Министерство образования и науки Украины

Донбасская государственная машиностроительная академия

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к курсовому проектированию

по дисциплине

«ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ СЛОЖНЫХ СИСТЕМ»

(для студентов специальности 151)

Утверждено
на заседании
методического совета
Протокол № от

Краматорск 2018

УДК 681.3.06 (075.8)

Методические указания к курсовому проектированию по дисциплине «Технология программирования сложных систем» (для студентов специальности 151 / Сост. А. А. Сердюк. – Краматорск: ДГМА, 2018. – 48 с.

Рассматриваются структура и состав курсового проекта. Приводятся методика моделирования средствами UML и рекомендации по разработке программной системы с применением CASE-средства Rational Rose. Демонстрируется пример полного цикла разработки программной системы – от начальной стадии анализа до этапа генерации программного кода.

Составитель Александр Александрович Сердюк, доц.

Отв. за выпуск Степан Павлович Сус, доц.

Редактор

Комп’ютерна верстка

Підп. до друку . Формат 60 х 84/16.
Папір офсетний. Ум. друк. арк. 4,75. Обл.-вид. арк. 3,46.
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.

СОДЕРЖАНИЕ

1 ОБЩИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

Курсовой проект по дисциплине «Технология программирования сложных систем» выполняется с целью приобретения умения системного восприятия и решения задачи разработки сложной программы, освоения методики объектно-ориентированного моделирования программных систем на языке **UML** ***(Unified Modeling Language)***, а также приобретения навыков работы с современным **CASE**-средством ***IBM Rational Rose 2003***.

Предметом проектирования является программная система, предназначенная для автоматизации процессов в определенной предметной области.

В технологии создания программного продукта используются методы, средства и процедуры.

Методы обеспечивают анализ системных и программных требований, проектирование структуры программы и структуры данных, генерацию программного кода и тестирование готовой программы. При разработке проекта следует руководствоваться объектно-ориентированной методологией, стандартным языком которой является ***UML***.

Средства (утилиты) технологии программирования – это инструменты, с помощью которых создается программа. При разработке проекта целесообразно применять ***CASE-технологию (Computer Aided Software Engineering)***, а из известных инструментов этой технологии – ***CASE-средство Rational Rose***, обеспечивающее инструментальную поддержку ***UML***.

Процедуры являются “клеем”, который соединяет методы и средства так, что весь процесс проектирования превращается в последовательность определенных операций, которые обеспечивают порядок моделирования и документирования, а также контроль качества программы.

Процесс программирования подчиняется основной парадигме технологии разработки программ – классическому жизненному циклу. Классический жизненный цикл составляет базовую платформу ***RUP***. Он представляет собой каскадную модель разработки программы, включающей в себя следующие этапы:

1. ***Системный анализ***, обеспечивающий формирование общего представления об интерфейсе программы с людьми, аппаратурой и базами данных.
2. ***Анализ требований*** к программной системе, позволяющий детализировать функции системы, её характеристики и интерфейс.
3. ***Проектирование*** на уровне моделей структуры и поведения системы.

4. ***Кодирование*** результатов моделирования, то есть создание текста программы на языке программирования.

Из существующих стратегий разработки программных систем (водопадная, инкрементная и эволюционная) для курсового проектирования целесообразно принять водопадную стратегию – ***однократный проход*** всех этапов создания программы. Несмотря на то, что такая стратегия обычно реализует только часть возможностей системы, для первого самостоятельного шага в создании сложной программной системы этого достаточно. Следует учесть, что дальнейшее усовершенствование программы возможно в курсовом проектировании по дисциплине "Цифровые системы управления и обработки информации", а также в процессе выполнения дипломного проекта.

Содержание курсового проекта должно раскрывать сущность этапов создания программы для заданной предметной области. Каждый этап работы над проектом должен быть описан по схеме:

- цели и задачи этапа;
- обоснование действий, выполняемых на данном этапе;
- реализация действий и результат этапа.

Основная часть расчетно-пояснительной записки должна раскрывать тему в полном объеме и состоять из следующих разделов:

- ✓ Введение.
- ✓ Описание и анализ предметной области.
- ✓ Моделирование функционального назначения системы.
- ✓ Разработка диаграмм взаимодействия объектов.
- ✓ Объектно-ориентированное моделирование структуры и поведения системы.
- ✓ Моделирование реализации и развертывания программной системы (при необходимости).
- ✓ Генерация программного кода.
- ✓ Выводы.

Приблизительная тематика курсовых проектов приведена в приложении А, образец титульного листа пояснительной записки представлен в приложении Б.

Результаты проектирования должны быть представлены в пояснительной записке на листах формата А4 и приложениях, включающих диаграммы UML и текст программы.

Пояснительная записка должна быть оформлена в соответствии с ГОСТ 2.105-85 «Общие требования к текстовым документам», а также ДСТУ 3008-95 «Документация. Отчеты в сфере науки и техники».

Защита проекта осуществляется перед комиссией из преподавателей кафедры в сроки, установленные графиком заседаний комиссии. Доклад по проекту должен быть построен так, чтобы были освещены, главным образом,

цели, задачи и мотивы принятия решений, а не само содержание диаграмм. Доклад должен быть иллюстрирован либо плакатами, либо слайдами.

Результаты защиты проектов оцениваются по следующим критериям.

- Оценкой ***A "отлично"*** оцениваются курсовые проекты, выполненные в соответствии с заданием, самостоятельно, имеющие оригинальные технические решения. При оценке проекта важную роль играют четкие ответы на поставленные вопросы. Повышает ценность курсового проекта его практическое использование на производстве или в учебном процессе.
- Оценкой ***B "хорошо"*** оцениваются курсовые проекты, имеющие частные недостатки или некоторые пробелы в проработке отдельных вопросов.
- Оценкой ***C*** "***хорошо***" оцениваются курсовые проекты, в которых выбрана ошибочная концепция в системном восприятии модели, однако методика моделирования применена в целом правильно.
- Оценкой ***D "удовлетворительно"*** оцениваются курсовые проекты, имеющие недостаточную проработку проекта, ошибки в методике моделирования и недостаточно аргументированные ответы на вопросы.
- Оценкой ***E "удовлетворительно"*** оцениваются курсовые проекты, имеющие существенные недостатки проекта, слабую проработку темы, неуверенные ответы на вопросы.

2 ОПИСАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

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.

Рекомендации по выполнению курсового проекта будут сопровождаться примером известной (в общих чертах) предметной области – автоматическим терминалом банка (банкоматом).

Тема примера: ***Разработать модель и каркас программного кода 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.

Таблица 2.1

Действия актеров	Отклик системы
1. Клиент вводит кредитную карточку в устройство чтения банкомата. ***Исключение 1***. Кредитная карточка недействительна.	1. Банкомат проверяет кредитную карточку. 2. Банкомат предлагает ввести ПИН-код.
2. Клиент вводит ПИН-код. ***Исключение 2***. 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. ***Исключение 3***. 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.

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

3 МОДЕЛИРОВАНИЕ ФУНКЦИОНАЛЬНОГО НАЗНАЧЕНИЯ СИСТЕМЫ

Рекомендации по разработке диаграмм вариантов использования.

Диаграмма вариантов использования отражает формализацию функциональных требований к поведению проектируемой программной системы. Каждый вариант использования должен представлять собой завершенную транзакцию между пользователем и системой.

Семантика построения диаграммы вариантов использования должна удовлетворять следующим правилам:

1. Экземпляр отдельного варианта использования инициализируется (запускается) посредством сообщения от экземпляра актера. Ответной реакцией варианта использования является последовательность действий, установленная для данного варианта использования, которая будет продолжаться до тех пор, пока соответствующий экземпляр актера не получит требуемый экземпляр сервиса.

2. Варианты использования могут быть специфицированы в виде текста, включающего предусловия и постусловия. Взаимодействие между вариантами использования и актерами может уточняться на диаграммах взаимодействия.

3. С системно-аналитической точки зрения диаграммы вариантов использования должны не только специфицировать функциональные требования к поведению проектируемой системы, но и выполнять исходную *структуризацию предметной области*. Для решения этой задачи необходимо, чтобы в диаграмме вариантов использования были выделены главные функции системы.

Последовательность разработки диаграмм вариантов использования в среде Rational Rose.

После запуска программы Rational Rose сначала необходимо изменить имя проекта, предложенного программой по умолчанию, и сохранить модель во внешнем файле на диске, например, под именем **ATMmodel.mdl**.

Далее следует активизировать диаграмму Use Case в окне диаграмм и ввести её имя, например: “Диаграмма вариантов использования”.

После активизации окна диаграммы появляется специальная панель инструментов, на которой присутствует только часть кнопок с пиктограммами элементов, которые могут быть использованы для разработки диаграмм. Эту панель следует дополнить кнопками с пиктограммами других необходимых элементов.

Так, например, для системы банкомата могут быть использованы элементы бизнес-деятельности: **Business Actor** (бизнес-актер), **Business Use**

Case (бизнес-вариант использования) и **Service** (сервис). На этом этапе окно броузера и панель инструментов приобретают вид, показанный на рисунке 3.1.

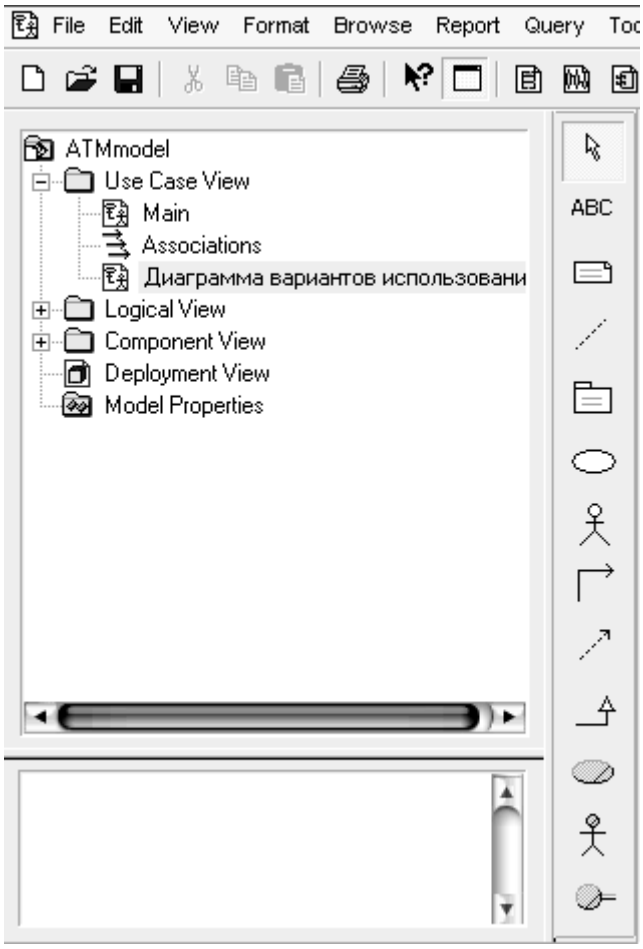

Рисунок 3.1 Вид окна броузера и панели инструментов

Далее в левую часть рабочего листа добавляется изображение актера. Предложенное программой имя актера следует изменить. 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** можно 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 **ОК** (рис. 3.2).

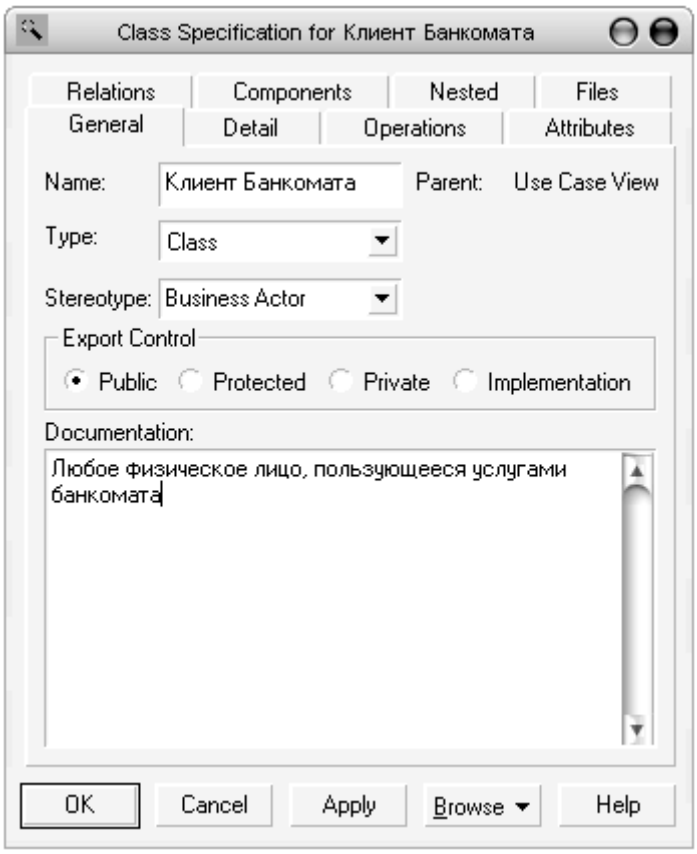

Рисунок 3.2 – Диалоговое окно спецификации свойств после изменения стереотипа и добавления текста документации для актера ***Клиент Банкомата***

Следующий шаг – добавление на рабочий лист элементов Use Case. Предложенное программой имя каждого варианта использования следует изменить. Так, например, 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 3.3.

Разместив на рабочем листе актера и все элементы Use Case, следует *добавить связи между ними*. При этом между актером и вариантом использования добавляется направленная ассоциация, а между вариантами использования – отношения зависимости. Так 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.

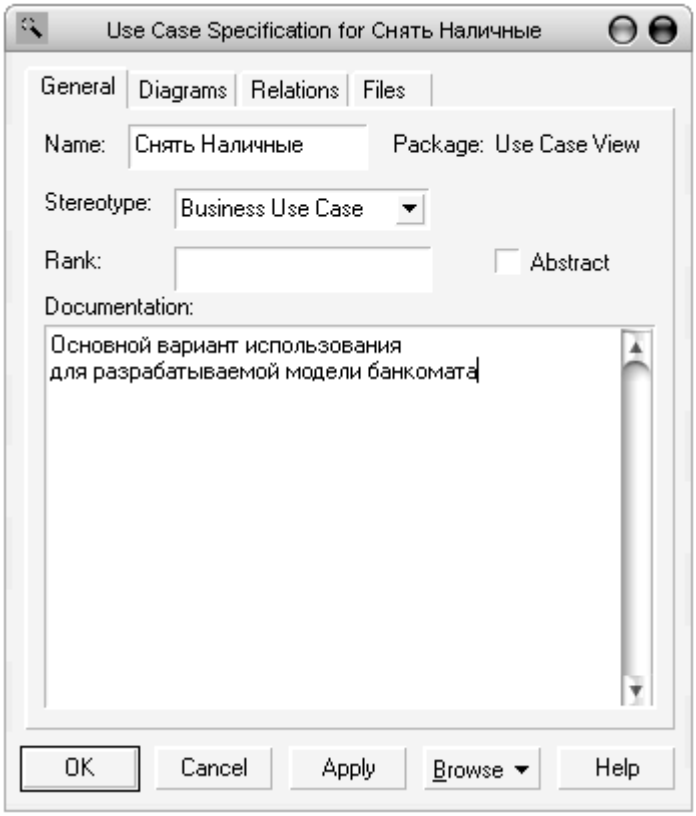

Рисунок 3.3 – Диалоговое окно спецификации свойств варианта использования ***Снять Наличные***

Для задания текстового стереотипа отношения зависимости <<**include**>> следует вызвать окно спецификации свойств этого отношения.

Для моделирования исключений на диаграмму добавляются другие элементы Use Case с использованием отношения зависимости со стереотипом <<**extend**>>. Так, например, 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.

При необходимости диаграмма может быть детализирована введением, например, таких прецедентов, как **Удалить Карточку**, **Печатать Чек** и др. Диаграмма должна иметь описания прецедентов.

Следует учесть, что если внутри элемента Use Case находятся основные и альтернативные потоки событий, то разработчик может выделить из этого элемента самостоятельный элемент Use Case и подключить его к базовому.

Окончательный вид диаграммы вариантов использования с добавлением необходимых элементов приведен на рисунке 3.4.

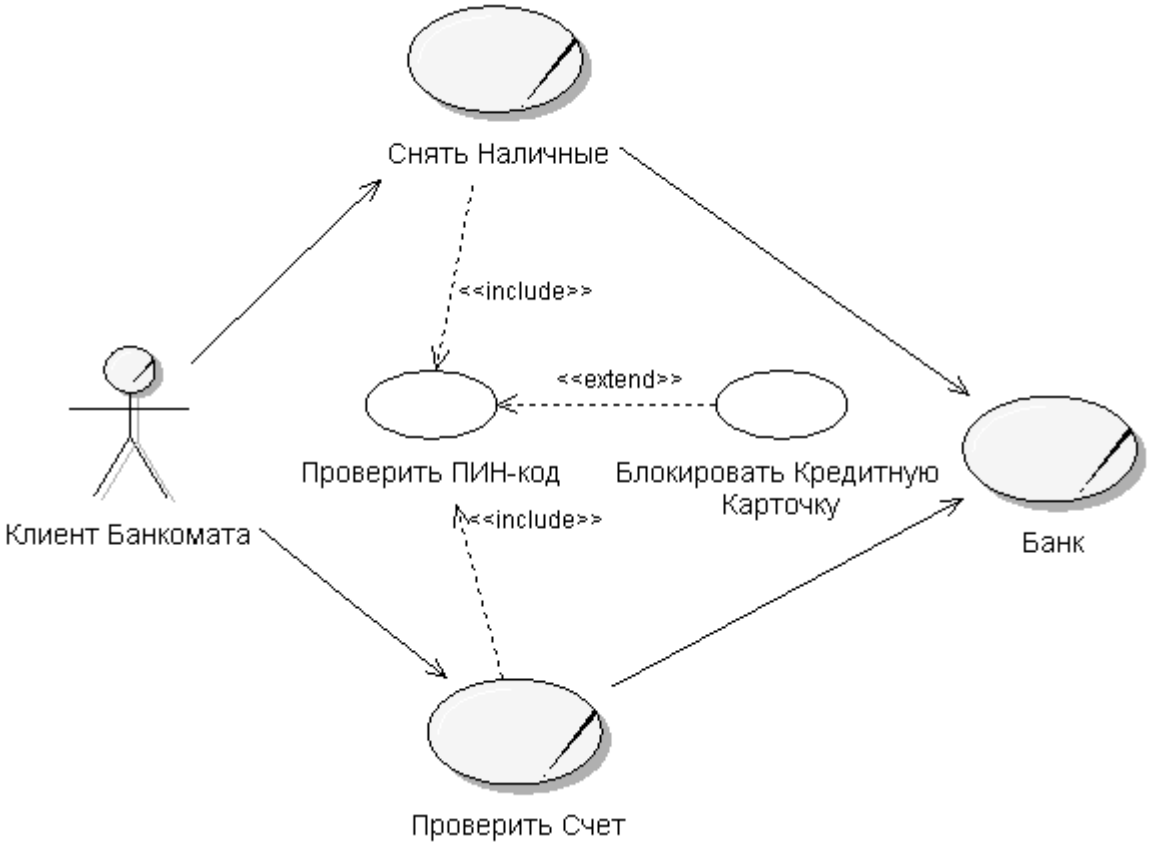

Рисунок 3.4 – Окончательный вид диаграммы Use Case для модели банкомата

4 РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ ДИАГРАММ ВЗАИМОДЕЙСТВИЯ

Диаграмма взаимодействия (**interaction diagram**) отображает один из процессов обработки информации в варианте использования. Если для одного варианта использования существует несколько альтернативных потоков, то для каждого потока нужно создать свою диаграмму взаимодействия.

Действие – это исполняемое предложение, которое образует абстракцию вычислительной процедуры. UML позволяет моделировать действия следующих видов:

- ***call*** (вызвать) – вызывает операцию, применяемую к объекту, причем объект может послать сообщение самому себе;
- ***return*** (возвратить) – возвращает значение вызывающему объекту;
- ***send*** (послать) – посылает объекту сигнал, явно описанный в классе, объект которого инициирует передачу сигнала;
- ***create*** (создать) – создает новый объект;
- ***event*** (событие) – выполнение операции по наступлении события;

- ***destroy*** (уничтожить) – удаляет объект.

Для моделирования взаимодействия применяются диаграмма последовательностей (**sequence diagrams**) и диаграмма кооперации (**collaboration diagrams**). Так как программа Rational Rose позволяет построить диаграмму кооперации по диаграмме последовательностей автоматически, то следует начать процесс моделирования взаимодействий с разработки диаграмм последовательностей.

Для разработки диаграммы последовательностей необходимо активизировать рабочее окно и внести имя диаграммы. В примере моделирования системы банкомата будет рассматриваться процесс построения диаграммы последовательностей для варианта использования **Снять Наличные**. Это имя вводится как имя диаграммы.

Для реализации варианта использования **Снять Наличные** задействованы следующие объекты (наименования объектов приведены без пробелов в соответствии с требованиями объектно-ориентированного программирования):

1. Устройство чтения карточки – **УстрЧтКарт**.
2. Контроллер банкомата – **Контроллер**.
3. Клавиатура банкомата – **Клавиатура**.
4. Экранная форма банкомата – **ЭкранФорма**.
5. Принтер банкомата – **Принтер**.
6. Устройство выдачи наличных – **УстрВыдНал**.
7. Транзакция банкомата – **Транзакция**.
8. Интерфейс сервера банка – **IБанк**.

Поведение действующих лиц и системы банкомата для варианта использования **Снять Наличные** (без печати чека) может быть представлено в следующем порядке :

1. Для начала диалога с объектом класса-актера **Клиент** объект класса **Контроллер** активизирует объект класса **ЭкранФорма** операцией “показатьГлавФорму”.
2. Класс-актер **Клиент** запускает объект класса **УстрЧтКарт** операцией ввода карты.
3. Объект класса **УстрЧтКарт** идентифицирует карту.
4. Объект класса Контроллер выполняет считывание идентификатора в объекте класса **УстрЧтКарт**.
5. Объект класса **Контроллер** активизирует объект класса **ЭкранФорма** сообщением “вывестиНовуюФорму”.
6. Класс-актер **Клиент** активизирует объект класса **Клавиатура** сообщением “ввестиПИН-код”.
7. Объект класса **Клавиатура** обрабатывает процедуру ввода и формирует строку ПИН-кода.
8. Объект класса **Контроллер** создает объект класса **Транзакция**.

9. Объект класса **Контроллер** запускает объект класса **Транзакция** сообщением “проверитьПИН-код”.

10. Объект класса **Транзакция** осуществляет запросы, вводы и выводы данных в процедуре “обработатьСобытие”.

11. Объект класса **Контроллер** активизирует объект класса **ЭкранФорма** сообщением “показатьМеню”.

12. Класс-актер **Клиент** активизирует объект класса **Клавиатура** выбором типа варианта использования **“Снять наличные”**.

13. Объект класса **Контроллер** активизирует объект класса **ЭкранФорма** сообщением “вывестиМенюСуммы”.

14. Класс-актер **Клиент** передает сообщение “ввестиСумму” объекту класса **Клавиатура**.

15. Объект класса **Клавиатура** обрабатывает сообщение о сумме наличных, запрашиваемой класс-актером **Клиент**.

16. Объект класса **Контроллер** запускает объект класса **Транзакция** сообщением “передатьСумму”.

17. Объект класса **Контроллер** запускает объект класса **Транзакция** процедурой “проверитьБаланс”.

18. Объект класса **Контроллер** вызывает процедуру “открытьСчет” с использованием объекта класса **IБанк**.

19. Объект класса **Контроллер** вызывает процедуру “уменьшитьБаланс” с использованием объекта класса **IБанк**.

20. Объект класса **Контроллер** активизирует объект класса **УстрЧтКарт** сообщением “выдатьКарту”.

21. Объект класса **Контроллер** активизирует объект класса **ЭкранФорма** для вывода текстового сообщения клиенту.

22. Объект класса **Контроллер** активизирует объект класса **УстрВыдНал** сообщением “выдатьНаличные” при условии, что новое значение баланса имеет положительную величину.

23. Объект класса **УстрВыдНал** рекурсивной операцией “определитьКолБанкнот” формирует требуемую сумму.

24. Объект класса **Контроллер** передает сообщение “завершитьТранзакцию” объекту класса **Транзакция**.

На этом сеанс обслуживания клиента по варианту **«Снять наличные»** заканчивается.

Построение диаграммы последовательностей в среде Rational Rose сводится к добавлению и редактированию свойств отдельных объектов и сообщений.

При добавлении сообщений должна быть включена нумерация сообщений в окне спецификации свойств модели. Это окно можно вызвать операцией главного меню **Tools ► Options.** В окне нужно открыть вкладку **Diagram** и выставить отметку выбора строки **Sequence numbering** (нумерация сообщений на диаграмме последовательностей), как показано на рисунке 4.1.

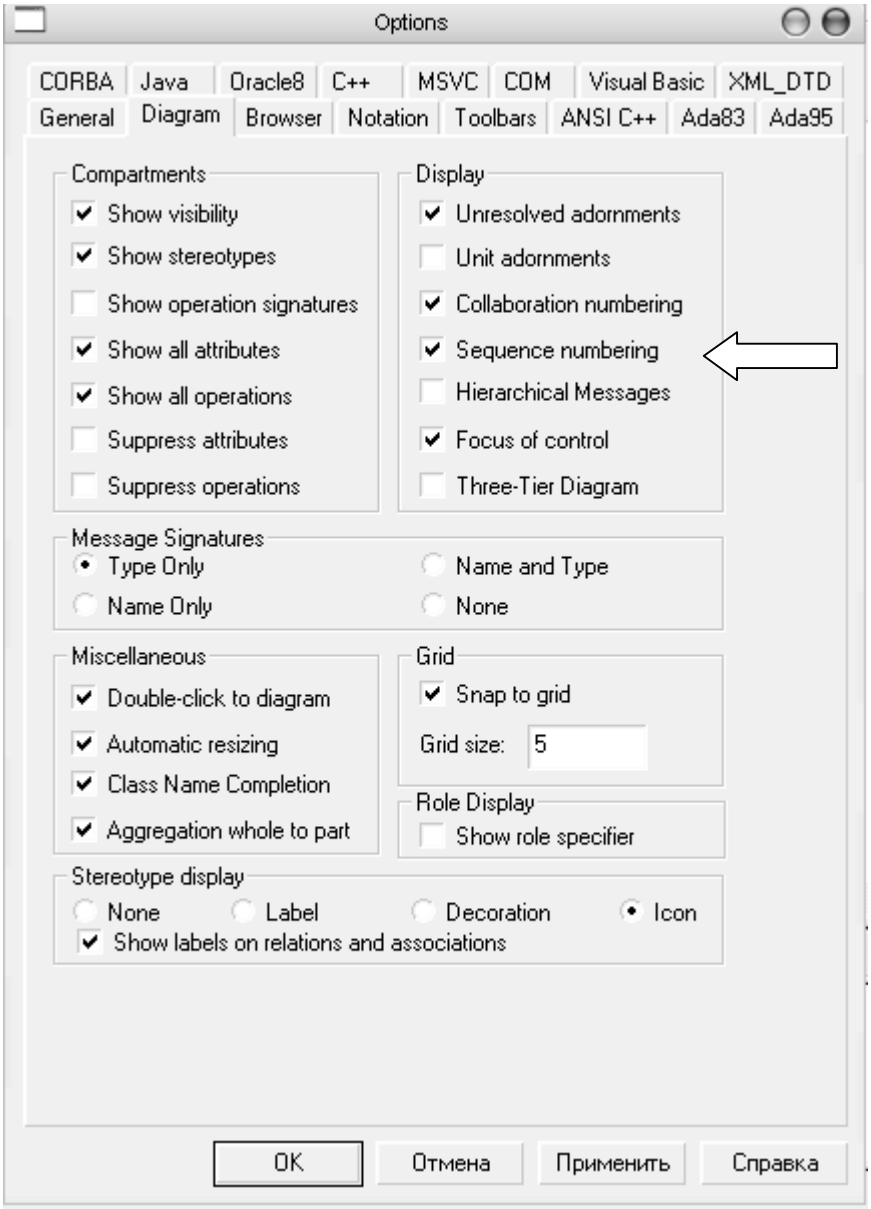

Рисунок 4.1 – Установка нумерации сообщений в диалоговом окне спецификации свойств модели

Фрагмент диаграммы последовательности для варианта использования **Снять Наличные** системы управления банкоматом приведен на рисунке 4.2.

Вид окна броузера проекта для диаграммы последовательностей варианта использования **Снять Наличные** показан на рисунке 4.3.

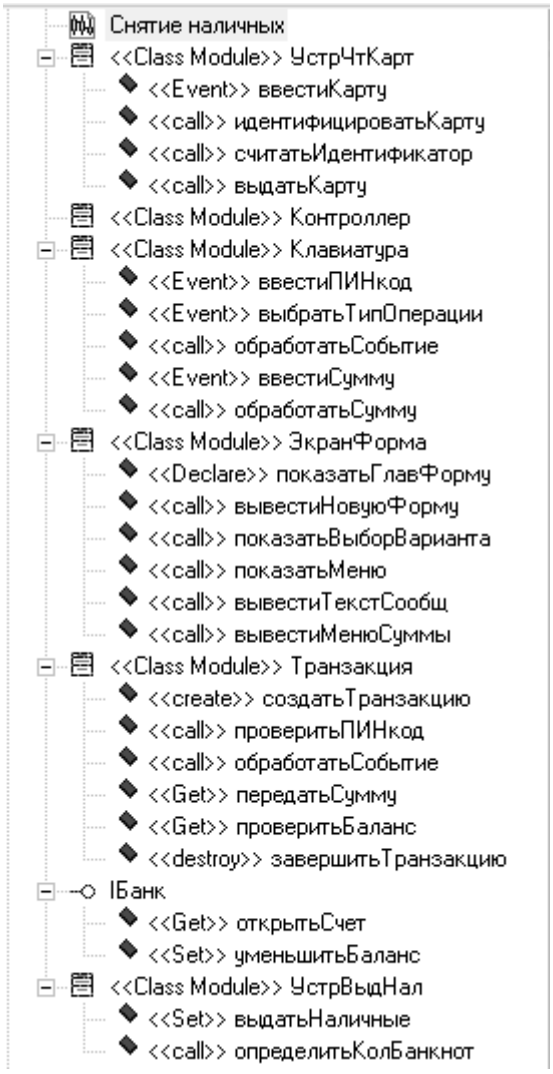

Рисунок 4.3 – Вид броузера проекта диаграммы последовательностей

Для создания диаграммы кооперации достаточно нажать функцию <**F5**> или выбрать в главном меню **Browser ► Create Collaboration Diagram**. На автоматически созданной диаграмме необходимо путем перемещением элементов упорядочить расположение объектов и сообщений.

Диаграмма кооперации для варианта использования **Снять наличные** (рис. 2.6) показана на рисунке 4.4.

5 РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ ДИАГРАММ КЛАССОВ

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

Рабочее окно диаграммы классов появляется по умолчанию при запуске программы Rational Rose. Предложенное программой имя диаграммы **Main** следует изменить. Для этого указатель мыши подводится к наименованию **Main**, щелчком правой кнопки вызывается контекстное меню и в открывшемся окне выбирается **Rename** (Переименовать). Для разрабатываемого примера внесено имя **Диаграмма классов ATM.**

Выбор стереотипов и редактирование свойств классов.

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***. При этом следует сразу определиться с языком программирования. В качестве языка программирования для данного примера принят язык **Visual Basic 6.**

Сначала следует изменить имена классов. В результате получаем следующий набор классов для рассматриваемого примера:

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

При добавлении классов вместе с именем класса следует выбрать их стереотипы на вкладке **General** диалогового окна спецификации свойств класса. Для большинства классов применяется стереотип << **entity** >> (сущность), для управляющих классов – стереотип << **control** >>, для граничных классов – <<**boundary**>>, для интерфейсов – <<**Interface**>>.

Для добавления класса используется иерархическое логическое представление структуры проекта **Logical View.** После добавления класса на диаграмму в окне спецификации класса в поле **Documentation** следует ввести описание класса и выбрать стереотип класса (рис. 5.1).

При выборе стереотипа (атрибута класса) для рассматриваемого примера учитывалось следующее:

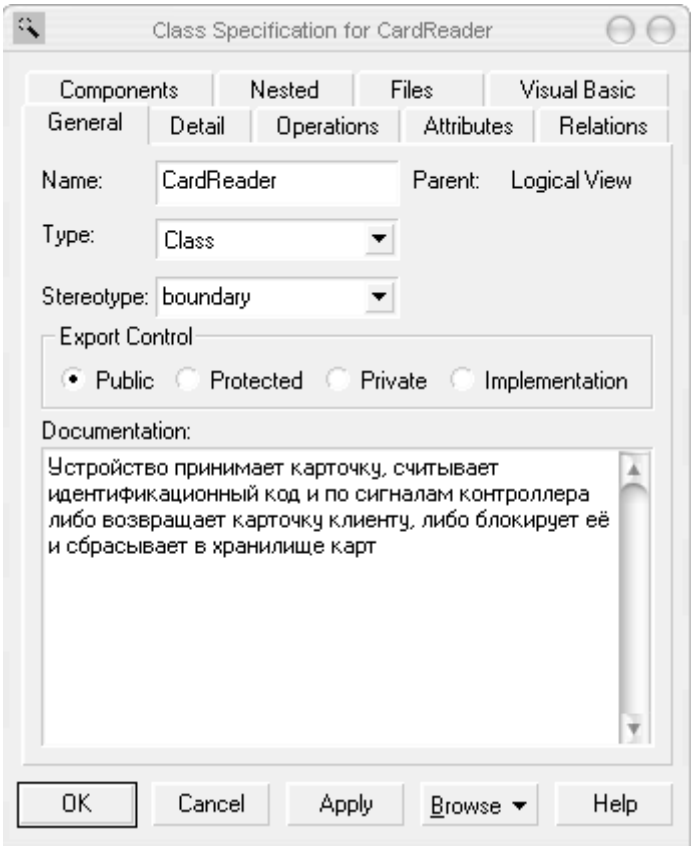

*Рисунок 5.1 – Вид окна спецификации свойств класса **CardReader***

- 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 вкладке **Detail** окна спецификации класса с помощью вложенного списка **Multiplicity** можно задать количество экземпляров класса. Так, например, для разрабатываемой модели 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 разрабатываемой модели 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 разрабатываемой в примере модели

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.

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

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 (Accound);
- сумма снимаемых со счета наличных (Sum);

- остаток на счете (Balance).
- перечень транзакций (mCol);

Редактирование атрибутов осуществляется в окне спецификации **Class Attribute Specification.** Так, например, для атрибута **IDCard** в качестве типа допустимых значений из вложенного списка **Type** следует выбрать тип **Long** (длинное целое). Для задания квантора видимости в группе **Export Control** (Управление экспортом) следует выбрать квантор **public**, так как этот параметр должен изменяться 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 **ОК**, чтобы сохранить результаты редактирования этих свойств атрибута.

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

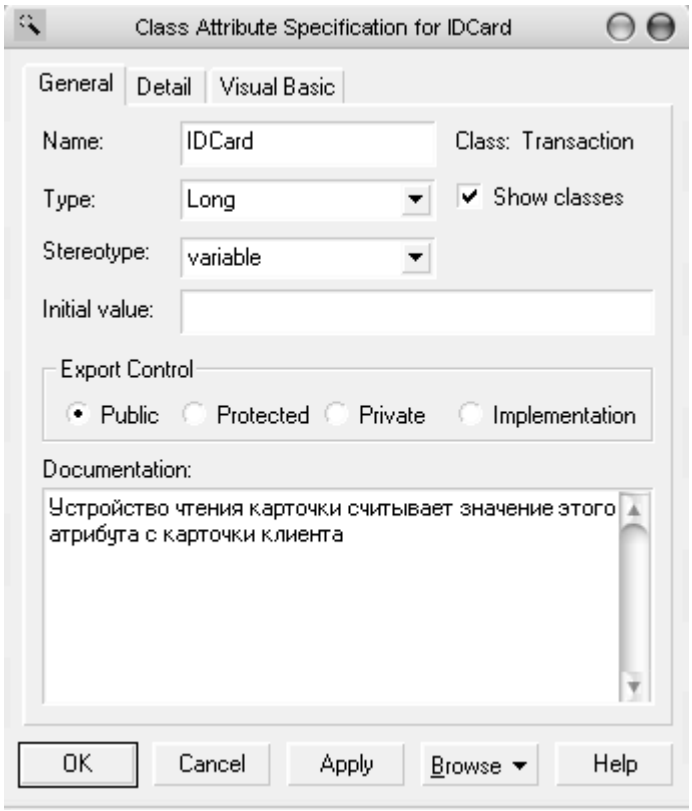

*Рисунок 5.2 – Диалоговое окно спецификации после редактирования свойств атрибута **IDCard***

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.

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 броузере проекта.

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 **ОК**, чтобы сохранить результаты редактирования свойств этой операции.

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

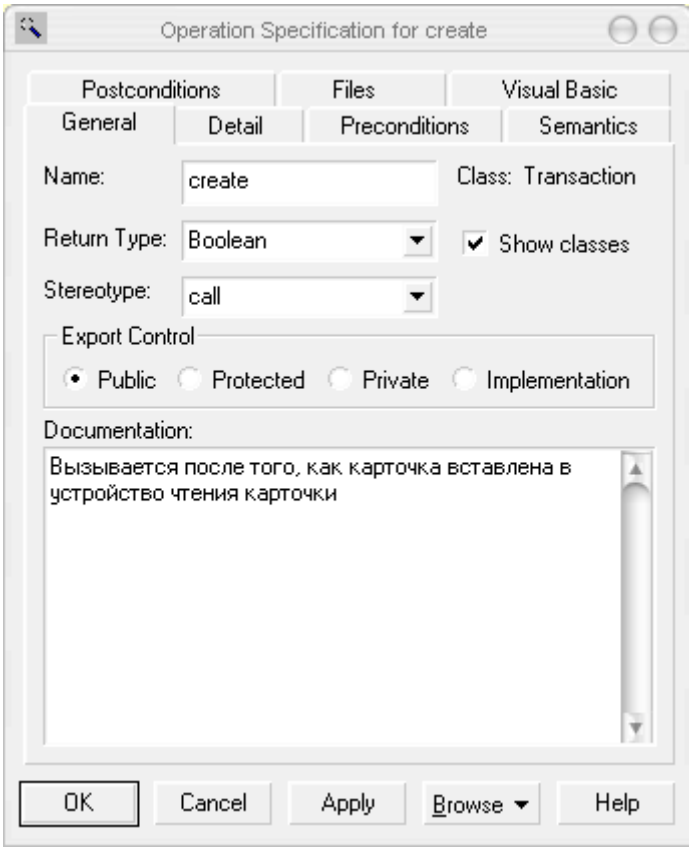

*Рисунок 5.3 – Диалоговое окно спецификации свойств операции **create***

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. 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.

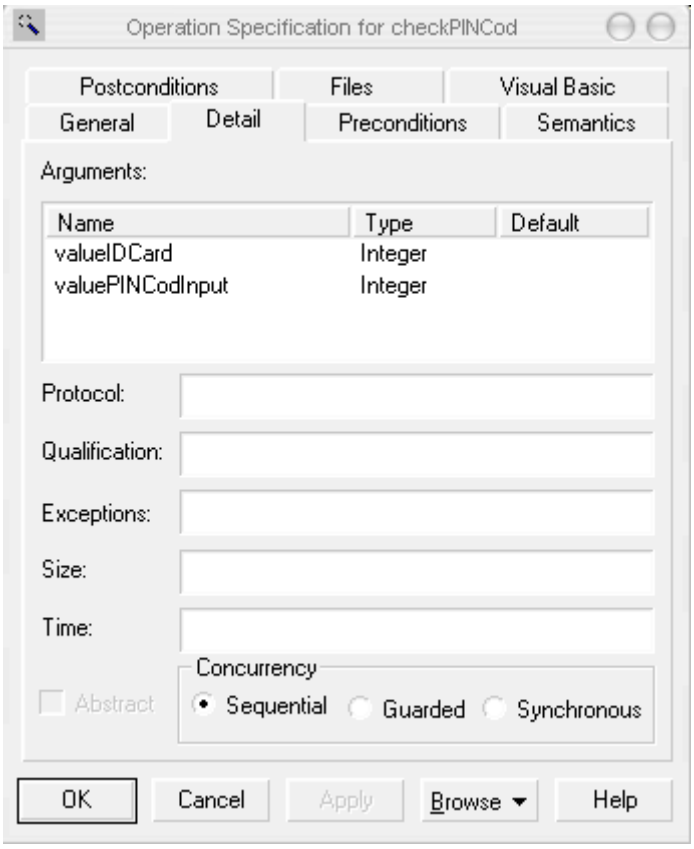

*Рисунок 5.4 – Окно спецификации свойств операции **checkPINCod()**, открытое на вкладке **Detail***

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

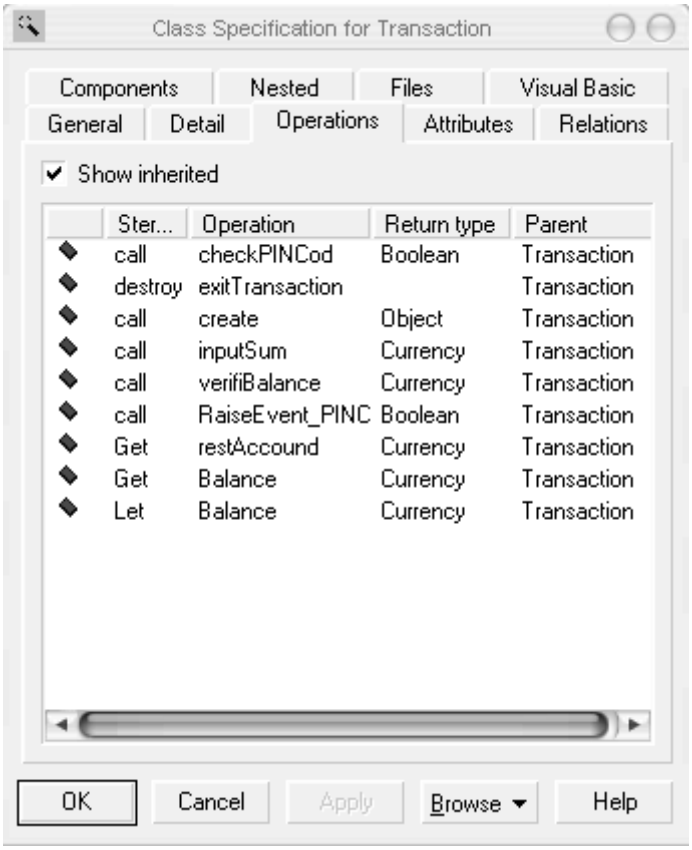

Рисунок 5.5 – Окно спецификации свойств операций класса, открытое на вкладке ***Operations***

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

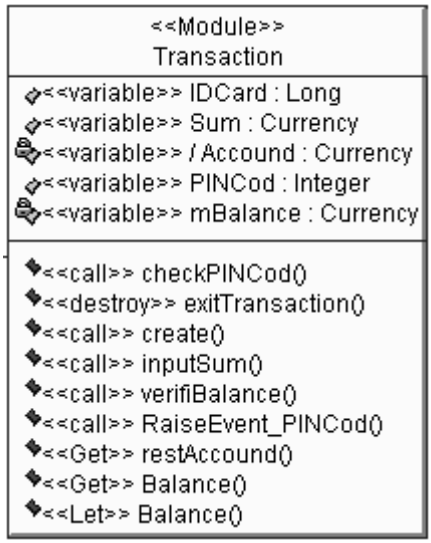

Рисунок 5.6 – Изображение класса ***Transaction*** *после добавления стереотипа, атрибутов и операций*

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 **ОК**, 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.

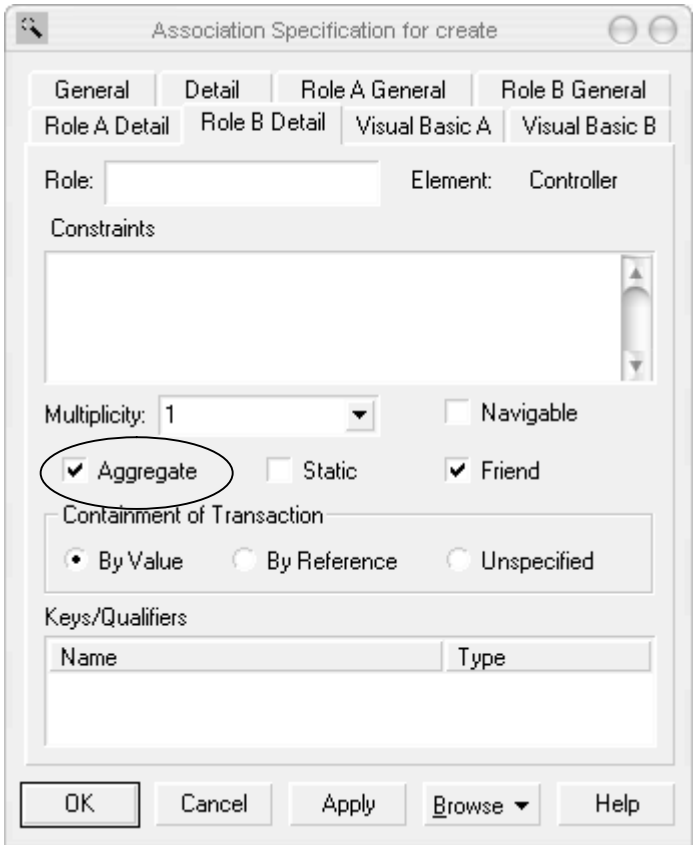

Рисунок 5.7 – Диалоговое окно спецификации свойств ассоциации ***create,*** *открытое на вкладке* ***Role B Detail***

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

6 МЕТОДИКА ПОДГОТОВКИ МОДЕЛИ ДЛЯ ГЕНЕРАЦИИ ПРОГРАММНОГО КОДА

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.

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 заглавии окна стоит **Untitled** – без заглавия) 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 6.1.

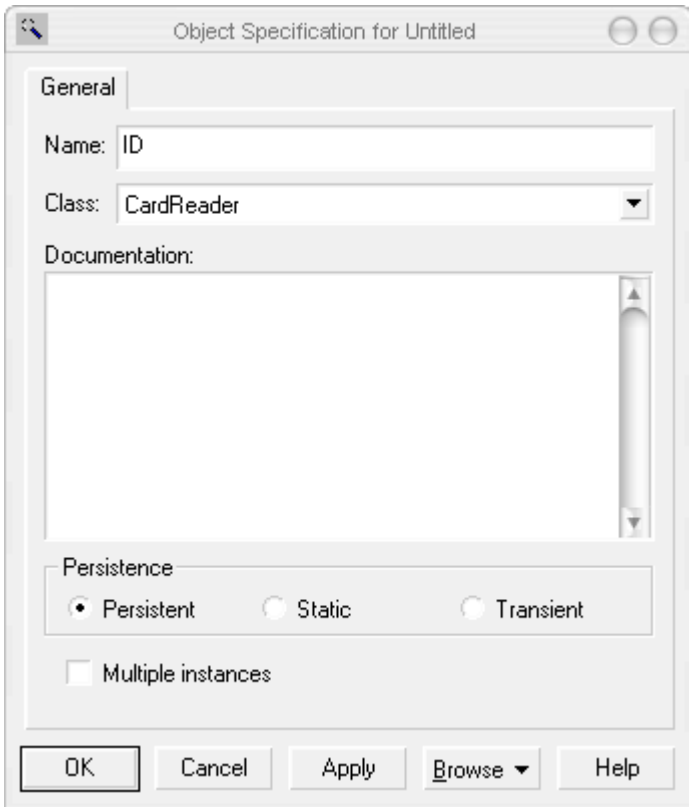

*Рисунок 6.1 – Окно спецификации свойств объекта **ID** (идентификационное устройство) класса **УстрЧтКарт** с новым именем **CardReader** после редактирования*

После редактирования всех классов следует отредактировать имена и свойства операций. Окончательный вид отредактированной диаграммы последовательностей показан на рисунке 6.2.

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.

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 ATM.** 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.

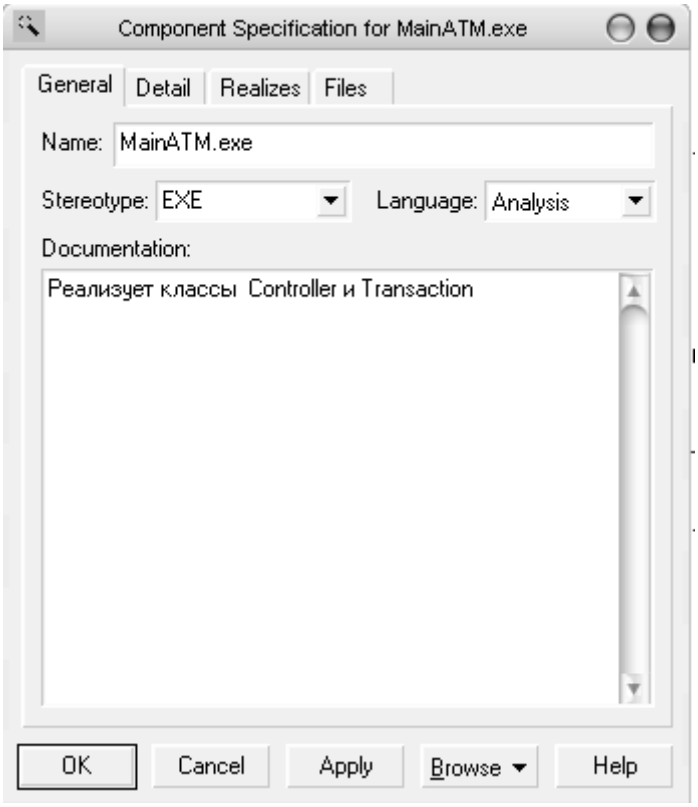

*Рисунок 6.3 – Диалоговое окно **Component Specification***

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.

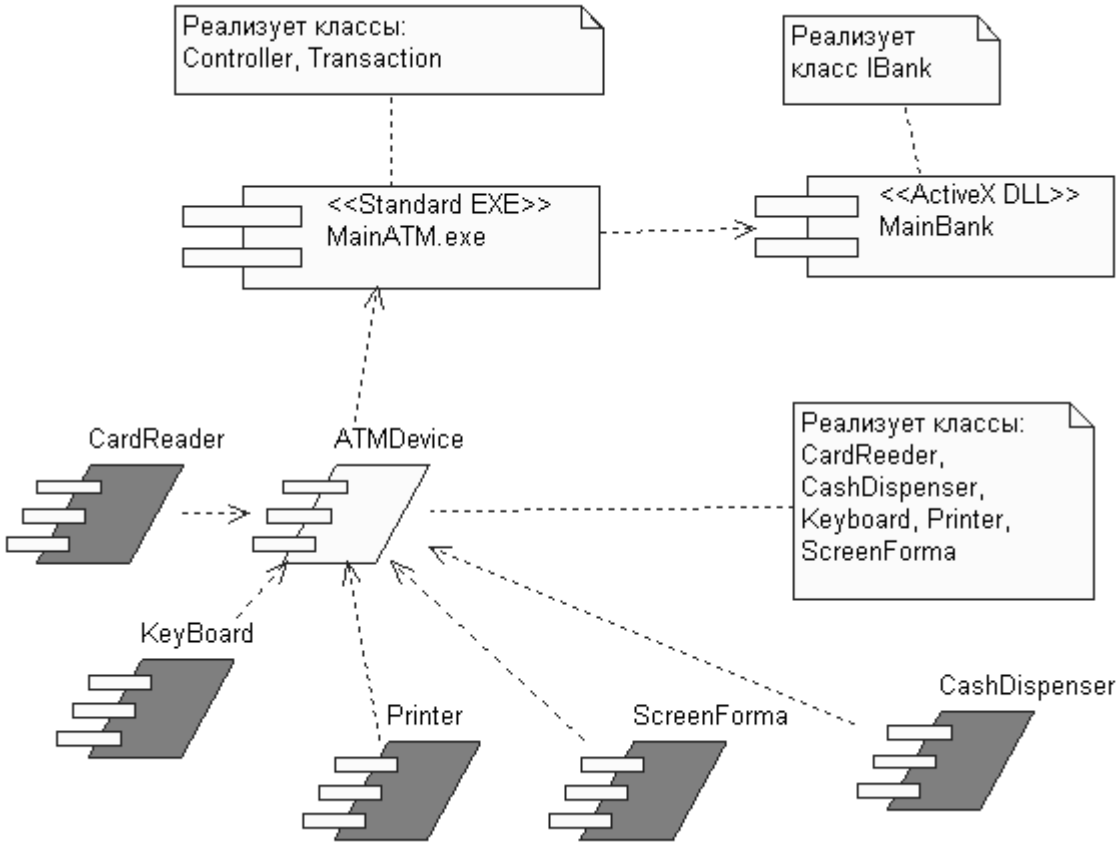

Рисунок 6.4 – Диаграмма компонентов системы управления ATM

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.

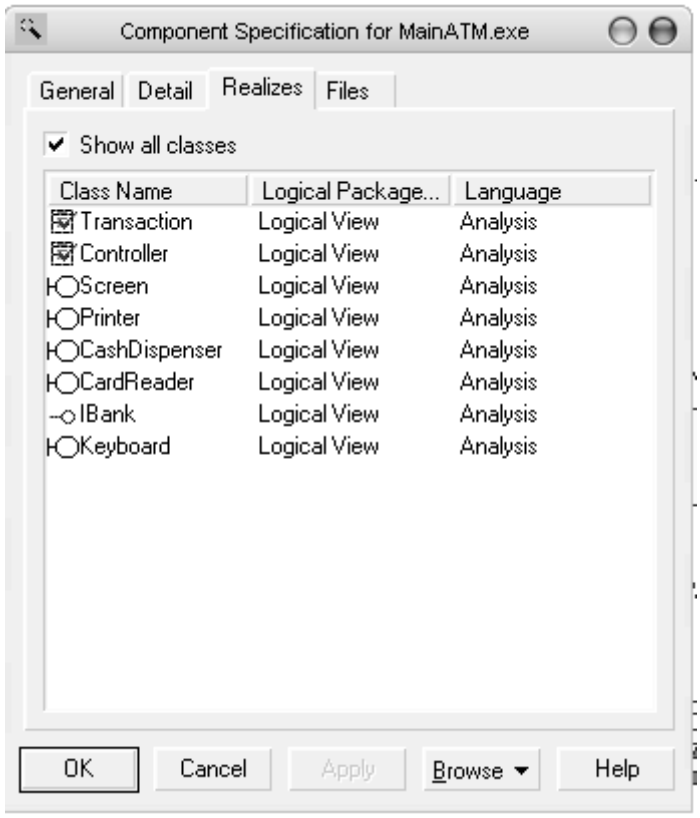

*Рисунок 6.5 – Диалоговое окно настройки свойств реализации классов в компоненте **MainATM.exe***

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 **ОК**.

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).

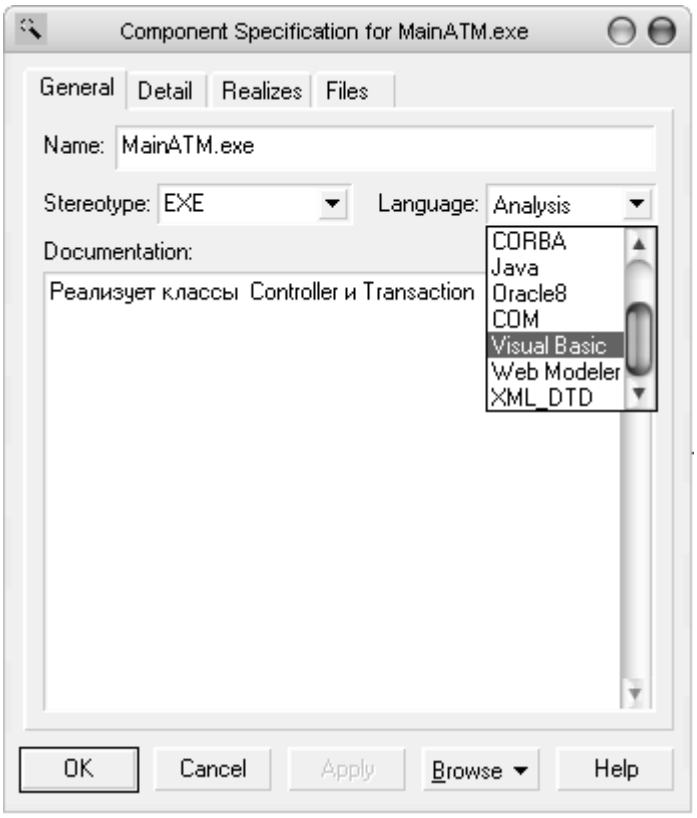

*Рисунок 6.6 – Окно спецификации свойств компонента **MainATM.exe** при выборе языка его реализации*

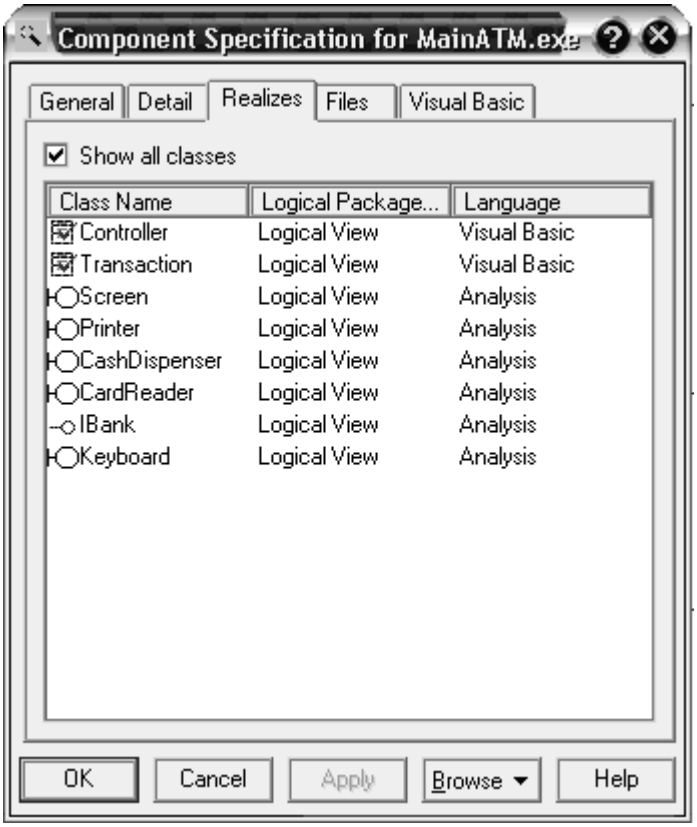

*Рисунок 6.7 – Окно спецификации свойств компонента **MainATM.exe**, открытое на вкладке **Realizes***

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 броузере 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 броузере проекта. Применительно к рассматриваемому примеру для генерации программного кода на языке Visual Basic выберем компонент **MainATM.exe**. Выделив компонент в броузере проекта, следует щелкнуть правой кнопкой мыши и в открывшемся контекстном меню выбрать операцию **Update Code**.

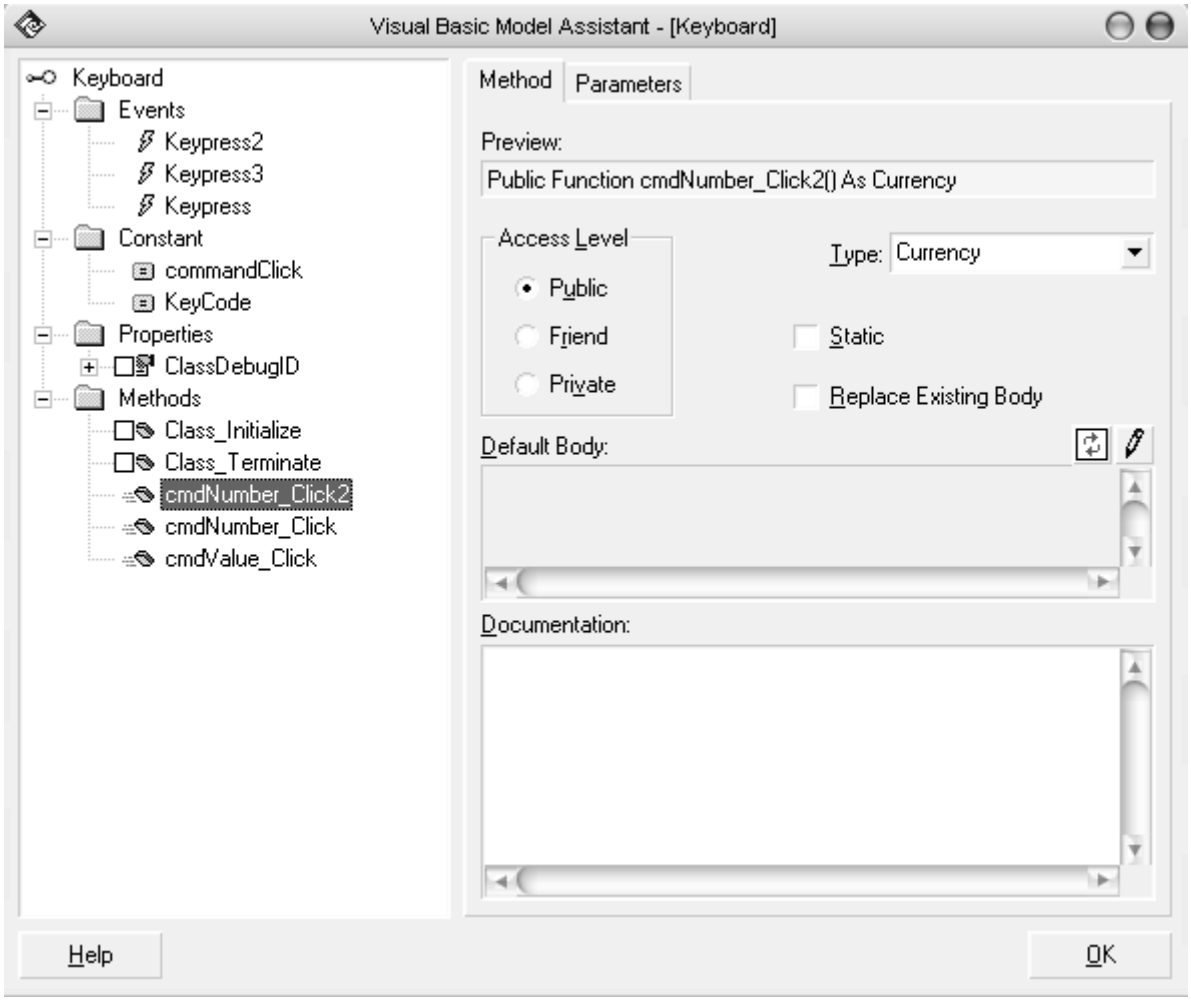

*Рисунок 6.8 – Окно **Visual Basic Model Assistant** для класса **Transaction***

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).

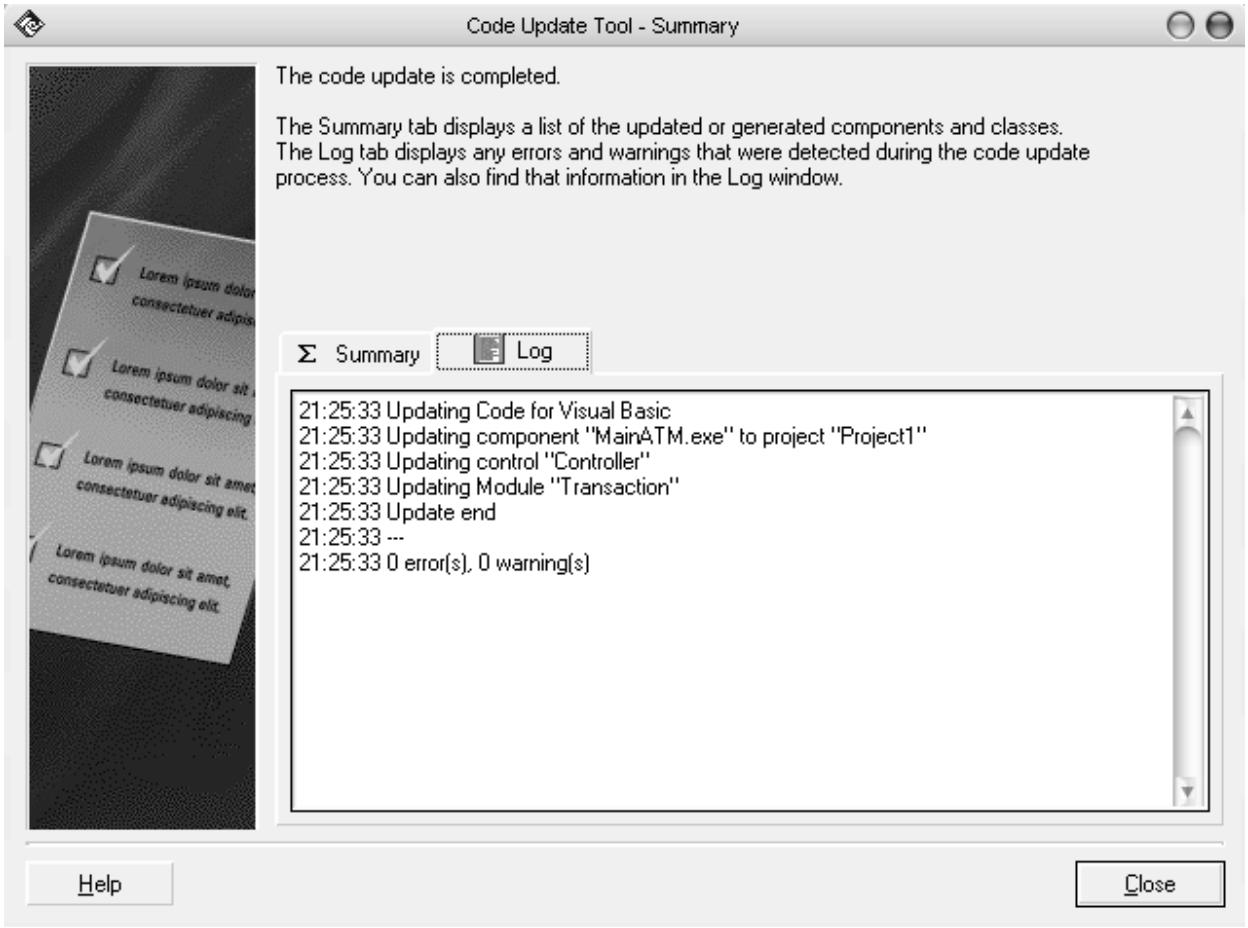

*Рисунок 6.10 – Окно вывода результата процесса генерации кода для компонента **MainATM.exe***

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_UniqueId" ,"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.

Литература

1. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. – М.: Конкорд, 1992. – 519 с.
2. Кватрани Терри. Визуальное моделирование с помощью Rational Rose 2002 и UML.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2003. – 192 с.
3. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose: Учебное пособие / – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 320 с.
4. Орлов С. А. Технология разработки программного обеспечения: Учебник для вузов. 3-е изд. / С. А. Орлов. – СПб.: Питер, 2004. – 527 с.
5. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов: Пер. с англ. – М.: ДМК Пресс, 2002. – 160 с.
6. Бенькович Е. С., Колесов Ю. Б., Сениченков Ю. Б. Практическое моделирование динамических систем – СПб.: БХВ-Петербург, 2002. – 464 с.
7. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК, 2000.
8. Леоненков А. В. Самоучитель UML – Санкт-Петербург: BHV, 2001.
9. Путилин А. Б., Юрагов Е. А. Компонентное моделирование и программирование на языке UML.: Практическое руководство по проектированию информационно-измерительных систем / – И.: НТ Пресс, 2005. – 664 с.
10. Новиков Л. Введение в Rational Unified Process.: *novikov@interface.ru*
11. Новичков А.Н. Эффективная разработка программного обеспечения с использованием технологий и инструментов компании Rational./ *http://www.compress.ru*
12. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002, 496 с.

Приложение А

Приблизительная тематика курсовых проектов

1. Разработать модель и каркас программного кода системы технического обучения персонала фирмы.
2. Разработать модель и каркас программного кода системы безопасности организации.
3. Разработать модель и каркас программного кода системы управления материальными ресурсами для малого предприятия.
4. Разработать модель и каркас программного кода системы учета энергетических ресурсов.
5. Разработать модель и каркас программного кода приложения для анализа графической информации.
6. Разработать модель и каркас программного кода приложения для задания параметров аналого-цифровой измерительной системы.
7. Разработать модель и каркас программного кода драйвера для платы сбора данных измерительной системы.
8. Разработать модель и каркас программного кода приложения для диагностики технологических объектов.
9. Разработать модель и каркас программного кода приложения для исследования поведения динамической системы.
10. Разработать модель и каркас программного кода системы управления технологическим процессом…(варианты определяются видом производства).
11. Разработать модель и каркас программного кода системы управления … (варианты: оборудование, бытовая техника, каналы передачи информации, транспортные средства и т.д.).
12. Разработать модель и каркас программного кода программной платформы для обеспечения связи “клиент-сервер”.

Приложение Б

Образец титульного листа

Донбасская государственная машиностроительная академия
Кафедра автоматизации производственных процессов

КУРСОВОЙ ПРОЕКТ

по дисциплине

«ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ СЛОЖНЫХ СИСТЕМ»

На тему:_____
_____

Выполнил студент гр. _____ _____
(фамилия и инициалы)

Руководитель _____
(фамилия и инициалы)

ОЦЕНКА:_____

Подписи членов комиссии_____

Краматорск____

Приложение В

Форма задания на проектирование

Донбасская государственная машиностроительная академия
Кафедра автоматизации производственных процессов

З А Д А Н И Е

на разработку курсового проекта по дисциплине

«Технология программирования сложных систем»

Студент гр._____

_____
(фамилия, имя, отчество)

Тема курсового проекта:_____

_____

_____

Задание выдал

_____ _____
(фамилия, инициалы)

Дата выдачи задания_____

Срок сдачи проекта_____