

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»

Кафедра программного обеспечения  
информационных технологий

**Л. В. БОЧКАРЁВА, М. В. КИРЕЙЦЕВ**

***СИСТЕМЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ  
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.  
РАБОТА В СРЕДЕ RATIONAL ROSE***

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ  
для студентов специальности  
«Программное обеспечение информационных технологий»  
всех форм обучения

Минск 2006

УДК 004.41(075.8)  
ББК 32.973 – 018.2 я 73  
Б 86

Р е ц е н з е н т:  
доцент кафедры МПСиС ИИТ БГУИР,  
кандидат технических наук В. Н. Мухаметов

**Бочкарёва, Л.В.**

Б 86 Системы автоматизации проектирования программного обеспечения. Работа в среде Rational Rose : учебно-метод. пособие для студ. спец. «Программное обеспечение информационных технологий» всех форм обуч. / Л. В. Бочкарёва, М. В. Кирейцев. – Мн. : БГУИР, 2006. – 38 с. : ил.

ISBN 985-488-051-6

Изложены особенности создания программного обеспечения с помощью инструментальных средств автоматизации процесса разработки. Рассмотрены особенности языка моделирования UML и его применение в среде автоматизированного синтеза Rational Rose. Предложены лабораторные работы по курсу «Системы автоматизации проектирования программного обеспечения».

УДК 004.41(075.8)  
ББК 32.973 – 018.2 я 73

ISBN 985-488-051-6

© Бочкарева Л. В., Кирейцев М. В., 2006  
© БГУИР, 2006

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. CASE-СРЕДСТВО RATIONAL ROSE .....	4
1.1. Работа в среде Rational Rose .....	5
1.2. Меню и диаграммы Rational Rose .....	6
2. ЛАБОРАТОРНЫЕ РАБОТЫ.....	7
Лабораторная работа № 1.....	7
Лабораторная работа № 2.....	10
Лабораторная работа № 3.....	12
Лабораторная работа № 4.....	14
Лабораторная работа № 5.....	16
Лабораторная работа № 6.....	19
Лабораторная работа № 7.....	21
Лабораторная работа № 8.....	24
Лабораторная работа № 9.....	31
ЗАКЛЮЧЕНИЕ.....	36
ЛИТЕРАТУРА.....	37

## ВВЕДЕНИЕ

В создании программ, как и в любом деле, важное место занимает творчество. Однако требования, предъявляемые к современным программным продуктам (ПП), заставляют ставить процесс их разработки на индустриальную основу [1,2]. Необходим инструмент, который позволит уменьшить долю ручного труда при программировании и предоставит большие возможности для творчества, освобождая программиста от выполнения рутинных операций и от ошибок [3]. Таким инструментом является CASE (Computer-Aided Software/System Engineering)- средство Rational Rose Enterprise Edition. Его можно использовать как отдельно, так и вместе с другими продуктами компании Rational Software, что позволяет создавать сложные программные системы (ПС) быстрее, качественнее и легче. Неслучайно Rational Rose включается во все конфигурации Rational Suite – инструмента для аналитиков, программистов и тестировщиков. Пользуясь Rational Rose, проектировщик может создать не абстрактное словесное описание системы, а его конкретную модель, которая затем дополняется описанием классов на языке программирования. Rational Rose поддерживает проектирование, основанное на двух способах: прямом и обратном. В первом режиме разработчик строит диаграммы классов и их взаимодействия, а на выходе получает сгенерированный код. Во втором режиме возможно построение модели на базе имеющегося исходного кода. Отсюда следует главная возможность для разработчика: повторное проектирование (round-trip). Программист описывает классы в Rational Rose, генерирует код, вносит изменения в модель и снова пропускает ее через Rational Rose для получения обновленного результата.

Сегодня уже очевидно, что моделирование позволяет значительно сократить время разработки, уложиться в бюджет и создать систему требуемого качества. Модели в виде UML (Unified Modeling Language) диаграмм, созданные в среде Rational Rose, имеют целый ряд замечательных особенностей. Они удобны для понимания алгоритмов работы, взаимосвязей между объектами системы и ее поведения в целом, а также позволяют непосредственно из проекта автоматически построить исходный текст программы на одном из языков, поддерживаемых Rational Rose.

Подведя итог вышесказанному, можно выделить следующие преимущества от применения Rational Rose.

- сокращение цикла разработки приложения “заказчик-программист-заказчик”;
- увеличение продуктивности работы программиста;
- улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;
- способность вести большие проекты и группы проектов;
- возможность повторного использования уже созданного программного обеспечения за счет упора на разбор его архитектуры и компонентов.

## 1. CASE-СРЕДСТВО RATIONAL ROSE

Rational Rose – это объектно-ориентированное средство автоматизированного проектирования ПС. В его основе лежит CASE-технология, комплексный подход и использование единой унифицированной нотации на всех этапах жизненного цикла создания ПС.

Для работы с Rational Rose необходим UML – графический язык описания архитектуры системы. Программы на UML представляются в виде диаграмм, состоящих из объектов и связей между ними или из этапов процесса проектирования [4,5]. Графические возможности продукта позволяют решать задачи, связанные с проектированием, на различных уровнях абстракции: от общей модели процессов предприятия до конкретной модели класса в создаваемом программном обеспечении (ПО). В среде Rational Rose проектировщик и программист работают в тандеме. Первый создает логическую модель системы, а второй дополняет ее моделями классов на конкретном языке программирования. В настоящее время ПП обеспечивает генерацию кода по модели на ряде языков программирования: Microsoft Visual C++, Ada, Java, Visual Basic, CORBA, XML, COM, Oracle. Кроме того, разрабатываются специальные мосты к не входящим в стандартную поставку языкам, например к Delphi.

### 1.1. Работа в среде Rational Rose

После запуска системы открывается ее главное окно, показанное на рис. 1.1. В верхней части экрана находится меню и стандартная панель инструментов. Она видна всегда, и ее кнопки соответствуют командам, которые могут использоваться для работы с любой диаграммой. Слева расположено окно Browser, представляющее собой иерархическую структуру и предназначенное для выполнения ряда действий:

- просмотр и добавление элементов к модели;
- просмотр существующих отношений между элементами модели;
- перемещение и переименование элементов модели;
- добавление элементов модели к диаграмме;
- связывание элемента с файлом или адресом Интернета;
- группирование элементов в пакеты;
- работа с детализированной спецификацией элемента;
- открытие диаграммы.

Каждый объект в Rational Rose имеет свое контекстное меню, посредством которого изменяются свойства и выполняются действия над объектом.

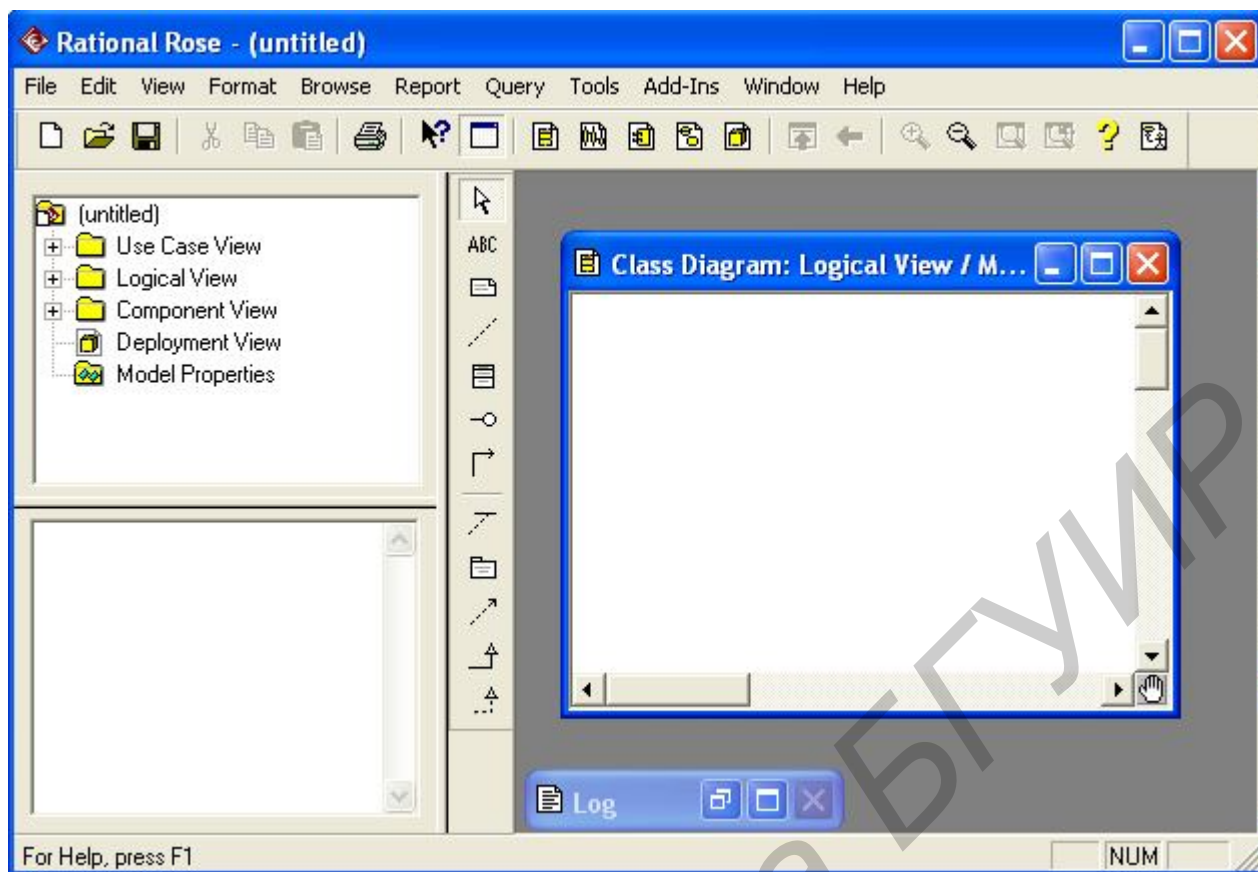


Рис. 1.1. Главное окно Rational Rose

Под Browser находится окно Documentation, предназначенное для документирования элементов модели Rational Rose. При описании класса вся информация из этого окна появится затем как комментарий в сгенерированном коде и в отчетах, создаваемых в среде Rational Rose. В случае смены активного элемента содержание Documentation автоматически обновляется.

В правой части экрана, называемой рабочим столом Rational Rose, находятся открытые в данный момент диаграммы. При создании новой модели на рабочем столе открывается Class Diagram (диаграмма классов). Окна Browser и Diagram разделены строкой инструментов, которая изменяется в зависимости от типа активной диаграммы. В низу рабочего стола видно свернутое окно Log (протокол), в котором фиксируются все действия, выполненные над диаграммами, также туда попадают сообщения об ошибках, произошедших в течение работы. Информация заносится в окно Log независимо от того, свернуто оно или вообще закрыто.

При изменении диаграмм в области рабочего стола Rational Rose автоматически обновит структуру Browser. Аналогично при внесении изменений в элемент с помощью Browser Rational Rose автоматически обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.

## 1.2. Меню и диаграммы Rational Rose

Рассмотрим состав и назначение пунктов меню главного окна Rational Rose, приведенного на рис. 1.1.

- *File* используется для сохранения, загрузки, обновления проекта, печати диаграмм и дополнительных настроек.
- *Edit* предназначен для копирования и восстановления данных, а также для редактирования свойств и стилей объектов.
- *View* применяется для настройки представления окон меню и строк инструментов.
- *Format* позволяет изменять параметры отображения объектов, такие, как шрифт, заливка, формат линий и т.д.
- *Browser* предназначен для навигации между диаграммами и спецификациями диаграмм, представленных в модели.
- *Report* предназначен для получения различного вида справок и отчетов.
- *Query* предоставляет возможности контролировать, какие элементы модели будут показаны на текущей диаграмме, и выполнять различные манипуляции с объектами диаграмм: скрывать, добавлять, фильтровать.
- *Tools* предоставляет доступ к различным дополнительным инструментам и подключаемым модулям.
- *Add-Ins* предоставляет доступ к менеджеру подключаемых модулей.
- *Window* позволяет управлять окнами на рабочем столе.
- *Help* позволяет получать справочную информацию.

Все перечисленные возможности среды Rational Rose служат для моделирования прикладной программной системы в виде совокупности диаграмм на основе графических средств языка UML.

## 2. ЛАБОРАТОРНЫЕ РАБОТЫ

### Лабораторная работа №1

#### ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ ПРИ ПОМОЩИ ДИАГРАММЫ *USE CASE*

##### Цель работы:

- научиться строить диаграммы Use Case в среде автоматизированного синтеза Rational Rose;
- разработать диаграмму Use Case для проектируемой прикладной системы.

##### Задание:

Средствами диаграммы Use Case описать сценарии поведения объектов разрабатываемой системы и создать список операций, которые она выполняет.

##### Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Use Case по предложенной тематике.



## Описание диаграммы Use Case

Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто Use case называют диаграммой функций, так как на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых ею функций. Данный тип диаграмм используется при описании бизнес-процессов автоматизируемой предметной области, определении требований к будущей программной системе.

Для построения диаграммы Use case необходимо запустить Rational Rose и создать новую пустую модель, а затем в окне Browser перейти на диаграмму Use case. Имеется несколько способов создания новых элементов в модели.

1. При помощи контекстных меню.
2. При помощи Menu:Tools=>Create.
3. При помощи строки инструментов.

В первом случае элемент создается непосредственно в модели, но его значок не включается ни в одну диаграмму. Поэтому после создания элемента необходимо поместить его на выбранную диаграмму. В двух других случаях вместе с созданием элемента его значок помещается на текущую диаграмму автоматически.

После активизации диаграммы Use Case соответствующая ей строка инструментов по умолчанию состоит из десяти значков, главными из которых являются  - Use Case (варианты использования) и  - Actor (действующие лица). Вместе они определяют сферу применения создаваемой системы. При этом первые описывают все то, что происходит внутри системы, а вторые – то, что происходит снаружи. Кроме этого, на диаграмме Use Case и Actor объединяются при помощи соответствующих связей. На рис. 2.1 приведена диаграмма Use Case.

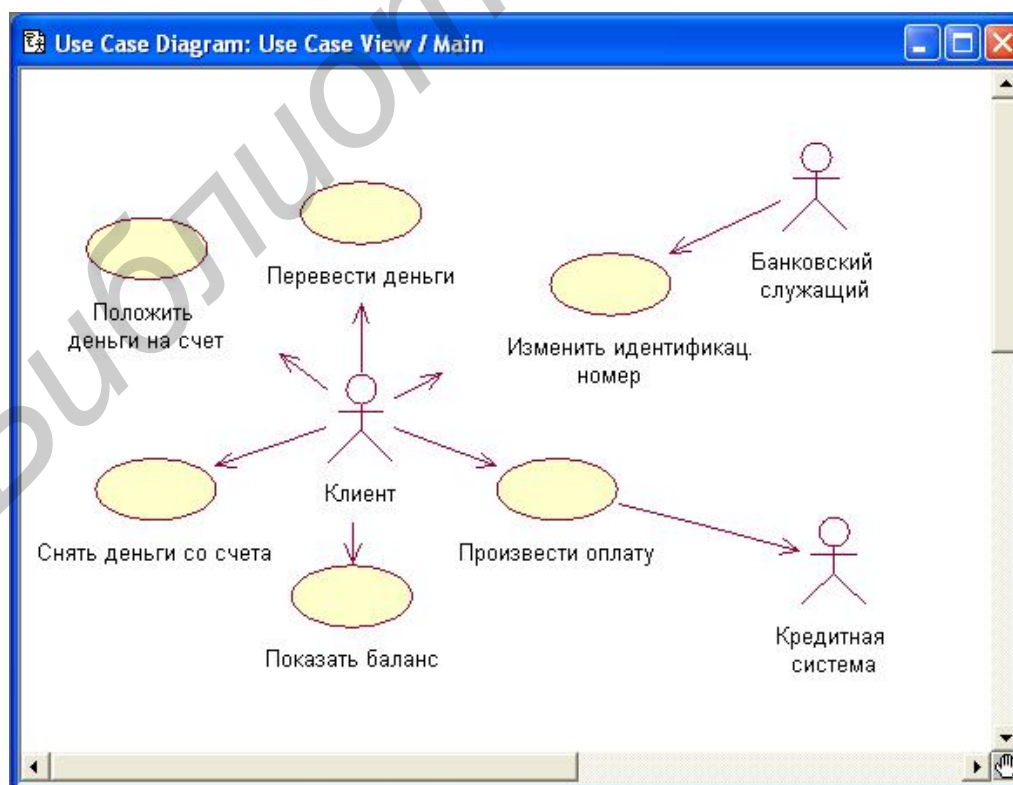


Рис. 2.1. Пример диаграммы Use Case



Здесь показаны три действующих лица: клиент, банковский служащий и кредитная система. Также предусмотрены шесть действий, выполняемых моделируемой системой: перевести деньги, положить деньги на счет, снять деньги со счета, показать баланс, изменить идентификационный номер и произвести оплату. Use Case и Actors соединены между собой однонаправленными связями. Обычно с диаграммы Use Case начинается проектирование ПС.

Для одной системы может создаваться несколько диаграмм Use Case. На диаграмме верхнего уровня, называемой в среде Rational Rose (Main), указываются только пакеты (группы) вариантов использования. Другие диаграммы описывают совокупности вариантов использования и действующих лиц. Конкретная реализация диаграмм Use Case зависит только от проектировщика. Главная диаграмма предлагается по умолчанию. Для получения доступа к ней необходимо в Browser выбрать элемент Use Case View, а затем с помощью контекстного меню выбрать пункт New => Use Case Diagram. После чего можно выделить новую диаграмму и ввести ее имя. Двойной щелчок на названии диаграммы в Browser открывает ее в области рабочего стола. Наполнение диаграммы элементами выполняется посредством ее панели инструментов. Рекомендуется придерживаться следующих правил, создавая диаграмму Use Case:

1. Не моделировать связи между Actors, так как по определению они находятся вне сферы действия системы. Следовательно, связи между ними также не относятся к ее компетенции.

2. Не соединять непосредственно два Use Case, поскольку данная диаграмма только перечисляет варианты использования, доступные системе, а не указывает порядок их выполнения.

3. Каждый вариант использования инициируется действующим лицом, поэтому должна быть связь, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Существуют два способа удаления элемента из диаграммы Use Case. Первый удаляет элемент из текущей диаграммы, но оставляет его в Browser и на остальных диаграммах системы. Для этого элемент выделяется в диаграмме и нажимается клавиша Delete. Второй метод удаляет элемент или диаграмму из модели, для чего необходимо его выделить в Browser и выбрать в контекстном меню пункт Delete. В среде Rational Rose невозможно отменить удаление диаграммы или удалить Main-диаграмму.

В последних версиях Rational Rose появились средства, позволяющие создавать модели производства. Их цели состоят в следующем:

- определение структуры и рабочих процессов организации, в которой будет использоваться разрабатываемая система;
- анализ проблем организации и поиск путей их решения;
- обеспечение общего понимания организации работы заказчиками и конечными пользователями;
- определение требований к системе, необходимых для поддержки производственных процессов организации.

Для моделирования производства Rational Rose предоставляет шесть дополнительных значков. Чтобы включить их в линейку инструментов диаграммы Use

Case, необходимо из контекстного меню ее инструментов выбрать *RClick=>Customize*. Отличительной особенностью этих значков является то, что они закрашены желтым цветом и имеют косую черту для выделения их на черно-белой печати.

### Вопросы для повторения

1. Для чего используется диаграмма Use Case?
2. Как создать новую диаграмму?
3. Какие значки находятся в строке инструментов диаграммы Use Case и каково их назначение?
4. Какие значки специфичны только для диаграммы Use Case?

## Лабораторная работа №2

### АНАЛИЗ УСТРОЙСТВ СРЕДСТВАМИ ДИАГРАММЫ *DEPLOYMENT*

#### Цель работы:

- научиться строить диаграммы Deployment в среде автоматизированного синтеза Rational Rose;
- разработать Deployment для проектируемой прикладной системы.

#### Задание:

С помощью диаграммы Deployment проанализировать аппаратную конфигурацию, на которой будут работать отдельные процессы системы, и описать их взаимодействие между собой и с другими аппаратными устройствами.



#### Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Deployment по предложенной тематике.

#### Описание диаграммы Deployment

Данная диаграмма предназначена для анализа аппаратной части системы. При помощи Deployment проектировщик может провести анализ необходимой аппаратной конфигурации, на которой будут работать процессы системы, и описать их взаимодействие между собой. Этот тип диаграмм также позволяет анализировать взаимодействие процессов, работающих на разных компьютерах сети.

С точки зрения используемых инструментов Deployment является самой простой диаграммой, так как в ней присутствуют только два вида основных значков.

Processor (процессор)  – это устройство, способное выполнять программы, и device (устройство)  – значок, отражающий на диаграмме все другие типы устройств. Процессором считается любая техническая система, способная обрабатывать данные. В эту категорию попадают серверы, рабочие станции и другие устройства, содержащие физические процессоры. Устройством на диаграмме обо-

значается аппаратура, не обладающая вычислительной мощностью, как, например, терминалы ввода/вывода, принтеры и сканеры. С помощью раздела спецификации вводится описание аппаратной части системы: стереотипы и характеристики. Стереотипы применяются для классификации процессоров и устройств, характеристики – это их физические особенности, например, скорость процессора и объем памяти.

Добавление и удаление элементов диаграммы Deployment выполняется теми же способами, которые были описаны для предыдущей диаграммы. Текстовое описание процессоров и устройств вводится в поле Documentation вкладки General раздела Specification.

На диаграмме Deployment может быть связана любая пара элементов. Чаще всего связи отражают физическую сеть соединений между узлами. Кроме того, это может быть ссылка Интернета, связывающая два узла. Для добавления связи на диаграмму можно использовать кнопку Connection панели инструментов. В Rational Rose разрешается назначать стереотипы для связей и задавать их характеристики, представляющие собой техническое описание физического соединения. Кроме процессоров и устройств на Deployment могут отображаться процессы, правила работы с которыми аналогичные. Процессом считается поток информации, например исполняемый файл, выполняющийся на процессоре.

Рассмотрим пример построения диаграммы Deployment для системы обработки заказов (рис. 2.2). Она включает в себя четыре процессора: *Сервер базы данных*, *Сервер приложений*, *Клиентская рабочая станция 1*, *Клиентская рабочая станция 2* и устройство *Принтер*.

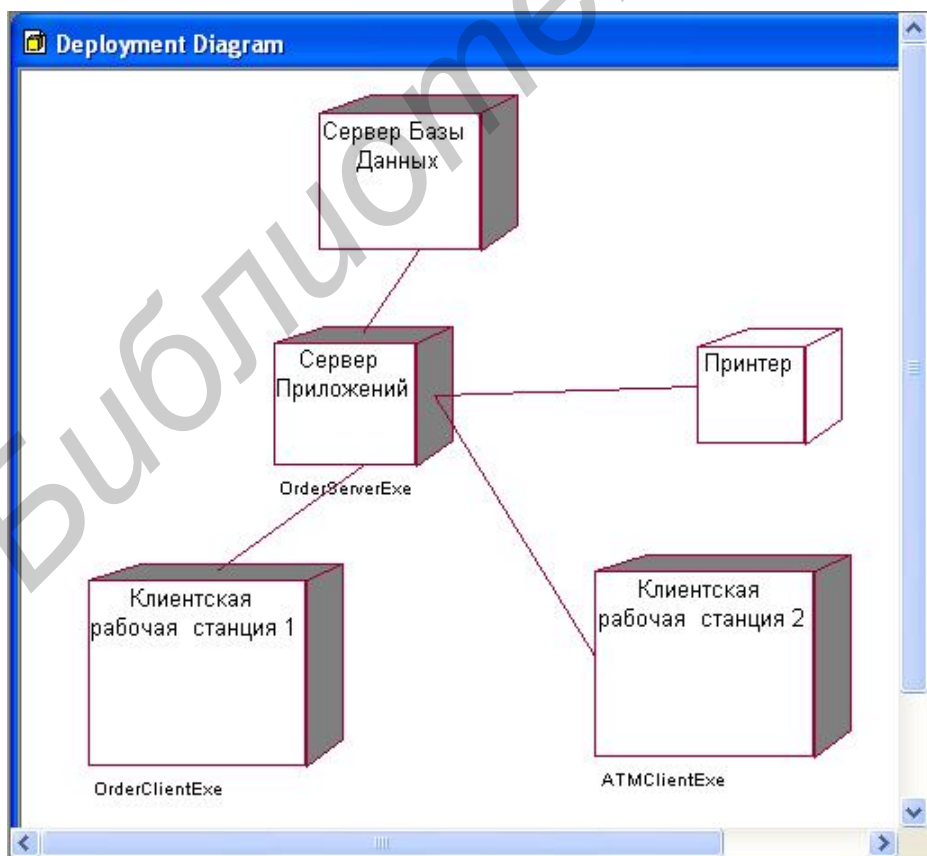


Рис. 2.2. Диаграмма Deployment

С процессором *Сервер приложений* связан процесс *OrderServerExe*, с *Клиентской рабочей станцией 1* связан процесс *OrderClientExe*, с *Клиентской рабочей станцией 2* – процесс *ATMClientExe*.

### Вопросы для повторения

1. Для чего используется диаграмма Deployment?
2. Каково назначение объектов Processor, Device, Connection?

## Лабораторная работа №3

### СОЗДАНИЕ МОДЕЛИ ПОВЕДЕНИЯ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММЫ STATECHART

#### Цель работы:

- научиться строить диаграммы Statechart в среде автоматизированного синтеза Rational Rose;
- разработать диаграмму Statechart для проектируемой прикладной системы.

#### Задание:




С помощью диаграммы Statechart описать состояния объектов системы и условия переходов между ними. Добавить в модель классы, на основе которых будут создаваться исследуемые объекты.


#### Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Statechart по предложенной тематике.

#### Описание диаграммы Statechart

Диаграмма состояний Statechart предназначена для изучения состояний объектов и условий переходов между ними. Модель состояний позволяет представить поведение объекта при получении им сообщений и взаимодействии с другими объектами. Кроме ранее предложенных способов создания диаграмм, Statechart можно построить из контекстного меню рассматриваемого класса. После активации диаграммы станет доступным набор ее инструментов.

Рассмотрим диаграмму Statechart, приведенную на рис. 2.3. Первым шагом на пути построения является создание точки начала работы с помощью инструмента Start State . Обычно следующим состоянием системы после начала ее работы является ожидание наступления событий. После чего в нашем случае создается состояние (State ) , которое соединяется стрелкой State Transition  с начальной точкой. Для повышения информативности состояниям и событиям, переводящим объекты из одних состояний в другие, присваиваются имена. С состоянием объекта могут быть связаны события и действия. Разница между ними

заключается в том, что действие осуществляется самим классом, для которого строится диаграмма Statechart, т.е. вызывается метод данного класса, а посылка сообщения направлена на объект другого класса, чей метод вызывается при помощи сообщения. Для того чтобы получить доступ к действиям и событиям, нужно через контекстное меню объекта перейти в окно его спецификации, где выбрать вкладку Actions, а в ней – кнопку Insert. На рис. 2.3 состояние «Создан кладовщиком при получении товара» связано с действием «Создан в двух экземплярах». Построение диаграммы завершается добавлением на нее значка  – End State, который отражает окончание работы. Направление перехода может быть установлено только в End State. Однако нет ограничений на количество самих элементов и переходов в них.

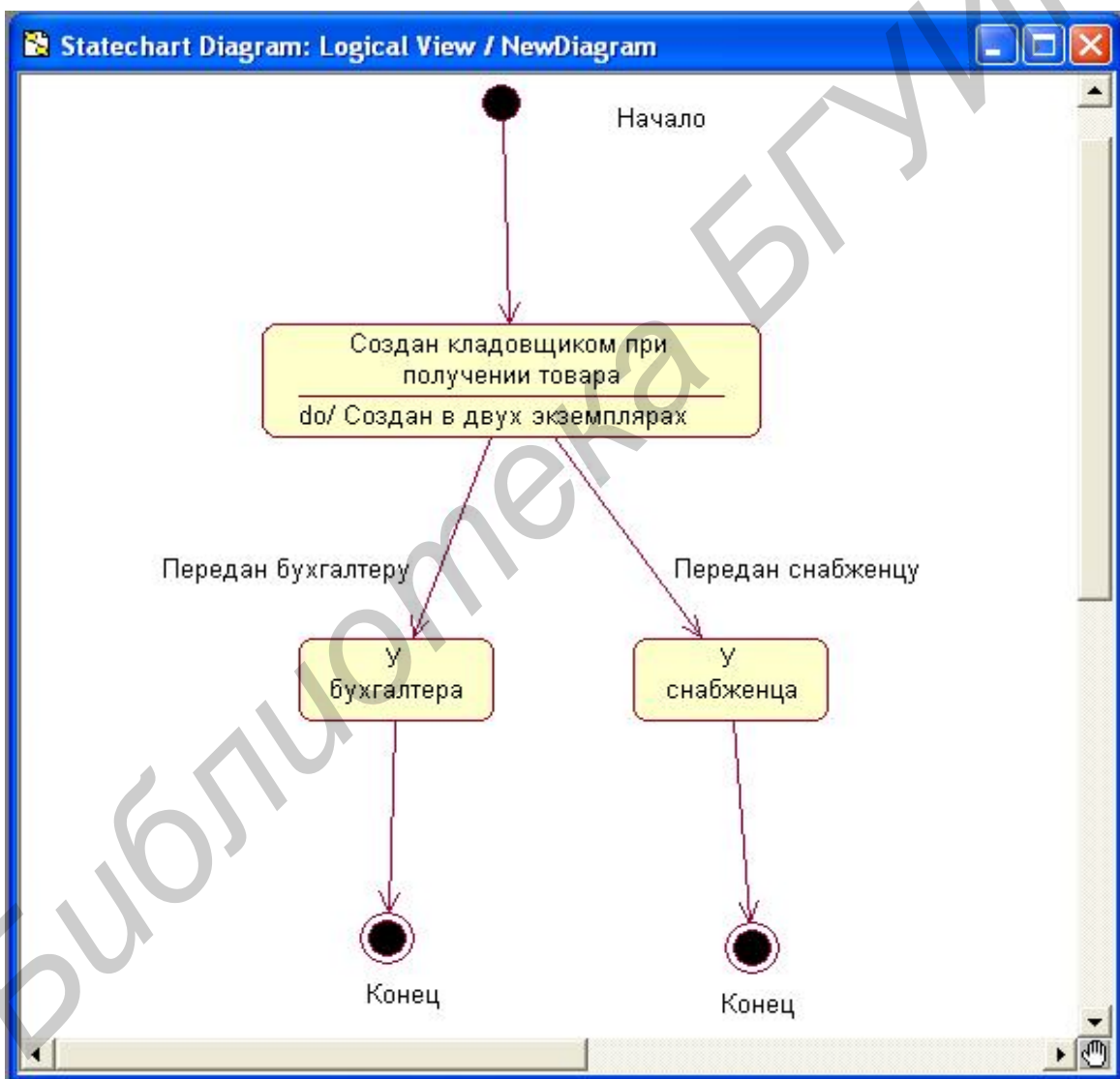


Рис. 2.3. Диаграмма Statechart

### Вопросы для повторения

1. Для чего предназначена диаграмма Statechart?
2. Какие инструменты доступны в диаграмме?
3. Какие бывают переходы между состояниями?

## Лабораторная работа №4

### СОЗДАНИЕ МОДЕЛИ ПОВЕДЕНИЯ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММЫ *ACTIVITY*

#### Цель работы:

- научиться строить диаграммы Activity в среде автоматизированного синтеза Rational Rose;
- разработать диаграмму Activity для проектируемой прикладной системы.

#### Задание:






С помощью диаграммы Activity промоделировать действия объектов проектируемой системы. Сравнить данную диаграмму с предыдущим типом диаграмм состояний и определить, в каких случаях следует применять каждую из них.

#### Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Activity по предложенной тематике.

#### Описание диаграммы Activity

Данный тип диаграмм является разновидностью диаграмм состояний. Главное различие между Activity и Statechart заключается в том, что первая характеризует действия, а вторая – статичные состояния. При этом Activity больше подходит для моделирования последовательности действия, а Statechart – для моделирования дискретных состояний объекта.

Для построения Activity доступны те же способы, что и для Statechart. При создании новой диаграммы состояний будет предложено выбрать из двух возможных вариантов: Statechart Diagram и Activity Diagram. После активизации Activity станет доступным набор ее инструментов. Как и Statechart diagram, Activity начинается значком Start State и завершается значком End State. Одноименный с диаграммой значок  обозначает выполнение определенных действий в течение жизни объекта. В отличие от , обычно обозначающего ожидание какого-либо события,  показывает непосредственное действие. Деятельности соединяются на диаграмме значком  – State Transition (переход состояния). Кроме этого,  показывает получение и обработку сообщения объектом. Переход состояния может происходить между Action-Action, State-State, State-Action, Action-State. Возможна установка нескольких переходов между двумя состояниями или действиями. Каждый такой переход уникален и показывает реакцию объекта на

определенное сообщение. Поэтому нельзя создать несколько переходов между двумя состояниями с указанием одного и того же сообщения.

Рассмотрим построение диаграммы Activity на примере описания работы склада в ходе получения им товара от продавца (рис. 2.4). На диаграмме изображены: виды деятельности, связанные с решением задачи и подлежащие автоматизации; входные и выходные документы или данные, связанные с конкретными действиями; исполнители действий и подразделения, в которых они выполняются.

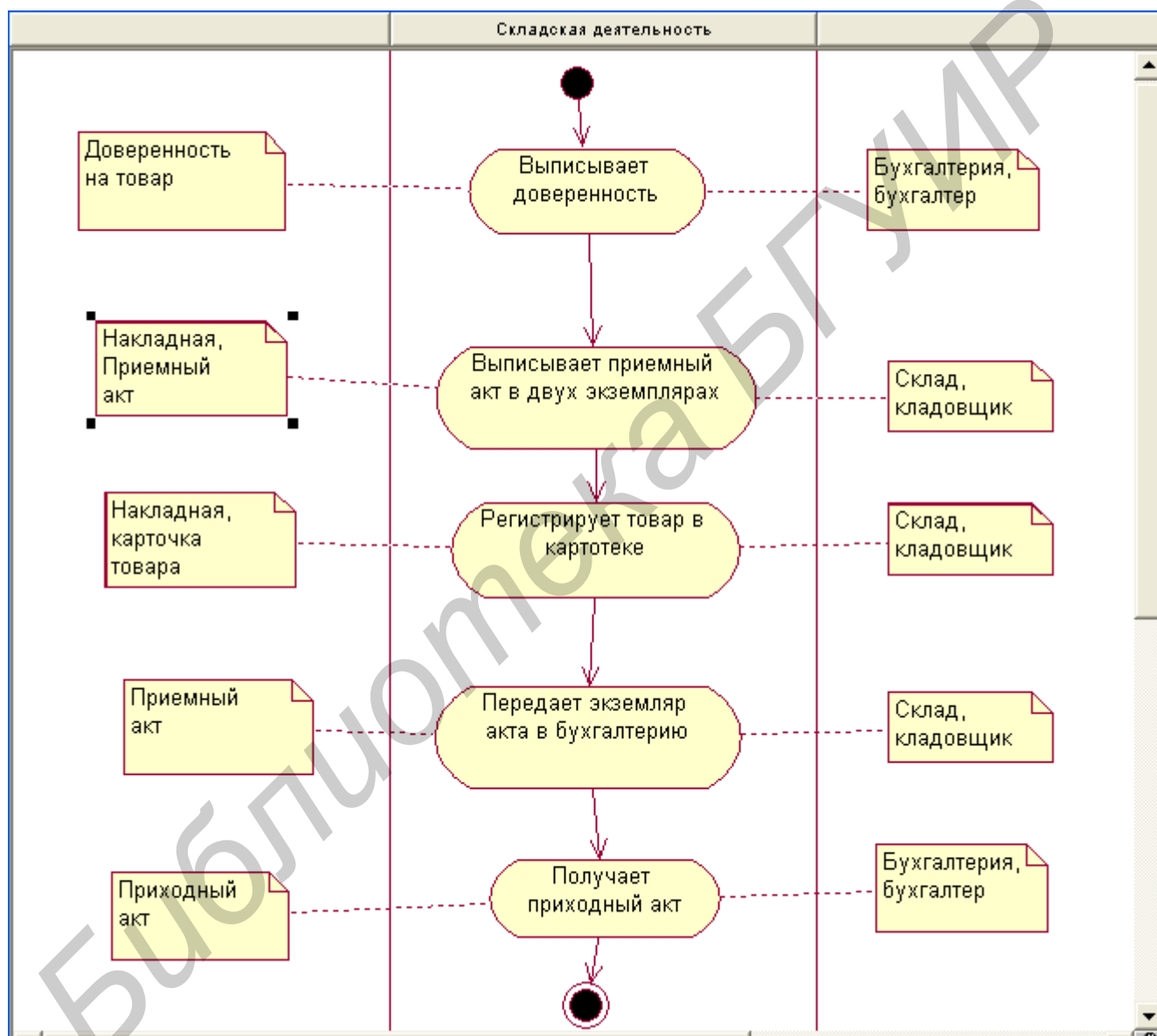




Рис. 2.4. Описание работы склада при получении товара от продавца

Для повышения наглядности диаграммы каждую деятельность поясняет значок  – Note (замечание), который связан с ней при помощи значка  – Anchor to Note (якорь для замечания).

Уникальным инструментом диаграммы Activity является значок  – Swimlanes (плавательные дорожки), который позволяет моделировать последова-

тельность действий различных объектов и связи между ними. С его помощью можно строить бизнес-процессы организации, отражая на диаграмме различные подразделения и объекты. Swimlanes помогают показать роли каждого участника бизнес-процесса. Для этого необходимо переместить соответствующие значки активности или состояний в определенную часть диаграммы, отделенную от остальных Swimlanes. На рис. 2.4 изображены три Swimlanes, хотя фактически представлены действия, относящиеся только к одной дорожке «Складская деятельность». Две другие дорожки поясняют работу склада в бизнес-процессе организации.

### **Вопросы для повторения**

1. Для чего предназначена диаграмма Activity?
2. Какие отличия между диаграммами Activity и Statechart?
3. Каковы основные инструменты диаграммы Activity и для чего используется элемент Swimlanes?

## **Лабораторная работа №5**

### **ОПИСАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММЫ SEQUENCE**

#### **Цель работы:**

- научиться строить диаграммы Sequence в среде автоматизированного синтеза Rational Rose;
- разработать диаграмму Sequence для проектируемой прикладной системы.

#### **Задание:**

С помощью диаграммы Sequence получить отражение во времени процесса обмена сообщениями между объектами создаваемой системы.

#### **Порядок выполнения работы**

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Sequence по предложенной тематике.




#### **Описание диаграммы Sequence**

После того как в системе изучено поведение каждого объекта, необходимо точно представить взаимодействие этих объектов между собой, определить клиентов, серверы и порядок обмена сообщениями между ними. Диаграмма Sequence позволяет получить отражение процесса обмена сообщениями во времени.



В течение работы системы объекты, являющиеся клиентами, посылают друг другу различные сообщения, а объекты-серверы обрабатывают их. В простейшем случае можно рассматривать сообщение как вызов метода какого-либо класса, в более сложных случаях сервер имеет обработчик очереди сообщений, и сообщения им обрабатываются асинхронно, т.е. сервер накапливает несколько сообщений в очереди, если не может обработать их сразу.

Диаграмма взаимодействия создается одним из способов, которые были предложены в предыдущих разделах. Выбор какого-либо из них повлечет за собой открытие окна, в котором предложено два типа диаграмм: Sequence и Collaboration. После активизации диаграммы Sequence станет доступным набор ее инструментов.

Рассмотрим построение диаграммы Sequence на примере сценария работы кладовщика с карточкой товара и накладной (рис. 2.5), в нем использованы основные инструменты Sequence. Значок  – Object (объект) позволяет включить объект в диаграмму. Каждый из них является реализацией какого-нибудь класса, поэтому в объекте можно указать соответствующий ему класс. Для повышения наглядности в примере использованы стереотипы классов. Кладовщик представлен как business worker, Накладная – как business entity, Карточка товара – как business entity. Значок  – Message (сообщение) предназначен для передачи сообщения от одного объекта к другому. Классы должны позволять отправку или прием сообщений. Инструмент  – Message to self (сообщение самому себе) показывает, что отправитель сообщения является одновременно и его получателем. В этом случае объект выполняет функции и сервера, и клиента. Для реализации объектов в окне параметров доступна опция Persistence, отражающая время жизни объекта. Это время от его создания до уничтожения. Обычно объекты существуют в пределах их области видимости и автоматически уничтожаются системой, когда выходят за эту область. Таким образом, можно настроить следующие параметры жизни объекта:

- Persistent – область видимости объекта превышает время жизни.
- Static – элемент существует на всем протяжении работы программы.
- Transient – время жизни объекта и области видимости совпадают, этот вариант присваивается по умолчанию и подходит в случае нашего примера.

Горизонтальные линии на рис. 2.6, проведенные от объекта к объекту, означают передачу сообщений между ними. Свойства задаются в окне спецификации с помощью двух групп кнопок.

1. *Synchronization* определяет порядок обмена сообщениями и может быть выбрана из следующих вариантов:

- Simple – простая посылка сообщения;
- Synchronous – операция происходит только в том случае, когда клиент посылает сообщение, а сервер может принять сообщение клиента;
- Timeout – клиент отказывается от выдачи сообщения, если сервер в течение определенного времени не может его принять;
- Balking – операция происходит только в том случае, когда сервер готов немедленно принять сообщение, иначе клиент не выдает сообщение;

- Procedure Call – клиент вызывает процедуру сервера и полностью передает ему управление;
- Return – возврат из процедуры;
- Asynchronous – клиент выдает сообщение и, не ожидая ответа сервера, продолжает выполнение своего программного кода.

2. *Frequency* определяет частоту обмена сообщениями:

- Periodic – сообщения поступают от клиента с заданной периодичностью;
- Aperiodic – сообщения поступают от клиента нерегулярно.

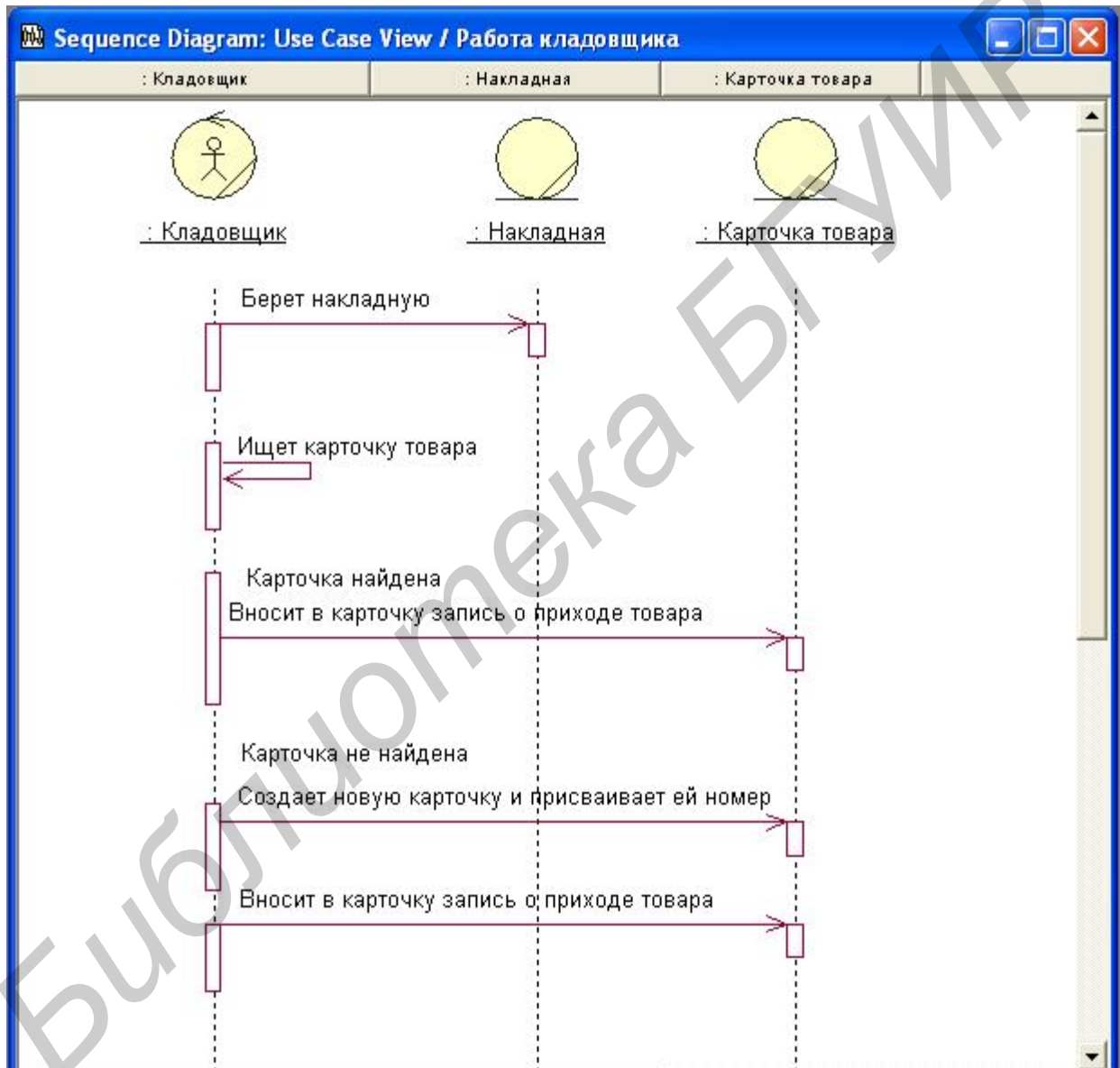


Рис. 2.5. Пример работы кладовщика с карточкой товара и накладной

### Вопросы для повторения

1. Для чего предназначена диаграмма Sequence?
2. Какие виды сообщений позволяет отразить диаграмма?
3. Как настроить отображение времени жизни объекта?

## Лабораторная работа №6

### ОПИСАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ СИСТЕМЫ ПРИ ПОМОЩИ ДИАГРАММЫ *COLLABORATION*

#### Цель работы:

- научиться строить диаграммы Collaboration в среде автоматизированного синтеза Rational Rose;
- разработать Collaboration для проектируемой прикладной системы.

#### Задание:


Средствами диаграммы Collaboration отразить взаимодействие всех объектов системы. Показать связь данной диаграммы с Sequence.




#### Порядок выполнения работы


1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Collaboration по предложенной тематике.

#### Описание диаграммы Collaboration

Второй тип диаграмм взаимодействия – это Collaboration. Данная диаграмма отличается от Sequence тем, что здесь не акцентируется внимание на последовательности передачи сообщений, а отражается наличие взаимосвязей между клиентами и серверами вообще. Поскольку на Collaboration для демонстрации сообщений не применяется временная шкала, диаграмма получается более компактной и оптимально подходит для представления взаимодействий сразу всех объектов. Однако такое представление является мгновенным снимком системы в некотором состоянии, так как объекты создаются и уничтожаются на всем протяжении работы программы. В связи с этим появляются такие понятия, как время жизни и область видимости объектов.

Удобной возможностью работы в Rational Rose является то, что на основе Sequence-диаграммы можно создавать Collaboration и наоборот. Для построения Collaboration нужно, находясь в диаграмме Sequence, выбрать Menu:Browse =>Create Collaboration diagram. При построении с помощью значка Interaction diagram  и выбора Collaboration в диалоговом окне создается пустая диаграмма, и в нее не перенесутся уже существующие объекты. На рис. 2.6 представлена диаграмма Collaboration, построенная на основе диаграммы Sequence (см. рис. 2.5).

Для построения приведенной диаграммы Collaboration были использованы ее основные инструменты. Значок Object  позволяет создавать объекты, которые имеют состояния и поведение. Значок  – Object Link (связь объекта) отражает взаимодействие объектов посредством показа их связей. При этом один из объектов может посылать сообщение другому. Значок  – Link To Self показывает,

что объект имеет обратную связь с самим собой. Значок  – Link Message (передача сообщения) позволяет отразить связь, которая подразумевает обязательную передачу сообщения. У каждой связи Link Message есть соответствующие свойства, которые позволяют настроить область видимости для связанных объектов. Для задания области видимости объектов используется диалоговое окно, приведенное на рис. 2.7. Сервер описывается в блоке Supplier visibility, клиент – в блоке Client visibility. В них доступны значения для выбора:

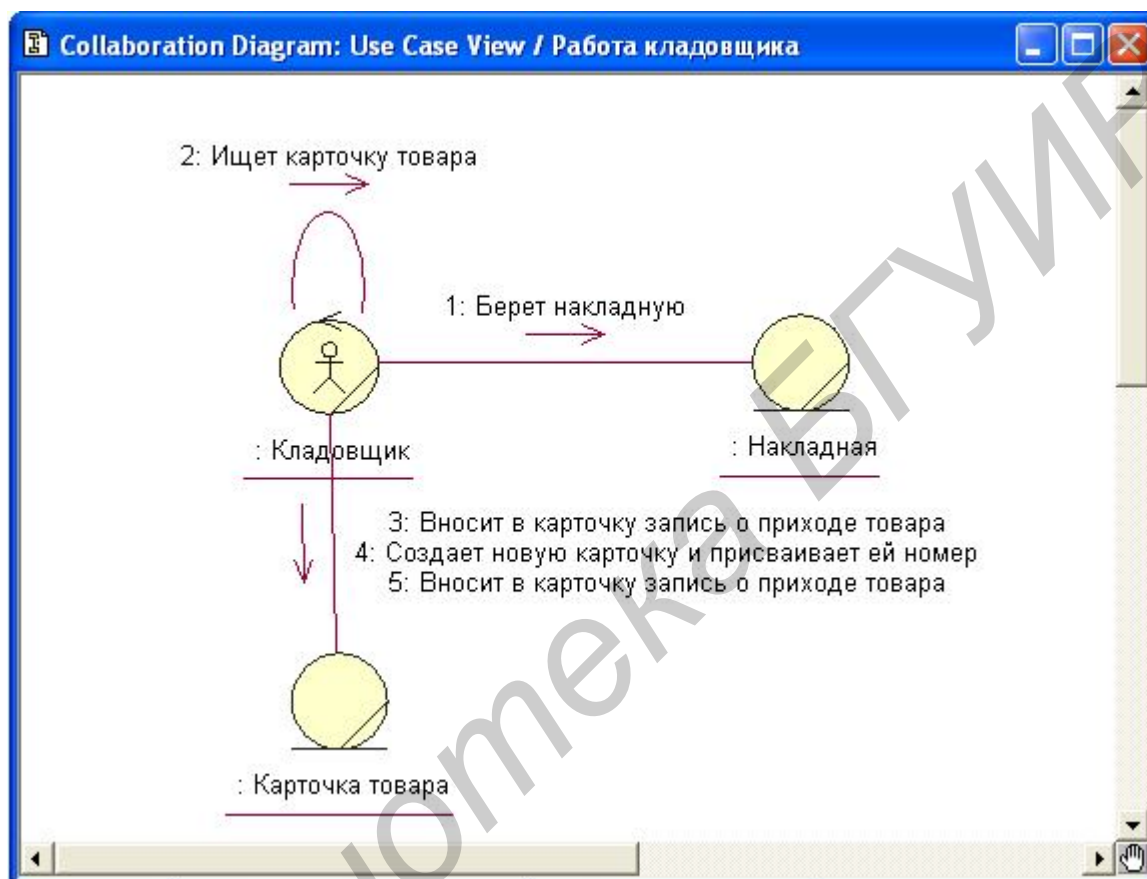


Рис. 2.6. Пример работы кладовщика с карточкой товара и накладной

- Unspecified – не определено, это значение присваивается по умолчанию;
- Field – объект включен в другой объект;
- Parameter – объект передается параметром в другой объект;
- Local – объект локально определен в границах другого объекта;
- Global – объект глобален по отношению к другому объекту.

При изменении области видимости на концах соединяющей линии появляется квадратик с указанной областью видимости.

На рис. 2.6 объект Кладовщик выделен как центральный, и все сообщения поступают к нему или исходят от него. Взаимодействия между Кладовщиком и Накладной, а также Карточкой товара изображаются линиями с добавленными стрелками, аналогичными тем, которые изображаются на Sequence-диаграмме. Все сообщения одного направления собираются вместе и предлагаются как подпись к одной стрелке.

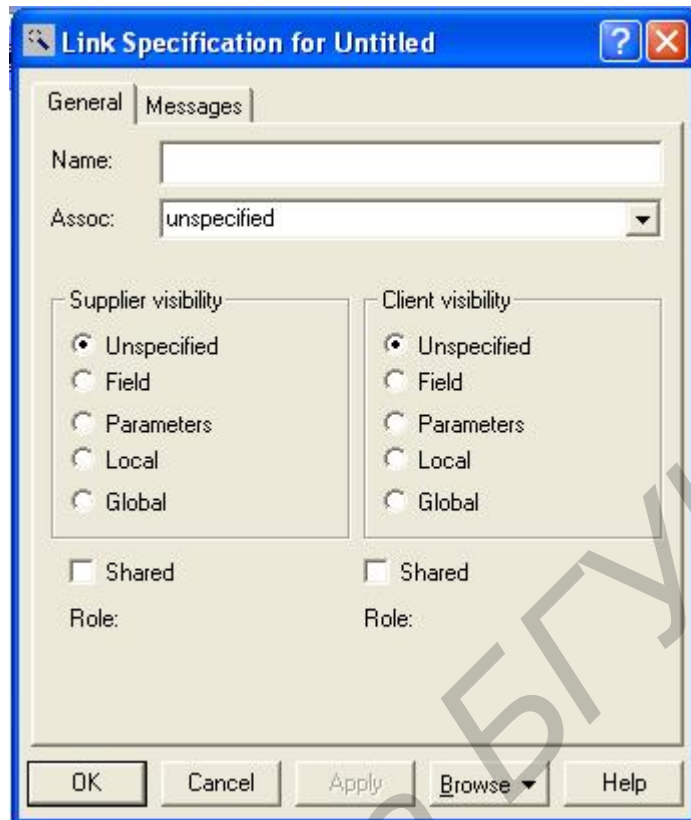


Рис. 2.7. Задание области видимости для Link Message

Находясь в области диаграммы Collaboration, можно создавать новые объекты и связи между ними. Если затем переключиться в диаграмму Sequence, то станет очевидным, что туда автоматически перенеслись все внесенные изменения.

### Вопросы для повторения

1. Для чего предназначена диаграмма Collaboration?
2. Как перенести данные между диаграммами Sequence и Collaboration?
3. Какие возможности имеются для настройки области видимости объектов?

## Лабораторная работа №7

### ДИАГРАММА *COMPONENT*

#### Цель работы:

- научиться строить диаграммы Component в среде автоматизированного синтеза Rational Rose;
- разработать Component для проектируемой прикладной системы.

## **Задание:**

Средствами диаграммы Component создать физическое отражение модели системы, т.е. показать организацию и взаимосвязи программных компонентов, представленных в исходном коде, двоичных или выполняемых файлах.

### **Порядок выполнения работы**


1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Component по предложенной тематике.

### **Описание диаграммы Component**

Диаграмма компонентов позволяет создать физическое отражение текущей модели. Component показывает организацию и взаимосвязи программных компонентов, представленных в файлах различных типов, а ее связи отражают зависимости одного компонента от другого. В текущей модели может быть создано несколько диаграмм компонентов для отражения пакетов, компонентов верхнего уровня или описания содержимого каждого пакета компонентов.

Для систем, состоящих из большого количества классов, целесообразно строить диаграмму компонентов, когда определены все связи классов и структура наследования. Но поскольку на всем протяжении проектирования системы, вплоть до выхода готового программного продукта в диаграммы будут вноситься изменения, оправдано создание Component для нескольких классов, чтобы получить практику работы с данным типом диаграмм.

В Rational Rose заложена возможность работы с программными библиотеками, их можно как создавать, так и пользоваться уже готовыми. Необходимо только указать, какие классы, в каких компонентах будут находиться. Для того чтобы обеспечить минимальные трудозатраты на разработку и сопровождение, тесно связанные между собой классы собираются в библиотеки.

Диаграмму компонентов можно построить двумя способами: с помощью меню Browse=>Component diagram или воспользовавшись значком Component diagram  на панели инструментов. После чего будет активизировано диалоговое окно выбора диаграммы, посредством которого создается, удаляется, переименовывается диаграмма.

Рассмотрим построения Component diagram на примере системы обслуживания банкоматов архитектуры клиент-сервер (рис. 2.8), на диаграмме показаны компоненты клиента. Система реализуется на языке программирования Visual C++. У каждого класса имеется свой собственный заголовочный файл и файл с расширением .cpp, поэтому каждый класс преобразуется в свои собственные компоненты на диаграмме, представляющие тело и заголовок класса. Темные компоненты соответствуют файлам тела класса на Visual C++, прозрачные компоненты – заголовочным файлам классов языка Visual C++.

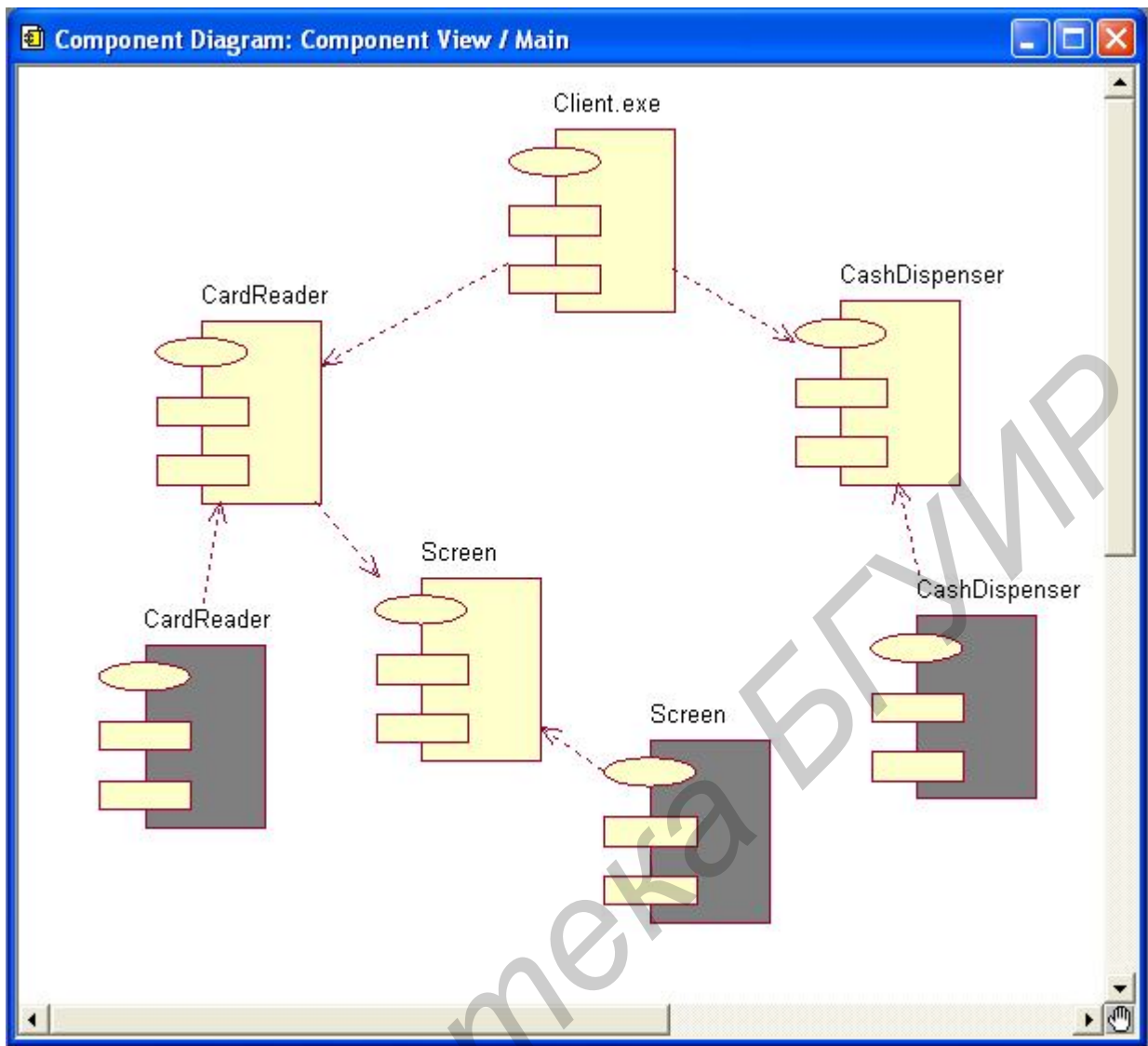





Рис. 2.8. Диаграмма компонентов

Для работы исполняемого файла Client.exe необходимы заголовочные файлы CardReader и CashDispenser, для того чтобы класс CardReader мог быть скомпилирован, класс Screen должен уже существовать. В свою очередь, заголовочные классы CardReader, CashDispenser и Screen используются для компиляции соответствующих им файлов на языке Visual C++. После компиляции всех классов может быть создан исполняемый файл Client.exe.

Для построения приведенной диаграммы компонентов были использованы некоторые из ее инструментов. Значок Package specification  позволяет отобразить определение пакета, а значок Package body  выполняет описание пакета. Обычно эти инструменты связаны между собой. Здесь Package specification – заголовочный файл с расширением .h, а Package body – файл с расширением .cpp. С помощью значка dependency  устанавливаются связи между компонентами. Этот тип связи показывает, что классы, содержащиеся в компоненте-клиенте, наследуются, содержат элементы, используют или каким-либо другим образом зависят от классов, которые экспортируются из компонента-сервера.

## Вопросы для повторения

1. Для чего предназначена диаграмма Component?
2. Какие инструменты предоставляет диаграмма?

## Лабораторная работа №8

### ДИАГРАММА CLASS

#### Цель работы:

- научиться строить диаграммы Class в среде автоматизированного синтеза Rational Rose;
- разработать диаграмму Class для проектируемой прикладной системы.

#### Задание:


Средствами диаграммы Class разработать внутреннюю структуру системы, описать наследование и взаимное положение классов относительно друг друга, используя соответствующие типы связей между ними.

#### Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить диаграмму Class по предложенной тематике.

#### Диаграмма Class

Диаграмма классов является основной для создания кода приложения. С ее помощью строится внутренняя структура системы. Обычно данная диаграмма строится для всех классов, становясь логической моделью системы. Кроме того, Rational Rose позволяет на основе Class diagram создавать исходный код приложения на любом языке программирования, который поддерживается генератором кода Rational Rose. С ее помощью возможно изменение свойств любого класса или его связей, при этом диаграммы или спецификации, связанные с изменяемым классом, будут автоматически обновлены.

Главная диаграмма классов (Main) уже присутствует во вновь созданной пустой модели, но возможно создание дополнительных диаграмм посредством контекстного меню Logical View в окне Browse или при помощи кнопки Class diagram . Создание нового класса и помещение его на диаграмму выполняется с помощью соответствующего значка на строке инструментов или из меню Tools =>Create=>Class. После добавления класса в диаграмму становится доступно его контекстное меню. Содержание меню изменяется при ассоциации класса с разными языками программирования.



Рассмотрим пункты меню для класса и свойства класса, не ассоциированного с каким-либо языком программирования (рис. 2.9, а, б).

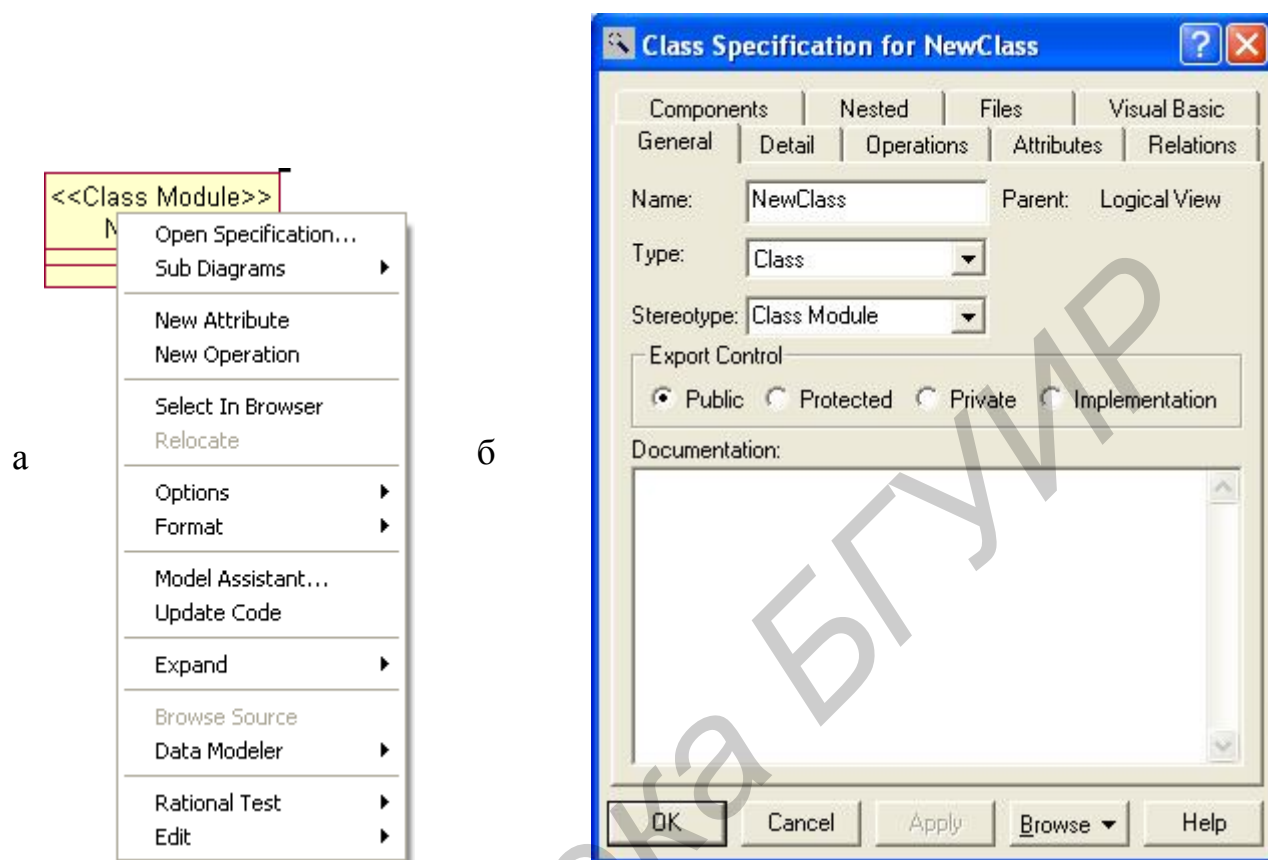


Рис. 2.9 Задание класса:

а – контекстное меню класса; б – окно спецификации класса

Назначение отдельных пунктов меню:

- Open Specifications – открытие диалогового окна заполнения спецификаций;
- Sub Diagrams – создание для текущего класса диаграммы активности и состояний или переход на поддиаграммы класса;
- New Attribute – добавление нового атрибута классу;
- New Operation – добавление новой операции классу;
- Select in Browser – выделение класса в окне Browser;
- Relocate – перемещение класса в новый пакет или на новое местоположение;
- Options – вызов подменю настройки значка класса;
- Format – вызов подменю настройки шрифта, цвета, заливки диаграммы.

Rational Rose позволяет устанавливать значительное количество свойств класса, которые влияют на генерацию его кода. Спецификация класса имеет несколько вкладок, и первой из них активизируется вкладка General (рис. 2.9, б). В этом окне задаются главные свойства класса: имя, тип, стереотип и доступ к нему, когда класс находится в пакете, а также документация к классу.

Вкладка Detail позволяет указать дополнительные характеристики класса (рис. 2.10):

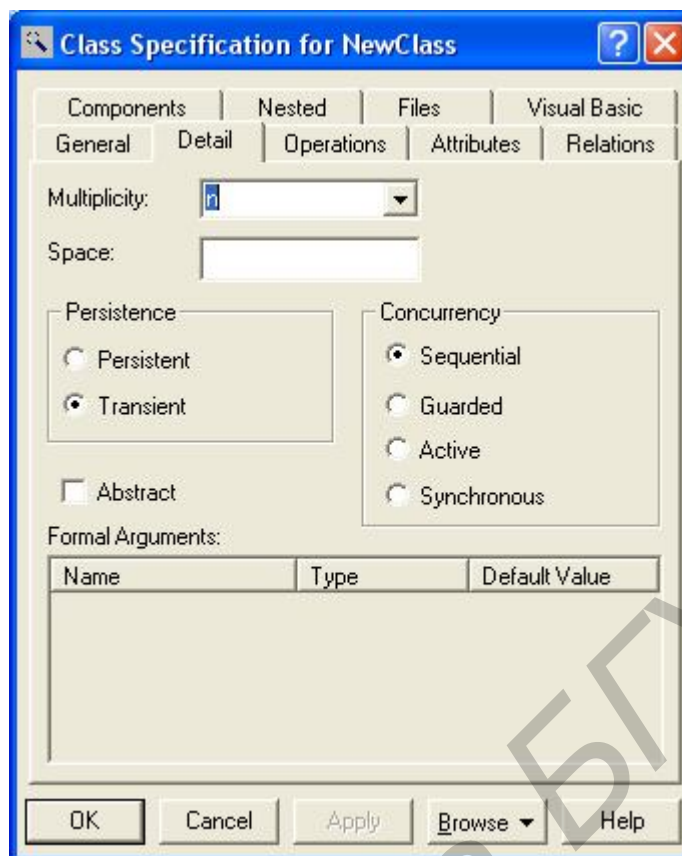


Рис. 2.10. Окно вкладки Detail

- Multiplicity – ожидаемое количество объектов, которые будут созданы на основе данного класса;
- Space – количество оперативной памяти, необходимой для создания объекта, учитывая накладные расходы на его создание плюс размер всех объектов, входящих в данный;
- Persistence – признак, указывающий время жизни объекта;
- Concurrency – поведение элемента в многопоточковой среде;
- Abstract adornment обозначает, что класс является абстрактным, т.е. базовым, который должен быть наследован подклассами;
- Formal Arguments заполняется только для параметризованных классов и утилит классов.

Вкладка Components отражает компоненты, с которыми ассоциирован класс (рис. 2.11). На вкладке рядом с иконками помечаются компоненты, которые должны включаться в модель, и могут быть показаны остальные компоненты модели.

Вкладка Attributes позволяет добавлять, удалять, редактировать атрибуты класса (рис. 2.12). На ней представлен список атрибутов класса, который можно редактировать при помощи контекстного меню.

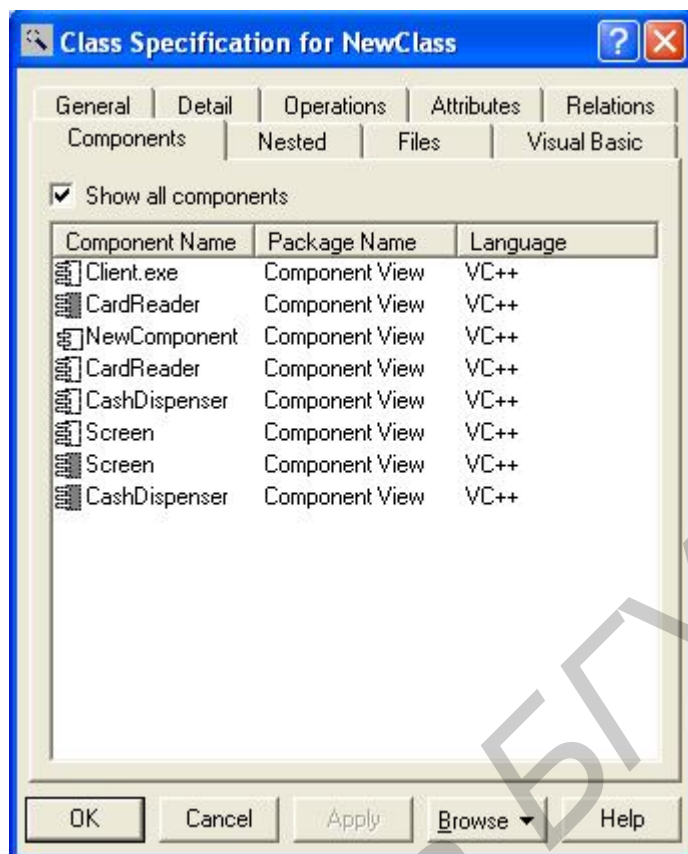


Рис. 2.11. Окно вкладки Components

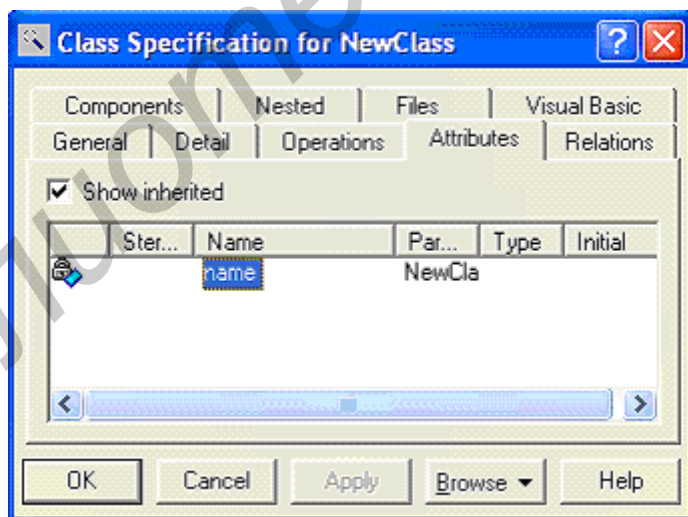


Рис. 2.12. Окно вкладки Attributes

Флажок Show inherited позволяет скрыть или показать доступные атрибуты родительских классов. Для того чтобы добавить атрибут, необходимо из контекстного меню выбрать пункт Insert. В диалоговом окне спецификаций атрибутов можно изменить его название, тип и стереотип, задать начальное значение и тип доступа к атрибуту. Вкладка Detail спецификаций атрибутов класса позволяет задать тип хранения атрибута в классе:

- By Value – по значению;
- By Reference – по ссылке;
- Unspecified – не указано.

Кроме того, можно указать, что атрибут является Static (статическим) или Derived (производным).

Вкладка Operations предназначена для добавления, удаления, редактирования операции класса (рис. 2.13), на ней представлен список операций класса, которые можно редактировать при помощи контекстного меню. Для того чтобы добавить операцию, необходимо из контекстного меню выбрать пункт Insert. С операциями связано окно их спецификаций.

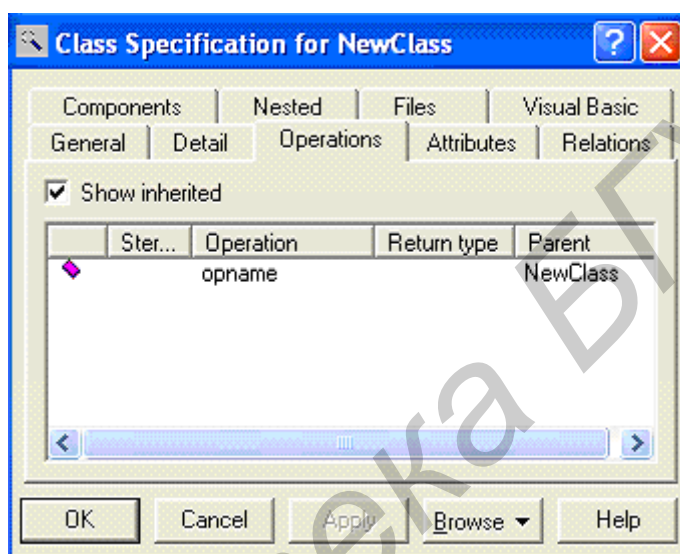


Рис. 2.13. Окно вкладки Operations

Вкладка Relations позволяет добавлять, удалять, редактировать связи класса. На ней представляется список связей класса, которые можно редактировать при помощи контекстного меню.

Вкладка Visual Basic, появившаяся после ассоциации класса с языком Visual Basic, предназначена для изменения свойств, связанных с данным классом. Ее поля не предназначены для редактирования.

На вкладке COM устанавливаются свойства для классов, которые связаны с созданием COM объектов в модели. Если такие объекты импортируются в модель, в них также появляется подобная вкладка.

### Назначение и виды связей в диаграмме Class

В большинстве случаев классы взаимодействуют друг с другом, что отображается при помощи различного вида связей, влияющих на получаемый при генерации код. В диаграмме классов используются следующие виды связей:

- Unidirectional association (однаправленная ассоциация);
- Dependency (зависимость);
- Association class (ассоциированный класс);

- Generalization (наследование);
- Realization (реализация).

Unidirectional association – это один из важных и сложных типов связи. Она показывает, что один класс включается в другой как атрибут по ссылке или по значению. На рис 2.14 приведен пример связи Unidirectional association.

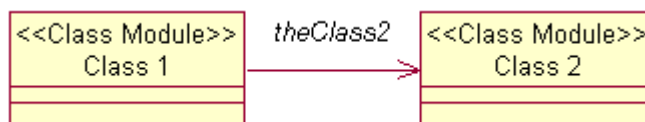


Рис. 2.14. Unidirectional association

Создаваемый код класса зависит от установленных спецификаций связи. При активизации окна спецификаций открывается ее вкладка General, где содержится следующая информация о связи:

- Name – имя связи;
- Parent – имя пакета, которому принадлежит связь;
- Stereotype – стереотип;
- Role A/Role B – имя роли, с которой один класс ассоциируется с другим;
- Element A/Element B – имя класса, который ассоциирован с данной ролью.

На вкладке Detail указываются дополнительные свойства связи, такие как:

- Name Direction – имя связанного класса;
- Constraints – выражение семантического условия, которое должно быть выполнено, в то время как система находится в устойчивом состоянии.

Вкладка Role General отражает настройки переменной, которая будет включена в класс. Поскольку направление связи на рис. 2.15 от Class1 к Class2, то Role A – это Class2, а Role B – это Class1. Рассмотрим вкладку Role A General. Она имеет следующие поля:

- Role – имя переменной для класса;
- Element – имя класса, для которого создается переменная;
- Export Control – доступ к элементу; имеется четыре переключателя: Public, Protected, Private, Implementation, которые указывают, в какой секции была создана переменная.

Вкладка Role Detail детализирует установки для связи и имеет поля:

- Role – имя переменной класса;
- Element – имя класса, для которого создается переменная;
- Constraints – выражение семантического условия, которое должно быть выполнено, в то время как система находится в устойчивом состоянии;
- Multiplicity – ожидаемое количество объектов данного класса, которые задаются числом, отображаемым рядом со стрелкой связи или буквой «n», указывающей, что количество не лимитировано;
- Navigable – направление, в котором действует связь. На какой класс будет направлена стрелка связи, тот и будет включаться в другой. Для того чтобы изменить направление связи, достаточно снять флажок с вкладки Role A Detail и уста-

новить его во вкладке Role B Detail. В случае, когда сняты флажки на обеих вкладках, ни один элемент не будет включен в другой, на диаграмме этому будет соответствовать просто линия;

- **Aggregate** – один класс содержит другой. Для того чтобы показать, что класс Class2 входит в класс Class1, необходимо установить этот флажок во вкладке Role B Detail. При этом стрелка связи на диаграмме приобретает ромб с обратной стороны стрелки (рис. 2.15);

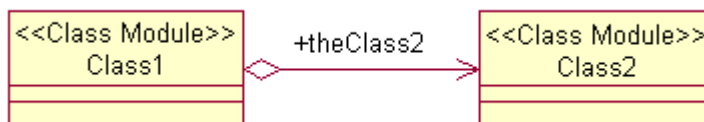


Рис. 2.15. Агрегирование класса

- **Static** – общность реквизита для всех объектов данного класса. При этом после инициализации к нему можно обращаться, даже если еще не было создано ни одного объекта класса. Static применяется, чтобы переменные такого типа не тиражировались при создании нового объекта класса;

- **Friend** – указанный класс является дружественным, т.е. имеет доступ к защищенным методам и атрибутам;

- **Key/qualifier** – атрибут, который идентифицирует уникальным образом единичный объект, что не влияет на генерацию кода.

Тип связи **Dependency** позволяет показать, что один класс использует объекты другого. Это может осуществляться при передаче параметров или вызове операций класса. В таком случае генератор кода Rational Rose включает заголовочный файл в класс, который использует операторы или объекты другого класса. Графическое изображение этого вида связи показано на рис. 2.16.



Рис. 2.16. Связь Dependency

Тип связи **Association class** используется для отображения свойства ассоциации. Свойства сохраняются в классе и соединяются связью **Association** (рис. 2.17). Этот тип не имеет своих спецификаций. Ассоциация предназначена для задания дополнительных атрибутов у связи. Она обозначает, что некоторый класс со своими атрибутами включается как элемент в два других, хотя при генерации кода это не отображается.

Тип связи **Generalization** позволяет указать, что один класс является родительским по отношению к другому, при этом будет создан код наследования класса. Пример такой связи показан на рис. 2.18.

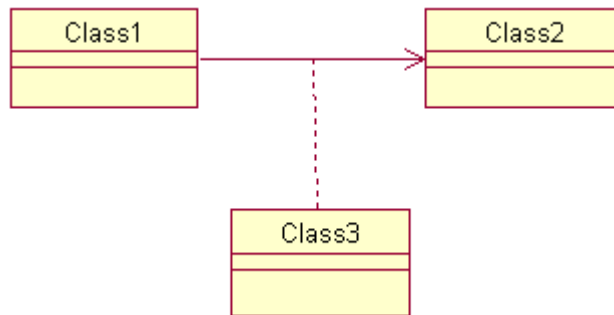


Рис. 2.17. Связь Association class



Рис. 2.18. Связь Generalization

Тип связи Realization позволяет показать, что один класс является реализацией, т.е. создан на основе шаблона другого. В Rational Rose для обозначения класса шаблона используется понятие параметризованный класс. Графическое изображение этого типа связи показано на рис. 2.19.



Рис. 2.19. Связь Realization

На практике чаще других используются два вида связей: Unidirectional association для агрегирования включения ссылок на классы и Generalization для создания иерархии наследования.

### Вопросы для повторения

1. Каково назначение диаграммы классов?
2. Как настроить свойства атрибутов класса и свойства методов класса?
3. Какие виды связей доступны в диаграмме классов?
4. Для чего используется каждый вид связи?

## Лабораторная работа №9

### СОЗДАНИЕ ПРИЛОЖЕНИЯ НА VISUAL C++

#### Цель работы:

- научиться строить приложения в среде автоматизированного синтеза Rational Rose;

- создать работающее приложение на VC++ на основе модели проекта в Rational Rose.

### **Задание:**

На основе диаграмм, построенных в среде автоматизированного синтеза Rational Rose, создать работающее приложение по заданной тематике.

### **Порядок выполнения работы**

1. Изучить теоретическую часть лабораторной работы.
2. Ответить на контрольные вопросы.
3. Построить шаблон приложения на основе библиотеки классов MFC, затем добавить функциональность в созданные классы и получить работающее программное приложение.

### **Создание кода класса на Microsoft Visual C++**

Прежде чем создавать приложение, рассмотрим процесс построения кода класса на заданном языке приложения. Наиболее удобный способ – это получить код класса на основе библиотеки классов Microsoft Foundation Class (MFC). Для этого не нужно вручную оперировать большим количеством установок, так как в пакет встроен модуль Model Assistant, который позволяет изменять все необходимые установки при помощи визуальных средств.

Для того чтобы генератор Rational Rose мог создавать на основе описанной модели программный код, для каждого класса необходимо указать язык и определить компонент, в котором класс будет храниться. В случае языка VC++ открывается доступ ко всей иерархии классов библиотеки MFC. Ассоциация класса с VC++ происходит в результате выполнения следующих действий: Menu=>Tools=>Visual C++=>Component Assigned Tools, после чего открывается окно, показанное на рис. 2.20. В его правой части выбирается класс и перетаскивается на значок VC++, а затем подтверждаются создание VC++ компонента и его связь с классом. Следующим открывается окно выбора проекта, где можно создать проект или выбрать уже существующий для помещения в него новый класс.

Model Assistant представляет собой окно (рис. 2.21), в котором создаются атрибуты и операции, а также изменяются их свойства. В окне имеется ряд полей:

- Preview показывает описание класса в текущий момент;
- Generate Code – ключ, определяющий необходимость создания для данного класса исходного текста на VC++; если ключ снят, то генерация кода не происходит и класс не показывается в списке классов для обновления кода;
- Class Type – установка типа класса: «class», «struct», «union»;
- Documentation позволяет задавать произвольные комментарии для класса.



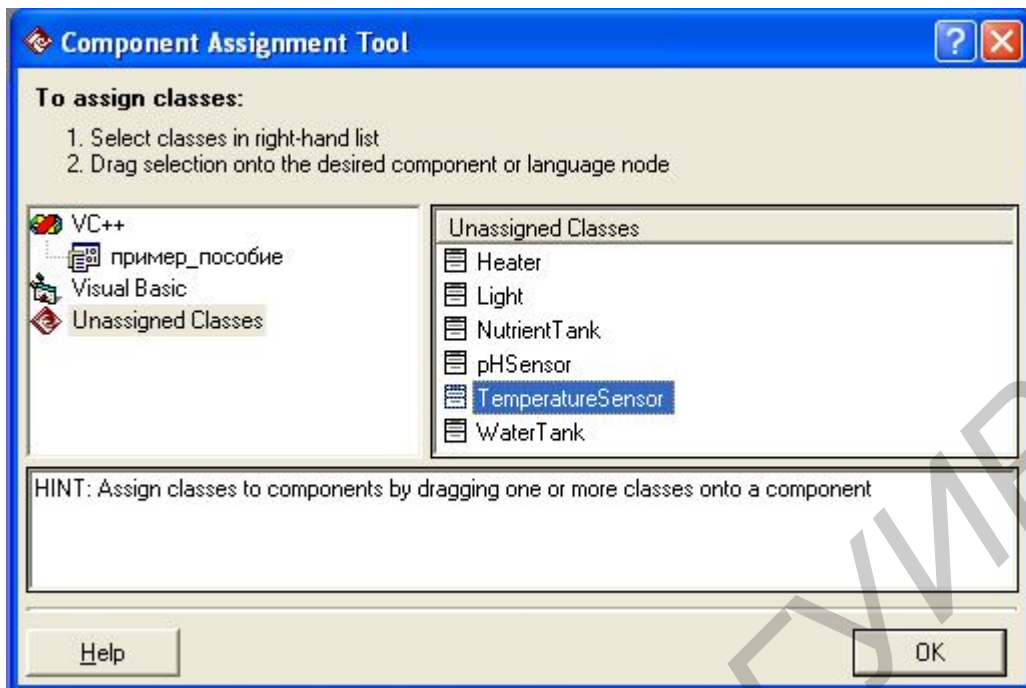


Рис. 2.20. Окно средства Component Assigned Tools

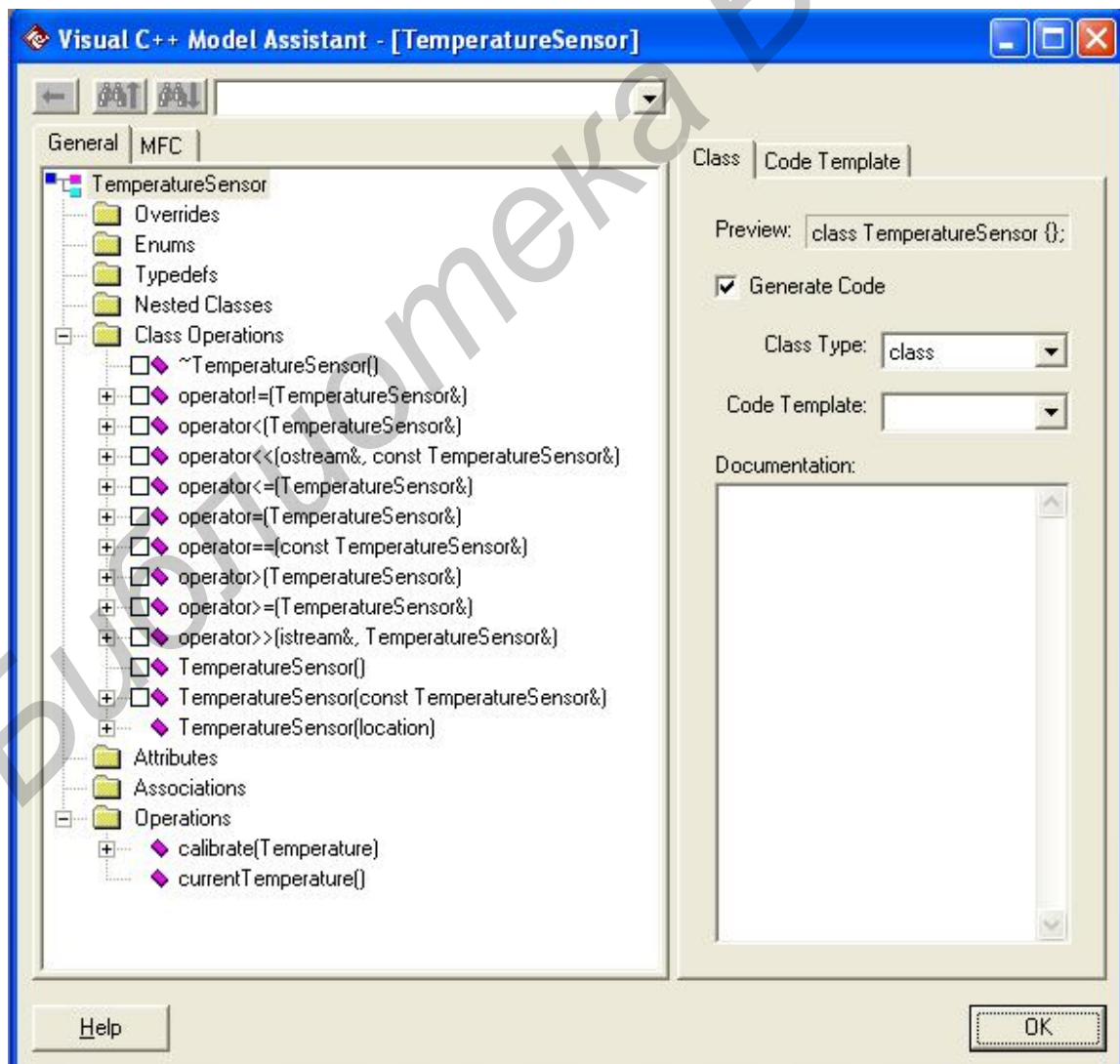


Рис. 2.21. Окно Model Assistant для класса TemperatureSensor

Rational Rose предоставляет возможности по интерактивной установке свойств методов класса. Если активизировать строку *calibrate* в окне на рис. 2.21, то откроется диалоговое окно (рис. 2.22.), предназначенное для установки и изменения атрибутов у операций. Имеются аналогичные возможности для работы со свойствами атрибутов.

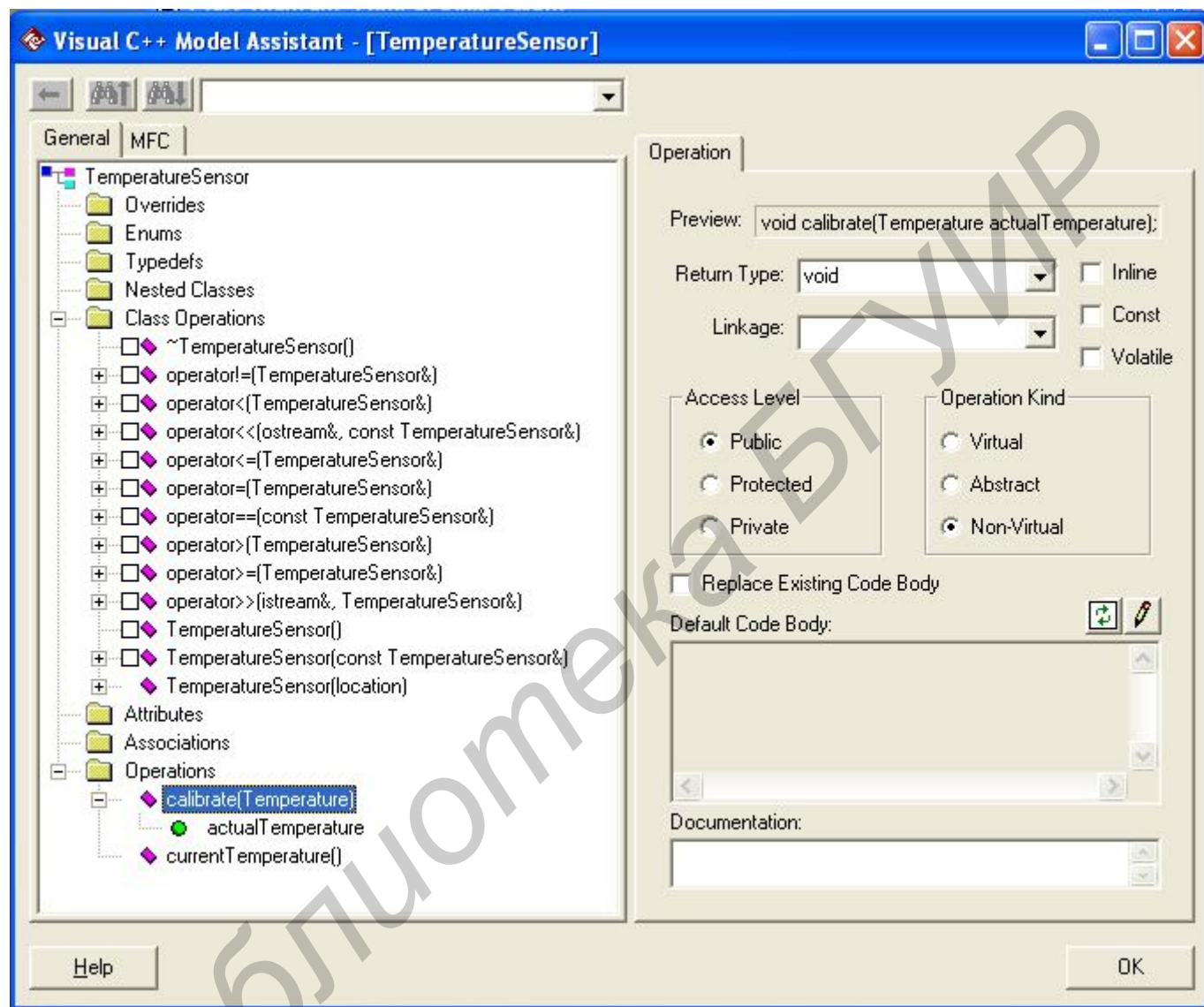


Рис. 2.22. Окно атрибутов операции *calibrate*

Для того чтобы получить преимущества использования инструмента Component Assignment Tool, необходимо создавать компоненты в окне (рис. 2.21), а не через окно Browser или в диаграмме компонентов. При этом созданные компоненты будут содержать всю необходимую информацию для генерации кода на выбранном языке программирования. Кроме того, инструмент позволяет просмотреть классы, которые еще не назначены в компоненты.

Одна из самых важных возможностей Rational Rose – это Update Code/Update Model. Данное средство позволяет создать проект Visual C++ по разработанной модели и обновить модель по уже готовому проекту, созданному при помощи

MFC. Предположим, уже создан проект, и известно, что после того как последний раз изменялась модель, исходный код определенного класса был исправлен, и его необходимо обновить. Для этого выбирается пункт Update Model, после чего появляется окно, в котором можно обновить либо все классы путем установки на них отметок, либо только выбранные, сняв с остальных отметки. Если классы модели еще не ассоциированы ни с одним проектом VC++, то это можно сделать из текущего окна. Затем Rational Rose получает информацию из проекта Visual C++, для чего загружается Microsoft Visual Studio, и активизируется нужный проект. Если в код были внесены изменения или удалены компоненты, программа предложит выполнить такие же изменения и в модели.

В случае, когда в проект Visual C++ были добавлены классы, и они еще не отражены в модели Rational Rose, необходимо провести обновление кода при помощи функции Update Code. Процесс обновления кода по изменениям в модели происходит аналогично обновлению модели. По завершении всех действий программой будет представлен отчет о том, как прошло обновление. Если все прошло нормально, то ошибок и предупреждений быть не должно.

### **Создание шаблона приложения на Microsoft Visual C++**

Будем рассматривать процесс создания приложения как логическое продолжение построения кода класса. Поэтому в нашем распоряжении находятся язык VC++ и библиотека MFC. Мастер создания приложений VC++ (AppWizard) может строить несколько типов приложений:

- Single document – приложение работает с одним документом;
- Multiple document – приложение работает с несколькими документами;
- Dialog based – приложение основано на окне диалога.

Для приложения, работающего с одним документом, мастер строит код следующих классов:

- главный класс приложения C\*\*\*App;
- класс документа C\*\*\*Doc;
- класс просмотра C\*\*\*View;
- класс для окна «О программе» CAboutDlg;
- класс основного окна программы CMainFrame.

Все приложения VC++ MFC являются объектами. Поэтому приложение – это главный класс, который включает в себя все необходимые для работы классы. Соблюдая соглашение об именах, мастер создает главный класс приложения с именем проекта, прибавляя к нему в начале букву C, а в конце App (в нашем случае это – C\*\*\*App, где \*\*\* – имя проекта). C\*\*\*App наследуется из библиотечного класса CWinApp. Класс документа C\*\*\*Doc, в котором должна проходить обработка данных, наследуется из библиотечного класса CDocument. Класс просмотра C\*\*\*View, отображающий данные на экране компьютера, наследуется из библиотечного класса CEditView. Данные отображаются в окне класса CMainFrame, наследуемого из библиотечного класса CFrameWnd.

Таким образом, на основе стандартных классов документа, предоставляемых MFC, строится приложение, в котором необходимо будет только добавить функциональность, а за отображение документа на экране отвечает библиотека.

Ассоциация проекта Rational Rose с проектом VC++ выполняется аналогично тому, как это делалось для одного класса. С помощью Component Assignment Tool в строку VC++ перетаскиваются все необходимые классы. Для простоты работы все классы, для которых необходимо создание исходного кода, помещаются в один компонент, заключенный в проект \*\*\*. Библиотека классов MFC импортируется в модель путем следующих действий: Menu:Tools=>Visual C++=>Quick Import MFC 6.0. Затем в модель Rational Rose можно загрузить перечисленные ранее классы, созданные в VC++. Для того чтобы они появились в проекте, необходимо обновить проект по готовому коду, выполнив действия: Menu:Tools => Visual C++=> Update Model From Code. После обновления в модели Rational Rose и в проекте VC++ содержатся одинаковые наборы классов. Теперь можно запустить Visual Studio, перейти в нужный проект, откомпилировать его и получить результат работы программы.

### Вопросы для повторения

1. Как создать исходный код VC++ по диаграмме классов?
2. Какова структура создаваемого кода?
3. Что такое Model Assistant?
4. Какова структура MFC-приложения?
5. Какие действия нужно предпринять для обновления модели по исходному коду?

## ЗАКЛЮЧЕНИЕ

В соединении с другими программными пакетами Rational Rose приобретает дополнительные возможности. В сочетании со средствами документирования Rational SoDA он может давать полное представление о проекте. Полностью интегрируясь с Microsoft Visual Studio, этот пакет позволяет получать исходный код взаимодействующих классов и строить визуальные модели по уже написанному исходному коду. Возможность интеграции со средствами управления требованиями Requisite Pro, со средствами тестирования SQA Suite, Performance Studio, со средствами конфигурационного управления ClearCase, PVCS выводит процесс разработки программного проекта на качественно новый уровень [6].

Таким образом, можно выделить следующие преимущества от применения Rational Rose:

- сокращение цикла разработки приложения «заказчик-программист-заказчик»;
- увеличение продуктивности работы программиста;
- улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;
- способность вести большие проекты и группы проектов;

- возможность повторного использования уже созданного программного обеспечения за счет упора на разбор их архитектуры и компонентов.

## ЛИТЕРАТУРА

1. Липаев В.В. Проектирование программных средств. – М.: Высш. шк., 1990.
2. Евгеньев Г.Б. Системология инженерных знаний. – М., 2001.
3. Вендров А.М. Проектирование ПО экономических информационных систем. – М., 2000.
4. Трофимов С.А. CASE-технологии: практическая работа в Rational Rose. – М.: Бинوم-Пресс, 2002.
5. Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML. – М.: Изд. Дом «Вильямс», 2003.
6. Крачтен Ф. Введение в Rational Unified Process. – М.: Изд. Дом «Вильямс», 2002.

Библиотека БГУИР

Учебное издание

**Бочкарёва** Лия Валентиновна  
**Кирейцев** Максим Валерьевич

СИСТЕМЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ  
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.  
РАБОТА В СРЕДЕ *RATIONAL ROSE*

Учебно-методическое пособие  
для студентов специальности  
«Программное обеспечение информационных технологий»  
всех форм обучения

Редактор Т. Н. Крюкова  
Корректор Е. Н. Батурчик

---

Подписано в печать 19.09.2006.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Печать ризографическая.	Усл. печ. л. 2,44.
Уч.-изд. л. 2,0.	Тираж 100 экз.	Заказ 506.

---

Издатель и полиграфическое исполнение: Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.  
220013, Минск, П. Бровки, 6