
    case 'B': ClientEventProvider.notify(ClientEventProvider.E22); break;
    case 'G': ClientEventProvider.notify(ClientEventProvider.E23); break;
    }
}

/**
 * @unimod.action.descr Авторизировать
 */
public void z1(StateMachineContext context) {
    try{
        bank = (Bank)Naming.lookup("rmi://localhost/bank");
    }catch(NotBoundException e){
        System.out.println("Not bound");
        e.printStackTrace();
    }catch(MalformedURLException e){
        System.out.println("MalformedURL");
        e.printStackTrace();
    }catch(RemoteException e){
        System.out.println("RemoteException");
        e.printStackTrace();
    };

    try{
        if(bank.checkCard(cardN, cardPIN))
            reply = ServerEventProvider.E10;
        else
            reply = ServerEventProvider.E11;
    }catch(RemoteException e){
        reply = ServerEventProvider.E15;
    }
    ClientEventProvider.notify(ClientEventProvider.E24);
}

/**
 * @unimod.action.descr Вернуть баланс
 */
public void z2(StateMachineContext context) {
    try{
        ServerQuery.ServerInfo.balance = bank.getBalance(cardN, cardPIN);
        reply = ServerEventProvider.E12;
    }catch(RemoteException e){
        reply = ServerEventProvider.E15;
    }
    ClientEventProvider.notify(ClientEventProvider.E24);
}

/**
 * @unimod.action.descr Снять деньги
 */
public void z3(StateMachineContext context) {
    try{
        if(bank.getMoney(cardN, cardPIN, money))
            reply = ServerEventProvider.E13;
        else
            reply = ServerEventProvider.E14;
```

```
        }catch(RemoteException e){
            reply = ServerEventProvider.E15;
        }
        ClientEventProvider.notify(ClientEventProvider.E24);
    }

    /**
     * @unimod.action.descr Ответить клиенту
     */
    public void z4(StateMachineContext context) {
        ClientEventProvider.notify("");
        ClientEventProvider.notify(reply);
    }

}
```

2.3. Сервер

2.3.1. *Server*

```
package server;

import java.rmi.*;
import java.rmi.server.*;
import java.net.*;

import server.Bank;

public class Server {
    public static void main(String[] args){
        Bank  bank = new BankImpl();
        try{
            System.out.println(1);
            UnicastRemoteObject.exportObject(bank);
            System.out.println(2);
            Naming.rebind("rmi://localhost/bank", bank);
            System.out.println(3);
        }catch(RemoteException e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }catch(MalformedURLException e){
            System.out.println("MalformedURL");
        }
    }
}
```

2.3.2. *Bank*

```
package server;

import java.rmi.*;

public interface Bank extends Remote{

    public void loadFromFile() throws RemoteException;
    public void saveToFile() throws RemoteException;
    public boolean checkCard(int N, int pin) throws RemoteException;
    public int getBalance(int N, int pin)throws RemoteException;
    public boolean getMoney(int N, int pin, int summ)throws
RemoteException;
```

```
}
```

2.3.3. *BankImpl*

```
package server;

import java.io.*;
import java.rmi.RemoteException;
import server.Bank;

public class BankImpl implements Bank{
    int dataSize;
    int[][] data;

    BankImpl(){
        try{
            loadFromFile();
        }catch(Exception e){
            System.out.println("-------------");
            e.printStackTrace();
            System.out.println("-------------");
        }
    }

    public void loadFromFile()throws RemoteException{
        try{
            BufferedReader f = new BufferedReader(new  InputStreamReader(new
FileInputStream("Bank.dat"), "windows-1251"));
            dataSize = Integer.parseInt(f.readLine());
            data = new int[dataSize][3];
            for(int i = 0; i < dataSize; i++){
                data[i][0] = Integer.parseInt(f.readLine());
                data[i][1] = Integer.parseInt(f.readLine());
                data[i][2] = Integer.parseInt(f.readLine());
            }
            f.close();
        }catch(Exception e){
            System.out.println("* - - = Base load error = - - *");
        }
    }

    public void saveToFile()throws RemoteException{
        try{
            BufferedWriter f = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("Bank.dat"), "windows-1251"));
            f.write("" + dataSize + '\n');
            for(int i = 0; i < dataSize; i++){
                f.write("" + data[i][0] + '\n');
                f.write("" + data[i][1] + '\n');
                f.write("" + data[i][2] + '\n');
            }
            f.close();
        }catch(Exception e){
            System.out.println("* - - = Base save error = - - *");
        }
    }

    public boolean checkCard(int N, int pin)throws RemoteException{
        System.out.println("N: '" + N + "'");
        System.out.println("pin: '" + pin + "'");
        boolean found = false;
        for(int i = 0; i < dataSize; i++)
            if((data[i][0] == N) && (data[i][1] == pin)){
```

```
                found = true;
                break;
            }
        return found;
    }

    public int getBalance(int N, int pin)throws RemoteException{
        int balance = -1;

        for(int i = 0; i < dataSize; i++)
            if((data[i][0] == N) && (data[i][1] == pin)){
                balance = data[i][2];
                break;
            }
        return balance;
    }

    public boolean getMoney(int N, int pin, int summ)throws RemoteException{
        boolean res = false;
        for(int i = 0; i < dataSize; i++)
            if((data[i][0] == N) && (data[i][1] == pin)){
                System.out.println(i);
                System.out.println("data: " + data[i][2] + " summ: " + summ);
                if(data[i][2] >= summ){
                    res = true;
                    data[i][2] -= summ;
                    saveToFile();
                }
                break;
            }
        System.out.println(res);
        return res;
    }
}
```

Приложение 3. Интерпретационный подход. Сгенерированное XML-описание

3.1. AClient

```
<?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">

    <!-- Объекты управления -->
  <controlledObject class="Client.FormPainter" name="o1"/>
  <controlledObject class="Client.ServerQuery" name="o2"/>
  <controlledObject class="Server.ServerReply" name="o3"/>

    <!-- Поставщики событий -->
  <eventProvider class="Client.HardwareEvent" name="p1">
    <association clientRole="p1" targetRef="AClient"/>
  </eventProvider>
  <eventProvider class="Client.HumanEvent" name="p2">
    <association clientRole="p2" targetRef="AClient"/>
  </eventProvider>
  <eventProvider class="Client.ServerEvent" name="p3">
    <association clientRole="p3" targetRef="AClient"/>
  </eventProvider>
  <eventProvider class="Server.ClientEvent" name="p4">
    <association clientRole="p4" targetRef="AClient"/>
  </eventProvider>

    <!-- Объявление -->
  <rootStateMachine>
    <stateMachineRef name="AClient"/>
  </rootStateMachine>

  <stateMachine name="AClient">
    <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>

      <!-- Ассоциации -->
    <association clientRole="AClient" supplierRole="o1" targetRef="o1"/>
    <association clientRole="AClient" supplierRole="o2" targetRef="o2"/>

      <!-- Состояния -->
      <state name="Top" type="NORMAL">
      <state name="10. Выдача денег" type="NORMAL">
        <outputAction ident="o1.z10"/>
      </state>
      <state name="12. Печать чека 2" type="NORMAL">
        <outputAction ident="o1.z12"/>
      </state>
      <state name="2. Ввод pin кода" type="NORMAL">
        <outputAction ident="o1.z2"/>
      </state>
      <state name="11. Печатать чек" type="NORMAL">
        <outputAction ident="o1.z11"/>
      </state>
      <state name="4. Главное меню" type="NORMAL">
        <outputAction ident="o1.z4"/>
      </state>
      <state name="8. Ввод суммы" type="NORMAL">
```

```
        <outputAction ident="o1.z8"/>
      </state>
      <state name="3. Авторизация" type="NORMAL">
        <stateMachineRef name="AServer"/>
        <outputAction ident="o2.z3"/>
      </state>
      <state name="13. Возврат карты" type="NORMAL">
        <outputAction ident="o1.z13"/>
      </state>
      <state name="1. Вставьте карту" type="NORMAL">
        <outputAction ident="o1.z1"/>
      </state>
      <state name="5. Запрос Баланса" type="NORMAL">
        <stateMachineRef name="AServer"/>
        <outputAction ident="o2.z5"/>
      </state>
      <state name="9. Запрос денег" type="NORMAL">
        <stateMachineRef name="AServer"/>
        <outputAction ident="o2.z9"/>
      </state>
      <state name="7. Печать чека 1" type="NORMAL">
        <outputAction ident="o1.z7"/>
      </state>
      <state name="s2" type="FINAL"/>
      <state name="6. Вывод на экран" type="NORMAL">
        <outputAction ident="o1.z6"/>
      </state>
      <state name="s1" type="INITIAL"/>
    </state>

      <!-- Переходы -->
    <transition event="e9" sourceRef="10. Выдача денег" targetRef="11. Печатать чек"/>
    <transition event="e8" sourceRef="12. Печать чека 2" targetRef="13. Возврат
карты"/>
    <transition event="e4" sourceRef="2. Ввод pin кода" targetRef="3. Авторизация"/>
    <transition event="e2" sourceRef="2. Ввод pin кода" targetRef="13. Возврат
карты"/>
    <transition event="e3" sourceRef="11. Печатать чек" targetRef="12. Печать чека
2"/>
    <transition event="e2" sourceRef="11. Печатать чек" targetRef="13. Возврат
карты"/>
    <transition event="e2" sourceRef="4. Главное меню" targetRef="13. Возврат карты"/>
    <transition event="e3" sourceRef="4. Главное меню" targetRef="5. Запрос Баланса"/>
    <transition event="e4" sourceRef="4. Главное меню" targetRef="8. Ввод суммы"/>
    <transition event="e4" sourceRef="8. Ввод суммы" targetRef="9. Запрос денег"/>
    <transition event="e2" sourceRef="8. Ввод суммы" targetRef="13. Возврат карты"/>
    <transition event="e10" sourceRef="3. Авторизация" targetRef="4. Главное меню"/>
    <transition event="e11" sourceRef="3. Авторизация" targetRef="13. Возврат карты"/>
    <transition event="e7" sourceRef="13. Возврат карты" targetRef="1. Вставьте
карту"/>
    <transition event="e0" sourceRef="1. Вставьте карту" targetRef="s2">
      <outputAction ident="o1.z0"/>
    </transition>
    <transition event="e6" sourceRef="1. Вставьте карту" targetRef="2. Ввод pin
кода"/>
    <transition event="e12" sourceRef="5. Запрос Баланса" targetRef="6. Вывод на
экран"/>
    <transition event="e15" sourceRef="5. Запрос Баланса" targetRef="13. Возврат
карты"/>
    <transition event="e13" sourceRef="9. Запрос денег" targetRef="10. Выдача денег"/>
    <transition event="e14" sourceRef="9. Запрос денег" targetRef="13. Возврат
карты"/>
    <transition event="e8" sourceRef="7. Печать чека 1" targetRef="13. Возврат
карты"/>
    <transition event="e3" sourceRef="6. Вывод на экран" targetRef="7. Печать чека
1"/>
```

```
    <transition event="e4" sourceRef="6. Вывод на экран" targetRef="13. Возврат
карты"/>
    <transition sourceRef="s1" targetRef="1. Вставьте карту"/>
  </stateMachine>

    <!-- Описание вложенного автомата -->
  <stateMachine name="AServer">
    <configStore
class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>

      <!-- Ассоциации -->
    <association clientRole="AServer" targetRef="AClient"/>
    <association clientRole="AServer" supplierRole="o3" targetRef="o3"/>

      <!-- Состояния -->
    <state name="Top" type="NORMAL">
      <state name="Авторизация" type="NORMAL">
        <outputAction ident="o3.z1"/>
      </state>
      <state name="s2" type="FINAL"/>
      <state name="s1" type="INITIAL"/>
      <state name="Чтение запроса" type="NORMAL">
        <outputAction ident="o3.z0"/>
      </state>
      <state name="Баланс" type="NORMAL">
        <outputAction ident="o3.z2"/>
      </state>
      <state name="Снятие денег" type="NORMAL">
        <outputAction ident="o3.z3"/>
      </state>
      <state name="Ответ клиенту" type="NORMAL">
        <outputAction ident="o3.z4"/>
      </state>
    </state>

      <!-- Переходы -->
    <transition event="e24" sourceRef="Авторизация" targetRef="Ответ клиенту"/>
    <transition sourceRef="s1" targetRef="Чтение запроса"/>
    <transition event="e23" sourceRef="Чтение запроса" targetRef="Снятие денег"/>
    <transition event="e21" sourceRef="Чтение запроса" targetRef="Авторизация"/>
    <transition event="e22" sourceRef="Чтение запроса" targetRef="Баланс"/>
    <transition event="e24" sourceRef="Баланс" targetRef="Ответ клиенту"/>
    <transition event="e24" sourceRef="Снятие денег" targetRef="Ответ клиенту"/>
    <transition event="*" sourceRef="Ответ клиенту" targetRef="s2"/>
  </stateMachine>
</model>
```

Приложение 4. Компилятивный подход. Исходные коды

4.1. *Model1EventProcessor*

```
/**
 * This file was generated from model [Model1] on [Thu Apr 06 16:48:05 MSD 2006].
 * Do not change content of this file.
 */

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

import java.util.BitSet;
import java.util.HashMap;
import java.util.Map;


public class Model1EventProcessor extends AbstractEventProcessor {

    private ModelStructure modelStructure;

    private static final int AClient = 1;
    private static final int AServer = 2;

    private int decodeStateMachine(String sm) {

        if ("AClient".equals(sm)) {
            return AClient;
        } else if ("AServer".equals(sm)) {
            return AServer;
        }

        return -1;
    }

    private AClientEventProcessor _AClient;
    private AServerEventProcessor _AServer;

    public Model1EventProcessor() {
        modelStructure = new Model1ModelStructure();

        _AClient = new AClientEventProcessor();
        _AServer = new AServerEventProcessor();
    }

    public ModelStructure getModelStructure() {
        return modelStructure;
    }

    public void setControlledObjectsMap(ControlledObjectsMap controlledObjectsMap) {
        super.setControlledObjectsMap(controlledObjectsMap);

        _AClient.init(controlledObjectsMap);
        _AServer.init(controlledObjectsMap);
    }

    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 AClient:
                    return _AClient.process(event, context, path, config);
                case AServer:
                    return _AServer.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);
            }
        }
    }

    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 AClient:
                    return _AClient.transiteToStableState(context, path, config);
                case AServer:
                    return _AServer.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);
            }
        }
    }

    private class Model1ModelStructure implements ModelStructure {

        private Map configManagers = new HashMap();

        private Model1ModelStructure() {
            configManagers.put("AClient", new
com.evelopers.unimod.runtime.config.DistinguishConfigManager());
            configManagers.put("AServer", new
com.evelopers.unimod.runtime.config.DistinguishConfigManager());
        }

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

```
        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 AClient:
                    return new StateMachineConfig("Top");
                case AServer:
                    return new StateMachineConfig("Top");
                default:
                    throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
            }
        }

        public boolean isFinal(String stateMachine, StateMachineConfig config) throws
EventProcessorException {
            // get state machine from path
            int sm = decodeStateMachine(stateMachine);
            int state;

            switch (sm) {
                case AClient:
                    state = _AClient.decodeState(config.getActiveState());
                    switch (state) {
                        case AClientEventProcessor.s2:
                            return true;
                        default:
                            return false;
                    }
                case AServer:
                    state = _AServer.decodeState(config.getActiveState());
                    switch (state) {
                        case AServerEventProcessor.s2:
                            return true;
                        default:
                            return false;
                    }
                default:
                    throw new EventProcessorException("Unknown state machine [" +
stateMachine + "]");
            }
        }
    }


    private class AClientEventProcessor {

        // states
        private static final int Top = 1;
        private static final int s1 = 2;
        private static final int _4__Главное_меню = 3;
        private static final int _5__Запрос_Баланса = 4;
        private static final int _3__Авторизация = 5;
        private static final int _11__Печатать_чек = 6;
        private static final int s2 = 7;
        private static final int _13__Возврат_карты = 8;
        private static final int _12__Печать_чека_2 = 9;
        private static final int _2__Ввод_pin_кода = 10;
        private static final int _8__Ввод_суммы = 11;
        private static final int _10__Выдача_денег = 12;
```

```
private static final int _9__Запрос_денег = 13;
private static final int _1__Вставьте_карту = 14;
private static final int _7__Печать_чека_1 = 15;
private static final int _6__Вывод_на_экран = 16;

private int decodeState(String state) {

    if ("Top".equals(state)) {
        return Top;
    } else if ("s1".equals(state)) {
        return s1;
    } else if ("4. Главное меню".equals(state)) {
        return _4__Главное_меню;
    } else if ("5. Запрос Баланса".equals(state)) {
        return _5__Запрос_Баланса;
    } else if ("3. Авторизация".equals(state)) {
        return _3__Авторизация;
    } else if ("11. Печатать чек".equals(state)) {
        return _11__Печатать_чек;
    } else if ("s2".equals(state)) {
        return s2;
    } else if ("13. Возврат карты".equals(state)) {
        return _13__Возврат_карты;
    } else if ("12. Печать чека 2".equals(state)) {
        return _12__Печать_чека_2;
    } else if ("2. Ввод pin кода".equals(state)) {
        return _2__Ввод_pin_кода;
    } else if ("8. Ввод суммы".equals(state)) {
        return _8__Ввод_суммы;
    } else if ("10. Выдача денег".equals(state)) {
        return _10__Выдача_денег;
    } else if ("9. Запрос денег".equals(state)) {
        return _9__Запрос_денег;
    } else if ("1. Вставьте карту".equals(state)) {
        return _1__Вставьте_карту;
    } else if ("7. Печать чека 1".equals(state)) {
        return _7__Печать_чека_1;
    } else if ("6. Вывод на экран".equals(state)) {
        return _6__Вывод_на_экран;
    }

    return -1;
}

// events
private static final int e2 = 1;
private static final int e11 = 2;
private static final int e15 = 3;
private static final int e9 = 4;
private static final int e14 = 5;
private static final int e10 = 6;
private static final int e3 = 7;
private static final int e8 = 8;
private static final int e4 = 9;
private static final int e13 = 10;
private static final int e0 = 11;
private static final int e7 = 12;
private static final int e12 = 13;
private static final int e6 = 14;

private int decodeEvent(String event) {

    if ("e2".equals(event)) {
        return e2;
    } else if ("e11".equals(event)) {
        return e11;
```

```
            } else if ("e15".equals(event)) {
                return e15;
            } else if ("e9".equals(event)) {
                return e9;
            } else if ("e14".equals(event)) {
                return e14;
            } else if ("e10".equals(event)) {
                return e10;
            } else if ("e3".equals(event)) {
                return e3;
            } else if ("e8".equals(event)) {
                return e8;
            } else if ("e4".equals(event)) {
                return e4;
            } else if ("e13".equals(event)) {
                return e13;
            } else if ("e0".equals(event)) {
                return e0;
            } else if ("e7".equals(event)) {
                return e7;
            } else if ("e12".equals(event)) {
                return e12;
            } else if ("e6".equals(event)) {
                return e6;
            }

            return -1;
        }

        private client.FormPainter o1;
        private client.ServerQuery o2;

        private void init(ControlledObjectsMap controlledObjectsMap) {
            o1 = (client.FormPainter) controlledObjectsMap.getControlledObject("o1");
            o2 = (client.ServerQuery) controlledObjectsMap.getControlledObject("o2");
        }

        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;
        }

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

            while (true) {
                switch (state) {
                    case s1:

                        return;
                    case _4__Главное_меню:

                        return;
                    case _5__Запрос_Баланса:
                        // 5. Запрос Баланса includes AServer

                        fireBeforeSubmachineExecution(context, event, path, "5. Запрос
Баланса", "AServer");
```

```
                        Model1EventProcessor.this.process(event, context, new
StateMachinePath(path,
                                "5. Запрос Баланса", "AServer"));

                        fireAfterSubmachineExecution(context, event, path, "5. Запрос
Баланса", "AServer");

                        return;
                    case _3__Авторизация:
                        // 3. Авторизация includes AServer

                        fireBeforeSubmachineExecution(context, event, path, "3.
Авторизация", "AServer");

                        Model1EventProcessor.this.process(event, context, new
StateMachinePath(path,
                                "3. Авторизация", "AServer"));

                        fireAfterSubmachineExecution(context, event, path, "3.
Авторизация", "AServer");

                        return;
                    case _11__Печатать_чек:

                        return;
                    case s2:

                        return;
                    case _13__Возврат_карты:

                        return;
                    case _12__Печать_чека_2:

                        return;
                    case _2__Ввод_pin_кода:

                        return;
                    case _8__Ввод_суммы:

                        return;
                    case _10__Выдача_денег:

                        return;
                    case _9__Запрос_денег:
                        // 9. Запрос денег includes AServer

                        fireBeforeSubmachineExecution(context, event, path, "9. Запрос
денег", "AServer");

                        Model1EventProcessor.this.process(event, context, new
StateMachinePath(path,
                                "9. Запрос денег", "AServer"));

                        fireAfterSubmachineExecution(context, event, path, "9. Запрос
денег", "AServer");

                        return;
                    case _1__Вставьте_карту:

                        return;
                    case _7__Печать_чека_1:

                        return;
                    case _6__Вывод_на_экран:
```

```
                    return;
                default:
                    throw new EventProcessorException("State with name [" +
config.getActiveState() + "] is unknown for state machine [AClient]");
            }
        }
    }

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

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

        switch (s) {
            case Top:


                fireComeToState(context, path, "s1");

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


                fireComeToState(context, path, "1. Вставьте карту");

                // 1. Вставьте карту [o1.z1]
                fireBeforeOutputActionExecution(context, path, "s1#1. Вставьте
карту##true", "o1.z1");

                o1.z1(context);

                fireAfterOutputActionExecution(context, path, "s1#1. Вставьте
карту##true", "o1.z1");

                return new StateMachineConfig("1. Вставьте карту");
        }

        return config;
    }

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


        BitSet calculatedInputActions = new BitSet(0);

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

        while (true) {
            switch (s) {
                case _4__Главное_меню:


                    switch (e) {
                        case e4:

// 4. Главное меню->8. Ввод суммы e4[true]/

                            fireTransitionCandidate(context, path, "4. Главное
меню", event, "4. Главное меню#8. Ввод суммы#e4#true");
```

```
                                fireTransitionFound(context, path, "4. Главное меню",
event, "4. Главное меню#8. Ввод суммы#e4#true");


                                fireComeToState(context, path, "8. Ввод суммы");

// 8. Ввод суммы [o1.z8]
                                fireBeforeOutputActionExecution(context, path, "4.
Главное меню#8. Ввод суммы#e4#true", "o1.z8");

                                o1.z8(context);

                                fireAfterOutputActionExecution(context, path, "4.
Главное меню#8. Ввод суммы#e4#true", "o1.z8");
                                return new StateMachineConfig("8. Ввод суммы");


                            case e2:

// 4. Главное меню->13. Возврат карты e2[true]/

                                fireTransitionCandidate(context, path, "4. Главное
меню", event, "4. Главное меню#13. Возврат карты#e2#true");


                                fireTransitionFound(context, path, "4. Главное меню",
event, "4. Главное меню#13. Возврат карты#e2#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "4.
Главное меню#13. Возврат карты#e2#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "4.
Главное меню#13. Возврат карты#e2#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            case e3:

// 4. Главное меню->5. Запрос Баланса e3[true]/

                                fireTransitionCandidate(context, path, "4. Главное
меню", event, "4. Главное меню#5. Запрос Баланса#e3#true");


                                fireTransitionFound(context, path, "4. Главное меню",
event, "4. Главное меню#5. Запрос Баланса#e3#true");


                                fireComeToState(context, path, "5. Запрос Баланса");

// 5. Запрос Баланса [o2.z5]
                                fireBeforeOutputActionExecution(context, path, "4.
Главное меню#5. Запрос Баланса#e3#true", "o2.z5");

                                o2.z5(context);

                                fireAfterOutputActionExecution(context, path, "4.
Главное меню#5. Запрос Баланса#e3#true", "o2.z5");
                                return new StateMachineConfig("5. Запрос Баланса");
```

```
                    default:

                        // transition not found
                        return config;
                }

            case _5__Запрос_Баланса:

                switch (e) {
                    case e15:

// 5. Запрос Баланса->13. Возврат карты e15[true]/

                        fireTransitionCandidate(context, path, "5. Запрос
Баланса", event, "5. Запрос Баланса#13. Возврат карты#e15#true");


                        fireTransitionFound(context, path, "5. Запрос
Баланса", event, "5. Запрос Баланса#13. Возврат карты#e15#true");


                        fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                        fireBeforeOutputActionExecution(context, path, "5.
Запрос Баланса#13. Возврат карты#e15#true", "o1.z13");

                        o1.z13(context);

                        fireAfterOutputActionExecution(context, path, "5.
Запрос Баланса#13. Возврат карты#e15#true", "o1.z13");
                        return new StateMachineConfig("13. Возврат карты");


                    case e12:

// 5. Запрос Баланса->6. Вывод на экран e12[true]/

                        fireTransitionCandidate(context, path, "5. Запрос
Баланса", event, "5. Запрос Баланса#6. Вывод на экран#e12#true");


                        fireTransitionFound(context, path, "5. Запрос
Баланса", event, "5. Запрос Баланса#6. Вывод на экран#e12#true");


                        fireComeToState(context, path, "6. Вывод на экран");

// 6. Вывод на экран [o1.z6]
                        fireBeforeOutputActionExecution(context, path, "5.
Запрос Баланса#6. Вывод на экран#e12#true", "o1.z6");

                        o1.z6(context);

                        fireAfterOutputActionExecution(context, path, "5.
Запрос Баланса#6. Вывод на экран#e12#true", "o1.z6");
                        return new StateMachineConfig("6. Вывод на экран");


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

```
                case _3__Авторизация:

                    switch (e) {
                        case e11:

// 3. Авторизация->13. Возврат карты e11[true]/

                                fireTransitionCandidate(context, path, "3.
Авторизация", event, "3. Авторизация#13. Возврат карты#e11#true");

                                fireTransitionFound(context, path, "3. Авторизация",
event, "3. Авторизация#13. Возврат карты#e11#true");

                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "3.
Авторизация#13. Возврат карты#e11#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "3.
Авторизация#13. Возврат карты#e11#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            case e10:

// 3. Авторизация->4. Главное меню e10[true]/

                                fireTransitionCandidate(context, path, "3.
Авторизация", event, "3. Авторизация#4. Главное меню#e10#true");


                                fireTransitionFound(context, path, "3. Авторизация",
event, "3. Авторизация#4. Главное меню#e10#true");


                                fireComeToState(context, path, "4. Главное меню");

// 4. Главное меню [o1.z4]
                                fireBeforeOutputActionExecution(context, path, "3.
Авторизация#4. Главное меню#e10#true", "o1.z4");

                                o1.z4(context);

                                fireAfterOutputActionExecution(context, path, "3.
Авторизация#4. Главное меню#e10#true", "o1.z4");
                                return new StateMachineConfig("4. Главное меню");


                            default:


                                // transition not found
                                return config;
                    }

                case _11__Печатать_чек:


                    switch (e) {
                        case e2:

// 11. Печатать чек->13. Возврат карты e2[true]/
```

```
                                fireTransitionCandidate(context, path, "11. Печатать
чек", event, "11. Печатать чек#13. Возврат карты#e2#true");


                                fireTransitionFound(context, path, "11. Печатать чек",
event, "11. Печатать чек#13. Возврат карты#e2#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "11.
Печатать чек#13. Возврат карты#e2#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "11.
Печатать чек#13. Возврат карты#e2#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            case e3:

// 11. Печатать чек->12. Печать чека 2 e3[true]/

                                fireTransitionCandidate(context, path, "11. Печатать
чек", event, "11. Печатать чек#12. Печать чека 2#e3#true");


                                fireTransitionFound(context, path, "11. Печатать чек",
event, "11. Печатать чек#12. Печать чека 2#e3#true");


                                fireComeToState(context, path, "12. Печать чека 2");

// 12. Печать чека 2 [o1.z12]
                                fireBeforeOutputActionExecution(context, path, "11.
Печатать чек#12. Печать чека 2#e3#true", "o1.z12");

                                o1.z12(context);

                                fireAfterOutputActionExecution(context, path, "11.
Печатать чек#12. Печать чека 2#e3#true", "o1.z12");
                                return new StateMachineConfig("12. Печать чека 2");


                            default:


                                // transition not found
                                return config;
                        }

                    case _13__Возврат_карты:


                        switch (e) {
                            case e7:

// 13. Возврат карты->1. Вставьте карту e7[true]/

                                fireTransitionCandidate(context, path, "13. Возврат
карты", event, "13. Возврат карты#1. Вставьте карту#e7#true");
```

```
                                fireTransitionFound(context, path, "13. Возврат
карты", event, "13. Возврат карты#1. Вставьте карту#e7#true");


                                fireComeToState(context, path, "1. Вставьте карту");

// 1. Вставьте карту [o1.z1]
                                fireBeforeOutputActionExecution(context, path, "13.
Возврат карты#1. Вставьте карту#e7#true", "o1.z1");

                                o1.z1(context);

                                fireAfterOutputActionExecution(context, path, "13.
Возврат карты#1. Вставьте карту#e7#true", "o1.z1");
                                return new StateMachineConfig("1. Вставьте карту");


                            default:


                                // transition not found
                                return config;
                        }

                    case _12__Печать_чека_2:


                        switch (e) {
                            case e8:

// 12. Печать чека 2->13. Возврат карты e8[true]/

                                fireTransitionCandidate(context, path, "12. Печать
чека 2", event, "12. Печать чека 2#13. Возврат карты#e8#true");


                                fireTransitionFound(context, path, "12. Печать чека
2", event, "12. Печать чека 2#13. Возврат карты#e8#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "12.
Печать чека 2#13. Возврат карты#e8#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "12.
Печать чека 2#13. Возврат карты#e8#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            default:


                                // transition not found
                                return config;
                        }

                    case _2__Ввод_pin_кода:


                        switch (e) {
                            case e4:
```

```
// 2. Ввод pin кода->3. Авторизация e4[true]/

                                fireTransitionCandidate(context, path, "2. Ввод pin
кода", event, "2. Ввод pin кода#3. Авторизация#e4#true");


                                fireTransitionFound(context, path, "2. Ввод pin кода",
event, "2. Ввод pin кода#3. Авторизация#e4#true");


                                fireComeToState(context, path, "3. Авторизация");

// 3. Авторизация [o2.z3]
                                fireBeforeOutputActionExecution(context, path, "2.
Ввод pin кода#3. Авторизация#e4#true", "o2.z3");

                                o2.z3(context);

                                fireAfterOutputActionExecution(context, path, "2. Ввод
pin кода#3. Авторизация#e4#true", "o2.z3");
                                return new StateMachineConfig("3. Авторизация");


                            case e2:

// 2. Ввод pin кода->13. Возврат карты e2[true]/

                                fireTransitionCandidate(context, path, "2. Ввод pin
кода", event, "2. Ввод pin кода#13. Возврат карты#e2#true");


                                fireTransitionFound(context, path, "2. Ввод pin кода",
event, "2. Ввод pin кода#13. Возврат карты#e2#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "2.
Ввод pin кода#13. Возврат карты#e2#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "2. Ввод
pin кода#13. Возврат карты#e2#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            default:


                                // transition not found
                                return config;
                        }

                    case _8__Ввод_суммы:


                        switch (e) {
                            case e4:

// 8. Ввод суммы->9. Запрос денег e4[true]/

                                fireTransitionCandidate(context, path, "8. Ввод
суммы", event, "8. Ввод суммы#9. Запрос денег#e4#true");
```

```
                                fireTransitionFound(context, path, "8. Ввод суммы",
event, "8. Ввод суммы#9. Запрос денег#e4#true");


                                fireComeToState(context, path, "9. Запрос денег");

// 9. Запрос денег [o2.z9]
                                fireBeforeOutputActionExecution(context, path, "8.
Ввод суммы#9. Запрос денег#e4#true", "o2.z9");

                                o2.z9(context);

                                fireAfterOutputActionExecution(context, path, "8. Ввод
суммы#9. Запрос денег#e4#true", "o2.z9");
                                return new StateMachineConfig("9. Запрос денег");


                            case e2:

// 8. Ввод суммы->13. Возврат карты e2[true]/

                                fireTransitionCandidate(context, path, "8. Ввод
суммы", event, "8. Ввод суммы#13. Возврат карты#e2#true");


                                fireTransitionFound(context, path, "8. Ввод суммы",
event, "8. Ввод суммы#13. Возврат карты#e2#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "8.
Ввод суммы#13. Возврат карты#e2#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "8. Ввод
суммы#13. Возврат карты#e2#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            default:


                                // transition not found
                                return config;
                        }

                    case _10__Выдача_денег:


                        switch (e) {
                            case e9:

// 10. Выдача денег->11. Печатать чек e9[true]/

                                fireTransitionCandidate(context, path, "10. Выдача
денег", event, "10. Выдача денег#11. Печатать чек#e9#true");


                                fireTransitionFound(context, path, "10. Выдача денег",
event, "10. Выдача денег#11. Печатать чек#e9#true");
```

```
                                fireComeToState(context, path, "11. Печатать чек");

// 11. Печатать чек [o1.z11]
                                fireBeforeOutputActionExecution(context, path, "10.
Выдача денег#11. Печатать чек#e9#true", "o1.z11");

                                o1.z11(context);

                                fireAfterOutputActionExecution(context, path, "10.
Выдача денег#11. Печатать чек#e9#true", "o1.z11");
                                return new StateMachineConfig("11. Печатать чек");


                            default:


                                // transition not found
                                return config;
                        }

                    case _9__Запрос_денег:


                        switch (e) {
                            case e13:

// 9. Запрос денег->10. Выдача денег e13[true]/

                                fireTransitionCandidate(context, path, "9. Запрос
денег", event, "9. Запрос денег#10. Выдача денег#e13#true");


                                fireTransitionFound(context, path, "9. Запрос денег",
event, "9. Запрос денег#10. Выдача денег#e13#true");


                                fireComeToState(context, path, "10. Выдача денег");

// 10. Выдача денег [o1.z10]
                                fireBeforeOutputActionExecution(context, path, "9.
Запрос денег#10. Выдача денег#e13#true", "o1.z10");

                                o1.z10(context);

                                fireAfterOutputActionExecution(context, path, "9.
Запрос денег#10. Выдача денег#e13#true", "o1.z10");
                                return new StateMachineConfig("10. Выдача денег");


                            case e14:

// 9. Запрос денег->13. Возврат карты e14[true]/

                                fireTransitionCandidate(context, path, "9. Запрос
денег", event, "9. Запрос денег#13. Возврат карты#e14#true");


                                fireTransitionFound(context, path, "9. Запрос денег",
event, "9. Запрос денег#13. Возврат карты#e14#true");


                                fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                                fireBeforeOutputActionExecution(context, path, "9.
Запрос денег#13. Возврат карты#e14#true", "o1.z13");
```

```
                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "9.
Запрос денег#13. Возврат карты#e14#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            default:


                                // transition not found
                                return config;
                        }

                    case _1__Вставьте_карту:


                        switch (e) {
                            case e0:

// 1. Вставьте карту->s2 e0[true]/o1.z0

                                fireTransitionCandidate(context, path, "1. Вставьте
карту", event, "1. Вставьте карту#s2#e0#true");


                                fireTransitionFound(context, path, "1. Вставьте
карту", event, "1. Вставьте карту#s2#e0#true");

                                fireBeforeOutputActionExecution(context, path, "1.
Вставьте карту#s2#e0#true", "o1.z0");

                                o1.z0(context);

                                fireAfterOutputActionExecution(context, path, "1.
Вставьте карту#s2#e0#true", "o1.z0");

                                fireComeToState(context, path, "s2");

// s2 []
                                return new StateMachineConfig("s2");


                            case e6:

// 1. Вставьте карту->2. Ввод pin кода e6[true]/

                                fireTransitionCandidate(context, path, "1. Вставьте
карту", event, "1. Вставьте карту#2. Ввод pin кода#e6#true");


                                fireTransitionFound(context, path, "1. Вставьте
карту", event, "1. Вставьте карту#2. Ввод pin кода#e6#true");


                                fireComeToState(context, path, "2. Ввод pin кода");

// 2. Ввод pin кода [o1.z2]
                                fireBeforeOutputActionExecution(context, path, "1.
Вставьте карту#2. Ввод pin кода#e6#true", "o1.z2");

                                o1.z2(context);

                                fireAfterOutputActionExecution(context, path, "1.
Вставьте карту#2. Ввод pin кода#e6#true", "o1.z2");
```

```
                        return new StateMachineConfig("2. Ввод pin кода");

                    default:

                        // transition not found
                        return config;
                }

            case _7__Печать_чека_1:

                switch (e) {
                    case e8:

// 7. Печать чека 1->13. Возврат карты e8[true]/

                        fireTransitionCandidate(context, path, "7. Печать чека
1", event, "7. Печать чека 1#13. Возврат карты#e8#true");

                        fireTransitionFound(context, path, "7. Печать чека 1",
event, "7. Печать чека 1#13. Возврат карты#e8#true");

                        fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
                        fireBeforeOutputActionExecution(context, path, "7.
Печать чека 1#13. Возврат карты#e8#true", "o1.z13");

                        o1.z13(context);

                        fireAfterOutputActionExecution(context, path, "7.
Печать чека 1#13. Возврат карты#e8#true", "o1.z13");
                        return new StateMachineConfig("13. Возврат карты");

                    default:

                        // transition not found
                        return config;
                }

            case _6__Вывод_на_экран:

                switch (e) {
                    case e4:

// 6. Вывод на экран->13. Возврат карты e4[true]/

                        fireTransitionCandidate(context, path, "6. Вывод на
экран", event, "6. Вывод на экран#13. Возврат карты#e4#true");

                        fireTransitionFound(context, path, "6. Вывод на
экран", event, "6. Вывод на экран#13. Возврат карты#e4#true");

                        fireComeToState(context, path, "13. Возврат карты");

// 13. Возврат карты [o1.z13]
```

```
                                fireBeforeOutputActionExecution(context, path, "6.
Вывод на экран#13. Возврат карты#e4#true", "o1.z13");

                                o1.z13(context);

                                fireAfterOutputActionExecution(context, path, "6.
Вывод на экран#13. Возврат карты#e4#true", "o1.z13");
                                return new StateMachineConfig("13. Возврат карты");


                            case e3:

// 6. Вывод на экран->7. Печать чека 1 e3[true]/

                                fireTransitionCandidate(context, path, "6. Вывод на
экран", event, "6. Вывод на экран#7. Печать чека 1#e3#true");


                                fireTransitionFound(context, path, "6. Вывод на
экран", event, "6. Вывод на экран#7. Печать чека 1#e3#true");


                                fireComeToState(context, path, "7. Печать чека 1");

// 7. Печать чека 1 [o1.z7]
                                fireBeforeOutputActionExecution(context, path, "6.
Вывод на экран#7. Печать чека 1#e3#true", "o1.z7");

                                o1.z7(context);

                                fireAfterOutputActionExecution(context, path, "6.
Вывод на экран#7. Печать чека 1#e3#true", "o1.z7");
                                return new StateMachineConfig("7. Печать чека 1");


                            default:


                                // transition not found
                                return config;
                        }

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


    }


    private class AServerEventProcessor {

        // states
        private static final int Top = 1;
        private static final int s1 = 2;
        private static final int Ответ_клиенту = 3;
        private static final int s2 = 4;
        private static final int Снятие_денег = 5;
        private static final int Авторизация = 6;
        private static final int Баланс = 7;
        private static final int Чтение_запроса = 8;

        private int decodeState(String state) {
```

```
            if ("Top".equals(state)) {
                return Top;
            } else if ("s1".equals(state)) {
                return s1;
            } else if ("Ответ клиенту".equals(state)) {
                return Ответ_клиенту;
            } else if ("s2".equals(state)) {
                return s2;
            } else if ("Снятие денег".equals(state)) {
                return Снятие_денег;
            } else if ("Авторизация".equals(state)) {
                return Авторизация;
            } else if ("Баланс".equals(state)) {
                return Баланс;
            } else if ("Чтение запроса".equals(state)) {
                return Чтение_запроса;
            }

            return -1;
        }

        // events
        private static final int e24 = 1;
        private static final int e22 = 2;
        private static final int e21 = 3;
        private static final int e23 = 4;

        private int decodeEvent(String event) {

            if ("e24".equals(event)) {
                return e24;
            } else if ("e22".equals(event)) {
                return e22;
            } else if ("e21".equals(event)) {
                return e21;
            } else if ("e23".equals(event)) {
                return e23;
            }

            return -1;
        }

        private callserver.ServerReply o3;

        private void init(ControlledObjectsMap controlledObjectsMap) {
            o3 = (callserver.ServerReply)
controlledObjectsMap.getControlledObject("o3");
        }

        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;
        }

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

```
            while (true) {
                switch (state) {
                    case s1:

                        return;
                    case Ответ_клиенту:

                        return;
                    case s2:

                        return;
                    case Снятие_денег:

                        return;
                    case Авторизация:

                        return;
                    case Баланс:

                        return;
                    case Чтение_запроса:

                        return;
                    default:
                        throw new EventProcessorException("State with name [" +
config.getActiveState() + "] is unknown for state machine [AServer]");
                }
            }
        }

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

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

            switch (s) {
                case Top:


                    fireComeToState(context, path, "s1");

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


                    fireComeToState(context, path, "Чтение запроса");

                    // Чтение запроса [o3.z0]
                    fireBeforeOutputActionExecution(context, path, "s1#Чтение
запроса##true", "o3.z0");

                    o3.z0(context);

                    fireAfterOutputActionExecution(context, path, "s1#Чтение
запроса##true", "o3.z0");

                    return new StateMachineConfig("Чтение запроса");
            }

            return config;
        }
```

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


            BitSet calculatedInputActions = new BitSet(0);

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

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


                        switch (e) {
                            default:

// Ответ клиенту->s2 *[true]/

                                fireTransitionCandidate(context, path, "Ответ
клиенту", event, "Ответ клиенту#s2#*#true");


                                fireTransitionFound(context, path, "Ответ клиенту",
event, "Ответ клиенту#s2#*#true");


                                fireComeToState(context, path, "s2");

// s2 []
                                return new StateMachineConfig("s2");


                        }

                    case Снятие_денег:


                        switch (e) {
                            case e24:

// Снятие денег->Ответ клиенту e24[true]/

                                fireTransitionCandidate(context, path, "Снятие денег",
event, "Снятие денег#Ответ клиенту#e24#true");


                                fireTransitionFound(context, path, "Снятие денег",
event, "Снятие денег#Ответ клиенту#e24#true");


                                fireComeToState(context, path, "Ответ клиенту");

// Ответ клиенту [o3.z4]
                                fireBeforeOutputActionExecution(context, path, "Снятие
денег#Ответ клиенту#e24#true", "o3.z4");

                                o3.z4(context);

                                fireAfterOutputActionExecution(context, path, "Снятие
денег#Ответ клиенту#e24#true", "o3.z4");
                                return new StateMachineConfig("Ответ клиенту");


                            default:
```

```
                                // transition not found
                                return config;
                            }

                        case Авторизация:


                            switch (e) {
                                case e24:

// Авторизация->Ответ клиенту e24[true]/

                                fireTransitionCandidate(context, path, "Авторизация",
event, "Авторизация#Ответ клиенту#e24#true");


                                fireTransitionFound(context, path, "Авторизация",
event, "Авторизация#Ответ клиенту#e24#true");


                                fireComeToState(context, path, "Ответ клиенту");

// Ответ клиенту [o3.z4]
                                fireBeforeOutputActionExecution(context, path,
"Авторизация#Ответ клиенту#e24#true", "o3.z4");

                                o3.z4(context);

                                fireAfterOutputActionExecution(context, path,
"Авторизация#Ответ клиенту#e24#true", "o3.z4");
                                return new StateMachineConfig("Ответ клиенту");


                                default:


                                // transition not found
                                return config;
                            }

                        case Баланс:


                            switch (e) {
                                case e24:

// Баланс->Ответ клиенту e24[true]/

                                fireTransitionCandidate(context, path, "Баланс",
event, "Баланс#Ответ клиенту#e24#true");


                                fireTransitionFound(context, path, "Баланс", event,
"Баланс#Ответ клиенту#e24#true");


                                fireComeToState(context, path, "Ответ клиенту");

// Ответ клиенту [o3.z4]
                                fireBeforeOutputActionExecution(context, path,
"Баланс#Ответ клиенту#e24#true", "o3.z4");

                                o3.z4(context);
```

```
                                fireAfterOutputActionExecution(context, path,
"Баланс#Ответ клиенту#e24#true", "o3.z4");
                                return new StateMachineConfig("Ответ клиенту");


                            default:


                                // transition not found
                                return config;
                        }

                    case Чтение_запроса:


                        switch (e) {
                            case e22:

// Чтение запроса->Баланс e22[true]/

                                fireTransitionCandidate(context, path, "Чтение
запроса", event, "Чтение запроса#Баланс#e22#true");


                                fireTransitionFound(context, path, "Чтение запроса",
event, "Чтение запроса#Баланс#e22#true");


                                fireComeToState(context, path, "Баланс");

// Баланс [o3.z2]
                                fireBeforeOutputActionExecution(context, path, "Чтение
запроса#Баланс#e22#true", "o3.z2");

                                o3.z2(context);

                                fireAfterOutputActionExecution(context, path, "Чтение
запроса#Баланс#e22#true", "o3.z2");
                                return new StateMachineConfig("Баланс");


                            case e21:

// Чтение запроса->Авторизация e21[true]/

                                fireTransitionCandidate(context, path, "Чтение
запроса", event, "Чтение запроса#Авторизация#e21#true");


                                fireTransitionFound(context, path, "Чтение запроса",
event, "Чтение запроса#Авторизация#e21#true");


                                fireComeToState(context, path, "Авторизация");

// Авторизация [o3.z1]
                                fireBeforeOutputActionExecution(context, path, "Чтение
запроса#Авторизация#e21#true", "o3.z1");

                                o3.z1(context);

                                fireAfterOutputActionExecution(context, path, "Чтение
запроса#Авторизация#e21#true", "o3.z1");
                                return new StateMachineConfig("Авторизация");
```

```
                            case e23:

// Чтение запроса->Снятие денег e23[true]/

                                fireTransitionCandidate(context, path, "Чтение
запроса", event, "Чтение запроса#Снятие денег#e23#true");


                                fireTransitionFound(context, path, "Чтение запроса",
event, "Чтение запроса#Снятие денег#e23#true");


                                fireComeToState(context, path, "Снятие денег");

// Снятие денег [o3.z3]
                                fireBeforeOutputActionExecution(context, path, "Чтение
запроса#Снятие денег#e23#true", "o3.z3");

                                o3.z3(context);

                                fireAfterOutputActionExecution(context, path, "Чтение
запроса#Снятие денег#e23#true", "o3.z3");
                                return new StateMachineConfig("Снятие денег");

                            default:
                                 // transition not found
                                return config;
                        }

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


    }

    private static boolean isInputActionCalculated(BitSet calculatedInputActions, int
k) {
        boolean b = calculatedInputActions.get(k);

        if (!b) {
            calculatedInputActions.set(k);
        }

        return b;
    }

}
```

4.2. *BankomatRun*

```
/**
*  Запускает программу
*/

import client.*;
import server.*;
import com.evelopers.unimod.runtime.*;
import com.evelopers.unimod.runtime.logger.*;
import com.evelopers.unimod.runtime.context.*;
```

```
import org.apache.commons.logging.*;
import org.apache.commons.logging.impl.*;
import com.evelopers.common.exception.*;

public class BankomatRun {
    public static void main(String[] args) {
        BankomatRun a = new BankomatRun();
        a.run();
    }

    private class ExitListener
            extends AbstractEventProcessorListener {
        private boolean done;

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

    private void run() {
        ModelEngine me;
        try {
            ControlledObjectsManager b = new ControlledObjectsManager() {
                FormPainter o1 = new FormPainter();
                ServerQuery o2 = new ServerQuery();
                ServerReply o3 = new ServerReply();

                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;
                    return null;
                }
            };
            me = ModelEngine.createStandAlone(new QueuedHandler(),
                    new Model1EventProcessor(), b,
                    new EventProvidersManager() {
                        HardwareEventProvider ep1 = new HardwareEventProvider();
                        HumanEventProvider ep2 = new HumanEventProvider();
                        ServerEventProvider ep3 = new ServerEventProvider();
                        ClientEventProvider ep4 = new ClientEventProvider();

                        public void init(ModelEngine engine) throws CommonException {
                            ep1.init(engine);
                            ep2.init(engine);
                            ep3.init(engine);
                            ep4.init(engine);
                        }

                        public void dispose() {
                            ep1.dispose();
                            ep2.dispose();
                            ep3.dispose();
                            ep4.dispose();
                        }

                        public EventProvider getEventProvider(String epName) {
                            if (epName.equals("ep1")) return ep1;
```

```
                            if (epName.equals("ep2")) return ep2;
                            if (epName.equals("ep3")) return ep3;
                            if (epName.equals("ep4")) return ep4;
                            return null;
                        }
                    });
            me.getEventProcessor().addEventProcessorListener(new
SimpleLogger(LogFactory.getLog(SimpleLogger.class)));
            ExitListener l1 = new ExitListener();
            me.getEventProcessor().addEventProcessorListener(l1);
            me.start();

            // wait for final state
            while (!l1.done) {
                Thread.sleep(10);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```