

**Министерство образования и науки Украины  
Донбасская государственная машиностроительная академия**

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

**к компьютерному практикуму**

по дисциплине

**«ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ СЛОЖНЫХ  
СИСТЕМ»**

(для студентов специальности 151)

Утверждено  
на заседании  
методического совета  
Протокол №     от

Краматорск 2018

## УДК 004.438.2

Методические указания к компьютерному практикуму по дисциплине "Технология программирования сложных систем". (для студентов специальности 151) / Сост. А. А. Сердюк. – Краматорск: ДГМА, 2018 – 89 с.

Приведено описание интерфейса инструментального средства визуального моделирования и автоматической разработки программного обеспечения Rational Rose. Изложены приемы визуализации, определения, документирования и конструирования компонентов программной системы. Даны методические рекомендации по разработке моделей и генерированию программного кода.

Составитель	А. А. Сердюк, доц.,
Отв. за выпуск	С. П. Сус, доц.

## СОДЕРЖАНИЕ

1 ИЗУЧЕНИЕ РАБОЧЕГО ИНТЕРФЕЙСА CASE-СРЕДСТВА RATIONAL ROSE	4
2 РАЗРАБОТКА ДИАГРАММ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ	21
3 КОНСТРУИРОВАНИЕ КЛАССОВ	31
4 РАЗРАБОТКА ДИАГРАММ ВЗАИМОДЕЙСТВИЯ	40
5 СОЗДАНИЕ СВЯЗЕЙ МЕЖДУ ОБЪЕКТАМИ, КЛАССАМИ И ПАКЕТАМИ	46
6 СОЗДАНИЕ АТТРИБУТОВ И ОПЕРАЦИЙ ДЛЯ ОПИСАНИЯ СТРУКТУРЫ И ПОВЕДЕНИЯ СИСТЕМЫ	52
7 РАЗРАБОТКА ДИАГРАММ СОСТОЯНИЙ	60
8 РАЗРАБОТКА ДИАГРАММ ДЕЙСТВИЙ (ДЕЯТЕЛЬНОСТИ)	66

# 1 ИЗУЧЕНИЕ РАБОЧЕГО ИНТЕРФЕЙСА CASE-СРЕДСТВА RATIONAL ROSE

*Цель работы:* освоение интерфейса и приемов работы с программным продуктом **Rational Rose**.

## 1.1 Общая характеристика CASE-средства IBM Rational Rose 2003

В настоящее время CASE-средство **IBM Rational Rose 2003** (release 2003.06.00) представляет собой современный интегрированный инструментарий для анализа, моделирования, проектирования архитектуры и разработки программных систем.

В рамках общего продукта IBM Rational Rose существуют различные варианты этого средства, отличающиеся между собой диапазоном предоставляемых возможностей. Базовым средством в настоящее время является **IBM Rational Rose Enterprise Edition**, которое обладает наиболее полными возможностями.

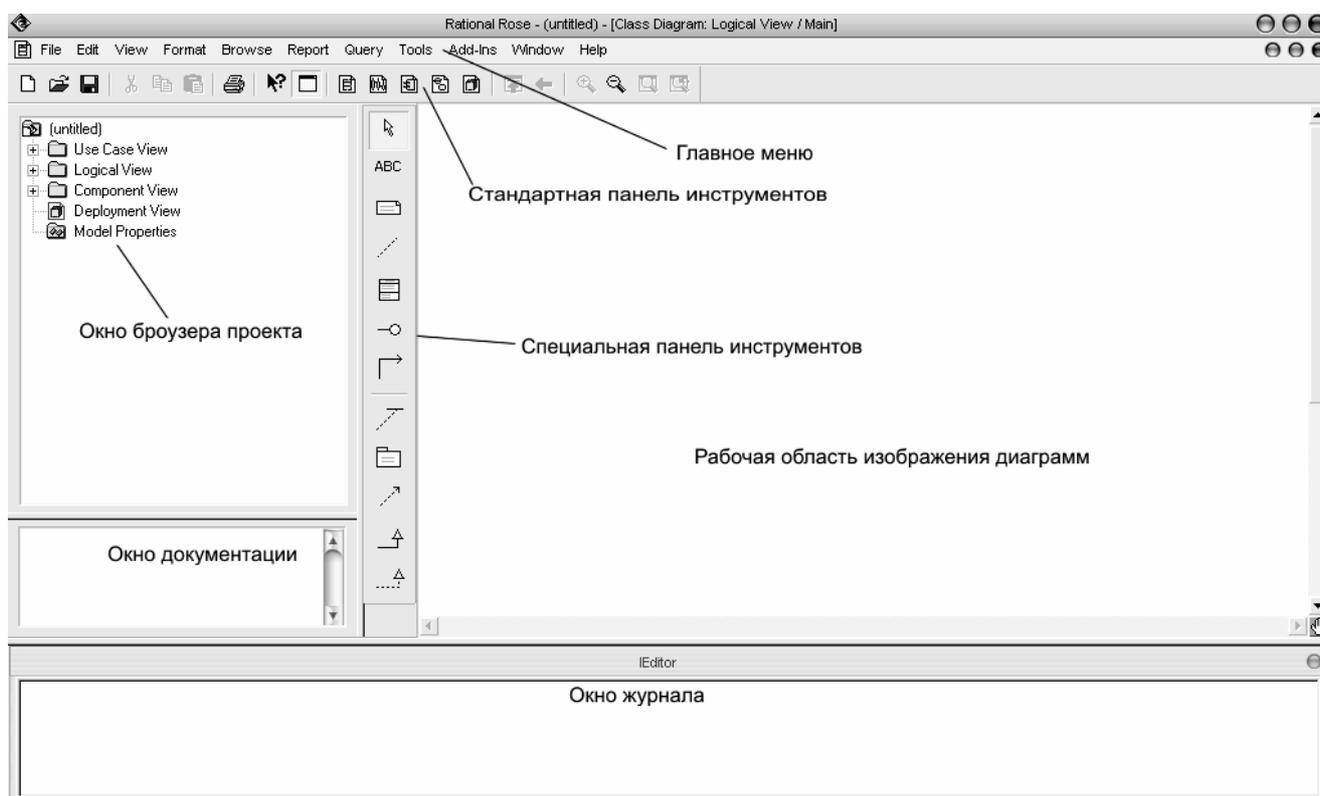
Функциональные особенности этой программы заключаются в следующем:

- интеграция с **MS Visual Studio 6**, которая включает поддержку на уровне прямой и обратной генерации кодов и диаграмм **Visual Basic** и **Visual C++** с использованием **Microsoft ATL (Active Template Library)**, **Web-Классов**, **DHTML** и протоколов доступа к различным базам данных;
- непосредственная работа (инжиниринг и реинжиниринг) с исполняемыми модулями и библиотеками форматов **EXE, DLL, TLB, OCX**.
- поддержка технологий **MTS (Microsoft Transaction Server)** и **ADO (ActiveX Data Objects)** на уровне шаблонов и исходного кода, а также элементов технологии Microsoft – **COM+ (DCOM)**;
- полная поддержка компонентов **CORBA** и **J2EE**, включая реализацию технологии компонентной разработки приложений **CBD (Component-Based Development)**, языка определения интерфейса **IDL (Interface Definition Language)** и языка определения данных **DDL (Data Definition Language)**;
- полная поддержка среды разработки Java-приложений, включая прямую и обратную генерацию классов **Java** формата **JAR**, а также работу с файлами формата **CAB** и **ZIP**.

## 1.2 Особенности рабочего интерфейса программы Rational Rose 2003

В CASE-средстве **IBM Rational Rose 2003** реализованы общепринятые стандарты на рабочий интерфейс программы, аналогично известным средам визуального программирования. Рабочий интерфейс программы **Rational Rose** (рис. 1.1) состоит из различных элементов, основными из которых являются:

- главное меню;
- стандартная панель инструментов;
- специальная панель инструментов;
- окно браузера (браузера) проекта;
- рабочая область изображения диаграммы или окно диаграммы;
- окно документации;
- окно журнала.



*Рисунок 1.1 – Общий вид рабочего интерфейса CASE-средства IBM Rational Rose 2003*

### 1.3 Главное меню и стандартная панель инструментов

Главное меню программы **IBM Rational Rose 2003** выполнено в общепринятом стандарте и имеет следующий вид (рис. 1.2).

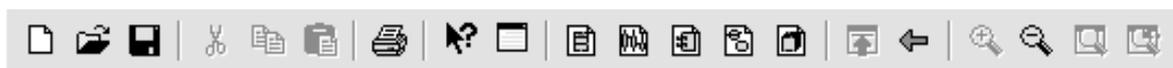


*Рисунок 1.2 – Внешний вид главного меню программы*

Отдельные пункты меню объединяют сходные операции, относящиеся ко всему проекту в целом. Некоторые из пунктов меню содержат хорошо знакомые

операции, такие как открытие проекта, вывод на печать диаграмм, копирование в буфер и вставка из буфера различных элементов диаграмм. Другие операции специфичны, что может потребовать дополнительных усилий для их изучения (свойства операций генерации программного кода или проверки согласованности моделей).

Стандартная панель инструментов располагается ниже строки главного меню и имеет вид, показанный на рисунке 1.3. Стандартная панель инструментов обеспечивает быстрый доступ к тем командам меню, которые выполняются разработчиками наиболее часто.



*Рисунок 1.3 – Внешний вид стандартной панели инструментов*

Пользователь может настроить внешний вид этой панели по своему усмотрению. Для этого необходимо выполнить операцию главного меню: **Tools** ► **Options** (Инструменты ► Параметры), открыть в появившемся диалоговом окне вкладку **Toolbars** (Панели инструментов) и нажать кнопку **Standard** (Стандартная). В дополнительно открытом окне можно переносить требуемые кнопки из левого списка в правый список, а ненужные кнопки – из правого списка в левый. Данным способом можно показать или скрыть различные кнопки инструментов, а также изменить их размер. Назначение отдельных кнопок стандартной панели инструментов приводится далее при рассмотрении операций главного меню.

#### **1.4 Назначение операций главного меню File и Edit**

Рабочий интерфейс средства **IBM Rational Rose 2003** имеет главное меню, которое позволяет пользователю загружать и сохранять информацию во внешних файлах, изменять внешний вид элементов графического интерфейса, вызывать справочную информацию, а также другие диалоговые окна для работы с программой **IBM Rational Rose 2003**.

Операции главного меню **File** (Файл) позволяют создавать новые модели в нотации языка UML, загружать и сохранять разрабатываемую модель во внешнем файле, распечатывать на принтере разработанные диаграммы. Назначение операций этого пункта главного меню представлено в таблице 1.1.

Операции главного меню **Edit** (Редактирование) позволяют выполнять действия по редактированию элементов модели и их свойств, а также выполнять поиск элементов в рамках разрабатываемого проекта. Назначение операций этого пункта главного меню представлено в таблице 1.2.

Таблица 1.1 – Операции пункта главного меню **File** (Файл)

Название операции меню	Назначение операции
<b>New</b>	Создает новую модель с именем <b>untitled</b> (по умолчанию)
<b>Open</b>	Вызывает стандартное диалоговое окно открытия внешнего файла с диска. При этом открыть можно либо файл модели (расширение <b>.mdl</b> ), либо файл подмодели (расширение <b>.ptl</b> )
<b>Save</b>	Сохраняет модель во внешнем файле на диске
<b>Save As</b>	Сохраняет модель под другим именем
<b>Save Log As</b>	Сохраняет содержимое журнала ошибок во внешнем файле на диске с именем <b>error.log</b>
<b>AutoSave log</b>	Автоматически сохраняет содержимое журнала ошибок во внешнем файле с именем <b>error.log</b>
<b>Clear Log</b>	Очищает содержимое журнала
<b>Load Model Workspace</b>	Загружает рабочую область из внешнего файла на диске. При этом файл должен иметь расширение <b>.wsp</b>
<b>Save Model Workspace</b>	Сохраняет рабочую область модели во внешнем файле на диске с расширением <b>.wsp</b>
<b>Save Model Workspace As</b>	Сохраняет рабочую область модели во внешнем файле на диске с расширением <b>.wsp</b> под другим именем
<b>Units</b>	Загружает категорию элементов модели из внешнего файла на диске. При этом открыть можно файл с расширением <b>.cat</b>
<b>Import</b>	Позволяет импортировать информацию из внешних файлов различных форматов
<b>Export Model</b>	Позволяет экспортировать информацию о модели во внешний файл
<b>Update</b>	Позволяет вставить информацию обратного проектирования из внешнего файла с расширением <b>.red</b> в разрабатываемую модель
<b>Print</b>	Позволяет распечатать отдельные диаграммы и спецификации модели путем их выбора из диалогового окна
<b>Print Setup</b>	Позволяет настроить свойства печати
<b>Edit Path Map</b>	Вызывает окно задания путей доступа к файлам Rational Rose

Таблица 1.2 – Операции пункта главного меню **Edit** (Редактирование)

Название операции меню	Назначение операции
<b>Undo</b>	Отменяет выполнение последнего действия по удалению или перемещению элементов модели
<b>Redo</b>	Восстанавливает изображение диаграммы после отмены операции перемещения
<b>Cut</b>	Вырезает выделенный элемент модели и помещает его в буфер
<b>Copy</b>	Копирует выделенный элемент модели и помещает его в буфер
<b>Paste</b>	Вставляет элемент из буфера обмена
<b>Delete</b>	Удаляет выделенные элементы из текущей диаграммы, но не из модели
<b>Select All</b>	Выделяет все элементы на текущей диаграмме
<b>Delete from Model</b>	Удаляет все выделенные элементы из модели
<b>Relocate</b>	Позволяет перемещать или отменять перемещение классов, ассоциаций или компонентов из одного пакета в другой
<b>Find</b>	Вызывает меню поиска элемента
<b>Reassign</b>	Позволяет заменить выделенный элемент другим элементом модели
<b>Compartment</b>	Позволяет отображать дополнительную информацию об объектах, классах, актерах или пакетах
<b>Change Info</b>	Позволяет изменить тип выделенного элемента диаграммы

### 1.5 Назначение операций главного меню **View**, **Format** и **Browse**

Операции главного меню **View** (Вид) позволяют отображать на экране различные элементы рабочего интерфейса и изменять графическое представление диаграмм. Назначение операций этого пункта главного меню представлено в следующей таблице (табл. 1.3).

Таблица 1.3 – Операции пункта главного меню *View (Вид)*

Название операции меню	Назначение операции главного меню
<b>Toolbars</b>	Позволяет настроить внешний вид рабочего интерфейса системы IBM Rational Rose 2003 и содержит дополнительные подпункты: <b>Standard</b> – делает видимой/невидимой стандартную панель инструментов (рис. 1.3); <b>Toolbox</b> – делает видимой/невидимой специальную панель инструментов текущей активной диаграммы; <b>Configure</b> – вызывает диалоговое окно настройки параметров модели, открытое на вкладке настройки панелей инструментов
<b>Status Bar</b>	Делает видимой/невидимой строку состояния
<b>Documentation</b>	Делает видимым/невидимым окно документации
<b>Browser</b>	Делает видимым/невидимым браузер проекта
<b>Log</b>	Делает видимым/невидимым окно журнала
<b>Editor</b>	Делает видимым/невидимым встроенный текстовый редактор
<b>Time Stamp</b>	Включает/выключает режим отображения времени в записях журнала
<b>Zoom to Selection</b>	Изменяет масштаб изображения выделенных элементов модели, так чтобы они разместились в одном окне
<b>Zoom In</b>	Увеличивает масштаб изображения
<b>Zoom Out</b>	Уменьшает масштаб изображения
<b>Fit in Window</b>	Изменяет масштаб изображения всех элементов текущей диаграммы, так чтобы все они разместились в одном окне
<b>Undo Fit in Window</b>	Отменяет изменение масштаба изображения для размещения элементов в одном окне
<b>Page Breaks</b>	Разбивает текущую диаграмму на страницы для последующей печати
<b>Refresh</b>	Перерисовывает текущую диаграмму
<b>As Booch</b>	Изображает элементы в нотации Г. Буча
<b>AsOMT</b>	Изображает элементы в нотации OMT
<b>As Unified</b>	Изображает элементы в нотации UML

Операции главного меню **Format** (Формат) позволяют выполнять действия по изменению внешнего вида элементов модели на различных диаграммах. Назначение операций этого пункта главного меню представлено в таблице 1.4.

Таблица 1.4 – Операции пункта главного меню **Format** (Формат)

Название операции меню	Назначение операции главного меню
<b>Font Size</b>	Изменяет масштаб используемого шрифта
<b>Font</b>	Вызывает диалоговое окно выбора шрифта
<b>Line Color</b>	Вызывает диалоговое окно выбора цвета линий
<b>Fill Color</b>	Вызывает диалоговое окно выбора цвета для изображения графических элементов диаграмм
<b>Use Fill Color</b>	Включает/выключает режим отображения цвета для изображения графических элементов диаграмм
<b>Automatic Resize</b>	Включает/выключает режим автоматического изменения размеров графических элементов диаграмм для отображения текстовой информации об их свойствах
<b>Stereotype</b>	Позволяет выбрать способ изображения стереотипов выделенных элементов диаграммы и содержит подпункты: <b>None</b> – стереотип не показывается; <b>Label</b> – стереотип отображается в форме текста; <b>Decoration</b> – стереотип отображается в форме небольшой пиктограммы в правом верхнем углу графического элемента; <b>Icon</b> – элемент диаграммы отображается в форме специального графического стереотипа
<b>Stereotype Label</b>	Включает/выключает режим отображения текстовых стереотипов (ассоциаций, зависимостей и пр.) диаграммы
<b>Show Visibility</b>	Включает/выключает режим отображения кванторов видимости
<b>Show Compartment Stereotypes</b>	Включает/выключает режим отображения текстовых стереотипов атрибутов и операций выделенных классов
<b>Show Operation Signature</b>	Включает/выключает режим отображения сигнатуры операций выделенных классов
<b>Show All Attributes</b>	Делает видимыми/невидимыми атрибуты выделенных классов
<b>Show All Operations</b>	Делает видимыми/невидимыми операции выделенных классов

Продолжение табл. 1.4

Название операции меню	Назначение операции главного меню
<b>Suppress Attributes</b>	Делает видимой/невидимой секцию атрибутов выделенных классов. Скрывает секцию атрибутов даже в том случае, когда выбрана опция <b>Show All Attributes</b>
<b>Suppress Operations</b>	Делает видимой/невидимой секцию операций выделенных классов. Скрывает секцию операций даже в том случае, когда выбрана опция <b>Show All Operations</b>
<b>Line Style</b>	Позволяет выбрать способ графического изображения линий взаимосвязей и содержит дополнительные подпункты: <b>Rectilinear</b> – линия изображается в форме вертикальных и горизонтальных отрезков; <b>Oblique</b> – линия изображается в форме наклонных отрезков; <b>Toggle</b> – промежуточный вариант изображения линии
<b>Layout Diagram</b>	Позволяет автоматически разместить графические элементы в окне диаграммы с минимальным количеством пересечений и наложений соединительных линий
<b>Autosize All</b>	Позволяет автоматически изменить размеры графических элементов текущей диаграммы таким образом, чтобы текстовая информация помещалась внутри изображений соответствующих элементов
<b>Layout Selected Shapes</b>	Позволяет автоматически разместить выделенные графические элементы в окне диаграммы с минимальным количеством пересечений и наложений соединительных линий

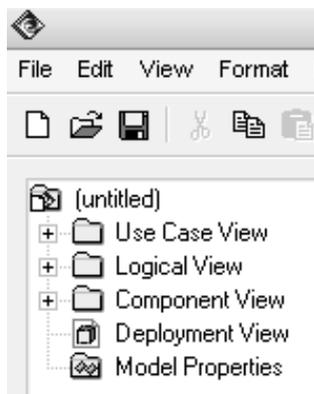
Операции главного меню **Browse** (Обзор) позволяют отображать рабочие окна с различными каноническими диаграммами разрабатываемой модели и вызывать диалоговые окна редактирования свойств отдельных элементов модели. Операции этого пункта главного меню приведены в таблице 1.5.

Таблица 1.5 – Назначение операций пункта меню **Browse** (Обзор)

Название операции меню	Назначение операции
<b>Use Case Diagram</b>	Позволяет отобразить в рабочем окне соответствующую диаграмму разрабатываемой модели
<b>Class Diagram</b>	
<b>Component Diagram</b>	
<b>Deployment Diagram</b>	
<b>Interaction Diagram</b>	
<b>State Machine Diagram</b>	
<b>Expand</b>	Отображает в рабочем окне первую из диаграмм выделенного пакета модели
<b>Parent</b>	Отображает в рабочем окне родителя выделенной диаграммы модели
<b>Specification</b>	Вызывает диалоговое окно свойств выделенного элемента модели
<b>Top Level</b>	Отображает в рабочем окне диаграмму самого верхнего уровня для текущей диаграммы модели
<b>Referenced Item</b>	Отображает в рабочем окне диаграмму классов, содержащую класс для выделенного объекта модели
<b>Previous Diagram</b>	Отображает в рабочем окне предыдущую диаграмму модели
<b>Create Message Trace Diagram</b>	Позволяет создать диаграмму трассировки сообщений
<b>Top Level</b>	Отображает в рабочем окне диаграмму самого верхнего уровня для текущей диаграммы модели
<b>Referenced Item</b>	Отображает в рабочем окне диаграмму классов, содержащую класс для выделенного объекта модели
<b>Previous Diagram</b>	Отображает в рабочем окне предыдущую диаграмму модели
<b>Create Message Trace Diagram</b>	Позволяет создать диаграмму трассировки сообщений

## 1.6 Окно браузера проекта

Окно браузера проекта по умолчанию располагается в левой части рабочего интерфейса ниже стандартной панели инструментов и имеет вид, показанный на рисунке 1.4.



*Рисунок 1.4 – Внешний вид браузера проекта с иерархическим представлением его структуры*

Броузер проекта организует представления модели в виде иерархической структуры, которая упрощает навигацию и позволяет отыскать любой элемент модели в проекте. При этом самая верхняя строка браузера проекта содержит имя разрабатываемого проекта (по умолчанию – **untitled**). Любой элемент, который разработчик добавляет в модель, сразу отображается в окне браузера. Соответственно, выбрав элемент в окне браузера, мы можем его визуализировать в окне диаграммы или изменить его спецификацию.

Броузер проекта позволяет также организовывать элементы модели в пакеты и перемещать элементы между различными представлениями модели. При желании окно браузера можно расположить в другом месте рабочего интерфейса либо скрыть вовсе, используя для этого операцию **Browser** главного меню **View** (Вид). Можно также изменить размеры браузера.

Иерархическое представление структуры каждого разрабатываемого проекта организовано в форме 4-х представлений:

- **Use Case View** – представление вариантов использования, в котором содержатся диаграммы вариантов использования и их реализации в виде вариантов взаимодействия;
- **Logical View** – логическое представление, в котором содержатся диаграммы классов, диаграммы состояний и диаграммы деятельности;
- **Component View** – представление компонентов, в котором содержатся диаграммы компонентов разрабатываемой модели;
- **Deployment View** – представление развертывания, в котором содержится единственная диаграмма развертывания разрабатываемой модели.

При создании нового проекта иерархическая структура формируется программой автоматически.

## 1.7 Специальная панель инструментов и окно диаграммы

Специальная панель инструментов располагается между окном браузера и окном диаграммы в средней части рабочего интерфейса. По умолчанию предлагается панель инструментов для построения диаграммы классов (рис. 1.5).



*Рисунок 1.5 – Внешний вид специальной панели инструментов для диаграммы классов*

Расположение специальной панели инструментов можно изменять, переместив рамку панели в нужное место. Программа Rational Rose позволяет настраивать состав кнопок данной панели, добавляя или удаляя отдельные кнопки, соответствующие тем или иным инструментам. Названия кнопок данной панели всегда можно узнать из всплывающих подсказок, появляющихся после задержки указателя мыши над соответствующей кнопкой.

Внешний вид специальной панели инструментов зависит не только от выбора типа разрабатываемой диаграммы, но и от выбора графической нотации для изображения самих элементов этих диаграмм. В Rational Rose 2003 реализованы три таких нотации: UML, OMT и Booch (последние две нотации практически не используются на практике).

Окно диаграммы является основной графической областью программы Rational Rose, в которой визуализируются различные представления модели проекта. По умолчанию окно диаграммы располагается в правой части рабочего интерфейса, однако его расположение и размеры также можно изменить. Если не был использован мастер проектов, окно диаграммы представляет собой чистую область, не содержащую никаких элементов модели (рис. 1.1). При разработке диаграмм в окне диаграммы будут располагаться соответствующие графические элементы модели (рис. 1.6).

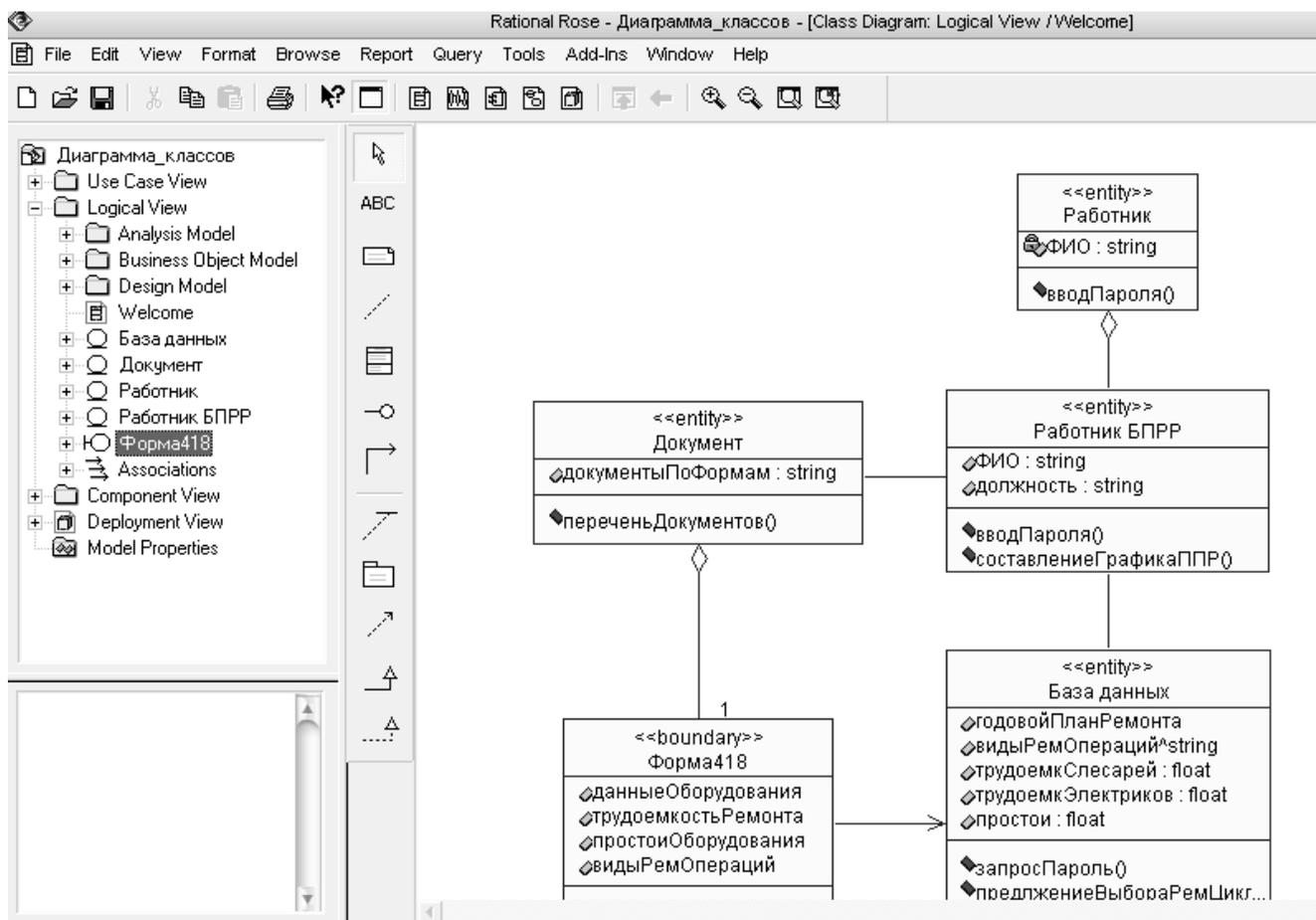


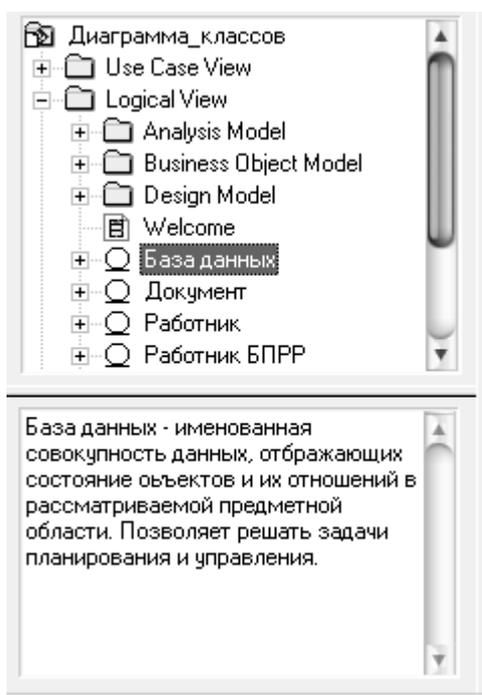
Рисунок 1.6 – Внешний вид окна диаграммы с диаграммой классов модели

Название диаграммы, которая является активной и располагается в данном окне, развернутом на всю область диаграммы, указывается в строке заголовка программы Rational Rose. Если же окно диаграммы не развернуто на всю область диаграммы, то название диаграммы указывается в строке заголовка окна диаграммы. Одновременно в графической области диаграмм могут присутствовать несколько окон диаграмм; при этом активной может быть только одна из них. Переключение между диаграммами можно осуществить выбором нужного представления на стандартной панели инструментов, а также с помощью выделения требуемой диаграммы в браузере проекта или с помощью операций главного меню **Window** (Окно). При активизации отдельного вида диаграммы изменяется внешний вид специальной панели инструментов, которая настраивается под конкретный вид диаграммы.

## 1.8 Окно документации и окно журнала

Окно документации по умолчанию должно присутствовать на экране после загрузки программы. Если по какой-то причине оно отсутствует, то его можно

отобразить через пункт меню **View ► Documentation** (Вид ► Документация), после чего окно документации появится ниже окна браузера проекта (рис. 1.7). Окно документации, как следует из его названия, предназначено для документирования элементов разрабатываемой модели. В него можно записывать различную текстовую информацию, и что важно – на русском языке. Эта информация при генерации программного кода преобразуется в комментарии и никак не влияет на логику выполнения программного кода.



*Рисунок 1.7 – Внешний вид окна документации (внизу) с информацией о классе “База данных”*

В окне документации активизируется та информация, которая относится к выделенному элементу диаграммы или к диаграмме в целом. Выделить элемент можно либо в окне браузера, либо в окне диаграммы. При добавлении нового элемента на диаграмму, например, класса, документация к нему является пустой (**No documentation**). Разработчик самостоятельно вносит необходимую пояснительную информацию, которая запоминается программой и может быть изменена в ходе работы над проектом. Размеры и положение окна документации можно изменять по своему усмотрению.

Окно журнала (**Log**) предназначено для автоматической записи различной служебной информации в ходе работы с программой. В журнале фиксируется время и характер выполняемых разработчиком действий, таких как обновление модели, настройка меню и панелей инструментов, а также сообщений об ошибках, возникающих при генерации программного кода. Окно журнала изображается поверх других окон в нижней области рабочего интерфейса программы (рис. 1.1).

Если окно журнала отсутствует на экране, то отобразить его можно с помощью операции главного меню **View ► Log** (Вид ► Журнал), для чего следует выставить отметку в соответствующей строке вложенного меню для данной операции. С целью увеличения размеров графической области диаграммы окно журнала чаще всего убирают с экрана, что можно выполнить с помощью кнопки закрытия этого окна в верхнем левом его углу или убрав отметку в соответствующей строке вложенного меню **View ► Log**.

## 1.9 Назначение операций главного меню **Report, Query и Tools**

Операции главного меню **Report** (Отчет) позволяют отображать различную информацию об элементах разрабатываемой модели и вызывать диалоговое окно выбора шаблона для генерации отчета о модели. Назначение операций этого пункта главного меню представлено в таблице 1.6.

Таблица 1.6 – Операции пункта главного меню **Report** (Отчет)

Название операции меню	Назначение операции главного меню
<b>Show Usage</b>	Отображает в диалоговом окне информацию об использовании выделенного элемента модели на различных диаграммах
<b>Show Instances</b>	Отображает в диалоговом окне информацию об использовании объектов выделенного класса модели на различных диаграммах
<b>Show Access Violations</b>	Отображает в диалоговом окне информацию о ссылках классов одного пакета на классы другого пакета при отсутствии соответствующей зависимости доступа или импорта между этими пакетами в модели
<b>SoDA Report</b>	Позволяет сгенерировать отчет о разрабатываемой модели в формате <b>MS Word</b> с использованием специального средства <b>IBM Rational SoDA</b>
<b>Show Participants in UC</b>	Отображает в диалоговом окне информацию о классах, компонентах и операциях, которые участвуют в реализации выделенного варианта использования модели на различных диаграммах

Операции главного меню **Query** (Запрос) позволяют добавлять существующие элементы разрабатываемой модели на редактируемую диаграмму, а также настраивать специальный фильтр отображения отношений между отдельными элементами модели. Назначение операций этого пункта главного меню представлено в таблице 1.7.

Таблица 1.7 – Операции пункта главного меню **Query** (Запрос)

Название операции	Назначение операции главного меню
<b>Add Classes</b>	Вызывает диалоговое окно с предложением добавить на текущую диаграмму классы, которые имеются в модели на различных диаграммах
<b>Add Use Cases</b>	Вызывает диалоговое окно с предложением добавить на текущую диаграмму варианты использования, которые имеются в модели на различных диаграммах
<b>Expand Selected Elements</b>	Вызывает диалоговое окно с предложением добавить на текущую диаграмму элементы модели, которые связаны с выделенным элементом на других диаграммах
<b>Hide Selected Elements</b>	Вызывает диалоговое окно с предложением удалить с текущей диаграммы элементы модели, которые связаны с выделенным элементом
<b>Filter Relationships</b>	Вызывает диалоговое окно, позволяющее включить/выключить режим отображения различных отношений на текущей диаграмме

Состав операций пункта главного меню **Tools** (Инструменты) зависит от установленных в программе Rational Rose конкретных расширений. Назначение операций этого пункта главного меню для типовой конфигурации программы представлено в таблице 1.8.

Таблица 1.8 – Операции пункта главного меню **Tools** (Инструменты) типовой конфигурации

Название операции	Назначение операции главного меню
<b>Model Properties</b>	Позволяет выполнить настройку свойств языка реализации для выделенного элемента модели и содержит подпункты: <b>Edit</b> – редактирование набора свойств; <b>View</b> – просмотр набора свойств; <b>Replace</b> – замена существующего набора свойств на новый, загружаемый из внешнего файла с расширением <b>.prp</b> или <b>.pty</b> ; <b>Export</b> – сохранение существующего набора свойств во внешнем файле с расширением <b>.prp</b> или <b>.pty</b> ; <b>Add</b> – добавление к существующему набору свойств нового набора свойств, загружаемого из внешнего файла с расширением <b>.prp</b> или <b>.pty</b> ; <b>Update</b> – обновление существующего набора свойств после его редактирования или дополнения

Продолжение табл. 1.8

Название операции	Назначение операции главного меню
<b>Check Model</b>	Проверяет разрабатываемую модель на наличие ошибок, информация о которых отображается в окне журнала
<b>Create</b>	Создает новый элемент модели из предлагаемого списка
<b>Options</b>	Вызывает диалоговое окно настройки параметров модели, открытое на вкладке <b>General</b>
<b>Open Script</b>	Вызывает стандартное диалоговое окно открытия внешнего файла, содержащего текст скрипта (файл с расширением <b>.ebs</b> ) для его редактирования в окне встроенного редактора скриптов
<b>New Script</b>	Открывает дополнительное окно встроенного редактора скриптов для создания, отладки, выполнения и сохранения нового скрипта во внешнем файле с расширением <b>.ebs</b>
<b>ANSI C++</b>	Позволяет выполнить настройку свойств языка программирования <b>ANSI C++</b> , выбранного в качестве языка реализации отдельных элементов модели
<b>CORBA</b>	Позволяет выполнить настройку свойств и спецификацию модели для генерации объектов <b>CORBA</b> для реализации отдельных элементов модели
<b>Java/J2EE</b>	Позволяет выполнить настройку свойств языка программирования <b>Java</b> , выбранного в качестве языка реализации отдельных элементов модели
<b>Oracle8</b>	Позволяет выполнить настройку свойств и спецификацию модели для генерации схем СУБД <b>Oracle 8</b> для отдельных элементов модели
<b>Quality Architect</b>	Позволяет выполнить настройку свойств и тестирование модели с помощью специального средства IBM Rational <b>Quality Architect</b>
<b>Rational Requisite Pro</b>	Позволяет выполнить настройку свойств модели для установления связей со специальным средством спецификации и управления требованиями
<b>Model Integrator</b>	Открывает рабочее окно специального средства интеграции моделей
<b>Web Publisher</b>	Позволяет выполнить настройку свойств модели для ее публикации в гипертекстовом формате
<b>TOPLink</b>	Вызывает мастер преобразования таблиц модели данных в классы языка программирования <b>Java</b> , выбранного в качестве языка реализации отдельных элементов модели
<b>COM</b>	Позволяет выполнить настройку свойств и спецификацию модели для генерации объектов <b>COM</b> с целью реализации отдельных элементов модели

Продолжение табл. 1.8

Название операции	Назначение операции главного меню
<b>Visual C++</b>	Позволяет выполнить настройку свойств и спецификацию модели для генерации программного кода <b>MS Visual C++</b> , выбранного в качестве языка реализации отдельных элементов модели
<b>Version Control</b>	Позволяет выполнить настройку свойств модели для установления связей со специальным средством управления и контроля версий модели
<b>Visual Basic</b>	Позволяет выполнить настройку свойств и спецификацию модели для генерации программного кода <b>MS Visual Basic</b> , выбранного в качестве языка реализации отдельных элементов модели
<b>XML_DTD</b>	Позволяет выполнить настройку свойств и спецификацию модели для ее публикации в формате расширяемого языка разметки <b>XML</b>
<b>Class Wizard</b>	Вызывает мастер создания нового класса и его размещения на выбранной диаграмме модели

### 1.10 Сведения об отчете

Отчет по данной работе не составляется.

Освоение и защита работы оцениваются по результатам тестирования. Содержание тестов определяется требованиями *знаний особенностей* применения программы Rational Rose и *умений использовать* это CASE-средство для моделирования программ и создания программного кода.

## 2 РАЗРАБОТКА ДИАГРАММ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

*Цель работы:* освоение приемов и методики разработки диаграмм вариантов использования с применением программы Rational Rose.

### 2.1 Рекомендации по разработке вариантов использования

Проектирование начинается с фазы исследований. На первом этапе должен быть проведен тщательный анализ предметной области с целью формулирования требования к программной системе.

Задачей этапа анализа предметной области является составление описания в контексте “Что должна делать система?”, подводящем к достижению цели – формулированию требований.

Характеристики поведения разрабатываемой системы фиксируются и документируются средствами модели, которая отображает функции продукта (*варианты использования – use case*), представляет окружение системы (множество *активных субъектов – actors*) и определяет связи между вариантами использования и активными субъектами (*диаграммы вариантов использования – use case diagrams*).

**Выбор вариантов использования.** Варианты использования (**use case**) позволяют моделировать диалог между активным субъектом и системой, отображая предоставляемые системой функции. Набор вариантов использования – это последовательность выполняемых системой транзакций, которая приводит к желаемому для активного субъекта результату.

При выборе вариантов использования системы необходимо сформулировать ответы на следующие вопросы.

1. Какие активные субъекты участвуют в работе системы, и какие задачи решает каждый из субъектов?
2. Какие данные в контексте системы тот или иной субъект может создавать, сохранять, изменять, удалять или считывать?
3. Какие варианты использования гарантируют обработку данных?
4. Должны ли активные субъекты сообщать системе о каких-либо непредвиденных внешних обстоятельствах?
5. Имеет ли субъект право получать информацию об определенных событиях, происходящих в системе?
6. Какие варианты использования связаны с поддержкой и администрированием системы?
7. Удовлетворяются ли вариантами использования все функциональные требования, предъявляемые к системе?

**Выбор активных субъектов.** Каждый из внешних *активных субъектов* (**actors**) отождествляется с чем-то или с кем-то, взаимодействующим с системой. Активный субъект способен выполнять различные функции:

- только вводить данные в систему;
- только получать информацию из системы;
- взаимодействовать с системой в обоих направлениях.

Помощь в отыскании активных субъектов могут дать ответы на следующие вопросы.

- Кто именно заинтересован в выполнении определенного требования?
- В каком подразделении организации должна использоваться система?
- Кто получит преимущества от внедрения системы в эксплуатацию?
- Кто будет поставлять системе те или иные данные, обращаться к ним и нести ответственность за их обновление и удаление?
- Кому предстоит выполнять обязанности администратора системы?
- Должна ли система использовать внешние ресурсы?
- Способен ли один и тот же активный субъект играть несколько различных ролей?
  - Разрешено ли нескольким субъектам осуществлять в системе одинаковые функции?
  - Будет ли система использоваться совместно с какими-либо существующими унаследованными системами?

Процедура определения активных субъектов системы отличается итеративным характером – первый вариант списка субъектов редко бывает окончательным. Внимательно анализируя роли сущностей, взаимодействующих с системой, можно прийти к адекватному множеству активных субъектов.

**Формирование потоков событий.** С каждым вариантом использования связан определенный поток событий, происходящих по мере выполнения соответствующих функций системы. При описании потока событий определяется, *что* необходимо осуществить, и игнорируются аспекты того, *как* это делается. Другими словами, события воспроизводятся средствами языка предметной области, а не в терминах практической реализации функций. Схема потока событий должна содержать ответы на следующие вопросы.

- Как и когда вариант использования инициируется и завершается?
- Каким образом активные субъекты взаимодействуют с системой?
- Какие данные затрагиваются вариантом использования?
- Что представляет собой нормальная последовательность событий, предусматриваемых вариантом использования?
- Существуют ли альтернативные потоки событий?

**Документирование потока событий.** Потоки событий обычно документируются по мере потребности. Описание потока событий для варианта

использования системы содержится в документе **Use Case Specification** (спецификация варианта использования). Для создания подобного документа в каждом проекте должен применяться некий стандартный шаблон. Рекомендуется применение шаблона из регламента **Rational Unified Process**, в котором предусмотрена следующая информация:

- 1.0. Наименование варианта использования.
  - 1.1. Краткое описание.
- 2.0. Потоки событий.
  - 2.1. Основной поток.
  - 2.2. Альтернативные потоки.
    - 2.2.\*. <Альтернативный поток \*>.
- 3.0. Специальные требования
  - 3.\*. <Специальное требование \*>.
- 4.0. Предусловия.
  - 4.\*. <Предусловие \*>.
- 5.0. Постусловия.
  - 5.\* <Постусловие \*>.
- 6.0. Дополнительные замечания.
  - 6.\*. Дополнительное замечание \*.

**Назначение связей в вариантах использования.** Между активным субъектом и вариантом использования системы допускается устанавливать *связь ассоциации (association relationship)*, которая выполняет коммуникативную функцию, сообщая о взаимодействии субъекта с системой в рамках определенного варианта использования. Направление связи указывает, кто (субъект или система) является инициатором взаимодействия. Двухнаправленная ассоциация представляется в модели в виде линии, соединяющей связываемые элементы. Возможность взаимодействия элементов только в одном направлении отображается с помощью стрелки.

Варианты использования могут соединяться *связями зависимости (dependency relationships)* двух типов – *включения (include)* и *расширения (extend)*.

Включающей связью соединяются определенный вариант использования и любой другой вариант, который предполагает выполнение функций, дополняющих вариант использования. Включающая связь отображается в модели с помощью стрелки, направленной от базового варианта к включаемому.

Расширяющие связи применяются для описания:

- множеств необязательных функций;
- поведения системы при возникновении нештатных ситуаций;
- различных потоков событий, инициируемых в зависимости от того, какие опции выбираются пользователем.

Примером может служить вариант использования системы управления конвейером, адресующий специальный "аварийный" вариант при возникновении технических неполадок. Расширяющая связь отображается в модели с помощью стрелки, направленной к базовому варианту от расширяющего варианта.

В UML поддерживается понятие *стереотипа (stereotype)*, которое обеспечивает возможность создания новых элементов модели на основе базовых и позволяет определять минимальное множество символов, пополняемое с целью описания набора сущностей, имеющих смысл в контексте разрабатываемой системы. Стереотипы применяются для конструирования требуемых связей вариантов использования. Имена стереотипов заключаются в угловые кавычки <<...>> и размещаются рядом со стрелками, описывающими связи.

Ассоциация может быть снабжена меткой с именем стереотипа <<communicate>>, подчеркивающей, что связь обладает коммуникативной функцией, хотя это не обязательно, поскольку ассоциация – единственный тип связей, которыми могут соединяться активные субъекты и варианты использования. Включающие и расширяющие связи, напротив, относятся к одной и той же категории связей – *зависимости* и поэтому *должны* снабжаться соответствующими метками. Пример диаграммы, содержащей ассоциацию и связи зависимости, приведен на рисунке 2.1.

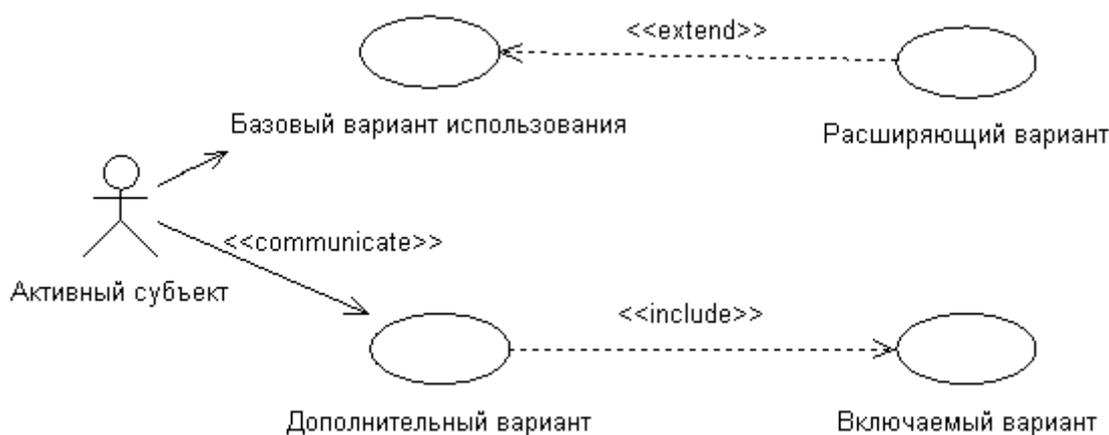


Рисунок 2.1 – Пример диаграммы с ассоциациями и связями зависимости

**Разработка диаграммы вариантов использования.** *Диаграмма вариантов использования (use case diagram)* – это графическое представление подмножеств активных субъектов, взаимодействующих с системой посредством тех или иных вариантов ее использования. При проектировании любой системы обычно конструируется *основная диаграмма (Main Use Case Diagram)*, представляющая множество пользователей (активных субъектов) и ключевые функции (варианты использования) системы. При необходимости создаются следующие диаграммы:

- диаграмма, представляющая все варианты использования, инициируемые определенным активным субъектом;
- диаграмма, изображающая варианты использования, подлежащие последовательной реализации;
- диаграмма, описывающая некоторый вариант использования и все его связи.

Таким образом, характеристики поведения разрабатываемой системы фиксируются и документируются средствами модели, которая отображает функции (варианты использования) продукта, представляет окружение системы (множество активных субъектов) и определяет связи между вариантами использования и активными субъектами (диаграммы вариантов использования).

## 2.2 Пример анализа предметной области

Тема примера: Разработать модель для системы управления банкомата.

### *Описание предметной области “Банкомат”.*

Банкомат – это автоматический терминал банка (АТМ) для выдачи наличных денег по кредитным пластиковым карточкам. В его состав входят следующие устройства: экран, панель управления с кнопками (клавиатура), приемник кредитных карт с устройством чтения карточек, устройство выдачи наличных денег, устройство блокирования кредитных карт, принтер для печати чеков (справок об остатке денег на счете). Банкомат подключен через локальную сеть к серверу (контроллеру) банка, хранящему сведения о счетах клиентов.

Обслуживание клиента начинается с момента ввода пластиковой карточки в приемник карт банкомата. После распознавания типа пластиковой карточки, банкомат выдает на дисплей сообщение ввести персональный код. После ввода ПИН-кода банкомат проверяет его правильность. Если код указан неверно, пользователю предоставляются еще две попытки для ввода правильного кода. В случае повторных неудач карта блокируется и перемещается в хранилище карт, а сеанс обслуживания заканчивается. После ввода правильного кода банкомат предлагает пользователю выбрать операцию. Клиент может либо снять наличные со счета, либо узнать остаток денег на его счету.

При снятии наличных со счета клиент банкомата должен ввести сумму, после чего банкомат запрашивает, нужно ли печатать чек. Затем банкомат посылает запрос на снятие выбранной суммы серверу банка. В случае получения разрешения на операцию, банкомат проверяет, имеется ли требуемая сумма в его хранилище денег, удаляет карточку из приемника и выдает указанную сумму в лоток выдачи. Если клиент затребовал чек, банкомат печатает справку по произведенной операции. Если клиент хочет узнать остаток на счете, то банкомат посылает запрос серверу банка и выводит сумму на дисплей.

### *Анализ предметной области.*

Перед разработчиком программной системы всегда стоит один из принципиальных вопросов: “**Какие объекты реального мира необходимо учесть в модели, и как они взаимосвязаны?**”.

Проектируя объектно-ориентированную систему, нужно, в конечном счете, создать модель абстракций предметной области, то есть **получить представление о структуре системы и ее поведении**. Начинать составление этой модели следует из анализа описания предметной области.

Именные группы и имена существительные в приведенном описании могут стать объектами или атрибутами классов.

Составив список имен возможных объектов, следует проанализировать этот список с целью исключения из него ненужных классов.

Из списка следует удалить:

- *избыточные наименования*, которые выражают одинаковую информацию (следует сохранить только одно из наименований);
- *нерелевантные* (не имеющие прямого отношения к проблеме) термины, необходимость представления которых в будущей системе сомнительна;
- *нечетко определенные* с точки зрения рассматриваемой проблемы выражения;
- *атрибуты*: некоторым существительным больше соответствуют не классы, а атрибуты, например, имя, возраст, вес, адрес и т. п.;
- *операции*: некоторым существительным больше соответствуют не классы, а имена операций; например, “телефонный вызов” вряд ли означает какой-либо класс;
- *роли*: некоторые существительные определяют имена ролей в объектной модели, например, владелец, водитель, начальник, служащий;
- *реализационные конструкции*: имена, которые больше связаны с программированием и компьютерной аппаратурой (подпрограмма, процесс, алгоритм, прерывание и т. п.), не следует на данном этапе сопоставлять с классом, так как они **не отражают особенностей** проектируемой прикладной системы.

После исключения всех ненужных (лишних) имен будет получен предварительный список объектов (классов), составляющих основу проектируемой программной системы.

Применительно к описанию предметной области “**Банкомат**” в предварительный список можно включить следующие объекты, из которых возможно создание классов:

1. Экранная форма банкомата.
2. Панель управления (клавиатура) банкомата.
3. Приемник карт с устройствами считывания и блокирования.

4. Устройство выдачи наличных денег.
5. Принтер для печати чека.
6. Контроллер банкомата.
7. Интерфейс сервера банка.
8. Транзакция банкомата, которая обеспечивает связь с внешней системой – банком и косвенно присутствует в описании.

В список можно также добавить абстрактный класс “Контроллер”, который позволит специфицировать системные атрибуты, связи и операции.

Придерживаясь методики “движение изнутри наружу”, применяемой для построения статической модели, устанавливаем, с какими объектами взаимодействует система банкомата. Из описания банкомата выделим внешние объекты – **Клиент Банкомата** и **Банк**. Они являются актерами по отношению к системе банкомата.

Определив структуру системы, следует перейти к анализу второго принципиального вопроса: “**Что должна делать система?**”.

Для определения поведения нужно выявить имеющиеся в описании прецеденты и провести грамматический анализ текста каждого прецедента.

Текст прецедента должен отвечать требуемому поведению системы и формулироваться с точки зрения действующего лица с использованием *глаголов настоящего времени в действительном залоге*.

Применительно к рассматриваемой задаче клиент может либо снять наличные, либо проверить остаток денег на счете. Из такого описания формулируем тексты двух прецедентов:

1. **Снять Наличные.**
2. **Проверить Счет.**

Для реализации каждого из этих двух прецедентов потребуется определенное поведение клиента и системы банкомата. В это поведение должны быть включены процедуры идентификации параметров карточки, проверки ПИН-кода, печать чека (по требованию клиента), выдача наличных (клиент должен ввести сумму) и другие. Описание поведения должно представлять собой *спецификацию потоков событий*, которая необходима для построения модели поведения с применением диаграмм вариантов использования (*use case diagram*) и диаграмм последовательностей (*sequence diagram*).

Спецификация потока событий представляет собой текстовый сценарий, составленный на основе шаблона. Пример сценария для прецедента “Снять наличные” приведен в таблице 2.1.

Завершая анализ предметной области, следует убедиться, что требуемое поведение может быть обеспечено структурой, представленной предварительным списком объектов системы.

Таблица 2.1

Действия актеров	Отклик системы
1. Клиент вводит кредитную карточку в устройство чтения банкомата. <i>Исключение 1.</i> Кредитная карточка недействительна.	1. Банкомат проверяет кредитную карточку. 2. Банкомат предлагает ввести ПИН-код.
2. Клиент вводит ПИН-код. <i>Исключение 2.</i> ПИН-код неверный.	3. Банкомат проверяет ПИН-код. 4. Банкомат отображает опцию меню.
3. Клиент выбирает опцию “Снять наличные”.	5. Банкомат предлагает ввести требуемую сумму.
4. Клиент вводит требуемую сумму. <i>Исключение 3.</i> Требуемая сумма превышает текущее состояние счета.	6. Система делает запрос в банк и выясняет текущее состояние счета. 7. Банкомат изменяет состояние счета, выдает карточку, наличные и чек.

### 2.3 Методика выполнения работы средствами Rational Rose

При открытии программы **Rational Rose** выводится мастер типовых проектов. Если технология *реализации* проекта еще не выбрана, то от мастера следует отказаться. В результате появится рабочий интерфейс программы с чистым окном активной диаграммы классов и именем проекта **untitled** по умолчанию. Для изменения имени следует сохранить модель во внешнем файле, указав новое имя.

*Активизирование диаграммы вариантов использования возможно двумя путями:*

1. Раскрыть представление вариантов использования **Use Case View** в браузере проекта и дважды щелкнуть на пиктограмме **Main** (Главная).
2. С помощью операции главного меню **Browse ► Use Case Diagram** (Браузер ► Диаграмма вариантов использования).

При этом появляется специальная панель инструментов с пиктограммами элементов, назначение которых показано на рисунке 2.2.

Состав инструментов на специальной панели можно изменить с помощью операции главного меню **Tools ► Options** (Инструменты ► Параметры), раскрыв вкладку **Toolbars** (Панель инструментов) и нажав соответствующую кнопку, например, **Use Case Diagram** в группе опций **Customize Toolbars** (Настройка

панели инструментов). В результате открывается диалоговое окно (рис. 2.3).

	Selection Tool - превращает изображение курсора в форму стрелки
ABC	Text Box - добавляет на диаграмму текстовую область
	Note - добавляет на диаграмму примечание
	Anchor Note to Item - добавляет связь примечания с элементом диаграммы
	Package - добавляет на диаграмму пакет
	Use Case - добавляет на диаграмму вариант использования
	Actor - добавляет на диаграмму актера
	Unidirectional Association - добавляет направленную ассоциацию
	Dependency or Instantiates - добавляет отношение зависимости
	Generalization - добавляет отношение обобщения

Рисунок 2.2 – Назначение кнопок специальной панели инструментов для диаграммы вариантов использования

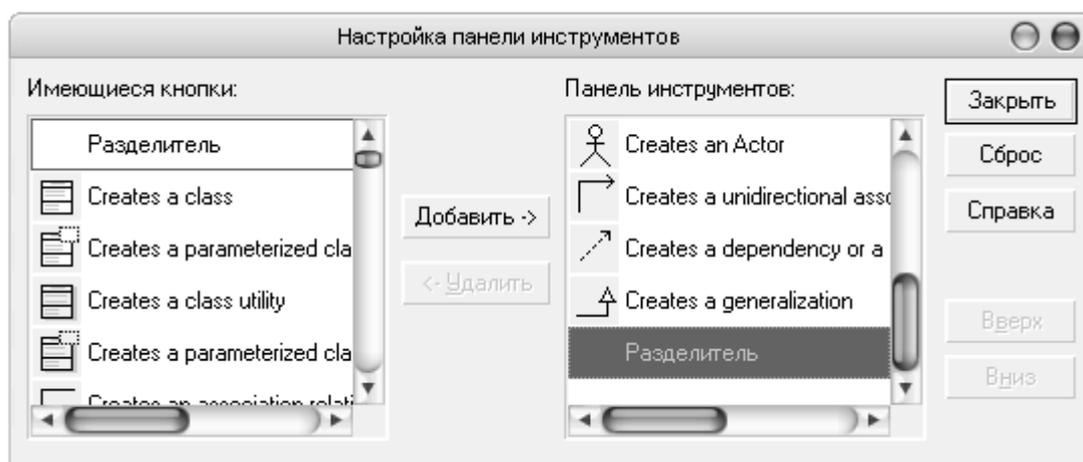


Рисунок 2.3 – Окно настройки специальной панели инструментов для диаграммы вариантов использования

### Создание активного субъекта:

1. Расположить курсор мыши над элементом **Use Case View** окна браузера (**Browser**) и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Actor**. Дерево, отображаемое в окне **Browser**, пополнится элементом **NewClass**, соответствующим новому активному субъекту.
3. Изменить название элемента **NewClass**, введя требуемое имя активного

субъекта, например. “Клиент”, “Менеджер”, “Оператор”, как показано на рисунке 2.4.

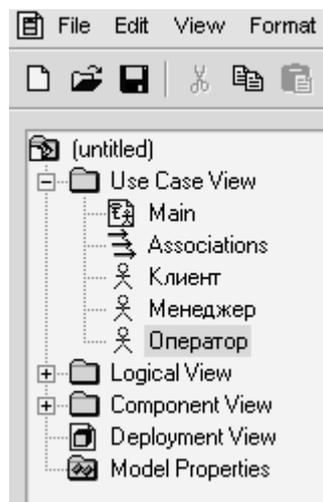


Рисунок 2.4 – Создание активного субъекта

#### **Документирование активного субъекта:**

1. Если окно документирования (Documentation) не отображается, его можно открыть, активизировав элемент меню **View ► Documentation**.
2. В окне **Browser** выбрать элемент дерева, соответствующий требуемому активному субъекту.
3. Разместить курсор в окне **Documentation** и ввести текстовое описание активного субъекта (рис. 2.5).

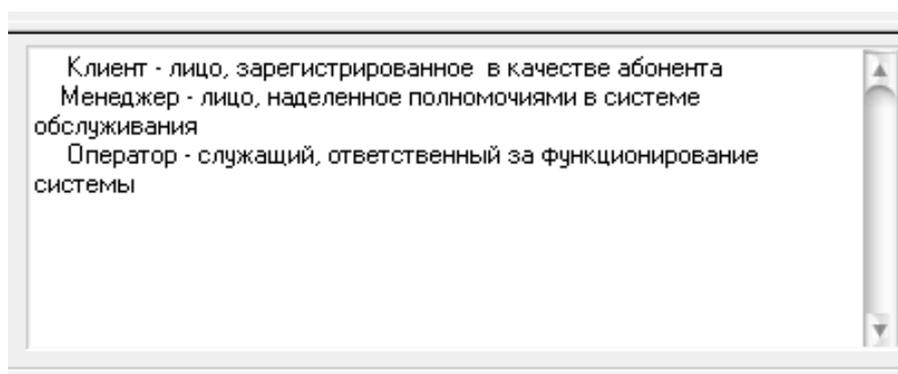


Рисунок 2.5 – Окно документирования активных субъектов

#### **Создание вариантов использования:**

1. Расположить курсор мыши над элементом **Use Case View** окна **Browser** и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Use Case**. Дерево, отображаемое в окне **Browser**, пополнится элементом **NewClass**, соответствующим новому варианту использования.

3. Изменить название элемента **NewClass**, введя требуемое имя варианта использования.

***Установление связи документа спецификации потока событий с вариантом использования:***

1. Расположить курсор мыши над элементом окна **Browser**, соответствующим требуемому варианту использования, и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **Open Specification**.
3. Перейти на вкладку **Files** диалогового окна **Use Case Specification**.
4. Расположить курсор мыши в пределах области окна и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
5. Выбрать элемент меню **Insert File**.
6. С помощью средств навигации диалогового окна **Открыть** выбрать требуемый файл.
7. Щелкнуть на кнопке **Открыть**.
8. Щелкнуть на кнопке **ОК** окна **Use Case Specification**.
9. Результат связывания документа спецификации с вариантом использования отображается в окне **Browser**.

***Создание коммуникативной ассоциации:***

1. Щелкнуть на пиктограмме **Unidirectional Association** специальной панели инструментов.
2. В окне диаграммы щелкнуть на символе активного субъекта и, не отпуская кнопку мыши, построить линию связи, направленную к символу соответствующего варианта использования.

***Создание метки стереотипа «communicate»:***

1. В окне диаграммы дважды щелкнуть на линии, представляющей ассоциацию, чтобы открыть диалоговое окно **Association Specification**.
2. Щелкнуть на кнопке со стрелкой в правой части поля **Stereotype** и выбрать в раскрывающемся списке опцию **communicate**.
3. Закрыть окно **Association Specification** щелчком на кнопке **ОК**.

***Создание включающей связи:***

1. Щелкнуть на пиктограмме **Dependency or Instantiates** специальной панели инструментов.
2. В окне диаграммы щелкнуть на символе базового варианта использования и, не отпуская кнопку мыши, построить линию связи, направленную к символу соответствующего включаемого варианта использования.
3. Дважды щелкнуть на линии, представляющей связь, чтобы открыть диалоговое окно **Dependency Specification**.
4. Щелкнуть на кнопке со стрелкой в правой части поля **Stereotype** и

- выбрать в раскрывающемся списке опцию **include**.
5. Закрывать окно **Dependency Specification** щелчком на кнопке **ОК**.

#### *Создание расширяющей связи:*

1. Щелкнуть на пиктограмме **Dependency or Instantiates** специальной панели инструментов.
2. В окне диаграммы щелкнуть на символе расширяющего варианта использования и, не отпуская кнопку мыши, построить линию связи, направленную к символу соответствующего базового варианта использования.
3. Дважды щелкнуть на линии, представляющей связь, чтобы открыть диалоговое окно **Dependency Specification**.
4. Щелкнуть на кнопке со стрелкой в правой части поля **Stereotype** и выбрать в раскрывающемся списке опцию **extend**.
5. Закрывать окно **Dependency Specification** щелчком на кнопке **ОК**.

#### *Создание основной диаграммы вариантов использования:*

1. Двойным щелчком на элементе **Main** поддерева **Use Case View**, отображаемого в окне **Browser**, открыть окно основной диаграммы вариантов использования.
2. В окне **Browser** выбрать элемент, соответствующий требуемому активному субъекту, и перетащить его в окно диаграммы.
3. Повторить действие, указанное в п. 2, для всех активных субъектов, подлежащих включению в диаграмму.
4. В окне **Browser** выбрать элемент, соответствующий требуемому варианту использования, и перетащить его в окно диаграммы.
5. Повторить действие, указанное в п. 4, для всех вариантов использования, подлежащих включению в диаграмму.

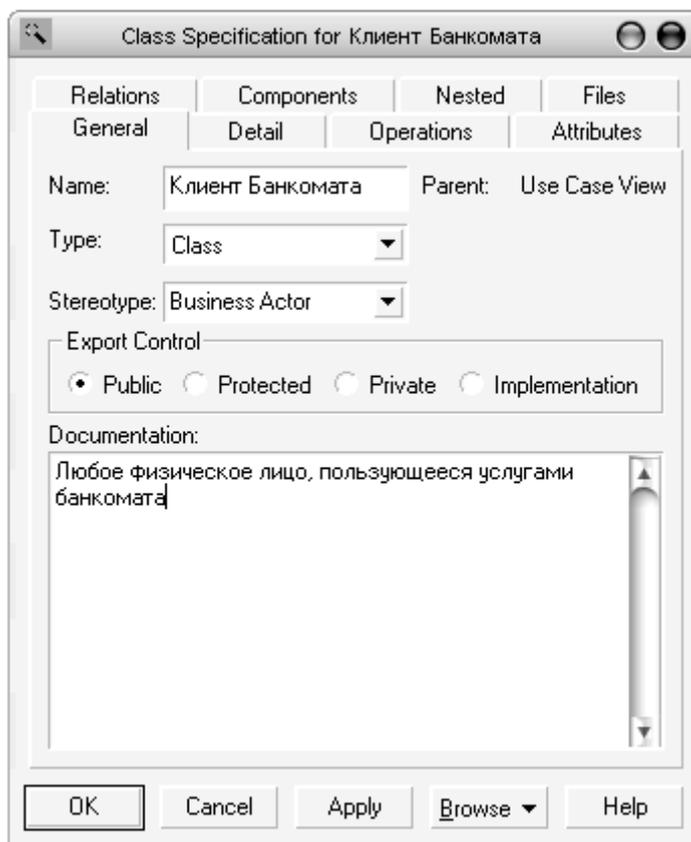
## 2.4 Пример построения диаграммы в среде Rational Rose

При построении диаграммы вариантов использования для системы банкомата могут быть использованы элементы бизнес-деятельности: **Business Actor** (бизнес-актер), **Business Use Case** (бизнес-вариант использования) и **Service** (сервис). Например, для разрабатываемой модели банкомата задается имя актера – **Клиент Банкомата**.

Хотя в среде Rational Rose актер является классом, для него некорректно специфицировать атрибуты и операции, поскольку актер является внешней по отношению к разрабатываемой системе сущностью.

Для актера **Клиент Банкомата** следует уточнить его назначение в модели. С этой целью открывается окно спецификации, в котором выбирается стереотип и добавляется текст документации. Для изменения стереотипа во вложенном

списке **Stereotype** выбирается строка **Business Actor** (бизнес-актер). Для добавления текста документации в секцию **Documentation** вводится текст: «Любое физическое лицо, пользующееся услугами банкомата», после чего нажимается кнопка **Apply** (Применить) или **OK** (рис. 2.6).



*Рисунок 2.6 – Окно спецификации свойств после изменения стереотипа и ввода текста документации для актера **Клиент Банкомата***

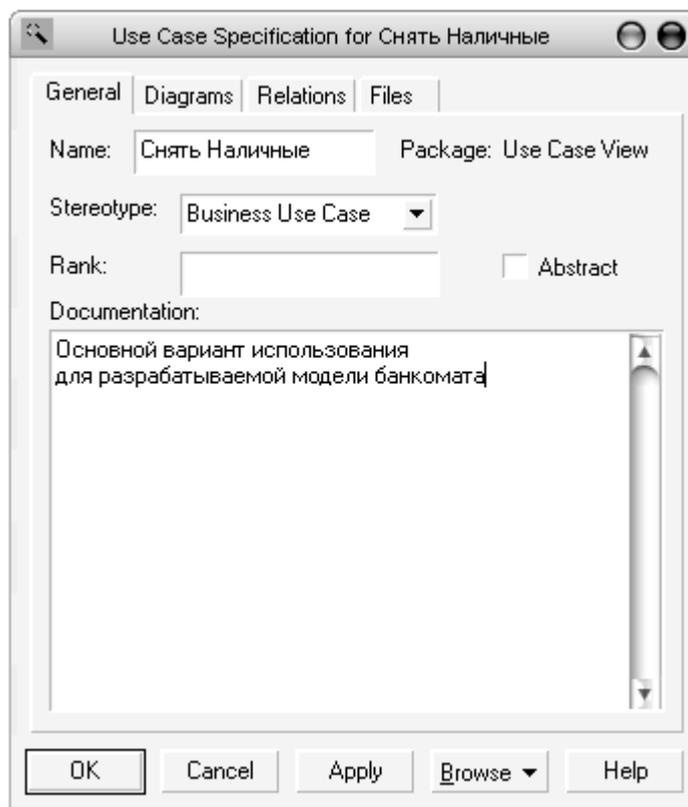
Следующий шаг – добавление элементов Use Case. Предложенное программой имя каждого варианта использования следует изменить. Для разрабатываемой модели банкомата в один из элементов Use Case вводится имя **Снять Наличные**, в другой – **Проверить Счет**.

Для уточнения свойств варианта использования во вложенном списке **Stereotype** выбирается стереотип **Business Use Case**. В секцию **Documentation** вводится текст: «Основной вариант использования для разрабатываемой модели банкомата».

Результаты выполненных действий показаны на рисунке 2.7.

Между актером и вариантом использования добавляется направленная ассоциация, а между вариантами использования – отношения зависимости. Для реализации основных вариантов использования системы банкомата **Снять Наличные** и **Проверить Счет** обязательным является введение прецедента

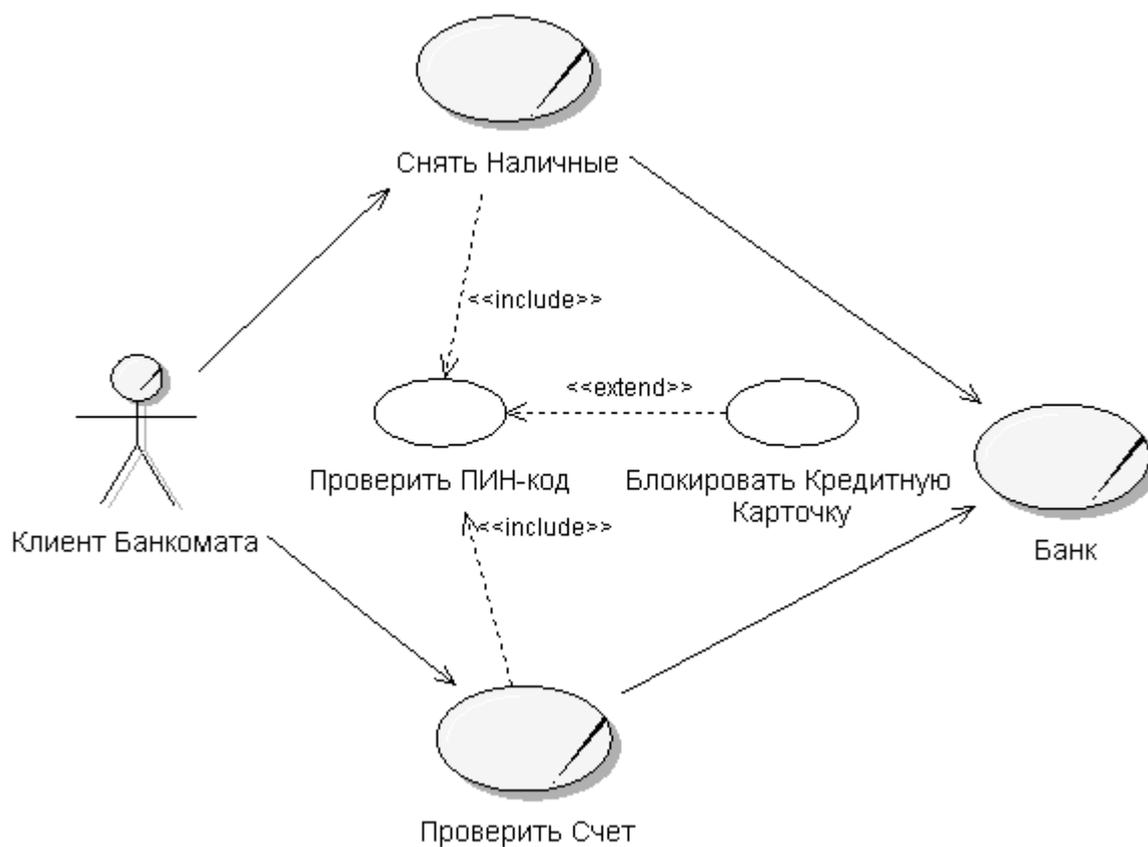
**Проверить ПИН-код.** Этот вариант использования связывается отношением зависимости <<include>> с основными вариантами использования. Задание текстового стереотипа отношения зависимости <<include>> производится в окне спецификации свойств этого отношения.



*Рисунок 2.7 – Диалоговое окно спецификации свойств варианта использования **Снять Наличные***

Для моделирования исключений на диаграмму добавляются другие элементы Use Case с использованием отношения зависимости со стереотипом <<extend>>. Для системы банкомата предусмотрен прецедент **Блокировать Кредитную Карточку**. Этот прецедент имеет отношение зависимости <<extend>> с прецедентом **Проверить ПИН-код**, поскольку он будет выполняться только в том случае, когда карточка и ПИН-код не соответствуют друг другу. Диаграмму можно детализировать введением таких прецедентов, как **Удалить Карточку**, **Печатать Чек** и др.

Окончательный вид диаграммы вариантов использования с добавлением необходимых элементов приведен на рисунке 2.8.



*Рисунок 2.8 – Окончательный вид диаграммы Use Case для модели банкомата*

## 2.5 Содержание отчета

Отчет по работе должен содержать описание и анализ предметной области, спецификацию потока событий и основную диаграмму вариантов использования с документированием активных субъектов и связей.

При защите отчета необходимо обосновать варианты использования, а также выбор актеров, типов связей, основных и альтернативных потоков событий.

### 3 КОНСТРУИРОВАНИЕ КЛАССОВ

*Цель работы:* освоение приемов и методики конструирования классов с применением программы Rational Rose.

#### 3.1 Рекомендации по определению и конструированию классов

Диаграмма классов – это логическое представление модели программной системы, её архитектура.

*Класс (class)* – это группа объектов с общими свойствами (атрибутами), поведением (функциями), семантикой и связями с другими объектами. Класс можно рассматривать как шаблон для создания объектов. Каждый объект является экземпляром *только одного класса*.

*Объект (object)* – это представление реальной или абстрактной сущности – автомобиля, персонального компьютера, металлорежущего станка, электродвигателя, датчика или банковской транзакции. Объект представляет собой понятие с явно оговоренными границами, смыслом и назначением в контексте программного приложения. Каждый объект системы обладает тремя характеристиками – *состоянием, поведением и идентификационным признаком*.

*Состояние (state)* объекта – это одно из возможных сочетаний условий его существования. Состояние объекта определяется набором *свойств-атрибутов* (attributes) и связей с другими объектами.

*Поведение (behavior)* охватывает функциональную сторону жизни объекта, определяет его реакцию на запросы со стороны других объектов и реализуется в виде набора операций.

*Идентификационный признак (identity)* задает свойство уникальности объекта – даже в том случае, если состояние последнего идентично состоянию других объектов.

Правильно сконструированный класс должен представлять одну и *только одну абстракцию*. Для именования классов следует пользоваться терминами, принятыми в соответствующей предметной области. В качестве имени класса принято употреблять существительное единственного числа, наилучшим образом описывающее моделируемое понятие.

Классам отвечают определенные стереотипы, позволяющие создавать новые разновидности элементов моделируемой системы.

К числу наиболее употребительных стереотипов классов относятся:

- *entity* – сущность;
- *boundary* – граница;

- *control* – управление;
- *utility* – прикладной класс;
- *exception* – исключение.

Названия стереотипов на диаграммах заключаются в угловые кавычки (<<...>>) и располагаются над именами классов. Система Rational Rose позволяет ассоциировать со стереотипом определенную пиктограмму или выделять его тем или иным цветом. Стереотипы позволяют аналитику и дизайнеру размежевать уровни предметной области.

**Класс сущностей (entity class)** моделирует структуру данных со стабильным характером. Класс подобного типа отражает качества сущности реального мира или применяется для выполнения внутренних функций системы. Классы сущностей обычно не зависят от окружения, они не чувствительны к внешним обстоятельствам. Для описания определенной функции подойдет краткое выражение с именем существительным. Классы сущностей часто называют "предметными", поскольку они представляют абстракции понятий и вещей реального мира.

**Классы границ (boundary classes)** обслуживают процессы взаимодействия между системой и ее окружением, обеспечивая интерфейсы для пользователей и сторонних систем (активных субъектов). Такие классы образуют ту часть системы, которая непосредственно "общается" с внешним миром. Для отыскания классов границ анализируется каждая пара вида "активный субъект/вариант использования". Классы границ определяются с учетом особенностей выбранных механизмов интерфейса.

**Классы управления (control classes)** применяются при реализации поведения системы и координируют события, возникающие по мере функционирования системы в рамках вариантов использования. Класс управления можно воспринимать как абстракцию, отображающую динамику варианта использования системы. Классы управления обычно тесно "привязаны" к особенностям конкретного приложения.

Для примера **Банкомат** определен следующий состав классов:

1. **Controller.**
2. **IBank.**
3. **CardReader.**
4. **Keyboard.**
5. **Printer.**
6. **Transaction.**
7. **CashDispenser.**
8. **ScreenForma.**
9. **Client.**

При выборе стереотипа (атрибута класса) для рассматриваемого примера

учитывалось следующее:

- класс **Controller** является управляющим классом и должен быть отмечен стереотипом **control**;
- класс **IBank** должен быть отмечен стереотипом **interface**;
- класс **Transaction**, который представляет собой пассивный класс для хранения данных и процедур, должен быть отмечен стереотипом **Class Module**;
- класс **ScreenForma**, представляющий собой несколько вариантов форм, должен иметь стереотип **collection**;
- остальные классы – **CardReader**, **Printer**, **CashDispenser** являются граничными классами с атрибутом **boundary**.

На ранних стадиях жизненного цикла системы для каждой пары вида "активный субъект/вариант использования" создается по одному классу управления, на который возлагаются обязанности по контролю за потоком событий, происходящих по мере выполнения этого варианта.

Класс управления должен быть "осведомлен" о том, *когда* следует выполнить операцию, но то, *как* это делать, является исключительной прерогативой класса сущностей.

При создании классов их следует документировать. Описание должно характеризовать назначение класса, а не его структуру.

При именовании или документировании класса возникают следующие ситуации:

- Описание класса совпадает с описанием другого класса. В этом случае следует изучить возможность объединения классов.
- Чрезмерно велик комментарий. В этом случае следует применить расщепление класса.
- Нельзя подобрать ни имя класса, ни вразумительный комментарий к нему. В этом случае необходим дополнительный анализ, который позволит обосновать или пересмотреть абстракцию.

Если система содержит большое количество классов, они группируются в пакеты.

**Пакет** – это собрание "родственных" вложенных пакетов и/или классов. Распределение классов по пакетам дает возможность подняться на более высокий уровень восприятия модели.

Каждый пакет обычно содержит интерфейс, образованный множеством *общедоступных* классов. К таким классам разрешено обращаться из кода классов, принадлежащих другим пакетам. Остальные классы пакета связаны со спецификой реализации и потому не допускают обращения извне.

Если проектируемая система отличается особой сложностью, пакеты могут конструироваться уже на ранних фазах ее жизненного цикла. В простых ситуациях созданные классы группируются в единственный пакет, "границы"

которого определяются системой в целом.

Подмножество пакетов и классов представляются в удобном графическом виде – **диаграммы классов (class diagrams)**.

*Основная диаграмма классов (Main Class Diagram)* модели изображает, как правило, **набор пакетов системы**.

### 3.2 Рекомендации по определению атрибутов и операций

Класс воплощает в себе множество функциональных обязанностей, которые обуславливают *поведение (behavior)* объектов этого класса. Обязанности реализуются посредством операций, определяемых в контексте класса. Операция должна осуществлять всего одну функцию, но успешно и во всей ее полноте. Каждая операция класса поступает в распоряжение всех его объектов.

*Структура (structure)* объекта определяется множеством атрибутов класса. Атрибут представляет собой определение порции данных, доступной для объекта. Объекты класса обладают собственными значениями атрибутов.

При определении атрибутов и операций, как и при создании классов, надлежит придерживаться единого стиля, позволяющего улучшить восприятие модели и программного кода, генерируемого на ее основе.

Рекомендуется следовать такому правилу: идентификатор начинается со строчной буквы, не содержит символов подчеркивания, а отдельные слова в нем пишутся слитно и выделяются заглавными буквами, например, **количествоСтудентов**.

Если объект не нуждается в атрибутах и/или операциях, стоит еще раз присмотреться к определению соответствующего класса. Это может свидетельствовать о том, что класс не является самодостаточным и потому подлежит объединению с каким-либо иным классом. С другой стороны, избыток данных или их разноречивость также недопустимы.

Сообщения в диаграммах взаимодействия, как правило, соотносятся с *операциями (operations)*, определенными в составе класса-приемника. Однако встречаются ситуации, когда сообщение не "превращается" в операцию. Если класс, которому адресовано сообщение, является классом границ, представляющим тип некоторого элемента графического интерфейса пользователя, сообщение выражает одно из требований, предъявляемых к интерфейсу. Подобные типы сообщений обычно реализуются в виде тех или иных управляющих компонентов интерфейса, например, сообщения диалоговых окон или их отдельных частей.

Каждую операцию следует именовать в терминах класса, который ее реализует, а не класса, запрашивающего требуемые функции.

Все операции должны быть документированы, чтобы пользователь, работающий с моделью, мог быстро уловить их смысл и назначение. Комментарий обязан сообщать о функциональной стороне операции, а также о типах входных параметров и возвращаемых значений. Набор типов входных и возвращаемого значений образует *сигнатуру* (**signature**) операции. Однако такая информация не всегда известна на начальных этапах проектирования.

Многие из атрибутов класса нетрудно найти, изучив постановку задачи, предметную область и потоки событий, предусматриваемые теми или иными вариантами использования.

Атрибуты также подлежат тщательному и четкому документированию. Описание атрибута должно отображать его назначение, но не структуру.

### **3.3 Рекомендации по определению связей между объектами, классами и пакетами**

Любая система состоит из многих классов и объектов. Характеристики поведения системы зависят от взаимодействия объектов. Средой, обеспечивающей возможность этого взаимодействия, являются *связи* (**relationships**). Для выявления потребности в использовании связи, соединяющей два класса (объекта), следует изучить соответствующие сценарии.

В процессе анализа системы выявляются связи ассоциации, обобщения, зависимости, реализации, агрегации и композиции.

Ассоциации может быть присвоено имя. В качестве имени обычно выбирается глагол или глагольное словосочетание, сообщающие смысл и назначение связи. Вместо имени связи, рассматриваемой в целом, можно задавать имена ее отдельных ролей. *Ролью* (**role**) называют конец линии связи, примыкающий к символу класса. В качестве имени роли используется существительное, которое обозначает функцию или возможность, определяемую фактом связи. На диаграмме классов имя роли располагается вблизи символа соответствующего класса.

Стандарта, который регламентировал бы способы употребления имен ассоциаций или их ролей, не существует. Однако вместо имен связей предпочтительно применять имена ролей, позволяющие быстрее донести смысл и значение элементов модели, поскольку подобрать глагольное словосочетание для имени связи довольно нелегко.

Именованные ассоциации или роли применяются только в случае необходимости. Например, если моделируется связь между классами "компания" и "человек", подчеркивающая факт трудоустройства "человека" в "компании", снабдить ассоциацию именем "работает" будет уместно. Однако если моделируется связь между классами "работодатель" и "служащий", то

потребность в именовании ролей просто не возникает – смысл связи становится очевидным.

Связи целесообразно снабдить признаками множественности.

*Признак множественности (multiplicity indicator)* определяет количество объектов класса, вовлеченных в связь. Для связи ассоциации или агрегирования задаются два признака множественности, по одному на каждую роль. Наиболее употребительными вариантами признака множественности являются:

1	Ровно один.
0..*	Нуль или более.
1..*	Один или более.
0..1	Нуль или один.
5..8	Диапазон значений от 5 до 8.

Возможна ситуация, когда взаимодействовать приходится нескольким объектам одного и того же класса. Для отображения подобного требования на диаграмме классов используются *возвратные связи (reflexive relationships)* ассоциации или агрегирования. Возвратные связи обычно снабжаются названиями ролей, а не именами ассоциаций.

Связь классов может обладать собственными структурой и поведением, то есть приходится иметь дело с элементами информации, относящимися к паре объектов, а не к каждому отдельному объекту. В таком случае ассоциация представляется специальным классом.

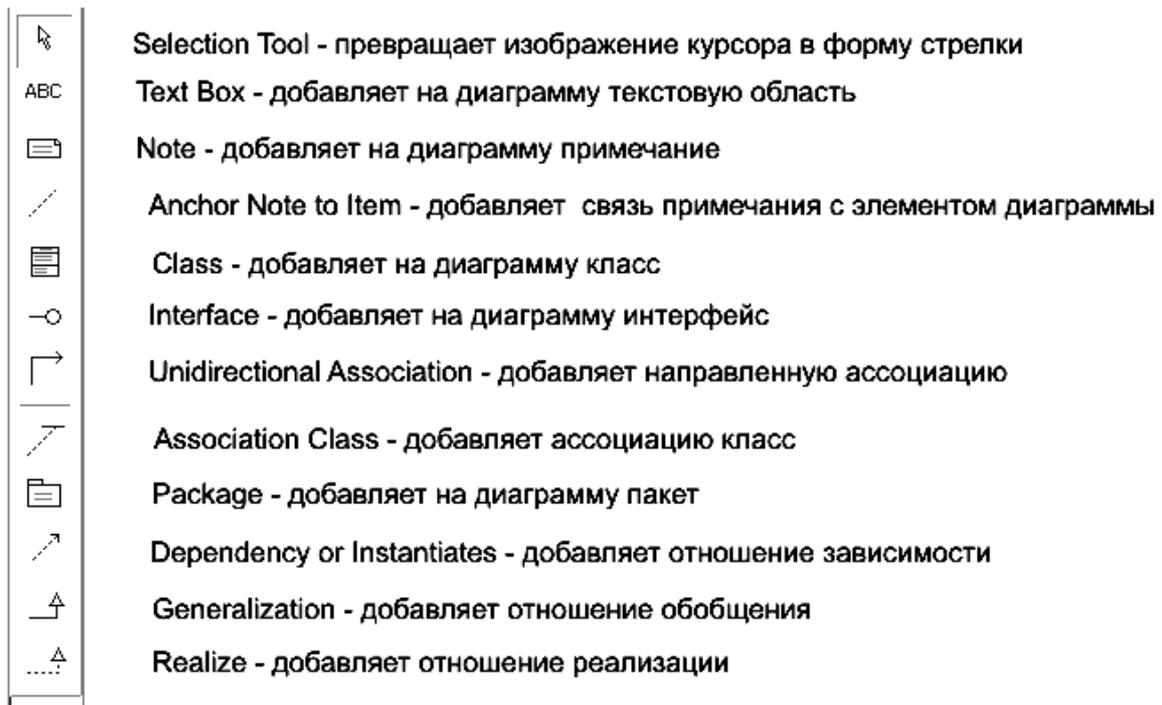
### **3.4 Методика конструирования и документирования классов в среде Rational Rose**

#### *Активизирование диаграммы классов.*

При создании нового проекта окно диаграммы классов появляется по умолчанию. В других случаях активизировать рабочее окно диаграммы классов можно одним из следующих способов:

1. Щелкнуть на кнопке с изображением диаграммы классов на стандартной панели инструментов.
2. Раскрыть логическое представление **Logical View** в браузере проекта и дважды щелкнуть на пиктограмме **Main** (Главная).
3. С помощью операции главного меню **Browse ► Class Diagram** (Браузер ► Диаграмма классов).

При этом появляется новое окно с чистым листом и специальная панель инструментов с пиктограммами элементов, назначение которых показано на рисунке 3.1.



*Рисунок 3.1 – Назначение кнопок специальной панели инструментов для диаграммы классов*

Имя диаграммы **Main**, предложенное программой, следует изменить. Для этого выделяется имя **Main** и щелчком правой кнопки вызывается контекстное меню, в котором выбирается команда **Rename** (Переименовать). Для рассматриваемого примера **Банкомат** внесено имя **Диаграмма классов АТМ**.

Построение диаграммы классов сводится к добавлению классов, атрибутов, операций и отношений. На начальном этапе для большей наглядности их имена можно записывать на русском или украинском языках. Однако следует учесть, что *при генерации кода диаграммы классов должны быть описаны только на английском языке.*

В связи с этим диаграмму классов рекомендуется строить с использованием *англоязычных терминов.*

***Добавление класса на диаграмму:***

1. Расположить курсор мыши над элементом **Logical View** окна **Browser** и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Class**. При этом дерево, отображаемое в окне **Browser**, пополнится элементом << >> **NewClass**.
3. Изменить название элемента **NewClass**, введя требуемое имя класса.

***Объявление стереотипа класса:***

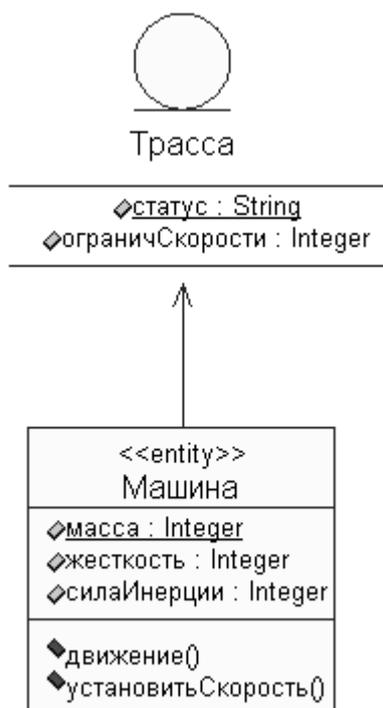
1. Расположить курсор мыши в окне **Browser** над элементом, соответствующим требуемому классу, и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.

2. Выбрать элемент меню **Open Specification**.
3. Перейти на вкладку **General** диалогового окна **Class Specification**.
4. Раскрыть список опций **Stereotype** и выбрать нужную опцию либо создать новый стереотип, введя в поле соответствующее наименование.
5. Закрыть окно **Class Specification** щелчком на кнопке **ОК**.

*Задание режима отображения стереотипа класса на диаграмме классов:*

1. Расположить курсор мыши над элементом, представляющим соответствующий класс в окне диаграммы классов, и щелкнуть правой кнопкой, активизируя контекстное меню.
2. Выбрать элемент меню **Options ► Stereotype Display** и одну из доступных опций: **None** – не воспроизводить стереотип, **Label** – показать наименование стереотипа в угловых кавычках (<< >>), **Icon** – использовать пиктограмму стереотипа, **Decoration** – изобразить класс стандартным символом UML с пиктограммой стереотипа.

На рисунке 3.14 приведена диаграмма классов, на которой стереотип класса **Трасса** воспроизводится в режиме **Icon**, а стереотип класса **Машина** – в режиме **Label**.



*Рисунок 3.14 – Варианты отображения стереотипов класса*

*Документирование класса:*

1. Если окно документирования (**Documentation**) не отображается, его можно активизировать командой **View ► Documentation**.
2. В окне **Browser** выбрать элемент, соответствующий требуемому классу.

### 3. Ввести текстовое описание класса в окне **Documentation**.

Окно **Documentation** с текстом, документирующим класс **Счет**, показано на рисунке 3.4.

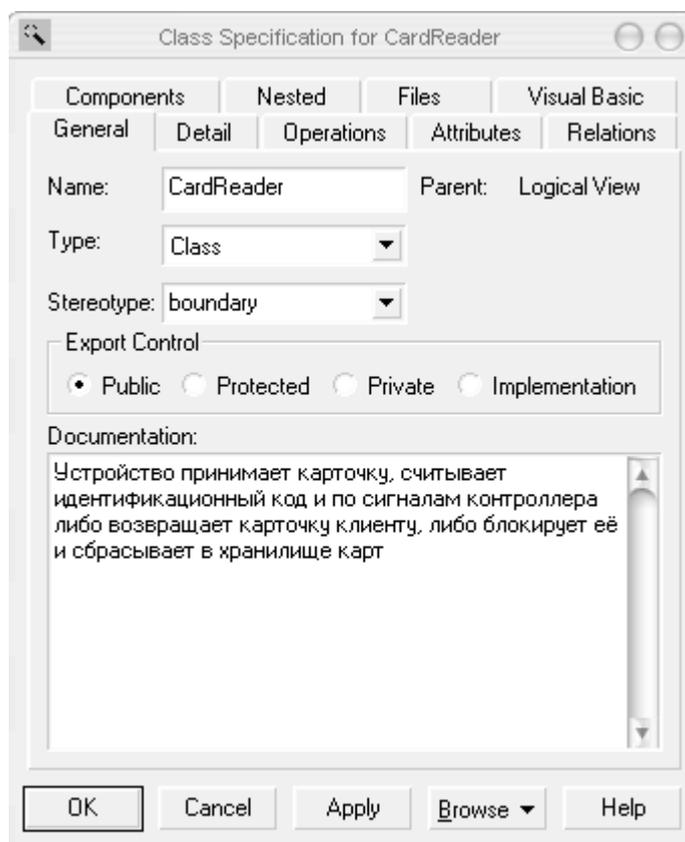


Рисунок 5.1 – Вид окна спецификации свойств класса **CardReader**

На вкладке **Detail** окна спецификации класса с помощью вложенного списка **Multiplicity** можно задать количество экземпляров класса. Так, например, для примера **Банкомат** большинство классов имеют по одному экземпляру, а классы **Transaction** и **ScreenForma** могут иметь *n* экземпляров.

Далее в группе выбора **Persistence** можно задать устойчивость класса. Применительно к примеру **Банкомат** для всех классов выбирается тип **Persistent**, то есть информация об объектах этих классов должна быть сохранена в системе.

В группе выбора возможностей реализации объектов **Concurrency** (Параллельность) для всех классов модели банкомата устанавливается свойство **Sequential** (Последовательный), то есть операции объектов должны выполняться последовательно.

Если какой-либо класс модели не имеет экземпляров, то на этой же вкладке **Detail** следует установить метку **Abstract** (Абстрактный). Для рассматриваемого примера отметка свойства **Abstract** оставляется пустой.

Для предотвращения потери информации о модели и результатов редактирования её свойств необходимо периодически сохранять модель во внешнем файле, нажимая комбинацию клавиш **Ctrl + S**.

### *Добавление атрибутов и их редактирование.*

В Visual Basic имена атрибутов и операций классов могут начинаться со строчной и прописной буквы. Видимость атрибутов на диаграмме классов изображается в форме специальных пиктограмм или украшений перед именем соответствующего атрибута.

#### *Создание атрибутов класса:*

1. Выделить класс в окне **Browser** и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Attribute**. Дерево, отображаемое в окне **Browser**, пополнится элементом **name**, отвечающим новому атрибуту класса.
3. Выбрать элемент **name** и изменить его название, введя требуемое имя атрибута.

На рисунке 3.10 показано дерево окна **Browser**, в котором отображены атрибуты и операции классов **Машина** и **Трасса**.

Для редактирования свойств атрибутов предназначено специальное диалоговое окно спецификации атрибута **Class Attribute Specification**.

#### *Документирование атрибутов класса:*

1. Если окно документирования (**Documentation**) не отображается, открыть его, активизировав элемент меню **View ► Documentation**.
2. Щелкнуть на пиктограмме “+” слева от элемента окна **Browser**, который представляет определенный класс, чтобы отобразить поддереву класса.
3. Щелчком выбрать элемент поддереву, соответствующий требуемому атрибуту.
4. Позиционировать курсор в окне **Documentation** и ввести текстовое описание атрибута.

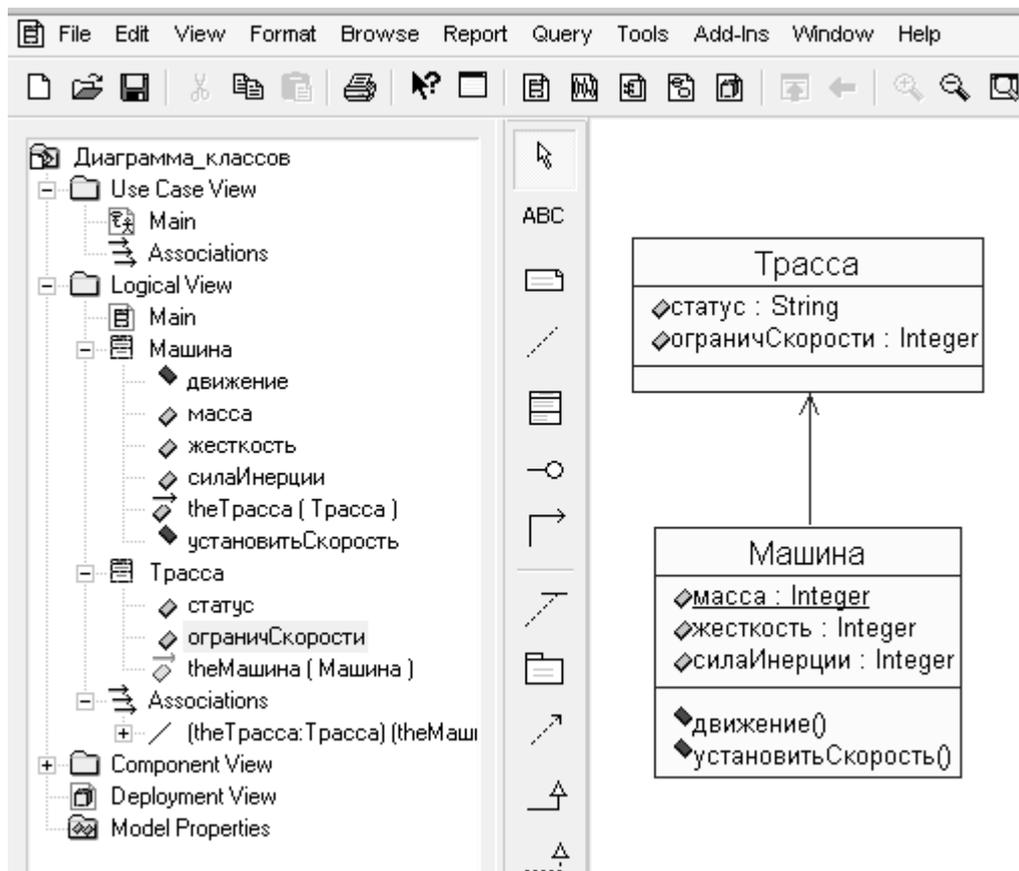


Рисунок 3.10 – Создание атрибутов и операций классов

Рассмотрим, например, какими атрибутами должен быть снабжен класс **Transaction**.

**Транзакция** – это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением. В своей работе транзакции осуществляют вводы, выводы и запросы, используя при этом внутренние и внешние файлы.

Данные могут поступать с устройств ввода, в том числе с формы, или из другого приложения. На вывод перемещаются данные вычислений или отчетов. По внешним запросам осуществляется ввод-вывод данных, как из внутренних логических файлов, так и из внешних интерфейсных файлов. При этом входная часть процесса не модифицирует внутренние логические файлы, а выходная часть не несет данных, вычисляемых приложением.

Из этих определений транзакции можно считать, что в системе банкомата существуют объекты-транзакции, которые используются контроллером банкомата для соединения его с интерфейсом банка с целью:

- 1) проверки соответствия параметров карточки персональному идентификационному коду и получения разрешения на продолжение сеанса работы с клиентом;

2) проверки счета клиента и определения возможности выдачи требуемой суммы наличных;

3) уменьшения суммы счета на величину выданной суммы наличных.

Для реализации таких процедур класс **Transaction** должен обладать следующим набором возможных атрибутов (свойств):

- номер карточки (IDCard);
- ПИН-код (PINCode);
- сумма на счете (Account);
- сумма снимаемых со счета наличных (Sum);
- остаток на счете (Balance).
- перечень транзакций (mCol);

Для редактирования атрибутов и операций предназначено специальное окно спецификаций **Class Specification for...**, которое открывается двойным щелчком на изображении класса (рис. 3.11).

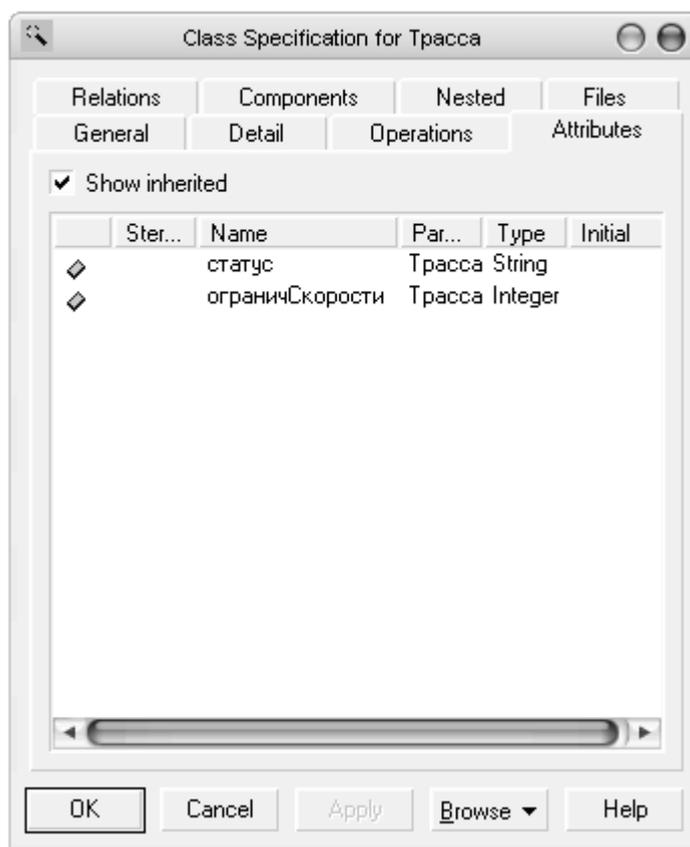


Рисунок 3.11 – Диалоговое окно спецификации свойств класса, открытое на вкладке **Attributes**

На рисунке 3.12 показано, что для атрибута “статус” тип данных соответствует строке (**string**), квантор видимости – открытый (**public**).

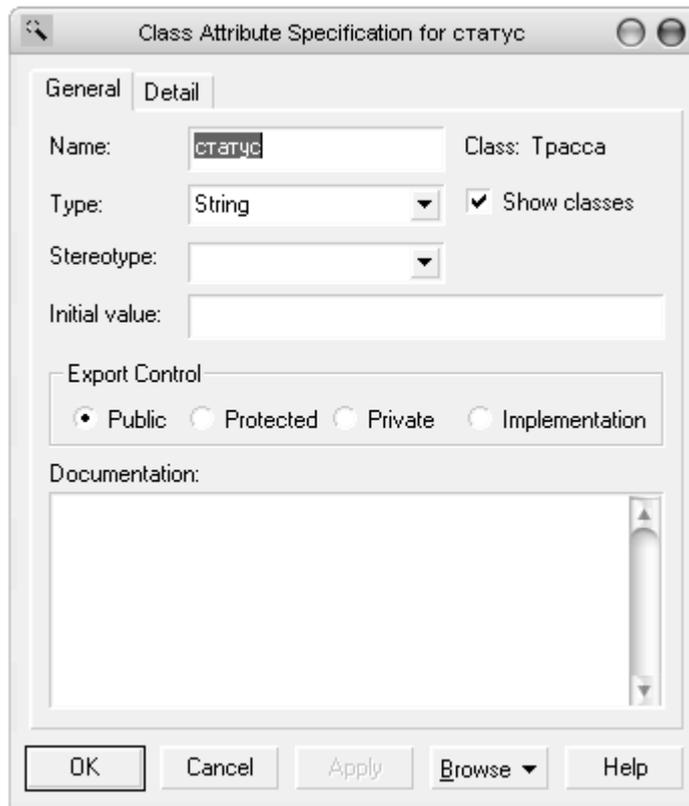


Рисунок 3.12 – Диалоговое окно спецификации свойств атрибута “статус”

На вкладке **Detail** (рис. 3.13) можно отредактировать дополнительные свойства.



Рисунок 3.13 – Диалоговое окно, открытое на вкладке **Detail**

В группе **Containment** (Локализация) специфицируются условия хранения атрибута. Здесь можно использовать следующие свойства:

- **By value** (По значению) – свойство по умолчанию, которое означает, что значения атрибута хранятся в пределах адресного пространства, выделенного для класса. Например, если имеется атрибут типа **string**, то значение этой строки содержится в пределах определения класса.
- **By reference** (По ссылке) – означает, что значение атрибута хранится вне адресного пространства, выделенного для объекта данного класса, но у объекта имеется указатель на этот атрибут.
- **Unspecified** (Не определен) – означает, что метод локализации данного атрибута не определен. При генерации программного кода будет выбрано значение **By value**.

Далее имеется возможность задать еще два свойства:

- **Static** (Статичный) – означает, что атрибут имеет одно и то же значение для всех объектов рассматриваемого класса.
- **Derived** (Производный) – означает, что значение атрибута может быть вычислено по значениям других атрибутов.

Рассмотрим пример объявления атрибута «номер карточки (IDCard)».

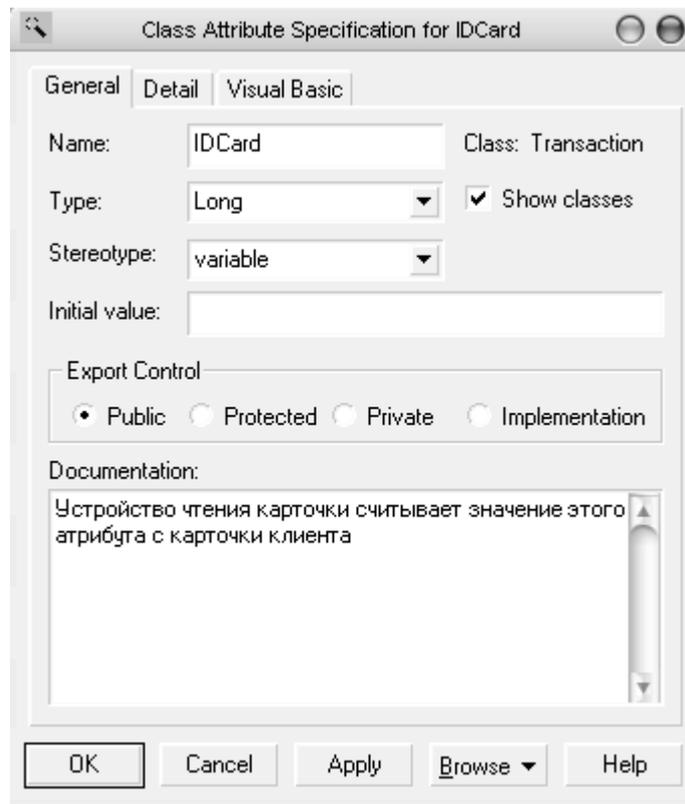
Для атрибута **IDCard** в качестве типа допустимых значений из вложенного списка **Type** следует выбрать тип **Long** (длинное целое), так как номер карточки состоит из 16 десятичных разрядов.

Для задания квантора видимости в группе **Export Control** (Управление экспортом) следует выбрать квантор **public** для того, чтобы этот параметр можно было изменять другим классом.

Поскольку начальное значение для данного атрибута не определено, соответствующее поле ввода следует оставить пустым, а в список стереотипов **Stereotype** ввести **variable** (переменная).

В секцию документации данного атрибута класса можно ввести поясняющий текст, например: «Устройство чтения карточки считывает значение этого атрибута с карточки клиента» и нажать кнопку **Apply** или **OK**, чтобы сохранить результаты редактирования этих свойств атрибута.

Окно спецификации свойств атрибута **IDCard** после редактирования его общих свойств показано на рисунке 5.2.



*Рисунок 5.2 – Диалоговое окно спецификации после редактирования свойств атрибута **IDCard***

Функционирование системы основано на выполнении тех или иных действий, которые представляются с помощью операций классов. Таким образом, следующий этап разработки диаграммы классов связан со спецификацией операций классов.

#### **Последовательность создания операции класса:**

1. Расположить курсор мыши над элементом окна **Browser**, представляющим определенный класс, и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Operation**. Дерево, отображаемое в окне **Browser**, пополнится элементом **opname**, отвечающим новой операции класса, как показано на рисунке 3.9.
3. Выбрать элемент **opname** и ввести требуемое имя операции.

#### **Редактирование операций класса.**

В контексте рассматриваемой модели банкомата в качестве имени первой операции для класса **Transaction** следует задать имя **create** (создать новую транзакцию). При задании имени операции скобки не записываются – программа Rational Rose добавляет их автоматически.

Каждая из операций классов имеет собственное диалоговое окно спецификации свойств **Operation Specification**, которое может быть открыто либо

по двойному щелчку на имени операции, либо в соответствующей вкладке спецификации класса, либо на имени этой операции в браузере проекта.

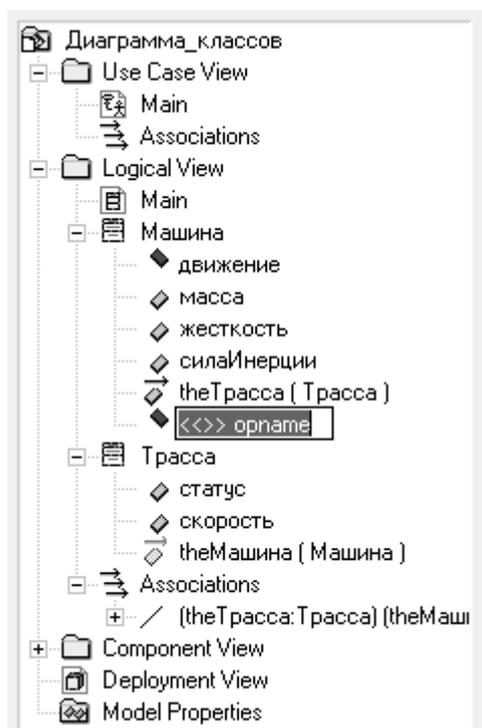


Рисунок 3.9 – Добавление операции класса

Для операции **create** в качестве квантора видимости следует выбрать из вложенного списка квантор **public**, в качестве стереотипа – **call**, а типа возвращаемого результата этой операции – **Boolean** (Логический).

В секцию документации данной операции класса вводится поясняющий текст, например: «Вызывается после того, как карточка вставлена в устройство чтения карточки».

После этого нужно нажать кнопку **Apply** или **OK**, чтобы сохранить результаты редактирования свойств этой операции.

Окно спецификации свойств операции **create** после редактирования ее свойств будет иметь вид, показанный на рисунке 5.3.

Для завершения спецификации операций следует определить и ввести параметры (атрибуты) операций. В сигнатуре операции можно указать ноль и больше параметров в следующем синтаксисе:

#### **Вид параметра Имя : Тип = ЗначениеПоУмолчанию**

Так, например, для операции **checkPINCod()** в качестве вида параметра принимается ключевое слово **in** (входной элемент не может модифицироваться), которое вводить не нужно – это слово устанавливается по умолчанию.

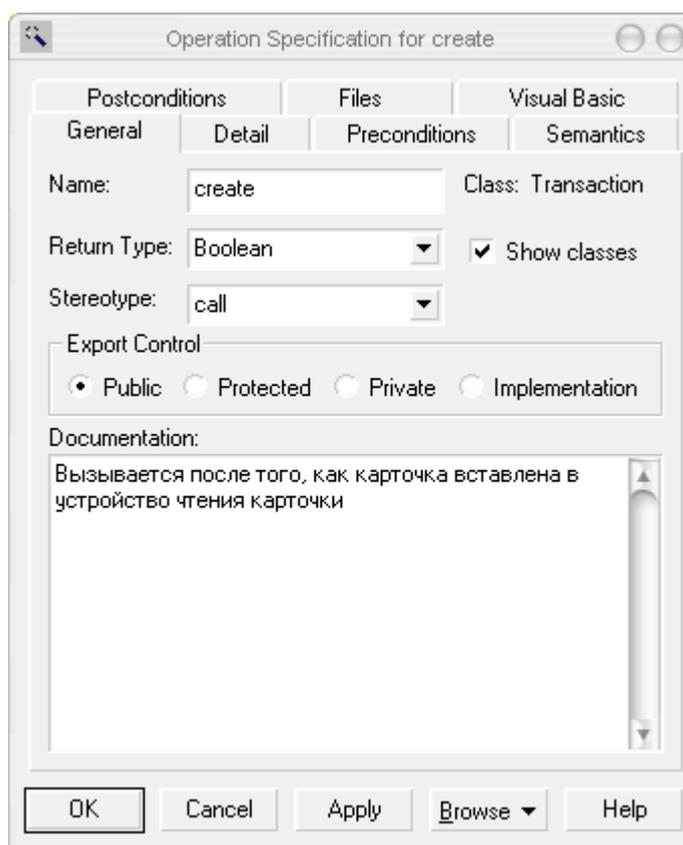


Рисунок 5.3 – Диалоговое окно спецификации свойств операции *create*

Далее для выполнения этой операции задаются два параметра (атрибута) с именами:

- **valueIDCard** (значение ПИН-кода, полученное при идентификации карточки);
- **valuePINCodInput** (значение ПИН-кода, введенное клиентом банкомата).

В качестве типа значений принимается **Integer**, значения по умолчанию не задаются.

Окно спецификации свойств операции **checkPINCod**, открытое на вкладке **Detail**, показано на рисунке 5.4.

На рисунке 5.5 показано окно спецификации свойств класса **Transaction**, открытое на вкладке **Operations**.

После добавления операций и атрибутов изображение класса **Transaction** принимает вид, показанный на рисунке 5.6.

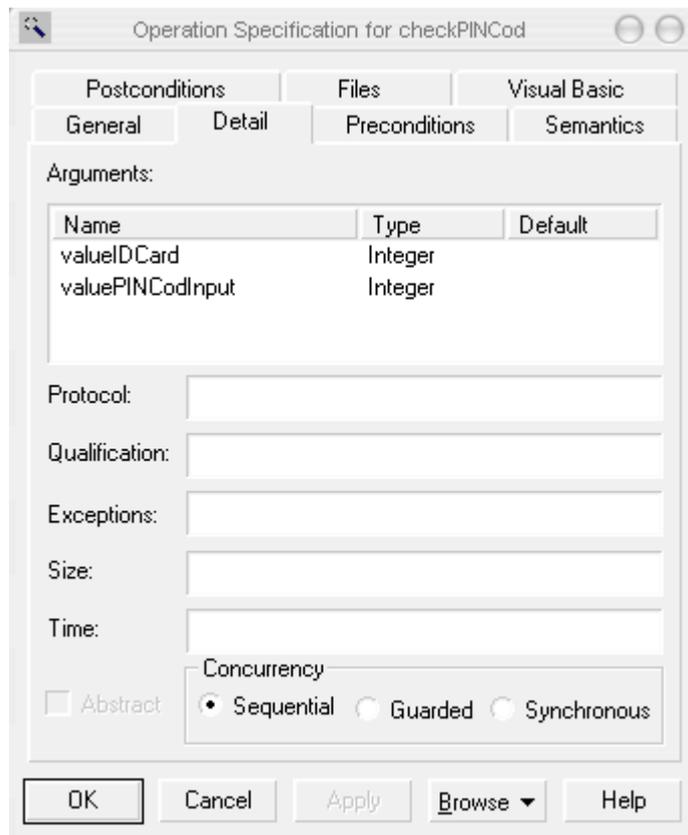


Рисунок 5.4 – Окно спецификации, открытое на вкладке **Detail**

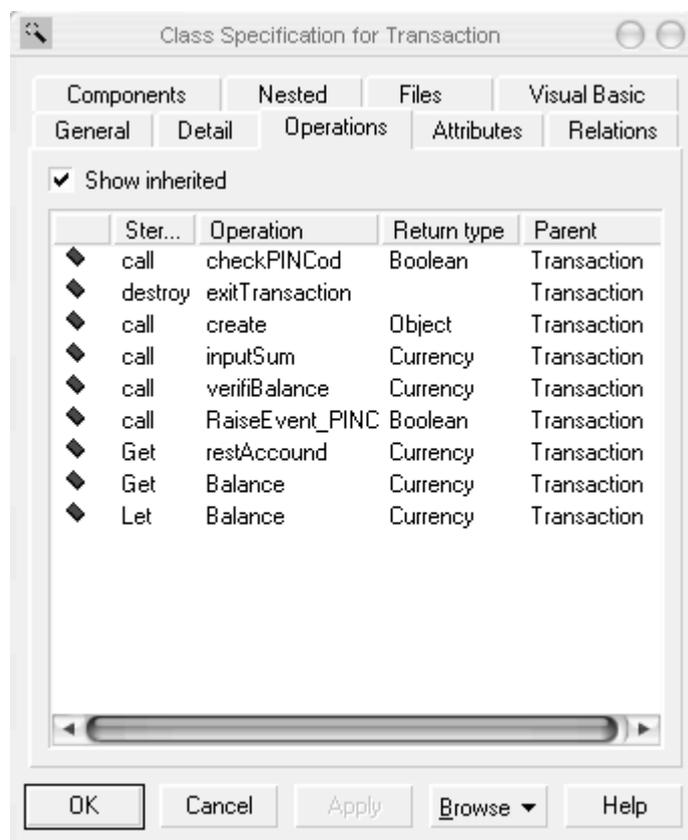


Рисунок 5.5 – Окно спецификации, открытое на вкладке **Operations**

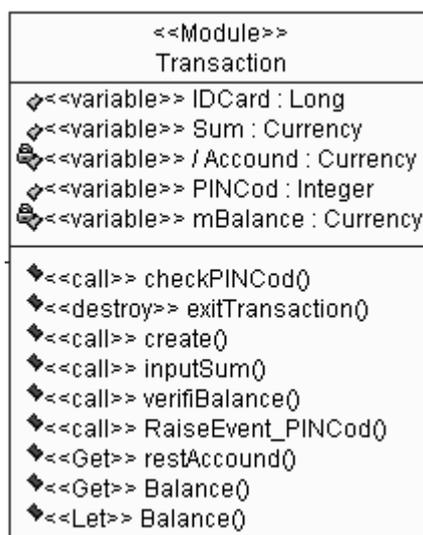


Рисунок 5.6 – Изображение класса *Transaction* после добавления стереотипа, атрибутов и операций

Приведенная последовательность процедур по редактированию атрибутов и операций выполняется для всех классов модели.

**Рекомендации по окончательному построению диаграммы классов.**

На этапе окончательного построения диаграммы классов следует сначала выявить, добавить на диаграмму и специфицировать отношения между классами. При добавлении отношений следует учитывать:

1. Если на диаграмме последовательностей класс **A** посылает сообщение классу **B**, то между этими классами должна быть установлена связь – ассоциация или зависимость.
2. Следует проверить классы на предмет наличия связей типа *целое-часть*. Любой класс, который состоит из других классов, может принимать участие в связях агрегации.
3. Следует обратить внимание на классы, которые имеют несколько типов или вариантов, возможно они имеют связи обобщения.

Одной из особенностей хорошо спроектированного приложения есть сравнительно небольшое количество связей в системе. Класс, у которого много связей, должен знать о большом числе других классов системы – каждая новая связь потребует добавления операций и атрибутов. В результате сложно будет вносить изменения в готовый программный продукт, причем, если изменится какой-либо из классов, это может повлиять на многие классы.

Последовательность процесса выявления и специфицирования отношений можно проследить на примере построения отношений между классами **Controller** и **Transaction**.

В качестве типа связи на начальном этапе выбирается направленная бинарная ассоциация, так как известен порядок следования, *кортеж* классов –

## **Controller ► Transaction.**

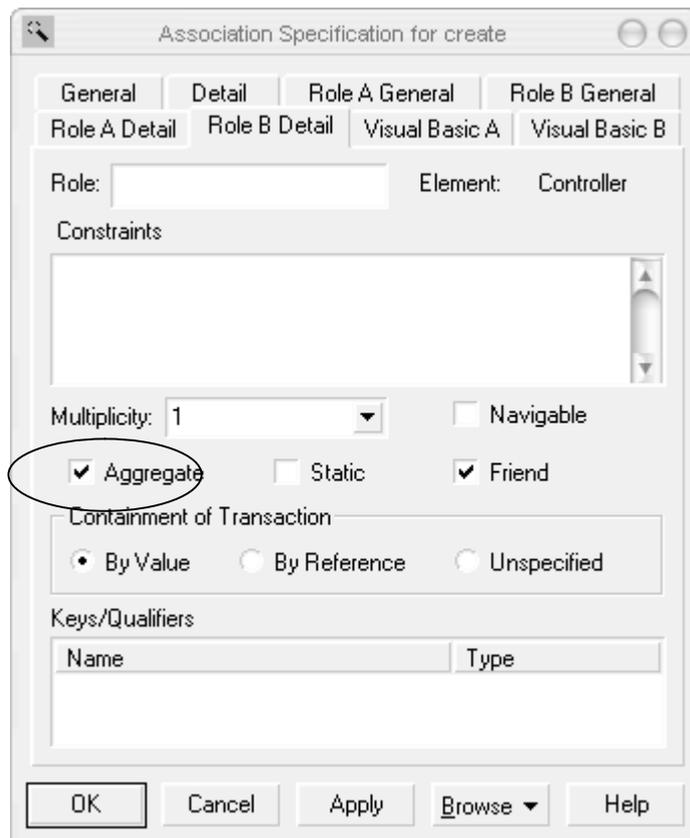
Для задания имени ассоциации следует открыть окно спецификации ассоциаций, а затем на вкладке **General** (Общие) в поле ввода **Name** (Имя) ввести текст ее имени **create** и нажать кнопку **Apply** или **OK**, чтобы сохранить результаты редактирования имени ассоциации.

Далее можно задать: кратность каждого из концов ассоциации, стереотип, ограничения, роли, а также некоторые другие свойства.

Зададим кратность конца ассоциации у класса **Controller** для добавленной на диаграмму ассоциации. Для этого следует в окне спецификации свойств ассоциации перейти на вкладку **Role B Detail** и выбрать значение **1** из вложенного списка **Multiplicity**. Аналогичным образом следует задать кратность конца ассоциации у класса **Transaction**, для чего на вкладке **Role A Detail** из вложенного списка **Multiplicity** следует выбрать значение **1...n**. Содержательно это будет означать, что каждый объект класса **Controller** может быть связан с одним или несколькими объектами класса **Transaction**. Задание кратности можно выполнить с помощью контекстного меню, которое вызывается после выделения линии ассоциации.

Следует учесть, что принятая в общей форме ассоциация может быть уточнена. В частности, между классом **Controller** и классом **Transaction** существует отношение *агрегации* – специальной формы ассоциации для представления отношения *<<часть-целое>>*. В отличие от другой специфической формы – композиции, представляющей физическое соединение целого и части, агрегация обозначает только наличие указателей на агрегируемый объект.

Для спецификации системных атрибутов и операций, необходимых при исполнении соответствующей программы, отношение агрегации будет означать, что класс **Controller** будет включать в себя в качестве составной части класс **Transaction**. При этом уничтожение любого объекта класса **Controller** не должно привести к уничтожению ассоциированных с ним объектов класса **Transaction**. Для специфицирования агрегации на вкладке **Role B Detail** следует установить метку в строке **Aggregate** (рис. 5.7). Остальные опции оставляются без изменений.



*Рисунок 5.7 – Диалоговое окно спецификации свойств ассоциации **create**, открытое на вкладке **Role B Detail***

Поместив на рабочий лист остальные классы, и обозначив их связи, получим окончательный вид диаграммы классов, показанный на рисунке 5.8.

### **3.7 Содержание отчета**

Отчет по работе должен содержать диаграммы классов, снабженные соответствующими связями, атрибутами, операциями и спецификациями. При защите отчета необходимо объяснить приемы работы с программой Rational Rose, а также обосновать принятые решения.

## 4 ПОДГОТОВКА МОДЕЛИ ДЛЯ ГЕНЕРАЦИИ ПРОГРАММНОГО КОДА

*Цель работы:* освоение приемов и методики генерации программного кода системы с применением программы Rational Rose.

Общая последовательность действий, которые необходимо выполнить для генерации программного кода в среде Rational Rose, состоит из следующих этапов:

1. Проверка модели на отсутствие ошибок.
2. Создание компонентов для реализации классов.
3. Отображение классов на компоненты.
4. Выбор языка программирования для генерации программного кода.
5. Установка свойств генерации программного кода.
6. Выбор класса, компонента или пакета.
7. Генерация программного кода.

Особенности выполнения каждого из этапов могут изменяться в зависимости от выбора языка программирования или схемы базы данных. В среде Rational Rose предусмотрено задание достаточно большого числа свойств, характеризующих как отдельные классы, так и проект в целом.

### 4.1 Проверка модели на отсутствие ошибок

В общем случае проверка модели может выполняться на любом этапе работы над проектом. Однако после завершения разработки графических диаграмм она является **обязательной**, поскольку позволяет выявить целый ряд ошибок разработчика. К числу таких ошибок и предупреждений относятся, например, не используемые ассоциации и классы, оставшиеся после удаления отдельных графических элементов с диаграмм, а также операции, не являющиеся именами сообщений на диаграммах взаимодействия.

В процедуру проверки включаются диаграммы последовательности и диаграммы классов. В связи с этим указанные диаграммы следует привести в соответствие друг другу – имена классов, операций и атрибутов должны быть одинаковыми. Кроме этого на диаграмме последовательностей необходимо задать имена объектов каждого класса.

Редактирование диаграммы последовательностей следует начать со спецификации объектов. Для этого нужно выделить класс на диаграмме, вызвать контекстное меню и выбрать **Open Specification...** (окно спецификации объекта). В этом окне имеется одна вкладка – **General**, на которой размещены: поле имени объекта **Name**, поле имени класса **Class**, поле группы свойств **Persistence** и переключатель **Multiple Instances**.

Группа свойств **Persistence** (Устойчивость) предназначена для спецификации устойчивости объектов:

- **Persistent** (Устойчивый) означает, что информация об объекте должна быть сохранена в системе;
- **Static** (Статический) означает, что информация об объекте сохраняется в течение всего жизненного цикла системы;
- **Transient** (Временный) означает, что информация об объекте хранится только в течение короткого времени, необходимого для выполнения операции.

Отметка у свойства **Multiple Instances** (Несколько экземпляров) ставится только в случае, если существует несколько экземпляров такого объекта.

Для изменения имени класса в окне спецификации свойств объекта **Object Specification** (по умолчанию в заголовке окна стоит **Untitled** – без заголовка) следует выбрать **Browse ► Browse Class**. При этом открывается окно **Class Specification**, в поле **Name** которого вводится новое имя класса.

Вид окна **Object Specification** после редактирования свойств класса **УстрЧтКарт** показан на рисунке 5.1.

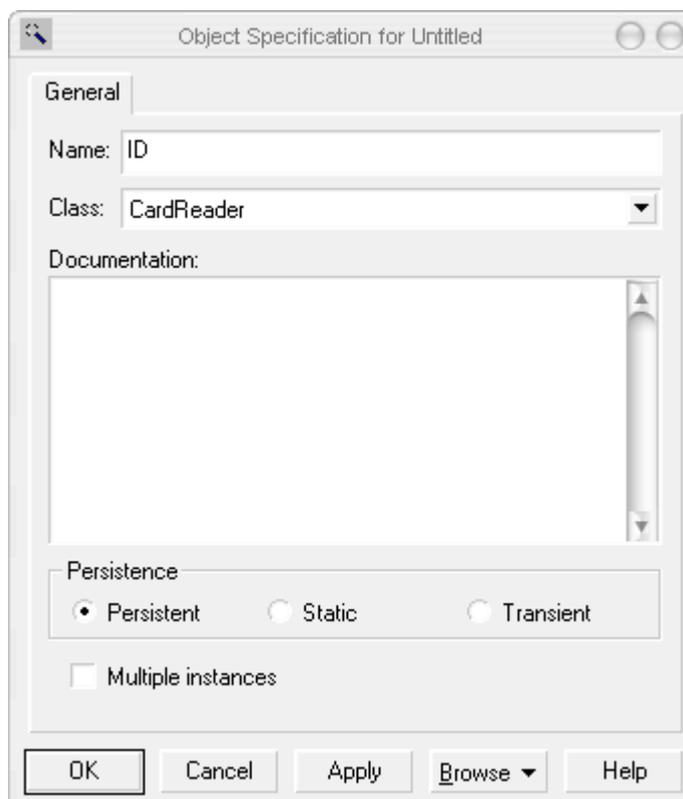


Рисунок 5.1 – Окно спецификации свойств объекта **ID** (идентификационное устройство) класса **УстрЧтКарт** с новым именем **CardReader** после редактирования

Для проверки модели нужно выполнить операцию главного меню **Tools ► Check Model** (Инструменты ► Проверить модель). Результаты проверки разработанной модели на наличие ошибок отображаются в окне журнала. Прежде чем приступить к генерации текста программного кода разработчику следует добиться устранения всех ошибок и предупреждений, о чем должно свидетельствовать чистое окно журнала.

## 5.2 Создание компонентов

Хотя программа Rational Rose позволяет генерировать программный код для каждого *класса* модели, имеет смысл разработать *диаграмму компонентов*, которая необходима для генерации программного кода *системы*.

После активизации диаграммы компонентов следует задать имя диаграммы, например: **Диаграмма компонентов АТМ**. Для первого добавленного компонента зададим имя – **MainATM.exe**.

Редактирование свойств произвольного компонента осуществляется с помощью диалогового окна спецификации свойств **Component Specification** (рис. 6.3). В частности, для компонента **MainATM.exe** из предлагаемого вложенного списка выбирается стереотип **<<EXE>>**, поскольку этот компонент будет реализован в форме исполнимого файла.

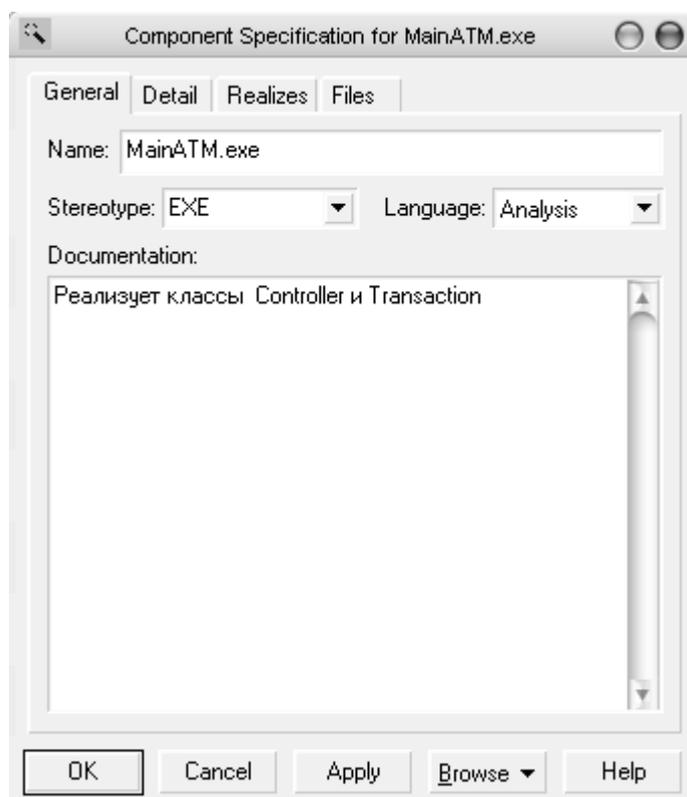


Рисунок 6.3 – Диалоговое окно *Component Specification*

Продолжая разработку модели банкомата, добавим на диаграмму компонентов второй компонент с именем **MainBank**, для которого выбираем стереотип **Main Program**. После этого нужно добавить отношение зависимости от компонента с именем **MainATM.exe** к компоненту с именем **MainBank**. Для наглядности диаграмму следует снабдить примечаниями, где указываются классы модели, реализованные в данных компонентах.

Окончательный вид диаграммы компонентов разрабатываемой модели управления банкоматом показан на рисунке 6.4.

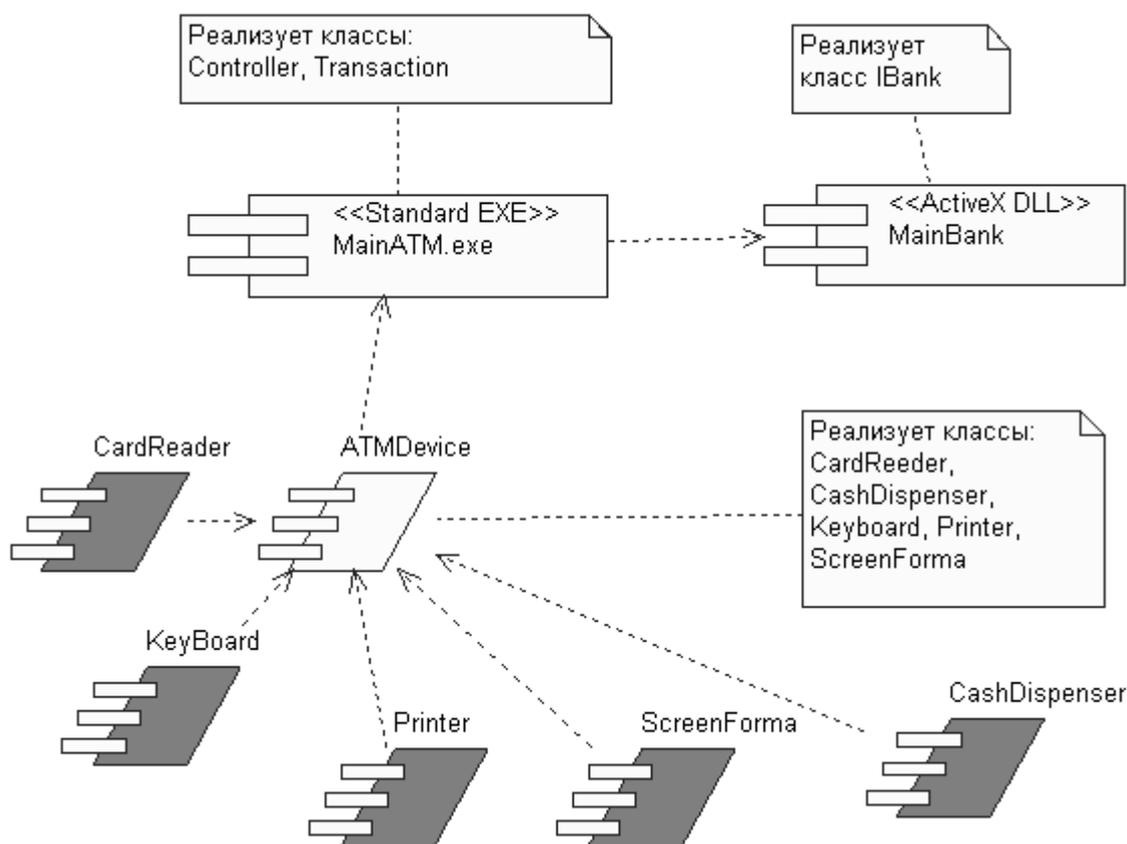


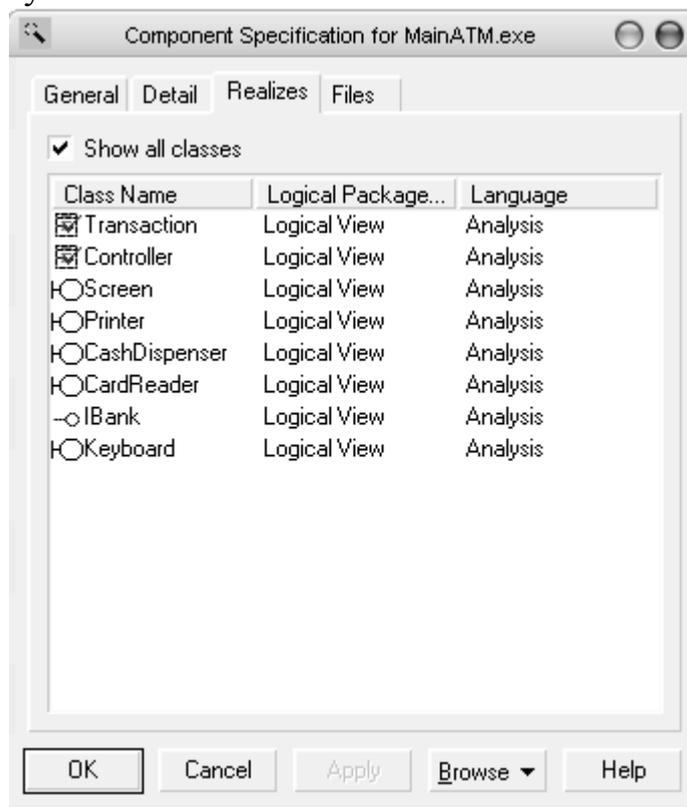
Рисунок 6.4 – Диаграмма компонентов системы управления АТМ

### Отображение классов на компоненты.

Для отображения классов на компоненты можно воспользоваться окном спецификации свойств компонента, открытого на вкладке **Realizes** (Реализует). Для включения реализации класса в данный компонент следует выделить требуемый класс на этой вкладке и выполнить для него операцию контекстного меню **Assign** (Назначить). В результате перед именем класса появится специальная отметка.

Применительно к модели банкомата для генерации программного кода

выбираются классы **Transaction** и **Controller** компонента **MainATM.exe**. В результате этого окно спецификации компонента на вкладке **Realizes** приобретает вид, показанный на рисунке 6.5.



*Рисунок 6.5 – Диалоговое окно настройки свойств реализации классов в компоненте **MainATM.exe***

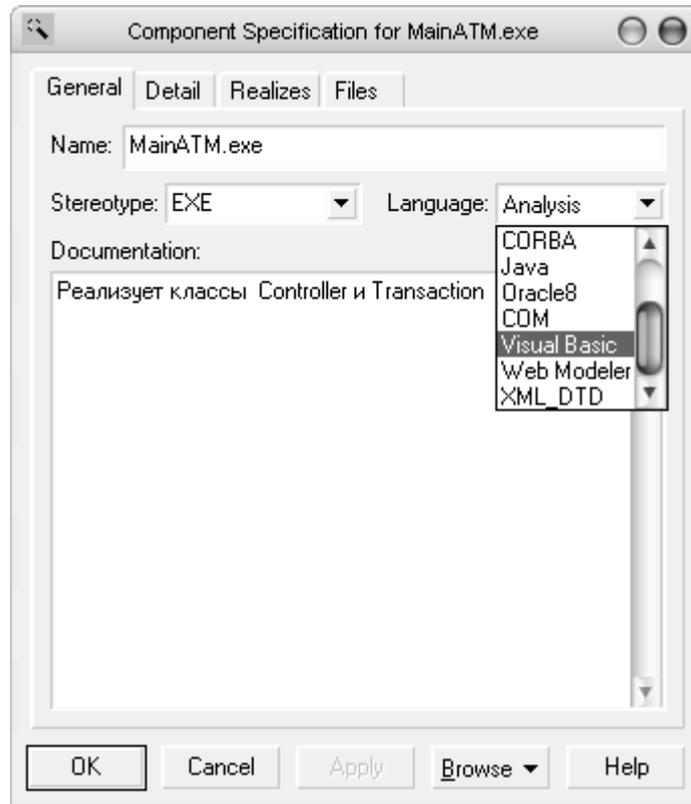
### **Выбор (назначение) языка программирования.**

Для назначения языка реализации модели следует вызвать диалоговое окно настройки параметров модели с помощью операции главного меню **Tools ► Options** (Инструменты ► Параметры). Далее на вкладке **Notation** (Нотация) в строке **Default Language** (Язык по умолчанию) из вложенного списка следует выбрать язык – **Visual Basic**.

После выбора языка программирования по умолчанию следует изменить язык реализации каждого из компонентов модели. С этой целью необходимо открыть диаграмму компонентов и выделить на ней тот компонент, для которого предполагается генерация кода. Далее с помощью операции контекстного меню нужно открыть окно спецификации свойств компонента, перейти на вкладку **General** (Общие) и изменить язык **Analysis** в строке **Language** (Язык) на **Visual Basic** (рис. 6.6), после чего закрыть окно спецификации, нажав **OK**.

Чтобы удостовериться в установке языка реализации выбранных классов, следует выделить на диаграмме соответствующий компонент (**MainATM.exe**), вызвать окно спецификации свойств компонента, в котором открыть вкладку

**Realizes.** При наличии метки **Show all classes** в списке классов выбранные для программной реализации классы **Controller** и **Transaction** будут иметь специальные отметки, а в столбце **Language** будет указан язык реализации – **Visual Basic** (рис. 6.7).



*Рисунок 6.6 – Окно спецификации свойств компонента **MainATM.exe** при выборе языка его реализации*

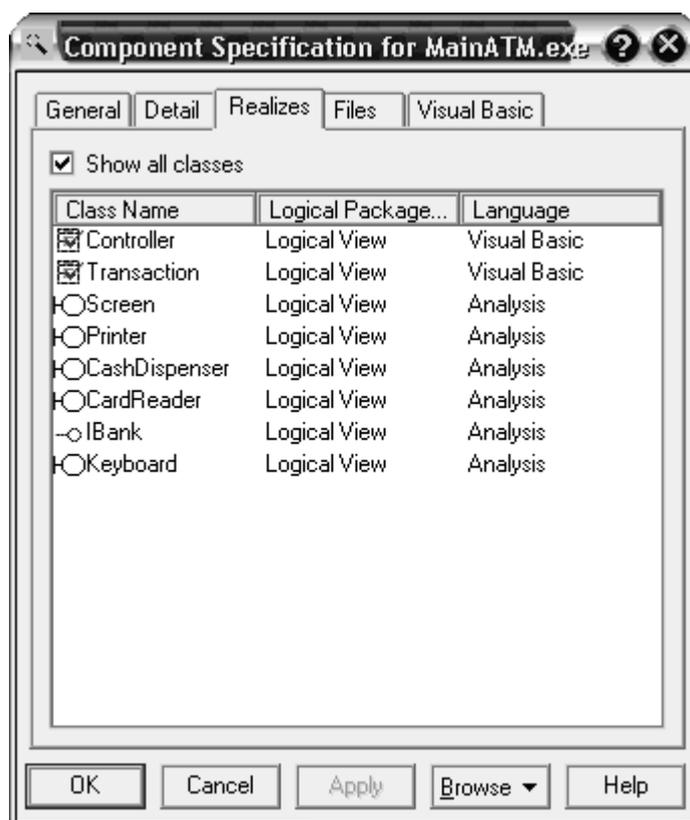


Рисунок 6.7 – Окно спецификации свойств компонента *MainATM.exe*, открытое на вкладке *Realizes*

Для классов, которые не соотнесены с компонентом **Visual Basic**, указывается язык по умолчанию – **Analysis**.

Следует заметить, что после выбора языка программирования необходимо привести в соответствие имена классов, операции, типы атрибутов и аргументов, а также типы возвращаемых значений операций. С этой целью нужно просмотреть все классы диаграммы классов и изменить те типы данных, которые не являются синтаксически допустимыми в выбранном языке программирования.

#### ***Задание свойств кода средствами приложения Model Assistant Tool.***

Приложение **Model Assistant Tool** позволяет отобразить элементы модели Rational Rose в соответствующих конструкциях языка Visual Basic или Visual C++. Для Visual Basic приложение позволяет создать и определить выражения, обработчики событий, методы и их параметры, процедуры **Get**, **Let** и **Set** для свойств класса и их ассоциаций.

Приложение **Model Assistant Tool** доступно, если выполнены два условия:

- В качестве языка программирования выбран язык Visual Basic или Visual C++;
- Класс соотнесен с компонентом Visual Basic или Visual C++.

Приложение **Model Assistant Tool** можно активизировать следующим образом:

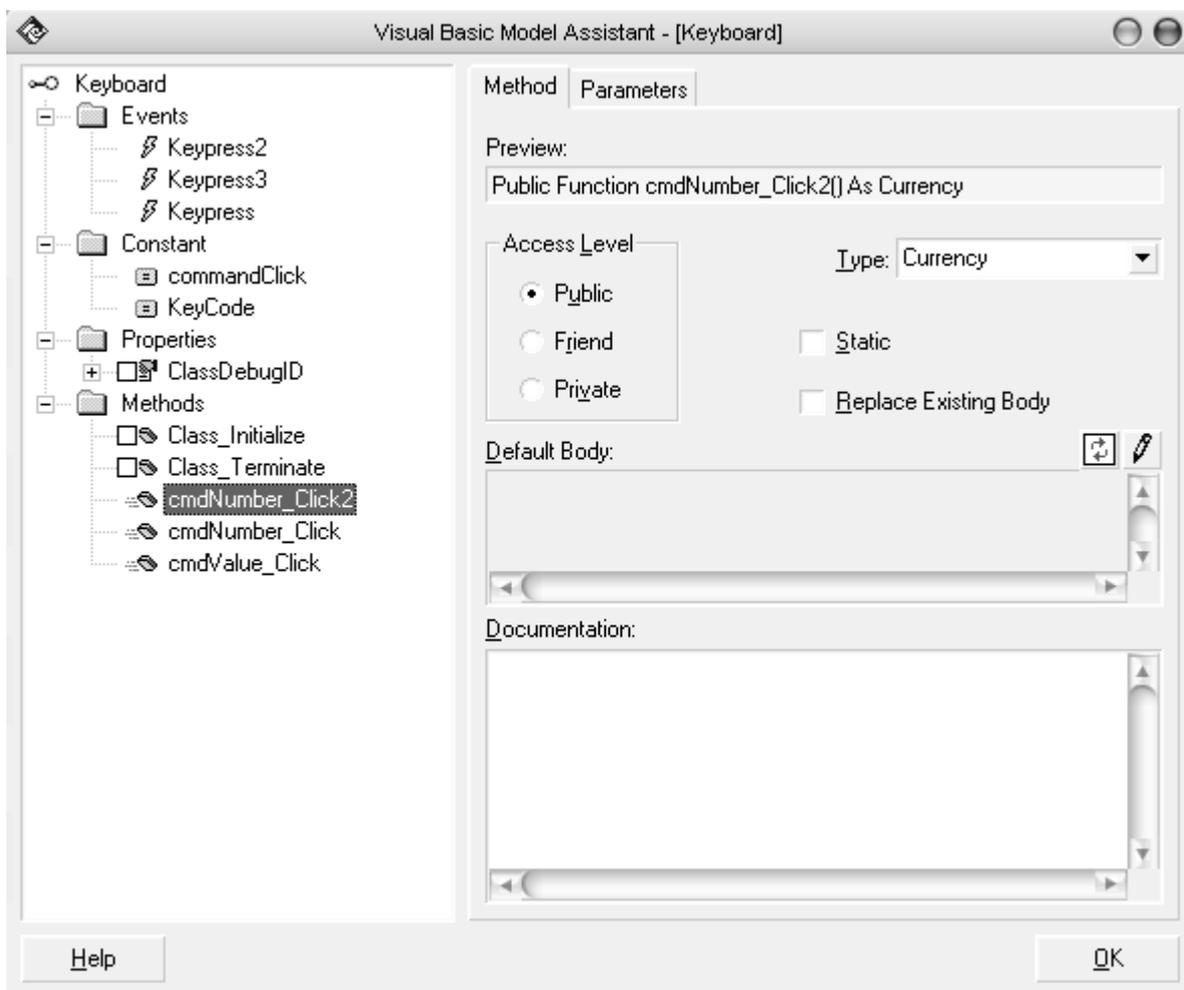
1. Щелчком правой кнопки мыши на имени класса в браузере или на диаграмме открыть контекстное меню.
2. Выбрать элемент меню **Model Assistant**.

На рисунке 6.8 показан вид окна **Visual Basic Model Assistant** для класса **Keyboard**. В поле **Preview** видно строку текста программного кода для метода **cmdNumber\_Click2**.

Просматривая события (events), константы (constant), свойства (propertis) и методы (methods), можно скорректировать текст будущей программы или привести её в соответствие с требованиями языка программирования.

### *Выбор класса, компонента или пакета для генерации кода.*

Выбор класса или пакета для генерации программного кода означает выделение соответствующего элемента в браузере проекта. Применительно к рассматриваемому примеру для генерации программного кода на языке Visual Basic выберем компонент **MainATM.exe**. Выделив компонент в браузере проекта, следует щелкнуть правой кнопкой мыши и в открывшемся контекстном меню выбрать операцию **Update Code**.



*Рисунок 6.8 – Окно **Visual Basic Model Assistant** для класса **Transaction***

### **Генерация программного кода.**

Управление процессом генерации программного кода осуществляется с помощью специального приложения **Code Update Tool**, которое выводит ряд окон для контроля хода этого процесса.

В окне выбора компонентов и классов (рис. 6.9) в левом поле следует установить метки в определенных компонентах и классах. При этом в правом поле можно отобразить (в зависимости от варианта выбора) не только выбранные компоненты, но также операции и атрибуты отдельных классов.

На рисунке 6.9 показан пример предварительного просмотра программного кода для класса **Transaction**. Выведенную в правом поле информацию можно отредактировать. Для этого необходимо выделить свойство, функцию или переменную и произвести редактирование программного кода, который выводится в нижней части окна. Закончив редактирование, нужно перейти к следующему этапу, нажав на кнопку **Next**.

В каждом сгенерированном методе Rose добавляет идентификатор – **modelID**, чтобы идентифицировать соответствующий метод в модели. Эти идентификаторы редактировать нельзя.

Следующее окно (**Finish**) приложения **Code Update Tool** показывает, какие классы и компоненты включены в процесс генерации программного кода. Убедившись в правильности хода процесса, нужно нажать **Finish**.

Результаты процесса генерации программного кода представляются в окне **Summary** (рис. 6.10).

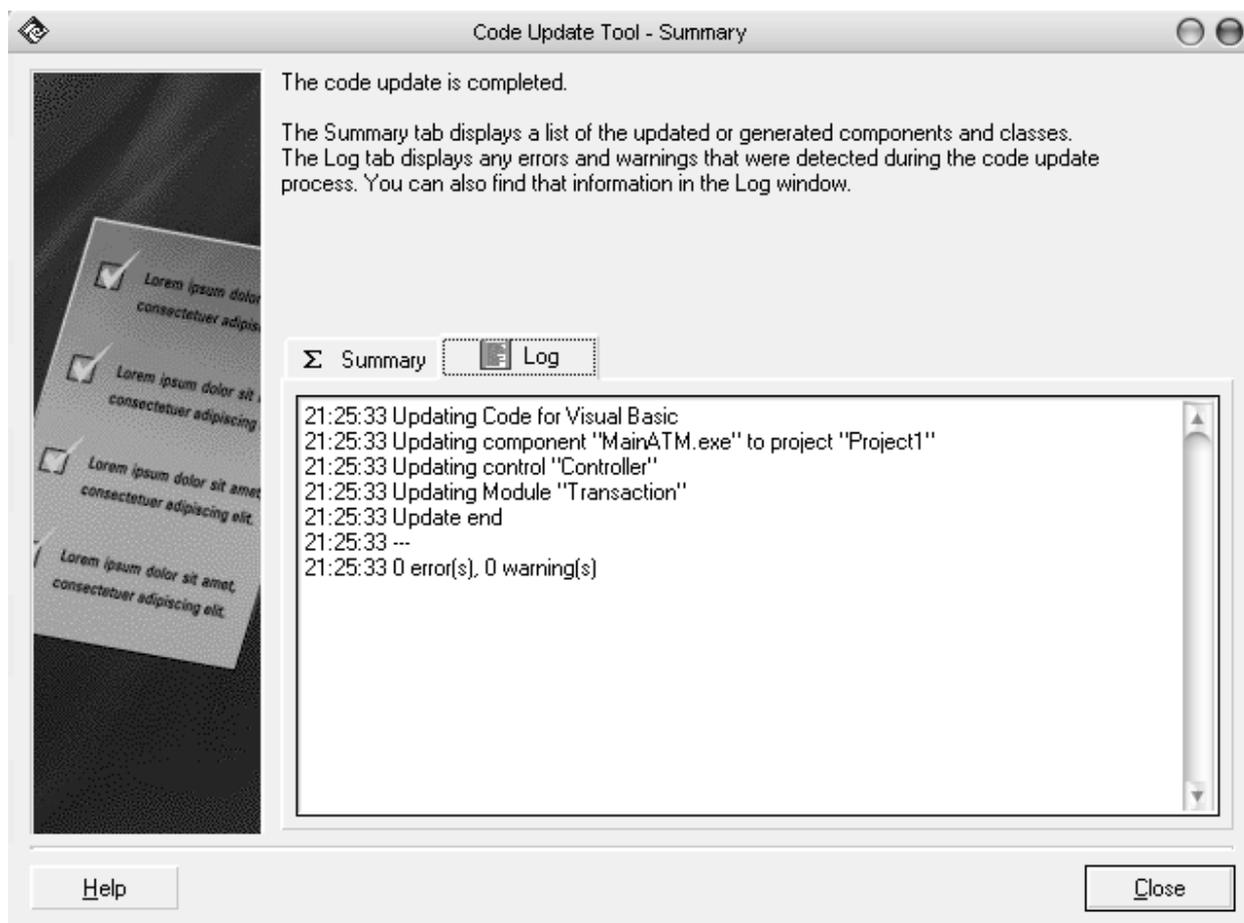


Рисунок 6.10 – Окно вывода результата процесса генерации кода для компонента *MainATM.exe*

После закрытия окна **Summary** программа Rational Rose запрашивает необходимость сохранения исходной модели, после чего устанавливает на панели задач кнопку с наименованием проекта программного кода. В проект включаются файлы со следующими расширениями:

- .cls** – программный код класса;
- .bas** – программный код класса отладки программы;
- .vbp** – программный код компонента.

Все эти файлы можно найти в пакете Microsoft Visual Studio в каталоге

VB98.

Ниже приведен текст программного кода для класса **Controller**, записанный в файле **Controller.cls**.

```
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
Attribute VB_Name = "Controller"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "RVB_Uniqueld" ,"453065FB02CE"
Attribute VB_Ext_KEY = "RVB_ModelStereotype" ,"control"
Option Explicit
Option Base 0
Option Compare Text

'##ModelId=4531FF5303A9
Public real As Controller

'##ModelId=4531FF530399
Public Virtual As Collection

'##ModelId=4534B8720177
Public Identification As CardReader

'##ModelId=4534B8780167
Public DelliveryOfMoney As CashDispenser

'##ModelId=4534B87D038A
Public interface As Keyboard

'##ModelId=4534B88801E4
Public mPrint As Printer

'##ModelId=45443437007D
Public output As Collection
```

```
'##ModelId=4554D5830222
```

```
Public Server As IBank
```

Из рассмотрения полученного заголовочного файла видно, что в нем содержится только объявление ассоциаций. Однако это объявление проведено в соответствии с правилами синтаксиса языка Visual Basic.

Ниже приведен файл реализации класса **CardReader** компонента **ATMDevice**, в котором объявляются атрибуты и функции, а также комментарий, внесенный в поле документации класса.

```
VERSION 1.0 CLASS
```

```
BEGIN
```

```
MultiUse = -1 'True
```

```
Persistable = 0 'NotPersistable
```

```
DataBindingBehavior = 0 'vbNone
```

```
DataSourceBehavior = 0 'vbNone
```

```
MTSTransactionMode = 0 'NotAnMTSObject
```

```
END
```

```
Attribute VB_Name = "CardReader"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = True
```

```
Attribute VB_PredeclaredId = False
```

```
Attribute VB_Exposed = False
```

```
Attribute VB_Description = "Устройство принимает карточку, считывает идентификационный код и по сигналам контроллера либо возвращает карточку клиенту, либо блокирует её и сбрасывает в хранилище карт"
```

```
Attribute VB_Ext_KEY = "RVB_UniqueId" ,"453065C5006D"
```

```
Attribute VB_Ext_KEY = "RVB_ModelStereotype" ,"boundary"
```

```
Option Explicit
```

```
Option Base 0
```

```
Option Compare Text
```

```
'##ModelId=454CE4F3001F
```

```
Private mIDCard As Long
```

```
'##ModelId=4534B4F600EA
```

```
Public Event lockCard()
```

```
'##ModelId=454E00F601C5
```

```
Public Event inputCard()
```

```
'##ModelId=4555793E0280
```

```
Public Event RaiseEvent_IDCard()
```

```
'##ModelId=4530B7530128
```

```
Public Function transferIDCode() As Long
```

```
End Function
```

```
'##ModelId=4530C1DB01B5
```

```
Public Sub ejectCard()
```

```
End Sub
```

```
'##ModelId=45576F920109
```

```
Public Property Get IDCard() As Variant
```

```
End Property
```

В сгенерированных Rational Rose файлах каждая из операций имеет пустое тело реализации, которое следует дополнить, исходя из функциональных требований модели и синтаксиса языка программирования.

В заключение необходимо отметить, что эффект от использования средства Rational Rose проявляется при разработке масштабных проектов в составе команды или проектной группы.

Действительно, может сложиться впечатление, что написать и отладить программу непосредственно в среде программирования гораздо проще без её моделирования.

Однако ситуация покажется не столь тривиальной, когда нужно выполнить проект с несколькими десятками вариантов использования и сотней классов. Именно для подобных проектов выявляется явное преимущество использования нотации языка UML и CASE-средства Rational Rose, позволяющих упростить *создание и документирование* программных систем.

## 4 РАЗРАБОТКА ДИАГРАММ ВЗАИМОДЕЙСТВИЯ

*Цель лабораторной работы:* освоение приемов и методики разработки диаграмм взаимодействия с применением программы Rational Rose.

### 4.1 Рекомендации по созданию диаграмм взаимодействия

Функции, охватываемые вариантом использования, фиксируются в потоке событий, а для описания способов реализации вариантов использования применяются сценарии.

*Сценарий (scenario)* – это экземпляр варианта использования, один из возможных потоков событий для этого варианта. Сценарии помогают идентифицировать объекты, разработать адекватные классы и выявить примеры взаимодействия объектов *в процессе выполнения функций*, предусмотренных вариантом использования. Сценарии документируют решения о том, каким образом функции, возлагаемые на вариант использования, распределяются между объектами и классами системы.

Поток событий для варианта использования описывается в текстовом виде, а сценарии представляются в форме *диаграмм взаимодействия (interaction diagrams)*. Различают два типа таких диаграмм – *диаграммы последовательностей (sequence diagrams)* и *диаграммы сотрудничества (collaboration diagrams)*.

*Диаграмма последовательностей* иллюстрирует очередность выполнения операций взаимодействия объектов во времени и отображает объекты и классы, вовлеченные в сценарий. На диаграммах последовательностей показываются также сообщения, которыми объекты обмениваются в ходе осуществления функций, предусмотренных сценарием. Диаграммы последовательностей обычно ассоциируются с реализациями вариантов использования, перечисленными в пакете **Logical View**.

В соответствии с требованиями UML объект на диаграмме последовательностей изображается в виде прямоугольника, содержащего *подчеркнутое наименование объекта*. Объект можно именовать тремя способами: указать только его название, задать имена объекта и класса либо ограничиться наименованием класса (для анонимного объекта).

Каждому объекту на диаграмме последовательностей ставится в соответствие временная отметка, обозначаемая отрезком штриховой линии, а сообщения, которыми обмениваются два объекта, представляются стрелкой, соединяющей объект-источник с объектом-приемником.

В диаграммы последовательностей целесообразно включать граничные классы, позволяющие отобразить факты взаимодействия системы с активными субъектами (пользователями и сторонними системами). На ранних стадиях анализа такой прием позволяет зафиксировать и документировать требования к интерфейсам.

При разработке диаграмм последовательностей следует стремиться к простоте их представления, что позволяет сразу видеть и объекты, и сообщения, которыми объекты обмениваются, и функции, охватываемые сценарием.

Если на диаграммах последовательностей должны быть реализованы условные логические выражения *если-то* и *иначе*, которые воспроизводят реалии предметной области, то их можно включить в одну диаграмму, используя дополнительные примечания для описания возможных вариантов выбора. Однако лучше создавать отдельные диаграммы – одну для случая *если-то*, а другую для иллюстрации ветви *иначе*. Это позволит упростить восприятие модели.

## 4.2 Методика разработки диаграмм взаимодействия

Активизировать рабочее окно диаграммы последовательностей можно следующими способами:

- Щелкнуть на кнопке с изображением диаграммы взаимодействия на стандартной панели инструментов (**Browse Interaction Diagram**), в открывшемся окне **Select Interaction Diagram** дважды щелкнуть левой кнопкой мыши на поле **<New>** и в открывшемся диалоговом окне **New Interaction Diagram** (рис. 4.1) выбрать диаграмму последовательностей (**Sequence**) и внести имя диаграммы в поле **Title**.

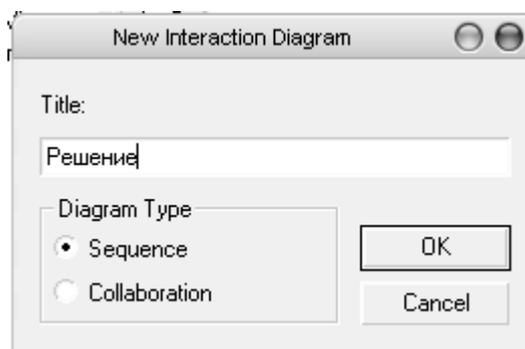
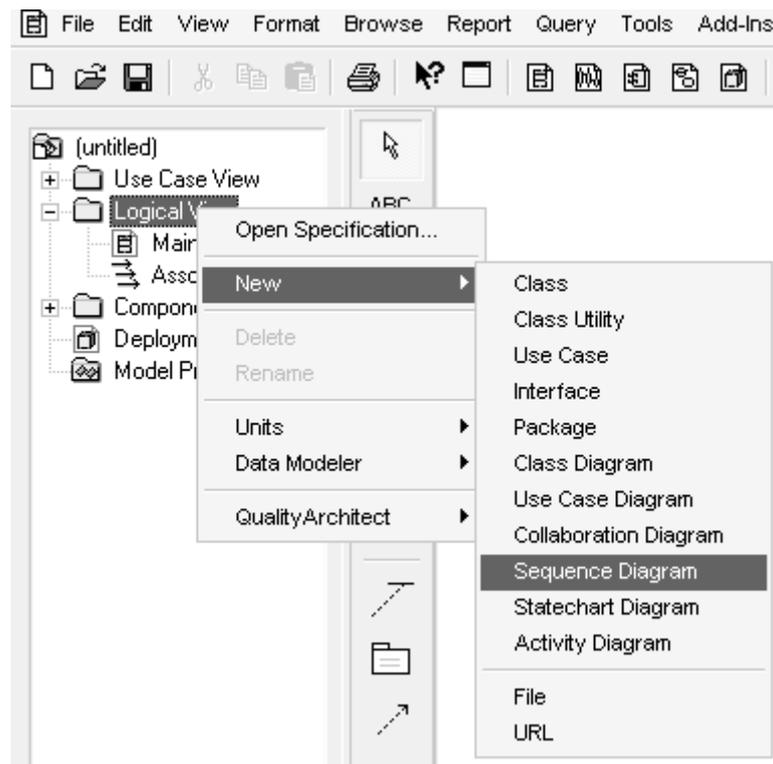


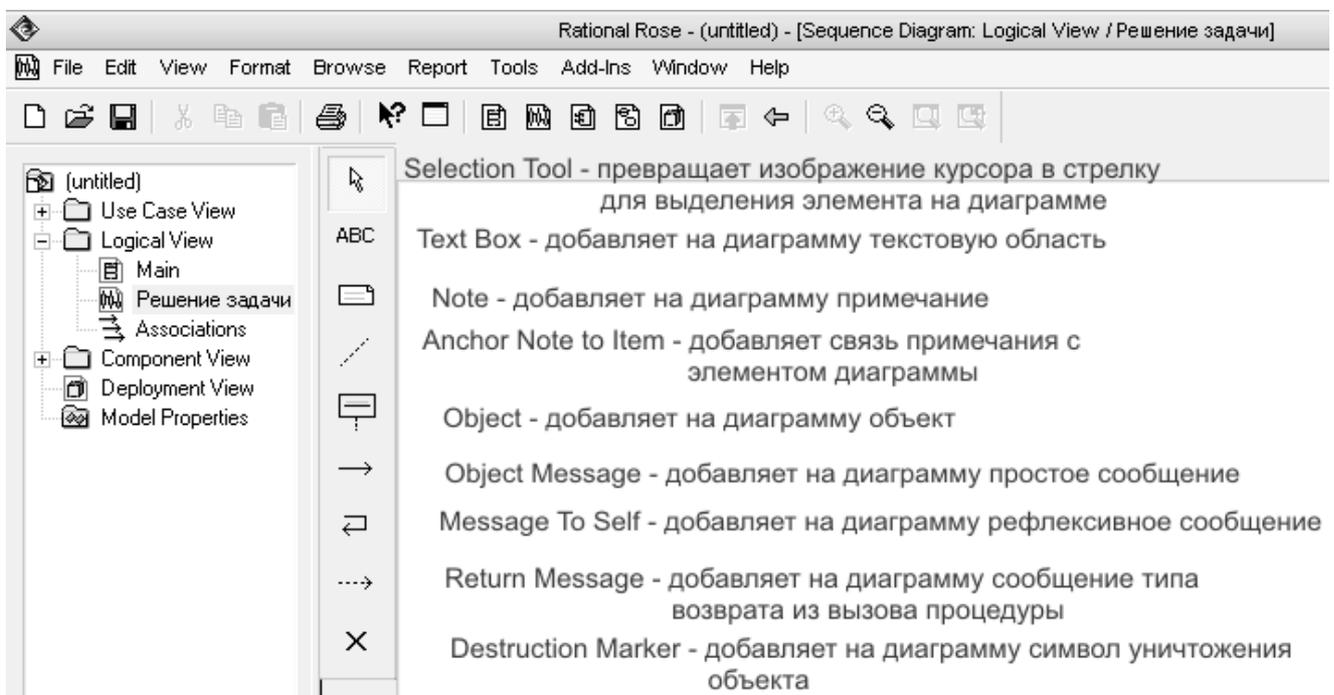
Рисунок 4.1 – Выбор диаграммы последовательностей и внесение её наименования

- Расположить курсор мыши над элементом **Logical View** окна **Browser**, активизировать правой кнопкой контекстное меню, выбрать элементы меню **New ► Sequence Diagram**, как показано на рисунке 4.2. Далее внести имя диаграммы в окне **Browser** и дважды щелкнуть по нему левой кнопкой мыши.

После выбора диаграммы последовательностей вместо специальной панели инструментов для диаграммы классов появляется специальная панель инструментов, содержащая кнопки с изображениями примитивов, необходимые для разработки диаграммы последовательностей (рис. 4.3).



*Рисунок 4.2 – Выбор диаграммы последовательностей с помощью контекстного меню*

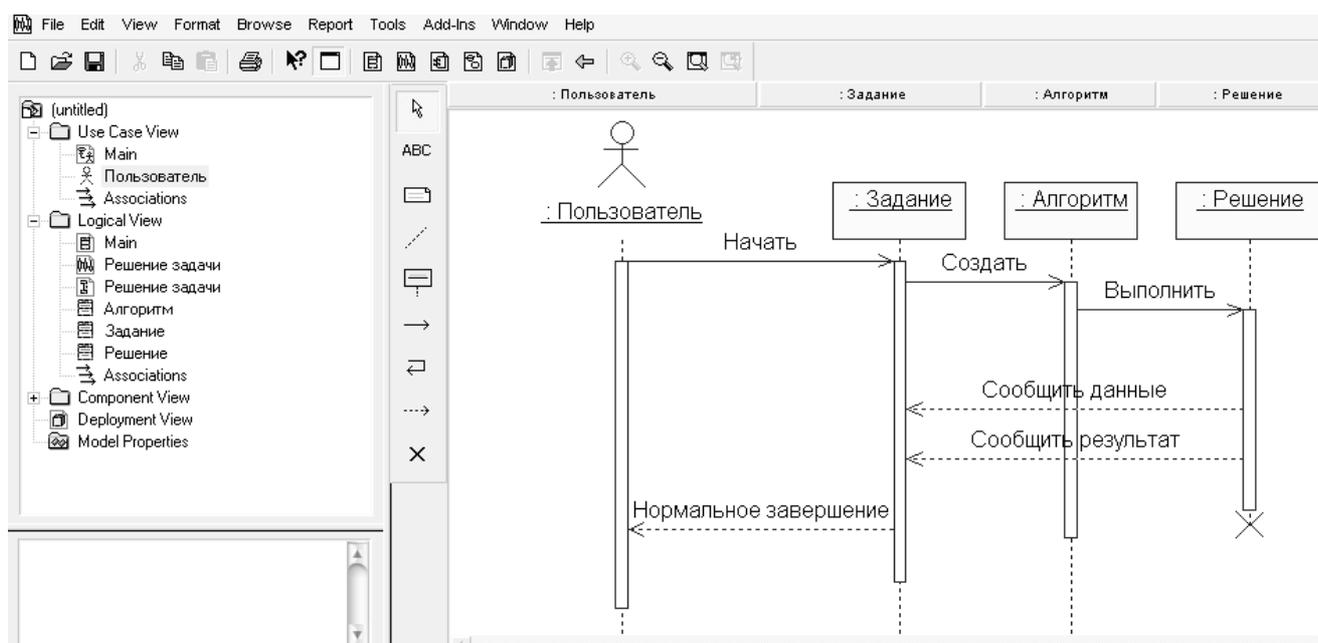


*Рисунок 4.3 – Специальная панель инструментов для разработки диаграммы последовательностей*

### *Создание объектов и сообщений в диаграмме последовательностей:*

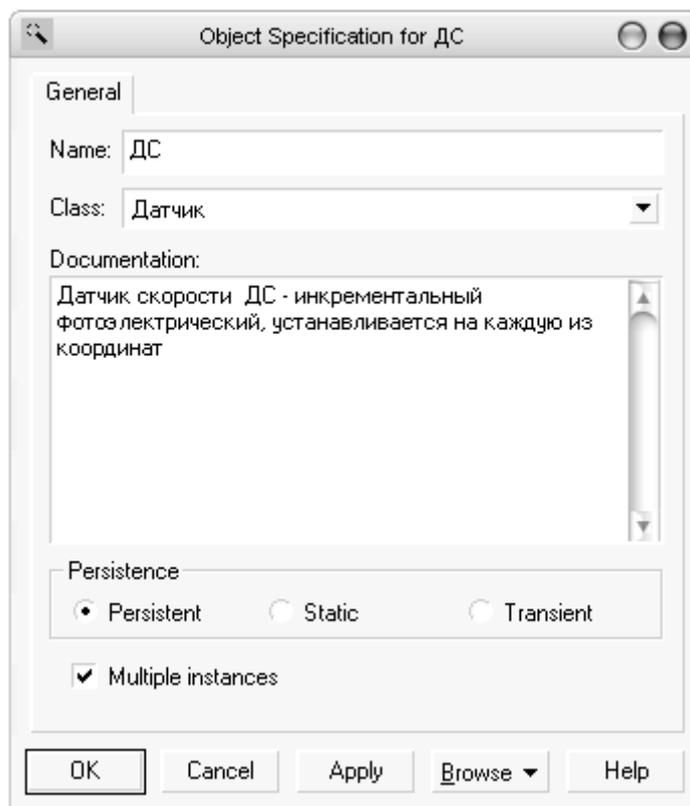
1. В окне **Browser** выбрать элемент, соответствующий требуемому активному субъекту, и перетащить его в рабочее окно диаграммы.
2. Щелкнуть на пиктограмме **Object** специальной панели инструментов.
3. Щелкнуть в соответствующей позиции рабочей области окна диаграммы, чтобы разместить в ней новый объект.
4. Выбрать объект на диаграмме и ввести его имя в окне спецификации объекта.
5. Повторить действия, указанные в пп. 2-4, для всех объектов, участвующих в реализации сценария.
6. Щелкнуть на пиктограмме **Object Message** специальной панели инструментов.
7. В окне диаграммы щелкнуть на символе активного субъекта (объекта), служащего источником, и, не отпуская кнопку мыши, построить линию сообщения, направленную к символу соответствующего активного субъекта (объекта), который является приемником.
8. Дважды щелкнуть на линии, представляющей сообщение, чтобы открыть диалоговое окно **Message Specification**, в поле **Name** ввести текст сообщения и закрыть окно щелчком на кнопке **ОК**.
9. Повторить действия, указанные в пп. 6-8, для всех сообщений, инициируемых при реализации сценария.

Диаграмма последовательностей для сценария "Решение задачи" изображена на рисунке 4.4.



*Рисунок 4.4 – Диаграмма последовательностей для сценария "Решение задачи"*

Каждый добавляемый на диаграмму объект по умолчанию считается анонимным – выводится двоеточие и наименование класса. Для внесения имени объекта следует вызвать диалоговое окно свойств объекта (рис. 4.5), дважды щелкнув на его изображении.



*Рисунок 4.5 – Диалоговое окно спецификации свойств объекта класса Датчик*

В окне спецификации свойств после введения имени, например, датчика скорости (ДС), необходимо также задать свойства устойчивости (**Persistence**) и множественности (**Multiple instances**).

Свойство **Persistent** (Устойчивый) означает, что информация об объектах данного класса должна сохраняться постоянно, независимо от используемого приложения.

Свойство **Static** (Статический) означает, что соответствующий объект хранится в памяти в течение работы программного приложения.

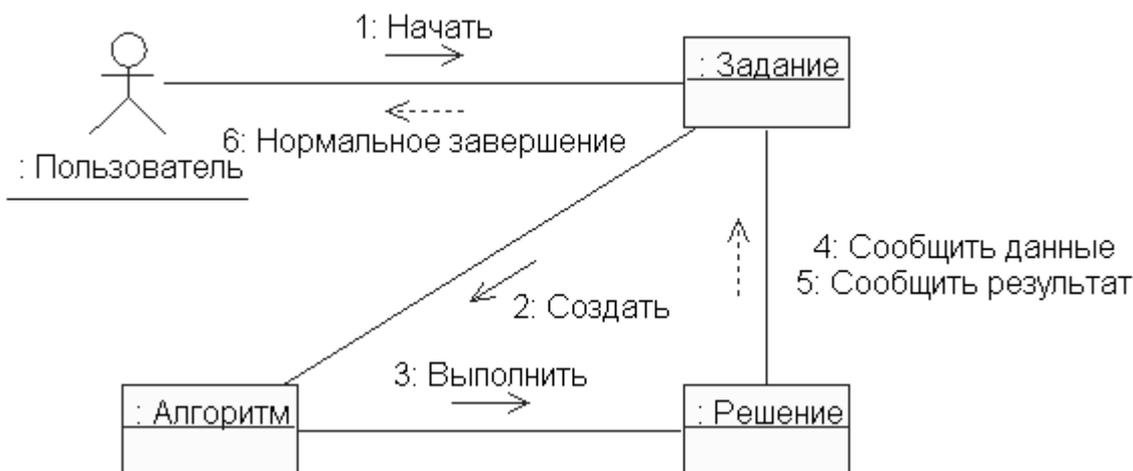
Свойство **Transient** (Временный) означает, что соответствующий объект хранится в памяти системы в течение короткого времени, необходимого только для выполнения операции.

Если в системе используется несколько экземпляров, следует установить метку **Multiple instances**.

### **Создание диаграммы сотрудничества на основе диаграммы последовательностей:**

1. При открытом окне диаграммы последовательностей выбрать элемент меню **Browse ► Create Collaboration Diagram** или нажать клавишу **<F5>**.
2. При необходимости упорядочить объекты и сообщения диаграммы сотрудничества.

Диаграмма сотрудничества для сценария “Решение задачи” показана на рисунке 4.6.



*Рисунок 4.6 – Диаграмма сотрудничества, полученная из диаграммы последовательности (рис. 4.4)*

### **4.3 Содержание отчета**

Отчет по работе должен содержать диаграммы последовательности и сотрудничества, снабженные соответствующими связями и спецификациями. При защите отчета необходимо объяснить приемы работы с программой Rational Rose, а также обосновать принятые решения.

## 7 РАЗРАБОТКА ДИАГРАММ СОСТОЯНИЙ

*Цель лабораторной работы:* освоение приемов и методики разработки диаграмм состояний с применением программы Rational Rose.

### 7.1 Рекомендации по определению состояний и переходов

Варианты использования и охватываемые ими сценарии служат инструментом описания поведения системы, то есть особенностей взаимодействия объектов в процессе ее функционирования. При создании программного обеспечения для систем управления процессами возникает необходимость в изучении поведения отдельно взятых объектов. С этой целью применяются *диаграммы состояний (statechart diagrams)*, отображающие состояния объекта, события или сообщения, инициирующие переход объекта из одного состояния в другое, а также действия, обусловленные сменой состояний.

Диаграммы состояний разрабатываются для таких классов, объекты которых отличаются повышенными динамическими характеристиками.

При анализе поведения объектов следует сосредоточить внимание на вопросах, касающихся того, "что" происходит в системе, игнорируя аспекты, связанные с тем, "каким образом" это делается.

*Состояние (state)* – это набор условий, при выполнении которых объект осуществляет определенное действие или ожидает наступления некоторого события. Состояние объекта характеризуется значениями одного или нескольких атрибутов класса. Помимо того, на состояние объекта способно воздействовать наличие связей с другими объектами. Рассматривая определенное состояние объекта, следует проверить значение признака множественности, выбранное для связи. Если возможность пребывания объекта в том или ином состоянии зависит от существования определенной связи, признак множественности для роли этой связи, "примыкающей" к соединенному объекту, должен включать значение "0" то есть отображать тот факт, что связь не является обязательной. Таким образом, для обнаружения возможных состояний объекта необходимо изучить множества его атрибутов и связей.

*Переход состояния (state transition)* отображает изменение одного состояния на другое (в частном случае состояния могут совпадать). Переход состояния сопровождается некоторым действием.

Существует два варианта активизации перехода – автоматический и условный. В первом случае переход осуществляется по завершении определенного набора операций объекта (с переходом не связано какое-либо событие). Условный переход осуществляется при возникновении некоторого именованного события, инициируемого другим объектом или внешней средой. Переходы обоих типов трактуются так, будто они выполняются мгновенно и не могут быть прерваны. На диаграмме переход отображается в виде стрелки, соединяющей два состояния.

В диаграмму состояний также включаются состояния двух специальных типов – *исходное (start state)* и *завершающее (stop state)*. Каждая диаграмма должна содержать единственное исходное состояние, поскольку после создания объекта его атрибуты всегда принимают строго определенные значения. Завершающих состояний объекта напротив, может быть несколько.

С переходом ассоциируются некоторые *действия* и/или контролируемые *условия*. Помимо того, переход способен инициировать возникновение определенного *события*. Действие сопряжено с выполнением набора функций, а событие связано с отправкой сообщения другому объекту системы. Контролируемое условие представляет собой выражение булева типа, формируемое на основе значений атрибутов объекта. Переход осуществляется только в том случае, если результатом вычисления условия служит величина "истина". Действия и контролируемые условия характеризуют поведение объекта и обычно определяются в виде операции класса. Обычно подобные операции объявляются как *закрытые (private)* – ими может воспользоваться только сам объект.

Действия, сопровождающие переход к состоянию, могут быть представлены как *входящие (entry)* операции, выполняемые объектом сразу после приобретения этого состояния. Аналогично действия, сопутствующие переходу из состояния, рассматриваются как *исходящие (exit)* операции, осуществляемые непосредственно перед изменением состояния. Пребывая в определенном состоянии, объект способен предпринимать определенные внутренние действия и/или посылать сообщения другим объектам. Подобное поведение объекта также отображается посредством операций класса. Поток *внутренних (do)* операций стартует после входящих операций и либо протекает до полного завершения, либо прерывается при активизации перехода из состояния.

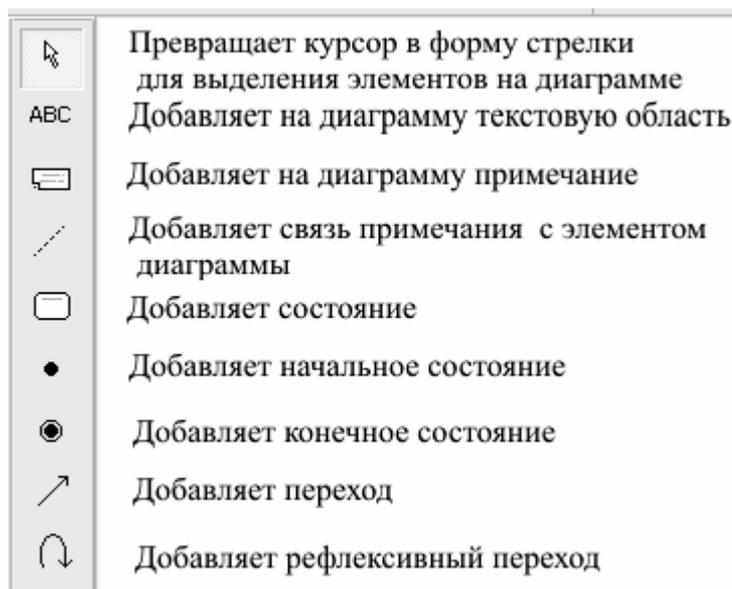
## 7.2 Методика разработки диаграмм состояний с применением программы Rational Rose

### *Активизация диаграммы состояний:*

1. Расположить курсор мыши над элементом окна **Browser**, представляющим определенный класс, и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Statechart Diagram**. Дерево, отображаемое в окне **Browser**, пополнится элементом **NewDiagram**, соответствующим новой диаграмме состояний.
3. Выбрать элемент **New Diagram** и изменить его название, введя требуемое имя диаграммы.
4. Дважды щелкнуть на элементе, чтобы открыть окно диаграммы и специальную панель инструментов.

На рисунке 7.1 показана специальная панель инструментов диаграммы

состояний и приведены назначения кнопок панели.



*Рисунок 7.1 – Назначение кнопок специальной панели инструментов диаграммы состояний*

#### **Создание состояний:**

1. Щелкнуть на пиктограмме начального состояния (**Start State**) специальной панели инструментов.
2. Включить элемент исходного состояния в диаграмму состояний, щелкнув в соответствующем месте рабочей области открытого окна диаграммы.
3. Щелкнуть на пиктограмме состояния (**State Transition**) специальной панели инструментов, затем повторить процедуру п. 2.
4. Для добавленного состояния двойным щелчком левой кнопки мыши на его изображении открыть окно спецификации и занести всю информацию по данному состоянию (рис. 7.2).

#### **Создание переходов:**

1. Щелкнуть на пиктограмме перехода (**State Transition**) специальной панели инструментов.
2. В окне диаграммы щелкнуть на символе состояние-источник, и, не отпуская кнопку мыши, построить линию перехода, направленную к символу состояния-приемника.
3. Если переход вызывается именованным событием, двойным щелчком на стрелке перехода открыть диалоговое окно **State Transition Specification**, перейти на вкладку **General**, ввести в поле **Event** строку наименования события (рис. 7.3) и закрыть окно щелчком на кнопке **OK**.

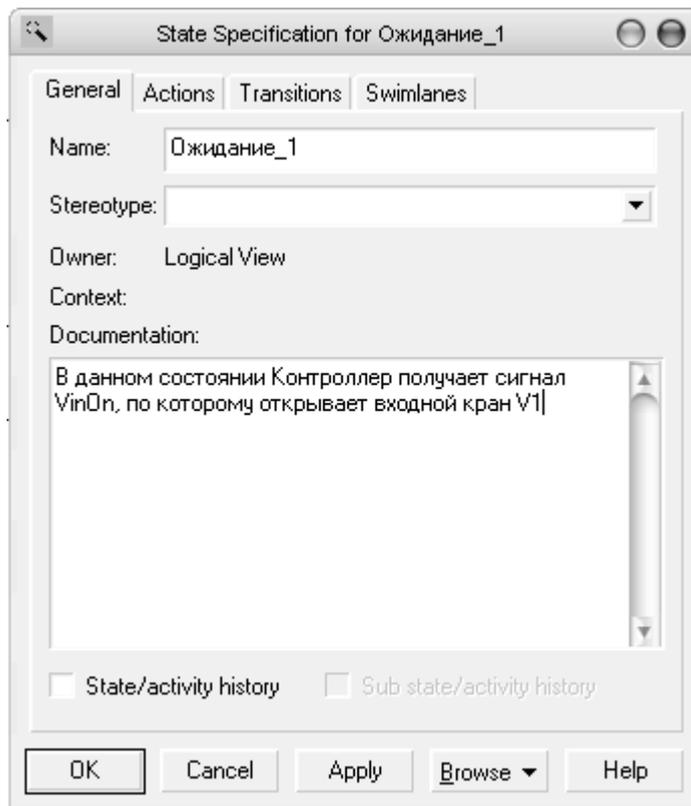


Рисунок 7.2 – Диалоговое окно спецификации свойств состояния

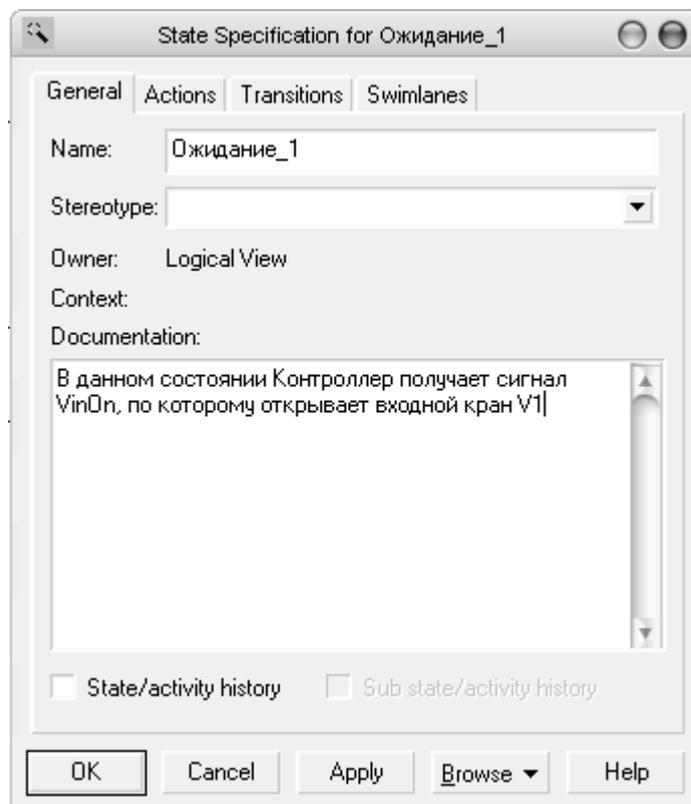


Рисунок 7.3 – Диалоговое окно спецификации свойств перехода, открытое на вкладке **General**

### ***Редактирование свойств состояний и переходов:***

При необходимости в диалоговом окне спецификации *свойств состояний* можно задать вложенное историческое состояние. Для этого следует выставить метку у свойства **State/activity history** (рис. 7.2) и нажать кнопку **Apply**. Внутренние действия на входе и выходе задаются на вкладке **Actions**. На вкладке **Transitions** (Переходы) можно определять и редактировать переходы, которые входят и выходят из рассматриваемого состояния. Последняя вкладка **Swimlanes** (Дорожки) служит для спецификации дорожек.

При редактировании свойств переходов следует обратить внимание на первые две строки вкладки **Detail** (Подробно). Первое поле **Guard Condition** служит для задания сторожевого условия, которое определяет правило срабатывания перехода. Во втором поле **Action** можно специфицировать действие, которое происходит при срабатывании перехода.

### ***Отображение данных перехода состояния:***

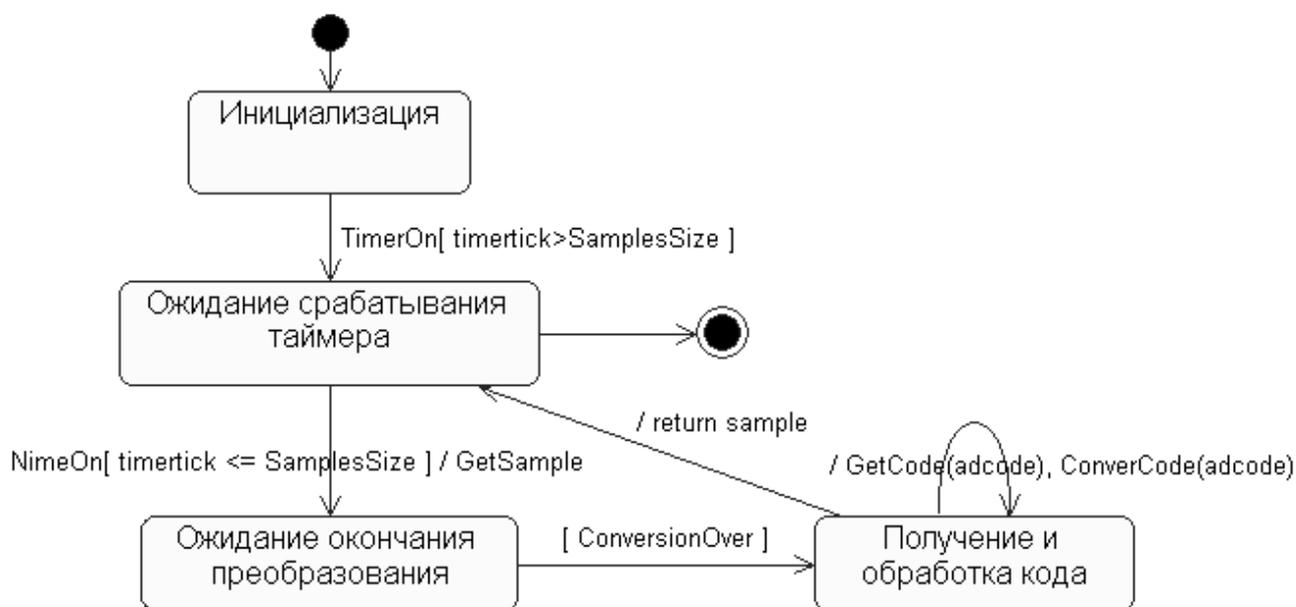
1. Расположить курсор мыши над стрелкой перехода состояния в окне диаграммы состояний и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **Open Specification**, чтобы открыть диалоговое окно **State Transition Specification**.
3. Перейти на вкладку **Detail**.
4. В поля **Guard Condition**, **Action** и/или **Send event** ввести описания контролируемого условия, действия и события, соответственно.
5. Закрыть диалоговое окно щелчком на кнопке **OK**.

### ***Определение входящих, исходящих и внутренних действий:***

1. Расположить курсор мыши над объектом состояния в окне диаграммы состояний и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **Open Specification**, чтобы открыть диалоговое окно **State Specification**.
3. Перейти на вкладку **Actions**.
4. Расположить курсор мыши над списком действий и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
5. Выбрать элемент меню **Insert** – это приведет к включению в список нового входящего (**entry**) действия.
6. Дважды щелкнуть на созданном элементе, чтобы открыть диалоговое окно **Action Specification**.
7. С помощью раскрывающегося списка **When** выбрать момент выполнения действия:
  - **On Entry** – входящее;
  - **On Exit** – исходящее;

- **Do** – внутреннее;
  - **On Event** – по наступлении события.
8. Если действие должно активизироваться по событию, в группе полей **On Event** ввести описания события, аргументов и условий.
  9. В раскрывающемся списке **Type** указать тип действия: **Action** – собственно действие или **Send Event** – отправка сообщения о событии.
  10. В поля **Name**, **Send arguments** и **Send target** ввести описания действия и информацию о событии (при необходимости).
  11. Щелчком на кнопке **OK** закрыть окно **Action Specification**.
  12. Щелчком на кнопке **OK** закрыть окно **State Specification**.

На рисунке 7.4 показан пример диаграммы состояний для моделирования динамики объекта ADCard (плата сбора данных) в процессе аналого-цифрового преобразования.



*Рисунок 7.4 – Диаграмма состояний для моделирования динамики платы сбора данных*

### 7.3 Содержание отчета

Отчет по работе должен содержать диаграммы состояний, снабженные соответствующими описаниями событий, переходов и условий. При защите отчета необходимо объяснить приемы работы с программой Rational Rose, а также обосновать принятые решения.

## 8 РАЗРАБОТКА ДИАГРАММ ДЕЙСТВИЙ (ДЕЯТЕЛЬНОСТИ)

*Цель лабораторной работы:* освоение приемов и методики разработки диаграмм действий (деятельности), а также моделирования бизнес-процессов с применением программы Rational Rose.

### 8.1 Особенности диаграмм действий

*Диаграммы действий (activities diagrams)*, отображают динамические характеристики системы, воспроизводя поток функций управления. Диаграммы действий показывают, какие ветви процесса могут выполняться параллельно, а также определяют альтернативные пути достижения целей.

Элементами диаграммы действий служат собственно *действия (activities)*, *переходы (transitions)* от одного действия к другому, *точки принятия решений (decision points)* и *полосы синхронизации (synchronization bars)*.

В UML для описания действия, перехода, точки принятия решения и полосы синхронизации применяются, соответственно, прямоугольник с округленными углами, направленная стрелка, ромб и отрезок утолщенной прямой (горизонтальный или вертикальный).

*Действие* описывает некоторый фрагмент поведения системы в контексте потока функций управления.

Элемент *перехода* применяется в диаграмме действий с целью обозначения направления передачи управления от одного действия к другому. Функция перехода обычно активизируется по завершении действия-источника.

В процессе моделирования поведения системы часто необходимо определить, в какие моменты и в каких точках поток управления претерпевает ветвление в зависимости от принимаемых системой или пользователем решений. Переход, который берет начало в *точке принятия решения*, содержит *контролируемое условие (guard condition)*, определяющее направление ветвления. Точки принятия решений совместно с соответствующими контролируемыми условиями позволяют наглядно продемонстрировать возможные альтернативные пути протекания процесса функционирования системы.

Процесс функционирования системы зачастую содержит стадии, которые могут выполняться параллельно. *Полосы синхронизации* позволяют указать, какие действия допускают одновременное выполнение или подлежат логическому объединению. Возможны ситуации, когда полоса синхронизации снабжается несколькими входящими связями и единственной исходящей и наоборот.

Диаграмма действий может быть разделена на *зоны (swimlanes)*, каждая из которых обычно связана с определенным активным субъектом, ответственным за выполнение соответствующего подмножества действий.

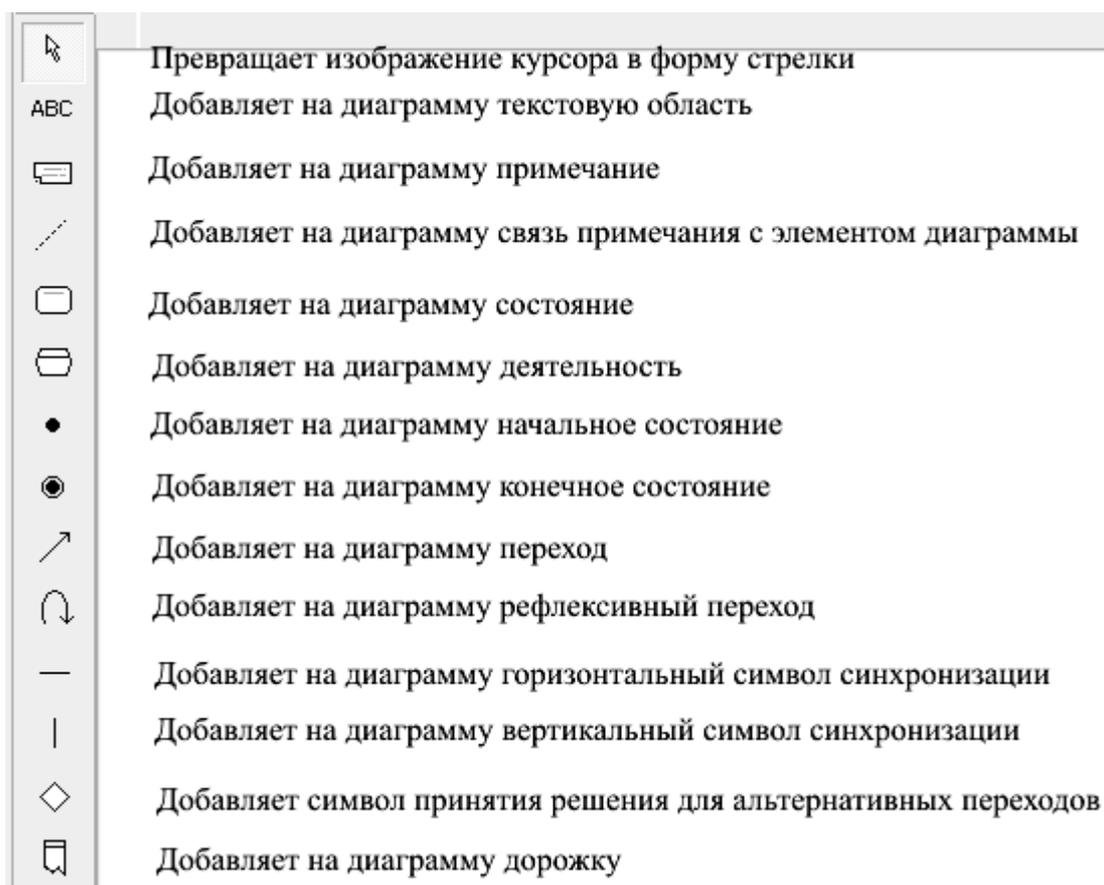
Для обозначения *исходного (start)* и *завершающего (end)* действий в диаграмме применяются специальные символы в виде круга и стилизованного

кольца. Обычно поток управления содержит одно исходное и несколько завершающих действий, по одному на каждый из возможных альтернативных путей протекания процесса функционирования системы.

## 8.2 Методика разработки диаграмм действий с применением программы Rational Rose

### *Создание диаграммы действий:*

1. Расположить курсор мыши над элементом **Use Case View** окна **Browser** и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
2. Выбрать элемент меню **New ► Activity Diagram**. Дерево, отображаемое в окне **Browser**, пополнится элементом **NewDiagram**, соответствующим новой диаграмме действий.
3. Выбрать элемент **NewDiagram** и изменить его название, введя требуемое имя диаграммы.
4. Двойным щелчком на элементе открыть окно диаграммы действий. При этом вместе с чистым листом диаграммы появляется специальная панель инструментов (рис. 8.1).



*Рисунок 8.1 – Графическое изображение и назначение кнопок специальной панели инструментов для диаграммы действий*

### **Создание действия:**

1. Щелкнуть на пиктограмме деятельности (**Activity**) специальной панели инструментов.
2. Включить элемент действия в диаграмму, щелкнув в соответствующем месте рабочей области открытого окна диаграммы действий.
3. Ввести наименование действия.

### **Создание перехода:**

1. Щелкнуть на пиктограмме перехода (**State Transition**) специальной панели инструментов.
2. В окне диаграммы щелкнуть на символе действия-источника и, не отпуская кнопку мыши, построить линию перехода, направленную к символу соответствующего действия-приемника.

### **Создание точки принятия решения:**

1. Щелкнуть на пиктограмме принятия решения (**Decision**) специальной панели инструментов.
2. Включить элемент точки принятия решения в диаграмму действий, щелкнув в соответствующем месте рабочей области открытого окна диаграммы.
3. Ввести наименование точки принятия решения.
4. Щелкнуть на пиктограмме перехода (**State Transition**) специальной панели инструментов.
5. В окне диаграммы щелкнуть на символе действия-источника и, не отпуская кнопку мыши, построить линию перехода, направленную к символу соответствующей точки принятия решения.

Пример введения точки принятия решения приведен на рисунке 8.2.

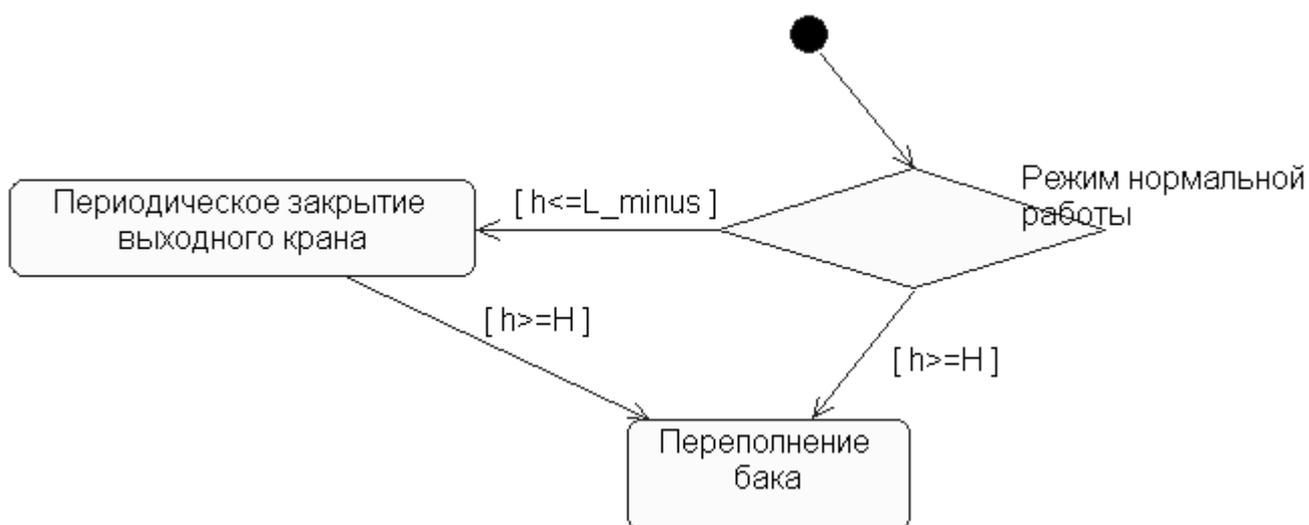


Рисунок 8.2 – Диаграмма действия системы регулирования жидкости в баке

### **Создание контролируемого перехода:**

1. Щелкнуть на пиктограмме перехода (**State Transition**) специальной панели инструментов.
  2. В окне диаграммы щелкнуть на символе точки принятия решения и, не отпуская кнопку мыши, построить линию перехода, направленную к символу соответствующего действия-приемника.
  3. Дважды щелкнуть на линии, представляющей переход, чтобы открыть диалоговое окно **State Transition Specification**.
  4. Перейти на вкладку **Detail**.
  5. В поле **Guard Condition** ввести текст контролируемого условия.
  6. Закрыть окно **State Transition Specification** щелчком на кнопке **OK**.
- Переходы с контролируемыми условиями показаны на рисунке 8.2.

### **Приведение линий диаграммы к ортогональному виду:**

1. В окне диаграммы щелчком левой кнопки мыши выбрать линию, подлежащую преобразованию (удерживая нажатой клавишу **<Shift>**, можно выбрать несколько линий одновременно).
2. Выбрать: элемент меню **Format ► Line Style ► Rectilinear**.
3. При необходимости щелкнуть на линии и, не отпуская кнопку мыши, переместить линию в нужном направлении.

На рисунке 8.3 показан вариант с ортогональными линиями диаграммы действий, приведенной на рисунке 8.2.

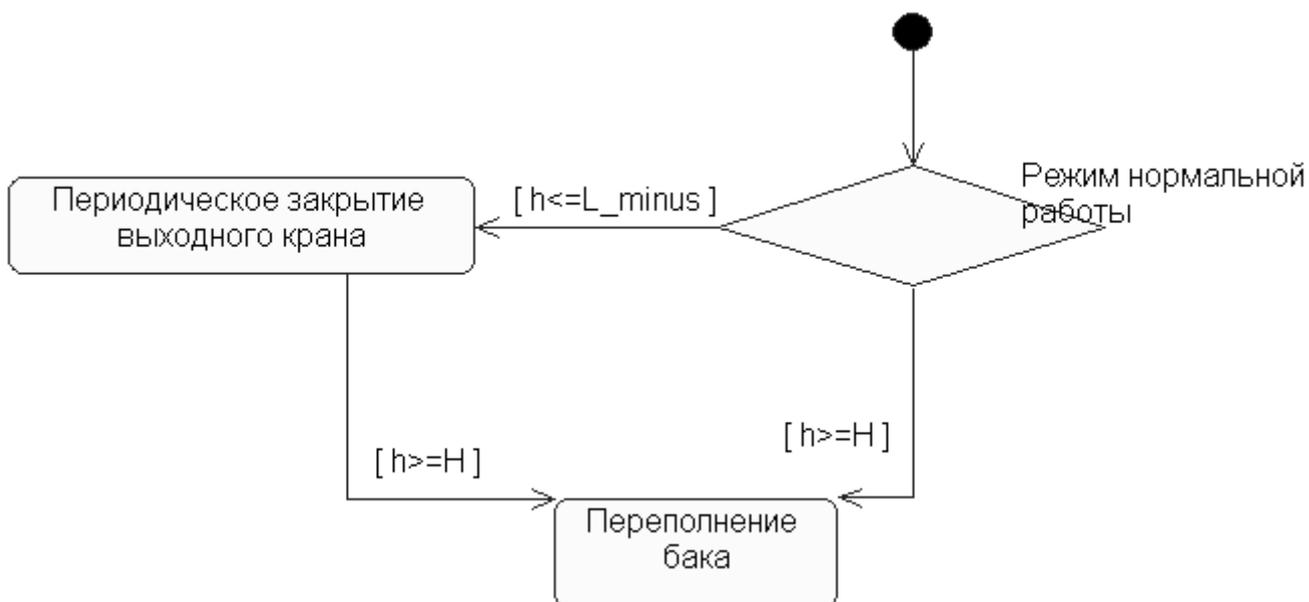


Рисунок 8.3 – Диаграмма действий (рис. 8.2) с ортогональными линиями

### **Создание полосы синхронизации:**

1. Щелкнуть на пиктограмме **Horizontal Synchronization** или **Vertical Synchronization** специальной панели инструментов.
2. Включить элемент полосы синхронизации в диаграмму действий, щелкнув в соответствующем месте рабочей области открытого окна диаграммы.
3. Щелкнуть на пиктограмме **State Transition** специальной панели инструментов и снабдить полосу синхронизации необходимой входящей (исходящей) связью.
4. Повторить операцию для создания остальных связей-переходов.

Пример создания полос синхронизации приведен на рисунке 8.4.

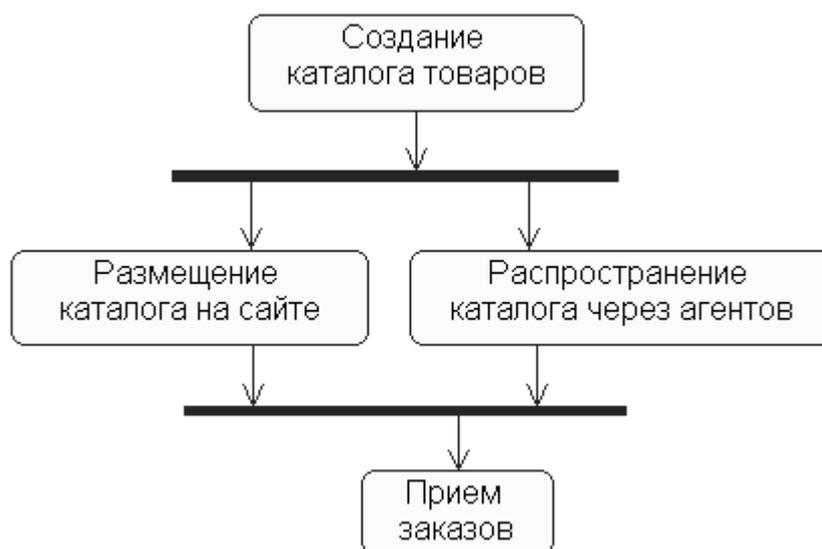


Рисунок 8.4 – Создание полос синхронизации на диаграмме действий

#### **Деление диаграммы действий на зоны:**

1. Щелкнуть на пиктограмме дорожки (**Swimlane**) специальной панели инструментов.
2. Включить дорожку в диаграмму действий, щелкнув в пределах рабочей области открытого окна диаграммы. Диаграмма пополнится новой зоной с именем **NewSwimlane**.
3. Двойным щелчком на текстовой метке **NewSwimlane** открыть диалоговое окно **Swimlane Specification**, в поле **Name** ввести требуемое имя зоны и щелкнуть на кнопке **OK**.
4. Для изменения размера зоны щелкнуть на границе зоны и, не отпуская кнопку мыши, переместить границу в нужном направлении.
5. Применяя технику перетаскивания, расположить в пределах зоны все нужные элементы диаграммы (или создать новые).

Пример диаграммы действий с двумя зонами показан на рисунке 8.5.

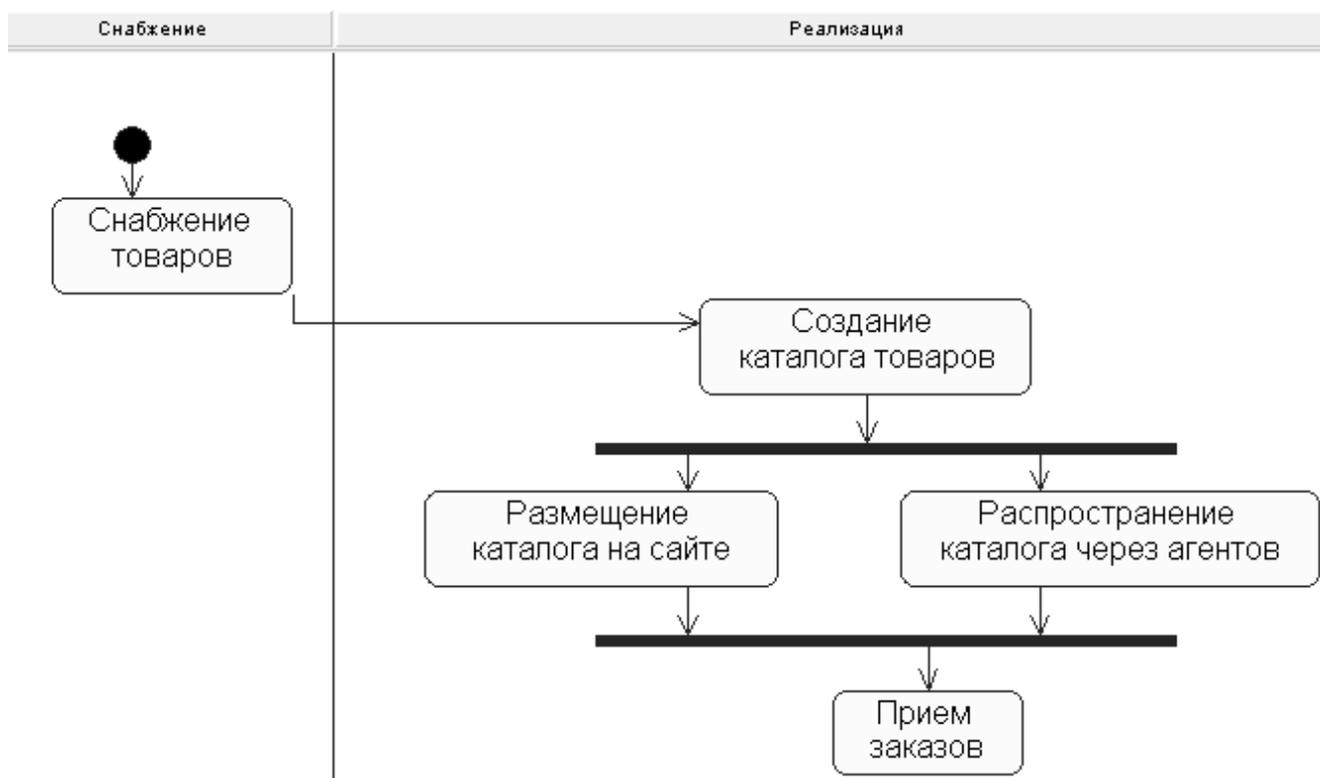


Рисунок 8.5 – Диаграмма действий с дорожками **Снабжение** и **Реализация**

### 8.3 Содержание отчета

Отчет по работе должен содержать диаграммы действий с соответствующим документированием. При защите отчета необходимо объяснить приемы работы с программой Rational Rose при создании диаграмм действий, а также обосновать принятые решения.

# **МЕТОДИЧНІ ВКАЗІВКИ**

до комп'ютерного практикуму

з дисципліни

## **«ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ СКЛАДНИХ СИСТЕМ»**

для студентів спеціальності 7.092501 «Автоматизоване управління технологічними процесами»

(Російською мовою)

Укладач    Олександр Олександрович Сердюк

Редактор

Комп'ютерна верстка

Підп. до друку                      .    Формат 60 x 84/16.  
Папір офсетний. Ум. друк. арк. 3,00. Обл.-вид. арк. 2,18.  
Тираж                      прим.    Зам. №

Видавець і виготівник  
«Донбаська державна машинобудівна академія»  
84313, м. Краматорськ, вул. Шкадінова, 72.  
Свідоцтво про внесення до Державного реєстру  
серія ДК №1633 від 24.12.03.