



## Глава 1. Что такое визуальное моделирование

*Визуальным моделированием* (visual modeling) называется способ представления идей и проблем реального мира с помощью моделей. Модель помогает понять проблему всем участникам, задействованным в реализации проекта на различных этапах: заказчику, эксперту, аналитику, проектировщику, автору документации, программисту и др. Моделирование обеспечивает более точную оценку необходимых ресурсов, четкую проработку планов и эффективное функционирование создаваемых систем.

*Модель* (model) – это абстракция, описывающая суть сложной проблемы или структуры без акцента на несущественных деталях, тем самым делая ее более понятной. Абстрагирование – одна из основных способностей человека, которая позволяет разбираться в сложных вещах. Инженеры, артисты, ремесленники используют модели на протяжении тысячи лет, чтобы сначала проверить изделие или замысел, а потом приступить к его реализации. Разработка программного обеспечения – не исключение. Для построения сложной системы необходимо сначала разделить ее на несколько абстрактных представлений и построить модели, используя принятые обозначения – *нотацию* (notation). Затем убедиться, что модели удовлетворяют всем потребностям системы, и постепенно добавлять детали для перехода от моделей к реализации.

Мы строим модель сложной системы, потому что не можем охватить и понять проект целиком. Существуют пределы в понимании сложных вещей. Это можно продемонстрировать на примере архитектуры. Если вы хотите построить сарай во дворе, вам достаточно просто начать строительство. Когда вы планируете построить новый дом, вам наверняка потребуется чертеж. А для возведения небоскреба он будет просто необходим. Этот же пример можно привести для программного обеспечения. Изучая работу отдельной формы в Visual Basic, программист не сумеет представить схему проекта целиком. Создание модели позволяет представить общую картину взаимодействия узлов системы без углубления в детали реализации отдельных элементов.

Модели помогают нам организовывать, отображать, понимать и создавать сложные вещи. Они призваны помочь в решении трудных задач при разработке программ сегодня и в будущем.

### Треугольник успеха

Я часто использую *треугольник успеха* (triangle for success), показанный на рис. 1.1, чтобы изобразить средства, необходимые для успешного проекта. Вам

потребуются все три грани – три составляющие – нотация, процесс и инструмент. Можно изучить нотацию, но если вы не знаете, как ее применить (организовать процесс), то, вероятно, потерпите неудачу. У вас могут быть хорошо организованные процессы, но если вы не сумеете описать порядок их взаимодействия (используя нотацию), то, скорее всего, вам не удастся довести проект до конца. И наконец, если вы неправильно определите орудия труда (инструменты), вас наверняка постигнет неудача.

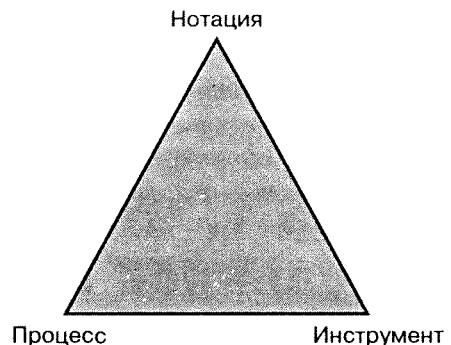


Рис. 1.1. Треугольник успеха

## Роль нотации

Нотация является важной составляющей любой модели – она служит связующим звеном между процессами. «Нотация выполняет три функции:

- является языком для описания взаимодействий, которые неочевидны или не могут быть получены непосредственно из кода;
- обеспечивает достаточную семантику, позволяющую охватить важные стратегические и тактические решения;
- предлагает конкретную форму, помогающую человеку рассуждать о предметной области, а средствам моделирования воплощать описанные идеи<sup>1</sup>.

Унифицированный язык моделирования (Unified Modeling Language – UML) предлагает достаточно полную нотацию, которая расширяется при переходе от анализа к проектированию. Определенные элементы нотации (например, классы, связи, агрегаты, наследование) используются на этапе анализа. Другие элементы (индикаторы реализации и свойства) вводятся на стадии проектирования.

## История UML

В 90-е годы появилось большое количество различных методологий с собственными наборами нотаций. Самые популярные – ОМТ (по Рамбо), Booch (по Бучу) и OOSE (по Джекобсону). Каждая из них имела свои преимущества. Методика ОМТ отличалась хорошими средствами анализа и слабыми сторонами в проектировании, а методика Booch 1991, наоборот, более подходила для проектирования, чем для анализа. В методике OOSE основное внимание уделено развитым средствам поведенческого анализа, а в других областях отмечено много недостатков.

Спустя некоторое время Буч опубликовал второе издание, в котором собрал лучшие идеи и решения в области анализа, предлагавшиеся в том числе Рамбо и Джекобсоном. В свою очередь, Рамбо написал серию статей, известных как

<sup>1</sup>Booch, Grady. *Object Solutions*. Redwood City, CA: Addison-Wesley, 1995.

методика ОМТ-2, куда вошли предложения Буча в области проектирования. Перечисленные методики были достаточно похожи, но отличались разными нотациями – один и тот же символ имел в них различные значения. Например, закрашенный круг был индикатором множественности в методике ОМТ и символом агрегата в нотации Буча. Вы, наверное, слышали фразу «война методов», употреблявшуюся в период, когда класс обозначался либо в виде облака, либо в виде прямоугольника? Трудно понять, что же лучше.

Конец войне методов положила нотация, принятая в языке UML. «Язык UML служит для определения, отображения и описания элементов объектно-ориентированных систем в процессе их создания. Он объединяет объектную модель, нотации Буча и ОМТ, а также лучшие идеи, предложенные авторами других методик (рис. 1.2). Таким образом, язык UML является стандартом де-факто в области объектно-ориентированного анализа и проектирования»<sup>1</sup>.

Универсальный язык UML – это попытка стандартизировать инструменты анализа и проектирования семантических моделей, синтаксических нотаций и диаграмм. Первая общедоступная версия (0.8) появилась в октябре 1995 года. Джекобсон и другие разработчики предложили несколько вариантов, которые были реализованы в последующих двух версиях (0.9 – в июле и 0.91 – в октябре 1996 года). Версия 1.0 была представлена для стандартизации в ассоциацию Object Management Group (OMG) в июле 1997 года. Дополнительные улучшения сделаны в версии 1.1, которая вышла в сентябре того же года, а в ноябре UML был утвержден ассоциацией OMG в качестве стандартного языка моделирования.

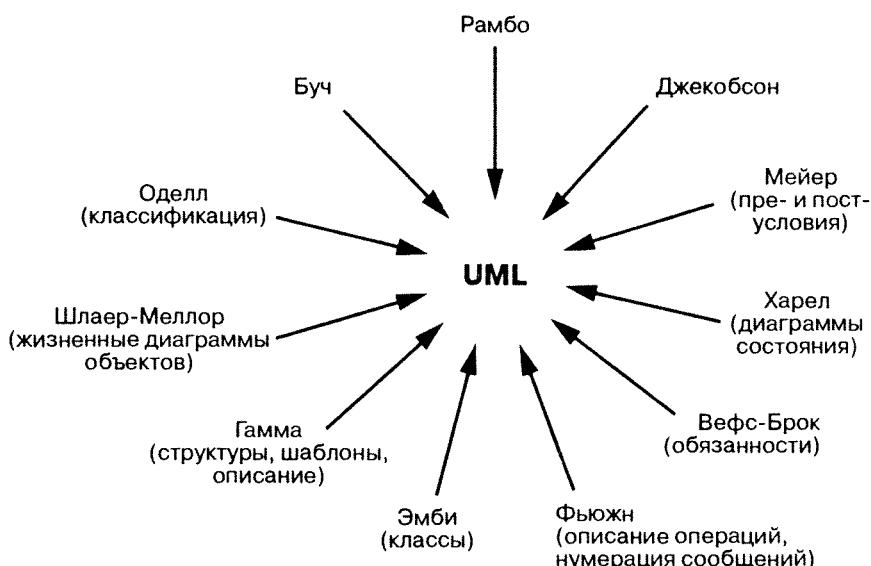


Рис. 1.2. Составные части языка UML

<sup>1</sup> *The Unified Method*, Draft Edition (0.8). Rational Software Corporation, October, 1995.

## Роль процессов

Успешно разработанный проект удовлетворяет или превосходит ожидание заказчика, выполняется в срок с оптимальными затратами и может быть адаптирован к изменению условий. Жизненный цикл разработки должен способствовать творческим и новаторским идеям. В то же время для своевременного завершения процесс разработки должен контролироваться. «Творчество естественно для создания всех хорошо структурированных объектно-ориентированных архитектур, но если разработчиков не контролировать, то они, возможно, никогда не достигнут конечного результата. Значит, для эффективной работы коллектива нужна дисциплина. Но слишком жесткая дисциплина приводит к развитию бюрократии, которая, в свою очередь, душит новаторские идеи»<sup>1</sup>. Правильно управляемый итеративный и инкрементальный жизненный цикл обеспечивает необходимый контроль и поддерживает творческий процесс на нужном уровне.

## Что такое итеративная и инкрементальная разработка

В итеративном и инкрементальном жизненном цикле (см. рис. 1.3) разработка осуществляется с помощью серии версий, которые развиваются в направлении конечной системы. Каждая версия состоит из одного или более компонентов процесса: построение бизнес-модели, определение требований к системе, анализ, проектирование, реализация, тестирование и внедрение. Разработчики допускают, что не все требования к системе известны в начале жизненного цикла. Корректировки возможны на любом этапе.

Такой тип жизненного цикла позволяет уменьшить риск. Технические риски оцениваются и группируются по приоритетам на ранней стадии цикла и корректируются

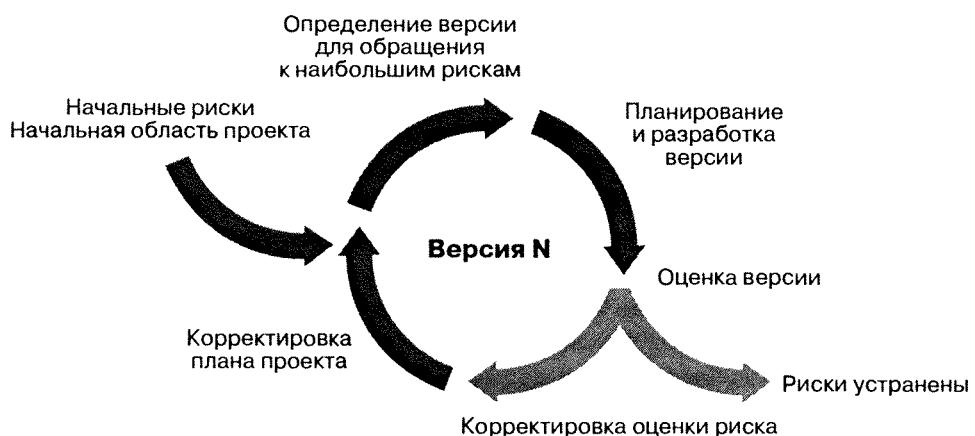


Рис. 1.3. Итеративная и инкрементальная разработка

<sup>1</sup> Booch, Grady. *Object Solutions*. Redwood City, CA: Addison-Wesley, 1995.

во время разработки каждой версии. Риски закреплены за каждой версией таким образом, что успешное завершение версии уменьшает риск, закрепленный за ней. Процесс планируется так, чтобы наибольшие риски были рассмотрены в первую очередь. Построение системы подобным образом выявляет и уменьшает риски на раннем этапе жизненного цикла. Результат такой модели жизненного цикла – уменьшение риска и затрат<sup>1</sup>.

## Методология Rational Unified Process

Для поддержки управления итеративным и инкрементальным жизненным циклом разработки используется методика Rational Unified Process, с помощью которой можно подробно описать технические и организационные аспекты создания программного обеспечения на стадиях определения требований к системе, анализа и проектирования.

Методология Rational Unified Process структурирована в двух направлениях:

- время (разделение жизненного цикла на фазы и версии);
- компоненты процесса (создание необходимого набора средств для выполнения четко определенных задач).

Оба направления должны быть хорошо проработаны для получения успешного проекта.

Работа над проектом состоит из следующих временных этапов:

- задумка* (inception) – определение общей идеи проекта;
- проработка* (elaboration) – планирование необходимых работ и ресурсов, указание особенностей и создание архитектуры;
- создание* (construction) – построение продукта при помощи серии последовательных версий;
- переходный период* (transition) – поставка продукта пользователям (производство, распространение, обучение).

В разрезе компонентов процесс делится на следующие стадии:

- построение бизнес-модели* (business modeling) – определение необходимых возможностей системы и потребностей пользователей;
- определение требований к системе* (requirements) – изложение общей идеи системы совместно с функциональными и нефункциональными условиями ее работы;
- анализ* (analysis) и *проектирование* (design) – описание способов исполнения системы на этапе реализации;
- реализация* (implementation) – кодирование и генерация работающих программных модулей системы;

<sup>1</sup> Дополнительную информацию об итеративном и инкрементальном процессе разработки программного обеспечения можно найти в статье «A Rational Development Process» by Philippe Kruchten, *CossTalk*, 9(7), July 1996, p.p. 11–16. Эта статья также доступна на сайте компании Rational по адресу: <http://www.rational.com>.

- тестирование (test)* – проверка функционирования системы;
- внедрение (deployment)* – поставка системы конечным пользователям и их обучение.

Каждая стадия в разрезе компонентов процесса обычно применяется к конкретной фазе временного направления (см. рис. 1.4.). Однако степень применения каждого компонента зависит от этапа разработки. Например, вы можете испытать концептуальный прототип системы на стадии задумки, и тогда вам потребуется не только определение требований – необходимо будет провести анализ, проектирование, реализацию и тестирование, чтобы завершить создание прототипа. Важность анализа выявляется на этапе проработки.

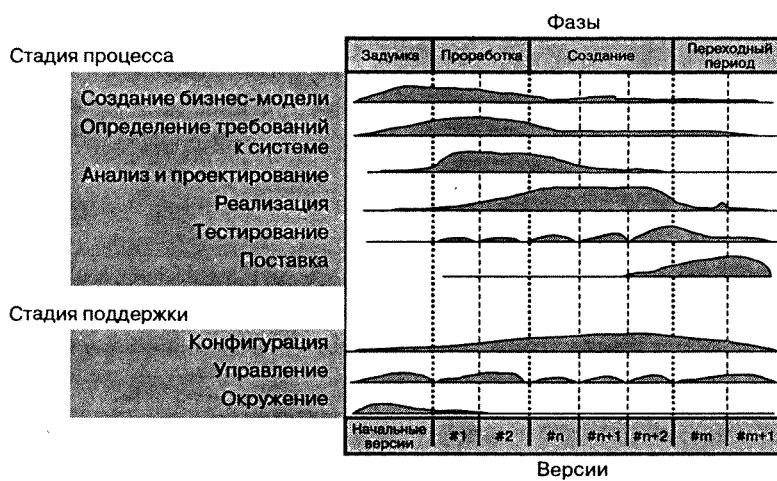


Рис. 1.4. Стадии разработки

Сначала предпочтительно выпустить несколько версий. Они обычно используются для проверки аналитических решений в области архитектуры системы. Таким образом, вы не только анализируете проблему. На стадии создания система строится с помощью серии версий. При любой структуре разработки дополнительные вопросы всегда возникают неожиданно, что требует проведения нового анализа.

Диаграмма должна быть основным руководством для отражения жизненного цикла вашего проекта. Основная мысль заключается в следующем: если вы только думаете над тем, что собираетесь создавать, в то время когда уже пишете код, вероятно, у вас возникнут проблемы. Заметьте, что тестирование применяется в ходе всего итерационного процесса. То есть вы не ждете, когда весь код будет написан для проверки его работы.

В этой книге применяется упрощенная версия Rational Unified Process, в которой сделан акцент на использовании языка UML для получения и документального описания решений на стадиях задумки и проработки проекта. В последних главах кратко рассказывается об этапе создания. Несмотря на то что тестирование – неотъемлемая часть процесса разработки, его описание выходит за рамки данной книги.

## Пакет Rational Rose

Методы создания программного обеспечения должны поддерживаться соответствующими инструментами разработки. Когда я впервые начала заниматься объектно-ориентированным моделированием, моими инструментами были бумага и карандаш. Теперь в продаже имеются более удобные программные инструменты – не только простые графические редакторы, но и сложные пакеты объектного моделирования. В этой книге для моделирования используется программа Rational Rose. На каждом этапе моделирования приводится описание необходимых действий.

Семейство продуктов Rational Rose призвано обеспечить разработчика программ полным набором инструментов визуального моделирования для эффективного решения сложных бизнес-задач с использованием архитектуры клиент/сервер, распределенных сред и систем реального времени. В продуктах Rational Rose отражен универсальный стандартизованный подход к построению моделей, позволяющий программистам моделировать логику приложений, а не программистам – бизнес-процессы. Демонстрационную версию пакета Rational Rose можно получить на сайте компании Rational Software Corporation по адресу: [www.rational.com](http://www.rational.com).

Хотя в этой книге в качестве инструмента моделирования используется Rational Rose, ряд диаграмм можно получить также средствами программы Microsoft Visual Modeler. Этот продукт предназначен специально для начинающих разработчиков и предоставляет следующие возможности:

- описывать и проектировать бизнес-объекты с последующим отображением их на программные компоненты;
- разделять сервисы в трехзвенной сервисной модели;
- распределять компоненты в сети;
- генерировать исходные модули на Visual Basic по созданной модели;
- использовать возвратное проектирование для создания модели по существующим компонентам и приложениям;
- синхронизировать модели с кодом.

Пакет Rational Rose по сравнению с Microsoft Visual Modeler позволяет анализировать требования к бизнес-системе и бизнес-сценарии с помощью диаграмм последовательности действий и диаграмм взаимодействий, моделировать состояния, генерировать код и поддерживать встроенный скрипт для доступа к внутренним компонентам программы.

## Резюме

Визуальное моделирование – это способ представления идей и проблем реального мира с помощью моделей. Модель помогает понять проблему всем участникам, задействованным в реализации проекта на различных этапах: заказчику, эксперту, аналитику, проектировщику, автору документации, программисту и др. Моделирование обеспечивает более точную оценку необходимых ресурсов, четкую проработку планов и эффективное функционирование создаваемых систем.

Нотация – важная составляющая любой модели, своего рода связующее звено между процессами. Унифицированный язык моделирования (UML) предлагает достаточно полную нотацию, которая расширяется при переходе от анализа к проектированию.

Успешно разработанный проект удовлетворяет или превосходит ожидание заказчика, выполняется в срок с оптимальными затратами и может быть адаптирован к изменению условий. Жизненный цикл разработки должен способствовать творческим и новаторским идеям. Правильно управляемый итеративный и инкрементальный жизненный цикл обеспечивает необходимый контроль и поддерживает творческий процесс на нужном уровне. В итеративном и инкрементальном жизненном цикле разработка осуществляется с помощью серии версий, которые развиваются в направлении конечной системы. Каждая версия состоит из одного или более компонентов процесса: построение бизнес-модели, определение требований к системе, анализ, проектирование, реализация, тестирование и внедрение.

В качестве средства управления итеративным и инкрементальным жизненным циклом разработки применяется методика Rational Unified Process, с помощью которой можно подробно описать технические и организационные аспекты разработки программного обеспечения на стадиях определения требований к системе, анализа и проектирования. В этой книге используется упрощенная версия Rational Unified Process.

Семейство продуктов Rational Rose призвано обеспечить разработчика программ полным набором инструментов визуального моделирования для эффективного решения сложных бизнес-задач с использованием архитектуры клиент/сервер, распределенных сред и систем реального времени.



## **Глава 2. Начало проекта**

### **Определение правильного проекта**

Главный вопрос при разработке системы не касается методологии. Это и не проблема технической реализации. Это с виду простой, но на самом деле достаточно сложный и важный вопрос: «Правильна ли создаваемая система?». К сожалению, он в большинстве случаев вообще не возникает или остается без ответа. Хотя неверная методология или технические проблемы могут привести к неудаче, иногда избыток ресурсов и титанические усилия талантливых людей спасают проект. Но ничто не поможет системе, которая не нужна или автоматизирует неправильные вещи.

Для начала проекта необходима идея. Зарождение идеи и определение общих требований и форм происходит на этапе задумки. Он заканчивается утверждением: «Наша система делает...». В процессе проработки идея приобретает ясные очертания, а предположения утверждаются или отвергаются. На этом этапе собираются и документируются основные идеи, предварительно описываются риски, внешние интерфейсы, общая функциональность системы и возможно создание тестовых прототипов для проверки общей концепции (proof of concept prototypes). Идеи поступают из разных источников (это могут быть заказчики, эксперты по предметной области, сами разработчики, отраслевые эксперты, результаты тестов, изучение существующих систем). Важно отметить, что любые прототипы, созданные в этом периоде, должны рассматриваться как временный код, предназначенный лишь для проверки предположений и не прошедший через стадии анализа и проектирования.

На этом этапе разработки процессы могут быть оформлены формально или проведены неформально, но они всегда включают анализ бизнес-требований, доступных ресурсов, возможностей использования различных технологий, а также идей и пожеланий конечных пользователей. Затем для выработки целевой концепции системы, описания задач и приоритетов могут быть использованы такие средства, как профессиональные и научные исследования, мозговой штурм, анализ эффективности, анализ функциональности и создание прототипов. Обычно в этот период происходят первичные сокращения запланированных ресурсов и времени. Для одних проектов концепцию можно изложить на обороте салфетки, для других описание идеи является формальным этапом, выполняемым при помощи итеративного процесса до достижения необходимого уровня точности и детализации.

Тщательно проработанная задумка помогает правильно оценить потребности и ресурсы для построения эффективной системы. Неправильно выполненный

этап задумки может привести к тому, что система станет ненужной, неосуществимой, слишком дорогой или никогда не будет доведена до конца.

## Несколько слов об университете ESU

Описание регистрации учебных курсов для университета Истерн (Eastern State University – ESU) будет использоваться в качестве основного примера книги.

После того как преподаватели ESU решат, какие курсы они будут вести в течение семестра, служба регистрации курсов внесет информацию в компьютерную систему. Затем для преподавателей распечатают сводный отчет по курсам, которые они будут читать, а для студентов – каталог курсов.

На этом этапе студенты заполняют специальную регистрационную форму, где указывают выбранные курсы, и отдают ее в службу регистрации. Обычно студент подписывается на четыре курса, после чего информация заносится в компьютер. Далее запускается ночная пакетная программа, которая распределяет студентов по курсам. При возникновении конфликтной ситуации служба регистрации уточняет студенческие данные. После успешного распределения студенту высыпается расписание для проверки. Обычно процесс регистрации на курсы занимает около недели, но в ряде случаев может потребоваться до двух недель, чтобы уладить все вопросы. Затем преподаватели получают список студентов для каждого курса, который они будут читать.

## Риски задачи регистрации курсов

Группа разработчиков определила, что главный риск системы связан с возможностью эффективно сохранять и получать информацию об учебных планах. С этой целью было создано несколько прототипов, чтобы оценить механизмы хранения и доступа к информации для каждой рассматриваемой системы управления базами данных. Результаты испытания прототипов показали, что риск неэффективной работы базы данных может быть уменьшен. Дополнительные прототипы были использованы для оценки аппаратных ресурсов, необходимых при создании онлайновой системы регистрации.

## Постановка задачи регистрации курсов

В начале каждого семестра студенты могут запросить каталог курсов, в который включен список учебных предметов, предлагаемых в данном семестре. Информация о курсах должна содержать фамилию преподавателя, название факультета и краткое описание, помогающее студентам сделать выбор.

Новая система позволит студенту выбрать четыре курса из предложенных в наступающем семестре. Кроме того, каждому студенту нужно дополнительно указать еще два варианта, на случай если курс будет переполнен или отменен. На курс не должно быть записано более десяти или менее трех студентов. Курс, на который запишутся менее трех студентов, будет отменен. По завершении регистрации система регистрации направляет информацию в систему оплаты для выставления счетов студентам.



Преподаватели должны иметь возможность онлайнового доступа к системе для указания курсов, которые они будут читать, и для просмотра списка записавшихся студентов.

В каждом семестре выделяется определенное время, в течение которого студенты могут менять свое расписание и получать доступ к системе для добавления или удаления выбранных курсов.

## **Резюме**

Фаза задумки – это этап открытия. Задача оговаривается и обсуждается в группе разработчиков с привлечением заказчиков. Все высказанные предположения и допущения могут быть утверждены или отклонены после апробации на прототипах. Результаты этой фазы – описание внешних интерфейсов, оценка начальных рисков, определение требований к системе. Заказчики, клиенты, пользователи и другие заинтересованные стороны обсуждают различные идеи и точки зрения и предлагают возможные пути получения необходимых ресурсов и инструментов.



## Глава 3. Создание прецедентов

### Поведение системы

Поведение разрабатываемой системы (то есть функциональность, обеспечивающая системой) описывается с помощью функциональной модели, которая отображает системные прецеденты (use cases), системное окружение (действующих лиц или актеров – actors) и связи между прецедентами и актерами (диаграммы прецедентов – use cases diagrams). Основная задача модели прецедентов – представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы.

Разработка модели прецедентов начинается на стадии задумки с выбора актеров и определения общих принципов функционирования системы. Затем на этапе проработки модель дополняется детальной информацией к существующим прецедентам, а при необходимости добавляются новые.

### Актеры

*Актеры* не являются частью системы – они представляют собой кого-то или что-то, что должно взаимодействовать с системой. Актеры могут:

- только снабжать информацией систему;
- только получать информацию из системы;
- снабжать информацией и получать информацию из системы.

Обычно актеры определяются из описания задачи или путем переговоров с заказчиками и экспертами. Для выявления актеров может быть использована следующая группа вопросов:

1. Кто заинтересован в определенном системном требовании?
2. Какую роль система будет выполнять в организации?
3. Кто получит преимущества от использования системы?
4. Кто будет снабжать систему информацией, использовать информацию и получать информацию от системы?
5. Кто будет осуществлять поддержку и обслуживание системы?
6. Использует ли система внешние ресурсы?
7. Выступает ли какой-либо участник системы в нескольких ролях?
8. Выступают ли различные участники в одной роли?
9. Будет ли новая система взаимодействовать со старой?

В языке UML актер изображается в виде фигуры человечка – см. рис. 3.1.

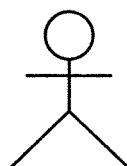


Рис. 3.1. Нотация языка UML  
для изображения актера

### Определение «хорошего» актера

Необходимо внимательно подходить к вопросу определения актеров для системы. Такое определение обычно происходит итеративным образом – первый из утвержденных списков актеров часто далек от конечного. Например, является ли новый студент актером другого вида нежели студент, вернувшийся из академического отпуска? Допустим, вы сначала ответили утвердительно. Следующий шаг – выяснить, как актер взаимодействует с системой. Если новый студент использует систему не так, как вернувшийся студент, то это разные актеры. Если они используют систему одинаковым образом – это один и тот же актер.

Другой вариант – создание актера для каждой роли, исполняемой участником. Здесь тоже можно получить избыточность. Хороший пример – ассистенты преподавателей в системе регистрации курсов университета. Ассистенты тоже проводят занятия с группами студентов. Средства, требующиеся для выбора курсов, уже заложены в языке для актеров в роли студентов и актеров в роли преподавателей. Таким образом, нет необходимости в актерах, исполняющих роль ассистентов преподавателей.

Отобрав утвержденных актеров и описания их ролей в использовании системы, вы итеративным путем получите нужный набор актеров для системы.

### Актеры в системе регистрации курсов университета

Перечислим ответы на ранее поставленные вопросы:

1. Студент хочет зарегистрироваться на курсы.
2. Преподаватель хочет выбрать курсы, которые он будет читать.
3. Регистратор должен создать учебный план и составить каталог на семестр.
4. Регистратор должен хранить информацию о курсах, преподавателях и студентах.
5. Система оплаты должна получать необходимую информацию из системы регистрации.

Основываясь на полученных ответах, можно выделить следующих актеров: студент (Student), преподаватель (Professor), регистратор (Register) и система оплаты (Billing system).

Алгоритм создания актеров в программе Rational Rose:

1. Щелкните правой кнопкой мыши по разделу **Use Case View** (Представление прецедентов) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Actor** (Создать ⇒ Актер). В список окна браузера будет добавлен новый актер с именем **New Class**.
3. Выбрав новый пункт списка, введите нужное имя актера.

Окно браузера со списком актеров для системы регистрации курсов показано на рис. 3.2.

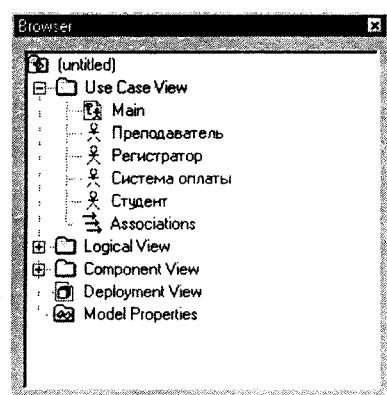


Рис. 3.2. Актеры

### Описание актеров

В модель желательно включить краткое описание каждого актера, в котором нужно указать роль актера при взаимодействии с системой.

Для системы регистрации курсов описание актеров может быть следующим:

- студент – человек, который регистрируется для посещения занятий в университете;
- преподаватель – человек, который читает лекции в университете;
- регистратор – человек, управляющий системой регистрации курсов;
- система оплаты – внешняя система, выполняющая функции расчетов за курсы.

Описание актеров в программе Rational Rose осуществляется при выполнении следующих действий:

1. Если окна описания нет на экране, откройте его, выбрав команду меню **View ⇒ Documentation** (Вид ⇒ Описание).
2. Из списка браузера выберите актера, щелкнув по нему мышью.
3. Установите курсор в окне описания и введите текст описания актера.

Пример описания актера студент показан на рис. 3.3.

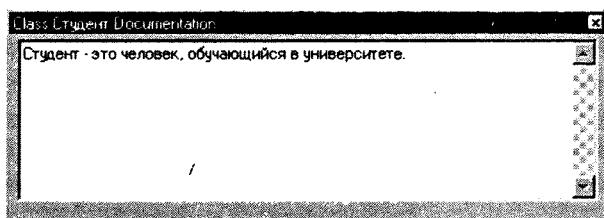


Рис. 3.3. Описание актера студент

## Прецеденты

С помощью *прецедентов* (use cases) моделируется диалог между актером и системой. Другими словами, они определяют возможности, обеспечиваемые системой для актера. Набор всех прецедентов системы определяет способы ее использования. Можно сказать, что прецедент – это последовательность транзакций, выполняемых системой, которая приводит к значимому результату для определенного актера.

Чтобы выделить прецеденты для системы, можно использовать следующую серию вопросов:

1. Каковы задачи каждого актера?
2. Будет ли актер создавать, хранить, изменять, удалять или получать информацию из системы?
3. Какой прецедент будет создавать, хранить, изменять, удалять или получать эту информацию?
4. Должен ли актер информировать систему о внезапных изменениях внешней среды?

5. Должен ли актер быть проинформирован об изменениях состояния системы?
6. Какие прецеденты будут поддерживать и обслуживать систему?
7. Могут ли все функциональные требования быть реализованы прецедентами?



Рис. 3.4. Нотация языка UML для прецедента

В языке UML прецедент изображается в виде овала – см. рис. 3.4.

### **Основы правильного прецедента**

На протяжении многих лет велись дискуссии на тему правильности прецедентов. Одной из проблем, с которой я столкнулась, является уровень их детализации. Насколько мал или велик он должен быть? Здесь нет однозначного ответа. Я обычно использую следующее правило: «Прецедент обычно определяет основной элемент функциональности и совершается от начала до конца. Он должен приносить что-то значимое для актера».

Например: в системе регистрации учебных курсов студент должен выбрать курсы для наступающего семестра; студент должен быть прикреплен к предлагаемым курсам; студенту должен быть выставлен счет. Это три прецедента или только один? Я бы сделала один, потому что функциональность действия определяет происходящее от начала до конца. Что бы получилось, если студента не прикрепили бы к выбранным курсам (или, по крайней мере, не известили об этом)? Или что произошло бы, если студент не получил бы счет (университет наверняка бы разорился, если бы курсы стали бесплатными)?

Другая проблема в том, как объединить функциональность различных действий, которые кажутся едиными. Например, регистратор должен добавлять курсы, удалять и изменять их. Три прецедента или один? Здесь я опять сделала бы один – работа с учебным планом, потому что действия инициируется одним актером (регистратором) и выполняются над одной сущностью системы (расписанием).

### **Прецеденты в системе регистрации курсов университета**

В системе должны обеспечиваться следующие потребности:

- актер студент использует систему для регистрации на курсы;
- по завершении выбора курсов в систему оплаты должна поступить необходимая информация;
- актер преподаватель использует систему для выбора курсов, которые он будет читать в наступающем семестре, и должен получать от системы расписание занятий;
- регистратор отвечает за составление каталога курсов на семестр, за управление информацией об учебных курсах, а также о студентах и преподавателях, работающих с системой.

На основании перечисленных потребностей можно выделить следующие прецеденты:

- регистрация на курсы;
- выбор курсов для преподавания;

- запрос расписания курсов;
- управление информацией о курсах;
- управление информацией о преподавателях;
- управление информацией о студентах;
- создание каталога курсов.

Для создания прецедентов в программе Rational Rose выполните следующие действия:

1. Щелкните правой кнопкой мыши по разделу **Use Case View** (Представление прецедентов) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Use Case** (Создать ⇒ Прецедент). В списке браузера появится новый прецедент.
3. Введите для него нужное название.

Окно браузера со списком прецедентов для системы регистрации курсов показано на рис. 3.5.

### Краткое описание прецедентов

В краткое описание прецедентов вносят информацию об их назначении. Такое описание обычно определяется на этапе задумки при выделении прецедентов для системы.

Для прецедента регистрация на курсах оно будет выглядеть так, как на рис. 3.6.

Этот прецедент инициируется студентом. Он обеспечивает возможность создавать, изменять, удалять и просматривать расписание студента в определенном семестре.

Для добавления краткого описания прецедента в программе Rational Rose:

1. В списке браузера выберите прецедент, щелкнув по нему мышью.
2. Установите курсор в окне описания и наберите краткое описание прецедента. Если окно невидимо, откройте его с помощью команды меню **View ⇒ Documentation** (Вид ⇒ Описание).

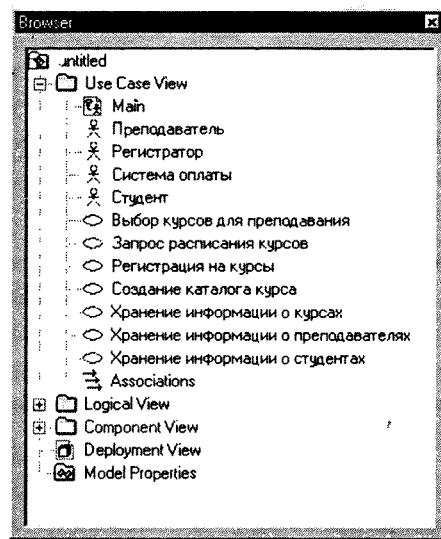


Рис. 3.5. Прецеденты

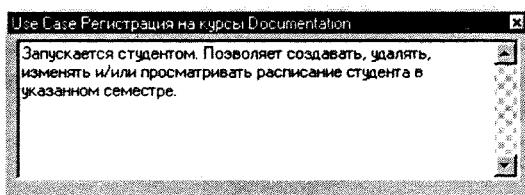


Рис. 3.6. Краткое описание прецедента

## Поток событий для прецедента

Поток событий (flow of events) для прецедента – это последовательность событий, необходимых для обеспечения требуемого поведения. Поток событий описывается в терминах того, «что» система должна делать, а не «как» она должна это делать. То есть он описывается на языке предметной области, а не терминами реализации. Поток событий должен определять:

- когда и как прецедент начинается и заканчивается;
- как он взаимодействует с актером;
- какие данные ему нужны;
- нормальную последовательность событий для прецедента;
- описание потоков в альтернативных и исключительных ситуациях.

Документация на потоки событий обычно составляется в момент проработки итеративным способом. Сначала дается только краткое описание необходимых шагов для нормального выполнения прецедента. В ходе анализа шаги уточняются. На завершающем этапе в прецедент добавляют потоки для исключительных ситуаций.

В каждом проекте должен использоваться стандартный шаблон для создания документа, описывающего поток событий. Самыми полезными я считаю следующие шаблоны:

- X. Поток событий для прецедента <имя>.
- X.1. Предусловия.
- X.2. Главный поток.
- X.3. Под-потоки (если применимы).
- X.4. Альтернативные потоки.

Здесь X – число от единицы до количества прецедентов.

Рассмотрим пример полного документа с описанием потока событий для прецедента выбор курсов для преподавания (Select Courses to Teach).

### Поток событий для прецедента «выбор курсов для преподавания»

#### 1.1. Предусловия

Под-поток создание учебных курсов (Create Course Offerings) прецедента управление информацией о курсах (Maintain Course Information) должен быть выполнен перед его началом.

#### 1.2. Главный поток

Прецедент начинает выполняться, когда преподаватель подключается к системе регистрации и вводит свой пароль. Система проверяет правильность пароля (E-1) и просит преподавателя выбрать текущий или будущий семестр (E-2). Преподаватель вводит нужный семестр. Система предлагает выбрать требуемую операцию: добавить (Add), удалить (Delete), просмотреть (Review), напечатать (Print) или выйти (Quit).

Если выбрана операция добавить (Add), S-1: выполняется поток добавить учебный курс (Add a Course Offering).

Если выбрана операция удалить (Delete), S-2: выполняется поток удалить учебный курс (Delete a Course Offering).

Если выбрана операция просмотреть (Review), S-3: выполняется поток просмотреть расписание (Review Schedule).

Если выбрана операция напечатать (Print), S-4: выполняется поток напечатать расписание (Print Schedule).

Если выбрана операция выйти (Quit): прецедент завершается.

#### 1.3. Под-потоки

S-1: добавить учебный курс (Add a Course Offering)

Система отображает диалоговое окно, содержащее поле для ввода названия и номера предмета. Преподаватель вводит название и номер предмета (E-3). Система отображает список учебных курсов для указанного предмета (E-4). Преподаватель выбирает учебный курс. Система закрепляет за преподавателем выбранный учебный курс (E-5). Затем прецедент начинается сначала.

S-2: удалить учебный курс (Delete a Course Offering)

Система отображает диалоговое окно, содержащее поле для ввода названия и номера учебного курса. Преподаватель выбирает название и номер учебного курса (E-6). Система удаляет взаимосвязь курса с преподавателем (E-7). Затем прецедент начинается сначала.

S-3: просмотреть расписание (Review Schedule)

Система получает (E-8) и отображает следующую информацию для всех учебных курсов, за которыми закреплен данный преподаватель: название предмета, номер предмета, номер учебного курса, день недели, время и место проведения занятий. Когда преподаватель отмечает, что просматривает список, прецедент начинается сначала.

S-4: напечатать расписание (Print Schedule)

Система распечатывает расписание преподавателя (E-9). Прецедент начинается сначала.

#### 1.4. Альтернативные потоки

E-1: введен неверный идентификационный номер преподавателя. Пользователь должен повторить ввод идентификационного номера или завершить прецедент.

E-2: введен неверный семестр. Пользователь должен повторить ввод семестра или завершить прецедент.

E-3: введено неверное название или номер предмета. Пользователь должен повторить ввод названия и номера предмета или завершить прецедент.

E-4: список учебных курсов не может быть отображен. Пользователю сообщается, что данная команда в настоящий момент недоступна. Прецедент начинается сначала.

E-5: преподаватель не может быть прикреплен к выбранному учебному курсу. Информация сохраняется, система осуществит прикрепление позже. Выполнение прецедента продолжается.

E-6: введено неверное название или номер учебного курса. Пользователь должен повторить ввод названия и номера учебного курса или завершить прецедент.

E-7: система не может удалить связь курса с преподавателем. Информация сохраняется, система удалит связь позже. Выполнение прецедента продолжается.

E-8: система не может получить информацию о расписании. Прецедент начинается сначала.

E-9: расписание не может быть распечатано. Пользователю сообщается, что данная опция в данный момент недоступна. Прецедент начинается сначала.

Документы с описанием потока событий составляются и хранятся отдельно от данных программы Rational Rose, но они связаны с прецедентами.

Для связи документов, описывающих потоки событий, с прецедентами в программе Rational Rose выполните следующие действия:

1. Щелкните правой кнопкой мыши по прецеденту в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. Щелкните по вкладке **Files** (Файлы).
4. Щелкните правой кнопкой мыши по списку файлов.
5. В появившемся контекстно-зависимом меню выберите команду **Insert File** (Добавить файл).
6. Укажите нужный файл в стандартном диалоговом окне выбора файла.
7. Щелкните по кнопке **Open** (Открыть), чтобы добавить указанный файл в список.
8. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров прецедента.

Связанные документы добавляются в список браузера. Связанный документ с описанием потока событий показан на рис. 3.7.

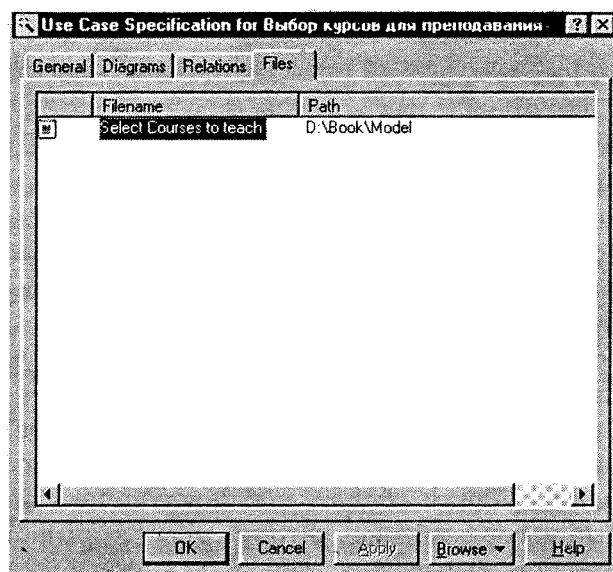


Рис. 3.7. Связанный документ с описанием потока событий

## Отношения прецедентов

Между актером и прецедентом может существовать ассоциативное отношение. Такой тип связи часто называют *коммуникативной ассоциацией* (communicate association), потому что она отражает связь между актером и прецедентом.

Ассоциативная связь может быть либо двухсторонней (от актера к прецеденту и от прецедента к актеру), либо односторонней (от актера к прецеденту или от

прецедента к актеру). Направление связи показывает, кто является ее инициатором (актер или прецедент). Такой тип отношений изображается в виде линии, соединяющей взаимодействующие элементы. Направление связи обозначается стрелками на линии связи.

Существует два типа отношений между прецедентами: включает и дополняет. Различные прецеденты могут иметь одинаково функционирующие фрагменты. Их обычно помещают в отдельный прецедент, чтобы не повторять несколько раз. Отношение включает (include relationship) создается, когда один из прецедентов использует другой. Например, каждый прецедент в системе регистрации учебных курсов начинается с аутентификации пользователя. Такие действия можно объединить в один прецедент, который будет применяться другими пользователями.

Отношение включает изображается как отношение зависимости, которое направлено от базового прецедента к используемому. (В программе Rational Rose 2000 вместо отношения зависимости необходимо использовать одностороннюю ассоциативную связь.)

Отношение дополняет (extend relationship) применяется для отражения:

- дополнительных режимов;
- режимов, которые запускаются только при определенных условиях, например сигнала тревоги;
- альтернативных потоков, которые запускаются по выбору актера.

Например, прецедент, контролирующий движение коробок на конвейере, может быть дополнен прецедентом сигнала тревоги при возникновении затора. Для системы регистрации курсов пока нельзя выделить каких-либо дополнительных прецедентов. Отношение дополняет изображается как отношение зависимости, которое направлено от дополнительного прецедента к базовому. (В программе Rational Rose 2000 необходимо использовать одностороннюю ассоциативную связь вместо отношения зависимости.)

В языке UML существует понятие *стереотипа* (stereotype), с помощью которого создаются новые элементы модели путем расширения функциональности базовых элементов. Таким образом, это понятие позволяет языку UML иметь минимальный набор символов, которые могут быть при необходимости дополнены для создания связующих элементов в разрабатываемой системе. Имя стереотипа заключается в двойные треугольные скобки и помещается рядом с линией связи. Стереотипы используются для создания нужных отношений между прецедентами. Стереотип <<communicate>> может добавляться к ассоциации, чтобы показать, что это коммуникативная ассоциация. Но в этом нет необходимости, поскольку ассоциация – это единственный допустимый тип связи между актером и прецедентом. Отношения включает и дополняет должны использовать стереотипы, потому что они отображаются как отношение зависимости.

Пример отношений прецедентов показан на рис. 3.8.

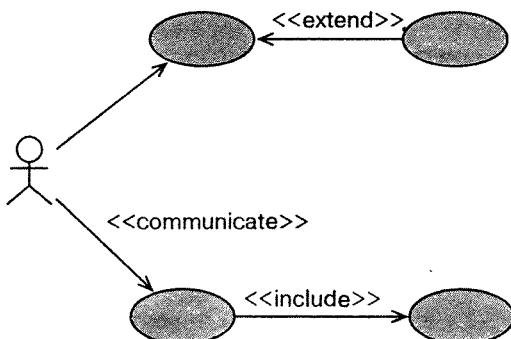


Рис. 3.8. Отношения прецедентов

## Диаграммы прецедентов

*Диаграмма прецедентов* (use case diagram) – это графическое представление всех или части актеров, прецедентов и их взаимодействий в системе. В каждой системе обычно есть главная диаграмма прецедентов, которая отображает границы системы (актеров) и основное функциональное поведение системы (прецеденты). Другие диаграммы прецедентов могут создаваться при необходимости. Приведу некоторые примеры:

- диаграмма, показывающая все прецеденты для определенного актера;
- диаграмма, показывающая все прецеденты, реализованные на данной итерации;
- диаграмма, показывающая определенный прецедент и все его отношения.

Для создания главной диаграммы прецедентов в программе Rational Rose:

1. Дважды щелкните по пункту **Main** (Главная диаграмма) в разделе **Use Case View** (Представление прецедентов) в списке браузера, чтобы открыть диаграмму.
2. В списке браузера выберите актера и перетащите его на диаграмму с помощью мыши.
3. Аналогичным образом поместите на диаграмму других нужных актеров.
4. В списке браузера выберите прецедент и перетащите его на диаграмму с помощью мыши.
5. Аналогичным образом поместите на диаграмму другие требуемые прецеденты.

Актеры и прецедент могут быть получены прямо на диаграмме с использованием панели инструментов.

Чтобы создать коммуникативные ассоциации в программе Rational Rose:

1. На панели инструментов щелкните по кнопке **Association** (Ассоциативная связь) или по кнопке **Unidirectional Association** (Однонаправленная ассоциативная связь). Если нужная кнопка отсутствует, щелкните правой

кнопкой мыши на панели инструментов, в появившемся контекстно-зависимом меню выберите команду **Customize** (Настройка), чтобы добавить кнопку.

2. Щелкните по актеру – инициатору связи – и перетащите возникшую линию связи на нужный прецедент.

Если нужно добавить стереотип, сделайте следующее:

1. Дважды щелкните по линии связи, чтобы открыть диалоговое окно **Specification** (Параметры).
2. В открывшемся списке **Stereotype** (Стереотип) выберите значение **communicate**.
3. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.
4. Аналогичным образом добавьте стереотип к другим связям.

Для создания отношения включает в программе Rational Rose нужно:

1. На панели инструментов щелкнуть по кнопке **Unidirectional Association**.
2. Щелкнуть по использующему прецеденту и перетащить возникшую линию связи на используемый.
3. Дважды щелкнуть по линии связи, чтобы открыть диалоговое окно **Specification**.
4. В открывшемся списке **Stereotype** выбрать значение **include**.
5. Щелкнуть по кнопке **OK**, чтобы закрыть диалоговое окно.

Создание отношения дополняет в программе Rational Rose предусматривает выполнение следующих действий:

1. На панели инструментов щелкните по кнопке **Unidirectional Association**.
2. Щелкните по прецеденту с дополнительными возможностями и перетащите возникшую линию связи на базовый.
3. Дважды щелкните по линии связи, чтобы открыть диалоговое окно **Specification**.
4. В открывшемся списке **Stereotype** выбрать значение **extend**.
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Specification**.

Главная диаграмма прецедентов для системы регистрации учебных курсов показана на рис. 3.9.

Порядок создания дополнительной диаграммы прецедентов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по разделу **Use Case View** (Представление прецедентов) в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Use Case Diagram** (Создать ⇒ Диаграмма прецедентов).
3. Введите название диаграммы.
4. Откройте диаграмму и поместите на нее необходимых актеров, прецеденты и связи.

Дополнительная диаграмма прецедентов показана на рис. 3.10.

## Диаграммы прецедентов

37

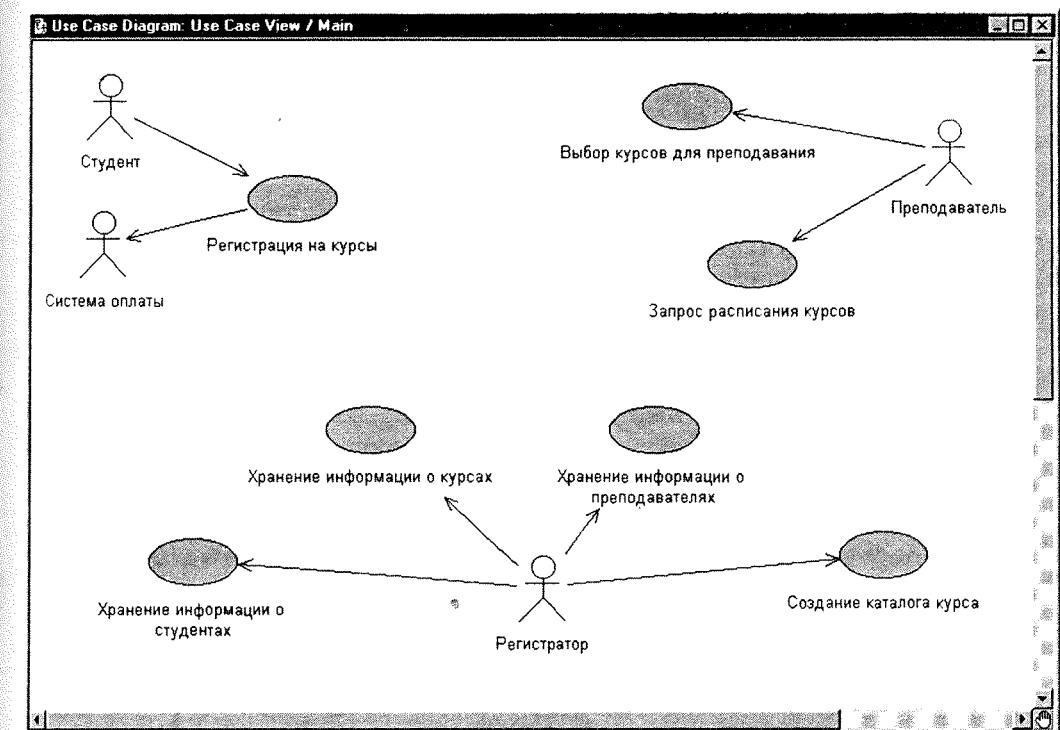


Рис. 3.9. Главная диаграмма прецедентов

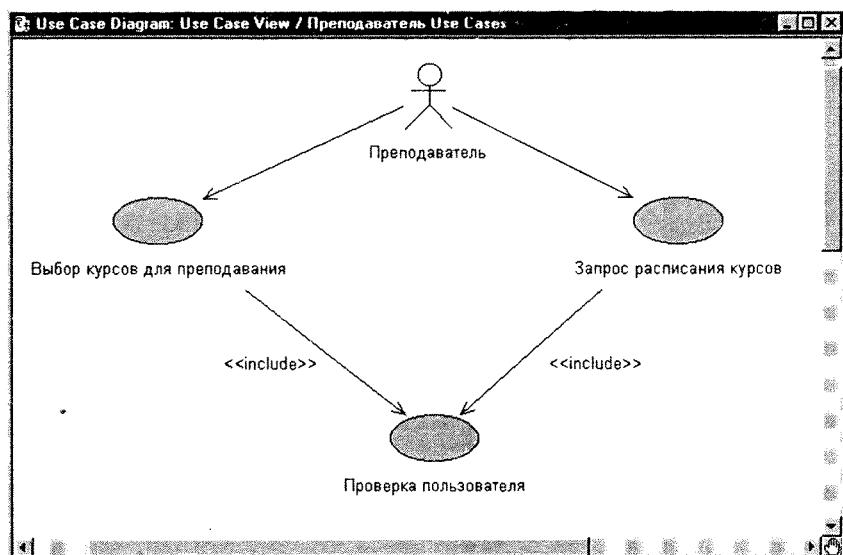


Рис. 3.10. Дополнительная диаграмма прецедентов

## Диаграммы действий

На данном этапе жизненного цикла также могут быть построены **диаграммы действий** (activity diagrams). Они отражают динамику проекта и представляют собой схемы потоков управления в системе от действия к действию, а также параллельные действия и альтернативные потоки.

В конкретной точке жизненного цикла диаграммы действий могут представлять потоки между функциями или внутри отдельной функции. На разных этапах жизненного цикла они создаются для отражения последовательности выполнения операции.

Диаграммы действий иллюстрируют действия, переходы между ними, элементы выбора и линии синхронизации. В языке UML действие изображается в виде прямоугольника с закругленными углами, переходы – в виде направленных стрелок, элементы выбора – в виде ромбов, линии синхронизации – в виде толстых горизонтальных или вертикальных линий (см. рис. 3.11).

Диаграммы действий в программе Rational Rose создаются следующим образом:

1. Щелкните правой кнопкой мыши по разделу **Use Case View** (Представление прецедентов) в списке браузера.

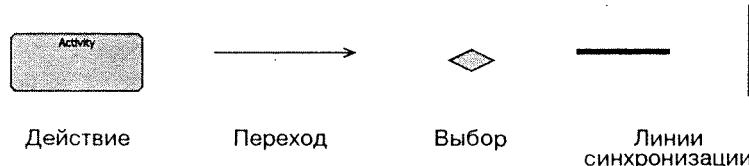


Рис. 3.11. Нотация языка UML для элементов диаграммы действий

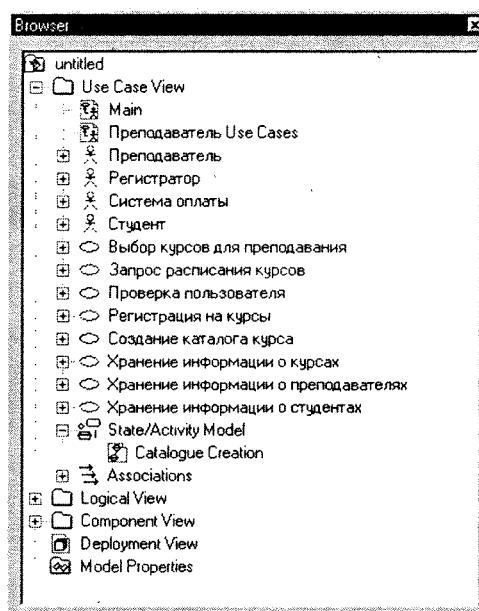


Рис. 3.12. Диаграмма действий в окне браузера

2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Activity Diagram** (Создать ⇒ Диаграмма действий). В список будет добавлена новая диаграмма с именем **New Diagram**.
  3. Введите название диаграммы.
  4. Чтобы открыть диаграмму, дважды щелкните по ней мышью в браузере.
- Окно браузера с диаграммой действий изображено на рис. 3.12.

## Действия

*Действием* называется исполнение определенного поведения в потоке управления системы (см. рис. 3.13).

Для создания действий в программе Rational Rose:

1. Щелкните по кнопке **Activity** (Действие) на панели инструментов.
2. Щелкните по диаграмме действий, чтобы поместить элемент, изображающий действие, на диаграмму.
3. Введите имя нового действия.

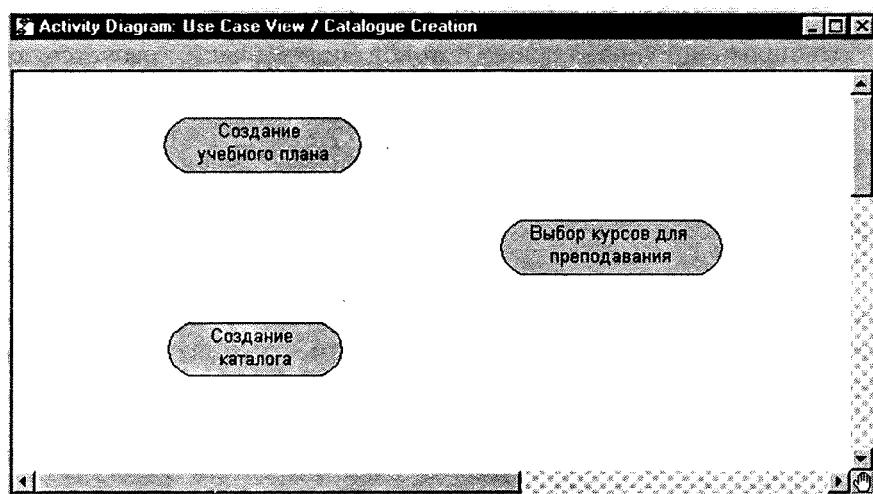


Рис. 3.13. Действия

## Переходы

Переходы используются для изображения пути потока управления от действия к действию (см. рис. 3.14). Они обычно осуществляются по завершении очередного действия.

Чтобы получить переходы в программе Rational Rose:

1. Щелкните по кнопке **State Transition** (Переход) на панели инструментов.
2. Щелкните по начальному действию на диаграмме и переместите стрелку перехода на последующее действие.

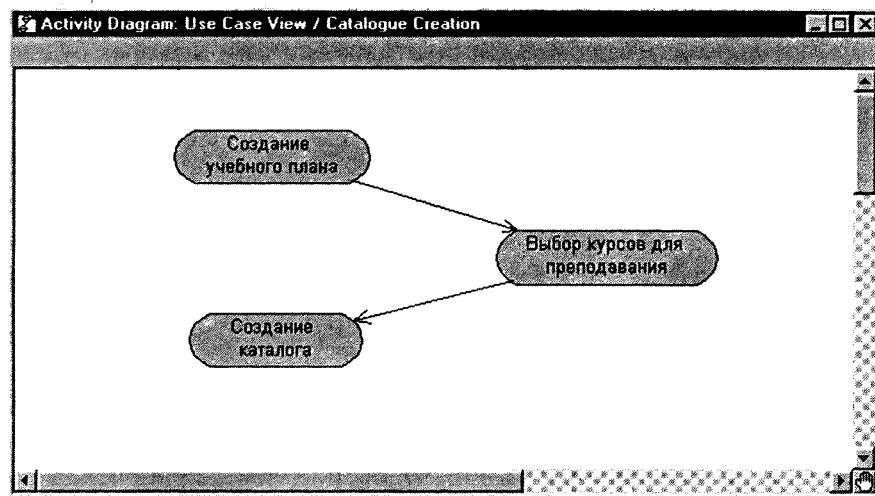


Рис. 3.14. Переходы

### Элементы выбора

При моделировании управляющих потоков системы часто требуется показать места их разделения на основе условного выбора. Переходы из элемента выбора содержат ограничительные условия, определяющие, какое направление перехода будет выбрано. Элементы выбора и условия позволяют задавать альтернативные пути потока управления.

Для создания элементов выбора в программе Rational Rose выполните следующие действия:

1. Щелкните по кнопке **Decision** (Элемент выбора) на панели инструментов.
2. Щелкните по диаграмме действий, чтобы поместить на нее элемент выбора.
3. Введите имя нового элемента.
4. Щелкните по кнопке **State Transition** на панели инструментов.
5. Щелкните по начальному действию на диаграмме и переместите стрелку перехода на элемент выбора.

Элемент выбора показан на рис. 3.15.

Последовательность создания условных переходов в программе Rational Rose:

1. Щелкните по кнопке **State Transition** на панели инструментов.
2. Щелкните по элементу выбора на диаграмме и переместите стрелку перехода на последующее действие.
3. Дважды щелкните по стрелке перехода, чтобы открыть диалоговое окно **Specification** (Параметры).
4. Щелкните по вкладке **Detail** (Подробно).
5. В поле ввода **Guard Condition** (Условие) введите условие перехода.
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.

Указатель перехода с условием изображен на рис. 3.16.

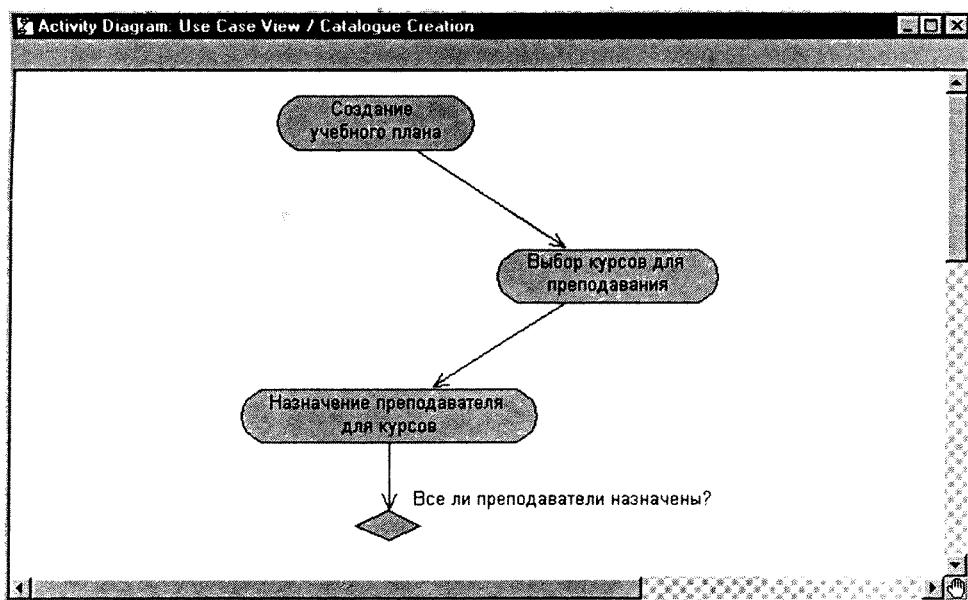


Рис. 3.15. Элементы выбора на диаграмме действий

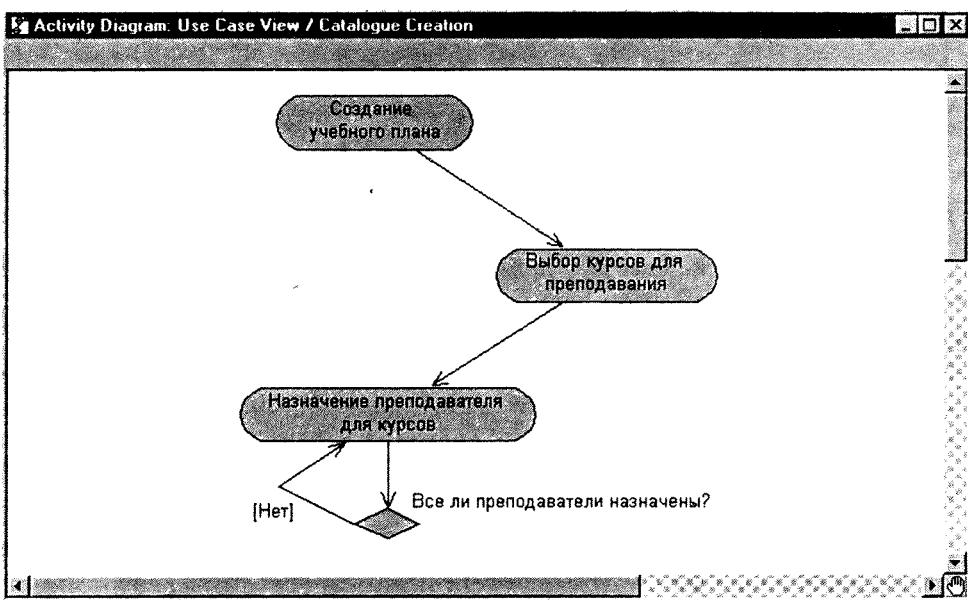


Рис. 3.16. Условные переходы

Чтобы получить прямолинейные линии переходов в программе Rational Rose:

1. Выберите линии переходов, которые вы хотите сделать прямолинейными (для выбора нескольких линий можно использовать клавишу **Shift**).

2. Выберите команду меню **Format** ⇒ **Style** ⇒ **Rectilinear** (Формат ⇒ Стиль ⇒ Прямолинейный).
3. Расположите линии нужным образом на диаграмме действий, перетаскивая их с помощью мыши.

Прямолинейные линии переходов показаны на рис. 3.17.

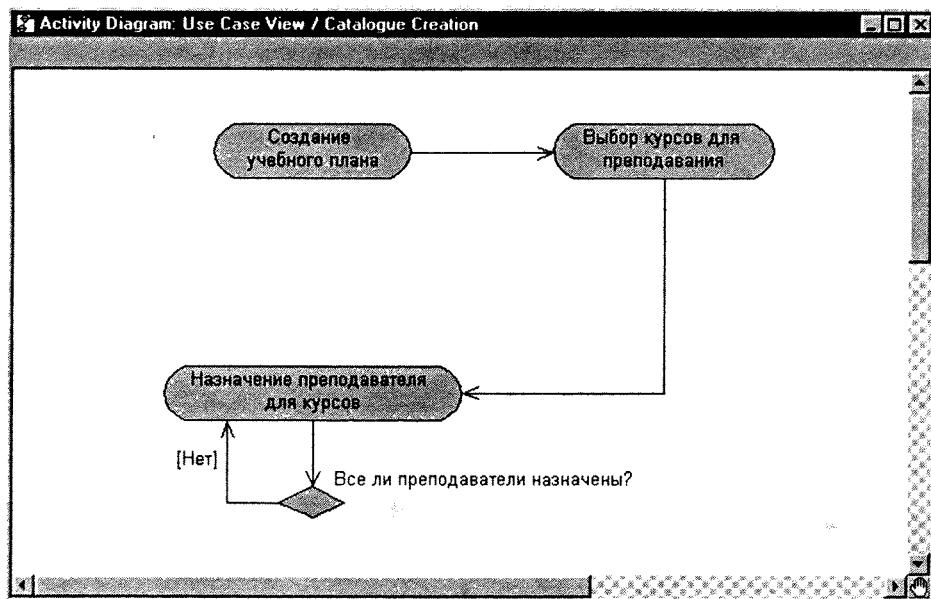


Рис. 3.17. Прямолинейные линии

### Линии синхронизации

В потоке обычно существуют действия, выполняемые параллельно. *Линия синхронизации* (synchronization bar) позволяет указать на необходимость их одновременного выполнения, а также обеспечивает единое выполнение действий в потоке (то есть указывает на необходимость завершения определенных действий для перехода к следующему) – см. рис. 3.18. Таким образом, линии перехода могут иметь несколько входящих линий переходов и одну исходящую либо одну входящую и несколько исходящих.

Для создания линий синхронизации в программе Rational Rose:

1. Щелкните по кнопке **Horizontal Synchronization** (Горизонтальная линия синхронизации) или **Vertical Synchronization** (Вертикальная линия синхронизации) на панели инструментов.
2. Щелкните по диаграмме действий, чтобы поместить на нее линию синхронизации.
3. Щелкните по кнопке **State Transition** (Переход) на панели инструментов и добавьте необходимые входящие и исходящие линии переходов к линии синхронизации.

Линии синхронизации показаны на рис. 3.18.

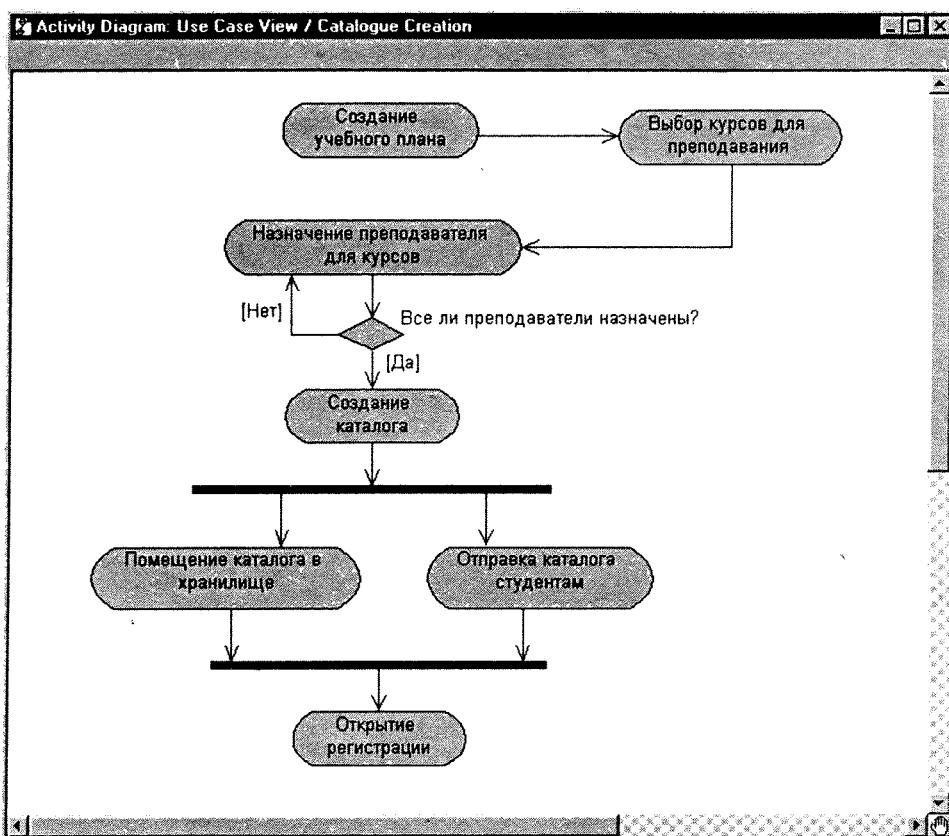


Рис. 3.18. Линии синхронизации

### Секции

*Секции* (swimlanes) делят диаграммы действий на несколько участков. Это нужно для того, чтобы показать, кто отвечает за выполнение действий на каждом участке.

Алгоритм создания секций в программе Rational Rose:

1. Щелкните по кнопке **Swimlane** (Секция) на панели инструментов.
2. Щелкните по диаграмме действий, чтобы создать на ней новую секцию с названием **New Swimlane**.
3. Дважды щелкните по названию новой секции, чтобы открыть диалоговое окно **Specification** (Параметры).
4. Введите нужное название секции в поле ввода **Name** (Название).
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.
6. Для изменения размеров секции переместите ее границу с помощью мыши.
7. Переместите все необходимые действия и переходы на диаграмме в новую секцию, где сразу сможете их создавать.

Диаграмма действий с разделительными линиями показана на рис. 3.19.

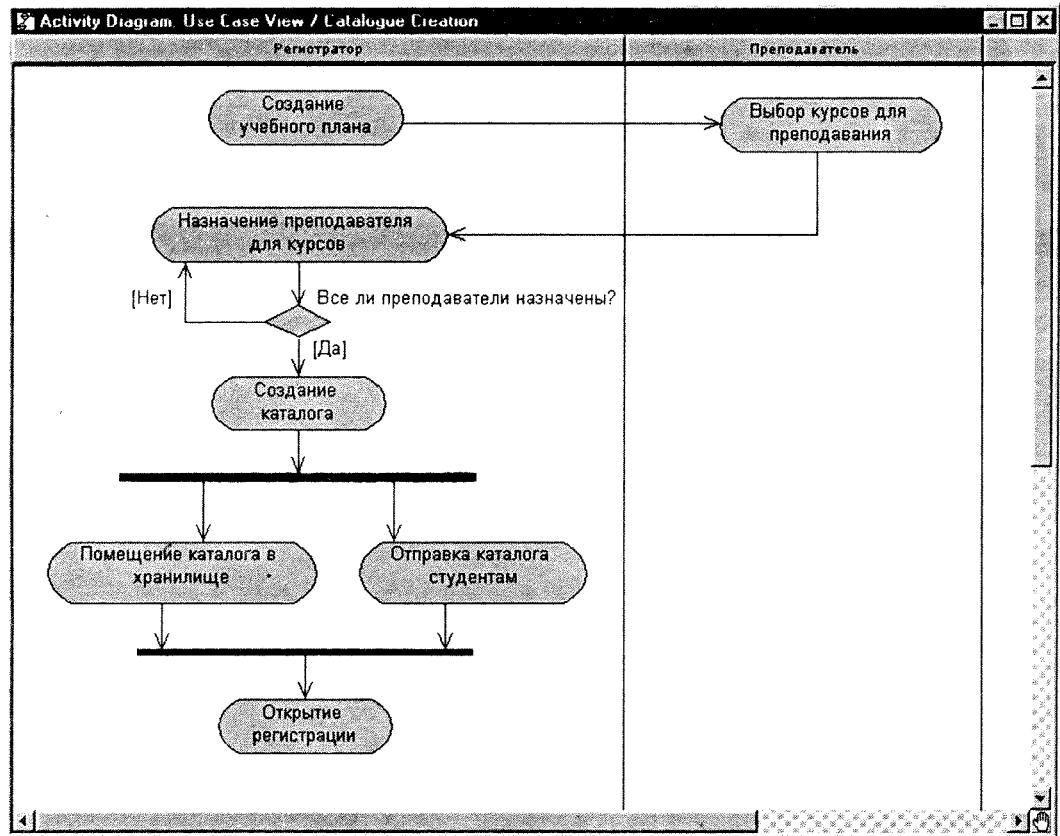


Рис. 3.19. Секции

### Начальное и конечное состояния

Для обозначения начального и конечного состояний в потоке управления системы используются специальные символы. Начальное состояние изображается в виде закрашенного круга, а конечное – в виде закрашенного круга, обведенного дополнительной окружностью. Обычно в потоке существуют одно начальное и несколько конечных состояний – для каждого альтернативного направления.

Последовательность создания начального и конечного состояний в программе Rational Rose:

1. Щелкните по кнопке **Start State** (Начальное состояние) или **End State** (Конечное состояние) на панели инструментов.
2. Щелкните по диаграмме действий, чтобы поместить на нее символ конечного или начального состояния.
3. Если вы добавили начальное состояние, щелкните по кнопке **State Transition** (Переход) на панели инструментов, а затем на символе начального состояния и выполните переход к первому действию в потоке.

4. Если вы добавили конечное состояние, щелкните по кнопке **State Transition** на панели инструментов, а затем на предшествующем действии и выполните переход к символу конечного состояния на диаграмме.

Диаграмма действий с начальным и конечным состояниями показана на рис. 3.20.

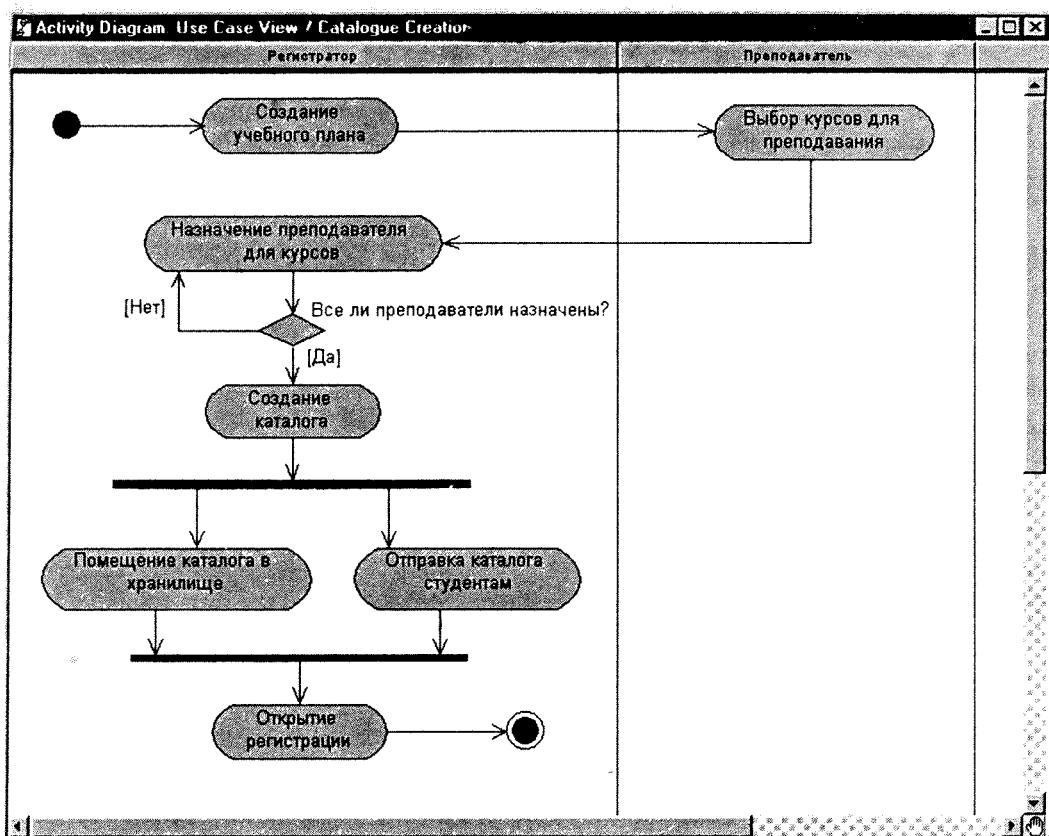


Рис. 3.20. Начальное и конечное состояния

## Резюме

Поведение системы описывается с помощью модели прецедентов, которая содержит системные прецеденты, системное окружение – актеров и связи между прецедентами и актерами – диаграммы прецедентов. Модель прецедентов должна представлять собой единое средство для обсуждения функциональности и поведения системы с заказчиком и конечным пользователем.

Разработка модели прецедентов начинается на стадии задумки с выбора актеров и определения общих принципов прецедентов системы. Затем модель дополняется на этапе проработки.



Актеры не являются частью системы – это кто-то или что-то, что должно взаимодействовать с системой. Прецеденты определяют функциональность системы. Они служат для моделирования диалога между актером и системой.

Для каждого прецедента указывается поток событий, описывающий те из них, которые необходимы для обеспечения требуемого поведения. Поток событий описывается в терминах того, «что» система должна делать, а не «как» она должна это делать. Диаграмма прецедентов – это графическое представление всех или части актеров, прецедентов и их взаимодействий в системе.

Наиболее распространенные типы отношений между прецедентами – включает и дополняет. Первый используется для выделения общих функциональных фрагментов в отдельный прецедент, второй – для придания прецеденту дополнительных элементов поведения.

Диаграммы действий отражают динамику системы. Они представляют собой схемы потоков управления в системе. В конкретной точке жизненного цикла диаграммы действий могут представлять потоки между прецедентами или внутри отдельного прецедента. На последующих этапах жизненного цикла диаграммы действий могут создаваться для отражения последовательности выполнения операции.



## Глава 4. Поиск классов

### Что такое объект

*Объект* (*object*) – это некая сущность реального мира или концептуальная сущность. Объект может быть чем-то конкретным, например грузовик Джо или мой компьютер, или концептуальным, как, например, химический процесс, банковская операция, торговый заказ, кредитная история или ставка прибыли.

Объектом называется концепция, абстракция или вещь с четко определенными границами и значением для системы. Каждый объект в системе имеет три характеристики: состояние, поведение и индивидуальность.

### Состояние, поведение и индивидуальность

*Состоянием* (*state*) объекта называется одно из условий, в которых он может находиться. Состояние системы обычно меняется во времени и определяется набором свойств, называемых *атрибутами* (*attribute*), значений свойств и отношений между объектами. Например, объект учебный курс (*CourseOffering*) в системе регистрации учебных курсов может находиться в одном из двух состояний: открыт для записи или закрыт для записи. Если количество студентов, зарегистрировавшихся на курс, меньше десяти, запись на курс продолжается. После регистрации десятого студента она прекращается.

*Поведение* (*behavior*) определяет, как объект реагирует на запросы других объектов и что может делать сам объект. Поведение реализуется с помощью набора *операций* (*operation*) для объекта. В системе регистрации курсов объект учебный курс может иметь операции добавить студента и удалить студента.

*Индивидуальность* (*identity*) означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта. Например: Алгебра 101, секция 1 и Алгебра 101, секция 2 – два объекта в системе регистрации курсов. Хотя они оба являются учебными курсами, каждый из них уникален.

В языке UML объект изображается в виде прямоугольников, а его имя пишется с подчеркиванием – см. рис. 4.1.

Алгебра 101, Раздел 1

Рис. 4.1. Нотация языка UML для объекта

### Что такое класс

*Класс* (*class*) – это описание группы объектов с общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами и семантикой. Таким образом, класс представляет собой шаблон для создания объекта.

Каждый объект является экземпляром конкретного класса и не может быть экземпляром нескольких классов. Например, класс учебный курс (CourseOffering) может определяться следующими характеристиками:

- атрибуты – место занятий, время занятий;
- операции – получить место занятий, получить время занятий, добавить студента на курс.

Алгебра 101, секция 1 и Алгебра 101, секция 2 – это объекты, принадлежащие классу учебный курс. Каждый объект имеет значения атрибутов и доступ к операциям, определенным классом учебный курс.

«Хороший» класс представляет одну и только одну абстракцию, то есть должен отражать одну основную сущность. Например, класс, способный хранить информацию о студентах и данные о курсах, которые студент посещает в течение нескольких лет, не является «хорошим» классом, потому что не представляет одну сущность. Такой класс необходимо разделить на два связанных класса: студент и история студента.

Названия классов выбираются в соответствии с понятиями предметной области. Имя должно быть существительным в единственном числе, наиболее точно характеризующим предмет. В качестве имени класса можно использовать *акроним*, если он имеет одинаковое значение для всех представляемых сущностей. При использовании акронима в описании класса желательно указать полное название.

Иногда трудно отличить объект от класса. Почему, например, Алгебра 101, секция 1 – объект, а не класс? Что отличает его от объекта Алгебра 101, секция 2? Ответы на эти вопросы субъективны. Изучив эти объекты, можно заключить, что у них одинаковая структура и поведение. Они лишь являются разными учебными предметами семестра. Кроме того, в системе регистрации курсов можно обнаружить множество схожих сущностей с одинаковой структурой и поведением: Музыка 101, секция 1; История 101, секция 1; История 101, секция 2 и т.п. Значит, допустимо создание единого класса учебный курс (CourseOffering).

В языке UML классы изображаются в виде разделенных прямоугольников. В верхней секции указывается имя класса, средняя секция содержит его структуру – атрибуты, а нижняя описывает его поведение – операции. Класс показан на рис. 4.2.

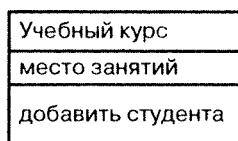


Рис. 4.2. Нотация языка UML для класса

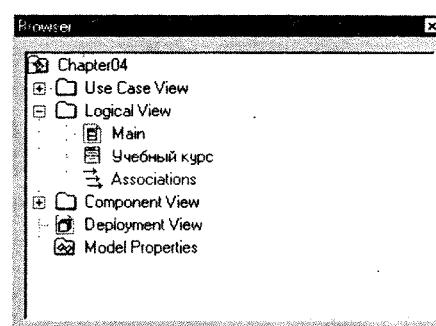


Рис. 4.3. Класс, созданный в окне браузера

Порядок создания классов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по разделу **Logical View** (Логическое представление) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class** (Создать ⇒ Класс). В список браузера будет добавлен новый класс с именем **New Class**.
3. Введите нужное имя класса.

Класс в окне браузера показан на рис. 4.3.

## Стереотипы и классы

Мы уже говорили о стереотипах отношений на диаграмме функций. Классы тоже могут иметь стереотипы. Как и ранее, стереотип используется для создания нового типа элемента моделирования, в данном случае для создания новых типов классов. Некоторые основные стереотипы класса – это сущность, граничный элемент, элемент управления, сервисный элемент и исключение.

Стереотип класса указывается под его именем и заключается в двойные треугольные скобки. Если требуется, стереотип можно отобразить графическим значком или выделить цветом. В программе Rational Rose есть изображения для стереотипов Rational Unified Process – управляющего элемента, сущности и граничного элемента. Эти стереотипы, а также пример класса со стереотипом исключение показаны на рис. 4.4.



Рис. 4.4. Классы со стереотипами

## Обнаружение классов

Рецептов для поиска классов не существует. Как сказал Грейди Буч: «Это действительно трудно!» Rational Unified Process содержит средства, помогающие обнаружить в системе классы типа управляющий элемент, граничный элемент и сущность. Эти три стереотипа соответствуют концепции «модель – представление –

управление» и позволяют аналитику отделить друг от друга представление, предметную область и управление в системе.

По причине того, что процесс анализа и проектирования является итеративным, список классов со временем изменится. Начальный набор классов, скорее всего, будет отличаться от итогового. Поэтому для описания начального набора классов, обнаруженных в системе, часто используется термин «класс-кандидат».

### **Классы-сущности**

*Класс-сущность* (entity class) используется для моделирования данных и поведения с длинным жизненным циклом. Этот тип классов может представлять сущности реального мира или внутренние элементы системы. Такие классы обычно не зависят от окружения, то есть они нечувствительны к взаимодействию окружающей среды с системой. Следовательно, они не зависят от приложения и могут использоваться в различных приложениях.

Первый шаг – изучить обязанности, описанные в потоке событий для выявления прецедентов (что система должна делать). Классы-сущности – это обычно те классы, которые требуются системе для выполнения определенных обязанностей. Использование существительных для описания обязанностей может стать хорошим началом. Исходный список нужно профильтровать, так как он будет содержать слова, не относящиеся к предметной области, языковые выражения, избыточные слова и существительные, описывающие структуру класса.

Классы-сущности обычно определяются на стадии проработки. Их часто называют классами предметной области, потому что они представляют собой абстракции предметов реального мира.

### **Границные классы**

*Границные классы* (boundary class) обеспечивают взаимодействие между окружающей средой и внутренними элементами системы. Такие классы предоставляют интерфейс для пользователя или другой системы (то есть для актера). Они составляют внешне зависимую часть системы и используются для моделирования интерфейсов системы.

Для обнаружения граничных классов изучают пары актер/сценарий. Такие классы, определенные на фазе проработки, обычно являются классами верхнего уровня. Например, вы можете смоделировать окно, но не моделировать его диалоговые элементы и кнопки. В этом случае вы опишете требования пользовательского интерфейса, но не реализуете его.

Требования к пользовательскому интерфейсу порой недостаточно ясны. Обычно используются термины «дружественный» и «гибкий». Но дружественный интерфейс разными людьми трактуется по-разному. Здесь могут пригодиться прототипы. Пользователь должен посмотреть и почувствовать систему, чтобы реально оценить, что значит «дружественный». И то, что это значит, затем представляется как структура и поведение граничного класса. На этапе проектирования такие классы совершенствуются и выносятся на обсуждение вопросов реализации пользовательского интерфейса.

Границные классы также используются для обеспечения связи с другими системами. На этапе проектирования эти классы совершенствуются и выносятся на обсуждение вопросов реализации протоколов взаимодействия.

### Управляющие классы

*Управляющие классы* (control class) служат для моделирования последовательного поведения одного или нескольких прецедентов и координации событий, реализующих заложенное в них поведение. Управляющие классы можно представить как классы, «исполняющие» прецедент и определяющие его динамику. Они обычно зависят от приложения.

На ранней стадии проработки управляющие классы добавляются для каждой пары актер/прецедент. Такие классы определяют поток событий в прецедентах.

Вопрос использования управляющих классов очень субъективный. Многие авторы утверждают, что их применение приводит к отделению данных от поведения. Это может случиться, если управляющие классы выбраны неаккуратно. Если управляющий класс реализует что-то большее, чем последовательное действие, то он делает слишком много. Например: в системе регистрации учебных курсов студент выбирает курс, и если курс доступен, студента на него записывают.

Кто должен знать, как прикрепить студента, – управляющий класс или класс, представляющий курс занятий? Правильный ответ – класс, представляющий курс занятий. Управляющий класс знает лишь, когда студент должен быть прикреплен. «Плохой» управляющий класс знает не только когда, но и как прикрепить студента.

Управляющий класс для каждой пары актер/прецедент создается на начальном этапе. При дальнейшем анализе и проектировании управляющие классы могут исключаться, разделяться или объединяться.

Этапы создания стереотипов для классов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по имени класса в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. Щелкните по вкладке **General** (Общие).
4. В открывшемся списке **Stereotype** (Стереотип) выберите нужный стереотип. Чтобы создать новый стереотип, введите его имя в поле открывшегося списка **Stereotype**.
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров класса.

Параметры для класса студент показаны на рис. 4.5. Если язык, выбранный

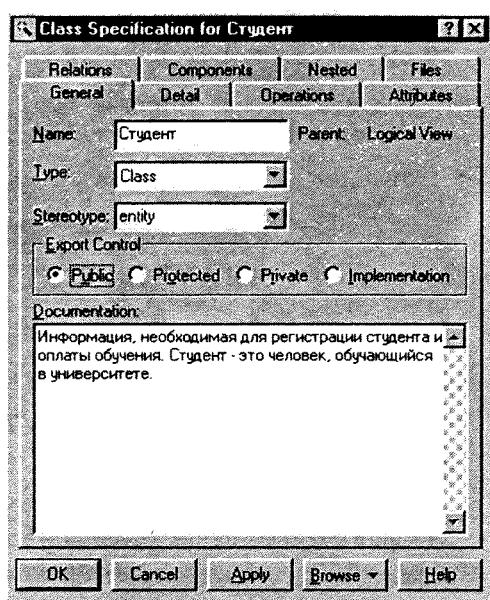


Рис. 4.5. Установка стереотипа класса

в модели по умолчанию, отличается от языка анализа (вкладка **Notation** (Нотация) диалогового окна **Options** (Настройки)), то в диалоговом окне параметров класса появится еще одна вкладка для языка.

## Документирование классов

После того как класс создан, информацию о нем необходимо отразить в документации. Документация предназначена для описания назначения класса, а не его структуры. Например, класс *студент* может быть описан следующим образом:

Информация, необходимая для регистрации студента и оплаты обучения. Студент – это человек, обучающийся в университете.

А вот пример неправильного описания:

Имя, адрес и телефон студента.

Оно раскрывает только структуру класса, которую можно увидеть, посмотрев на список атрибутов, и не поясняет, для чего нужен данный класс.

Трудности при выборе имени и описании класса могут свидетельствовать о том, что это недостаточно хорошая абстракция. В следующем списке перечислены возможные варианты:

- можно определить имя и дать краткое, четкое описание – хороший класс-кандидат;
- можно определить имя и выбрать описание, похожее на описание другого класса, – объединить классы;
- можно определить имя, но потребуется целая книга, чтобы описать назначение класса, – разделить класс;
- нельзя определить имя и дать описание – требуется дополнительный анализ для выделения правильных абстракций.

Чтобы описать классы в программе Rational Rose:

1. Выберите класс в списке браузера.
2. Установите курсор в окне описания и введите описание класса.

Описание класса представлено на рис. 4.6.

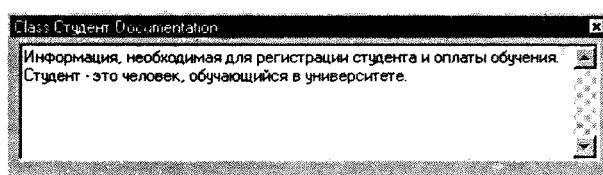


Рис. 4.6. Описание класса

## Пакеты

Если в системе существует немного классов, управлять ими достаточно легко. Многие системы состоят из большого количества классов, поэтому необходим механизм, позволяющий разбить их на группы и облегчающий управление и повторное использование. Здесь оказывается полезной концепция пакетов.

*Пакет* (package) в логическом представлении модели – это набор классов и других связанных пакетов. Путем объединения классов в пакеты мы можем получить представление модели на более высоком уровне. Изучая содержимое пакета, мы, наоборот, получаем более детальное представление.

Каждый пакет содержит интерфейс, реализуемый набором его общедоступных классов (public classes), то есть тех, с которыми могут общаться классы из других пакетов. Остальные классы пакета – это классы реализации (implementation classes), которые не взаимодействуют с классами в других пакетах.

В сложной системе для облегчения восприятия пакеты могут быть созданы на этапе проработки. В более простой системе классы, выделенные на этапе анализа, могут быть сгруппированы в один пакет, представляющий саму систему. В ходе дальнейшего анализа и проектирования пакеты нужны для группировки классов, используемых в системной архитектуре.

В языке UML пакеты изображаются в виде папок (см. рис. 4.7).

Чтобы создать пакеты в программе Rational Rose:

1. Щелкните правой кнопкой мыши по разделу **Logical View** (Логическое представление) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Package** (Создать ⇒ Пакет).
3. Введите нужное имя пакета.

Пакет, созданный в списке браузера, показан на рис. 4.8. После создания пакета в него можно поместить необходимые классы.

Последовательность перемещения классов в пакет в программе Rational Rose:

1. В списке браузера выделите нужный класс, щелкнув по нему мышью.
2. Удерживая кнопку мыши нажатой, перетащите класс в пакет.
3. Повторите те же действия для других классов, которые требуется переместить.

Перемещенные классы показаны на рис. 4.9.

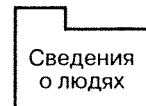


Рис. 4.7. Нотация языка UML для пакетов

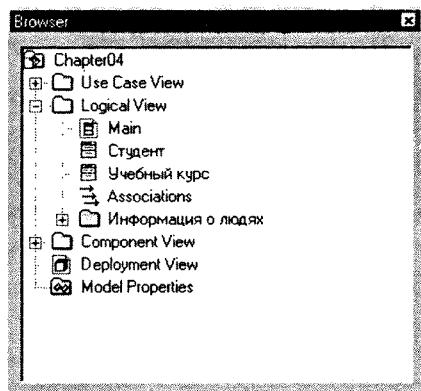


Рис. 4.8. Пакет, созданный в списке браузера

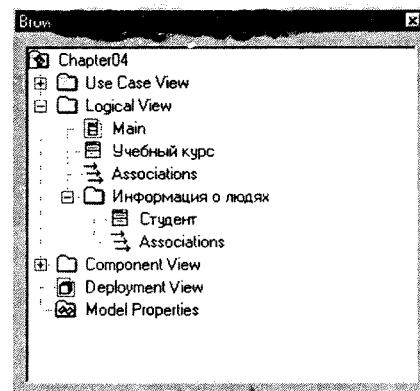


Рис. 4.9. Перемещенные классы

## Объекты и классы в системе регистрации курсов

Рассмотрим сценарий добавление учебного курса (Add a Course Offering to Teach), который является внутренним потоком для прецедента выбор предметов для преподавания (Select Courses to Teach). Данный сценарий позволяет преподавателю выбрать учебный курс для конкретного семестра.

Хотя мы рассматриваем этот процесс пошагово, на практике большинство шагов могут быть выполнены одновременно.

### Выбор граничных классов

Рассматриваемый прецедент взаимодействует только с актером преподаватель. Действие, выполняемое указанным сценарием, – это только одна из возможностей, обеспечиваемых прецедентом (он также определяет, что преподаватель может изменять, удалять, просматривать и печатать курсы). Это означает, что в системе должен быть механизм, позволяющий преподавателю выбирать желаемое действие. Для обеспечения потребностей преподавателя создается специальный класс – параметры курса преподавателя (ProfessorCourseOptions). Дополнительно мы можем указать класс, который служит для добавления новых курсов, доступных преподавателю, – добавление учебного курса (AddACourseOffering).

### Выбор классов-сущностей

Данный сценарий состоит из предметов, учебных курсов и назначения преподавателей. Мы можем выделить три класса-сущности: предмет (Course), учебный курс (CourseOffering) и преподаватель (Professor).

### Выбор управляемых классов

Добавим один управляемый класс с целью обработки потока событий для прецедента – менеджер курсов преподавателя (ProfessorCourseManager).

Выбранные классы (с установленными стереотипами сущность, управляющий элемент или граничный элемент) могут быть добавлены к модели (см. рис. 4.10). Так как актер преподаватель уже существует, при создании класса преподаватель программа Rational Rose предупредит, что одно и то же имя используется в разных разделах.

### Создание пакетов

Следующий шаг – объединить классы в пакеты. На данном этапе выделим шесть классов: предмет, учебный курс, преподаватель, параметры курса преподавателя, добавление учебного курса и менеджер курсов преподавателя. Их можно разделить на три логические группы: объекты, специфичные

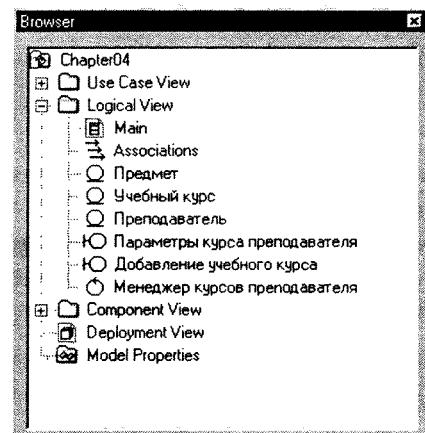


Рис. 4.10. Классы для сценария добавление учебного курса

для университета; объекты, содержащие информацию о людях; интерфейсы для актеров. Таким образом, мы можем создать следующие пакеты: Интерфейсы (Interfaces), Объекты университета (UniversityArtifacts) и Сведения о людях (PeopleInfo). Затем классы помещаются в соответствующие пакеты (см. рис. 4.11).

## Диаграммы классов

По мере того как новые классы добавляются в систему, их текстовое представление становится неудобным. *Диаграммы классов* (class diagrams) помогают графически представить некоторые или все классы в модели.

Главная диаграмма классов в логическом представлении модели обычно отображает пакеты системы. Каждый пакет также имеет свою главную диаграмму классов, которая обычно содержит общедоступные классы пакета. Другие диаграммы создаются по необходимости. Приведу типичные примеры использования диаграмм классов:

- просмотр всех классов реализации в пакете;
- просмотр структуры и поведения одного или нескольких классов;
- просмотр иерархии наследования классов.

Программа Rational Rose автоматически создает главную диаграмму классов в логическом представлении модели.

Чтобы добавить пакеты к главной диаграмме классов, сделайте следующее:

1. Дважды щелкните по пункту списка **Main diagram** (Главная диаграмма) в браузере, чтобы открыть диаграмму.
2. Выберите нужный пакет в списке, щелкнув по нему мышью.
3. Перетащите пакет на диаграмму.
4. Аналогичным образом перетащите на диаграмму другие пакеты.

Главная диаграмма классов для системы регистрации показана на рис. 4.12.

Этапы создания главной диаграммы классов пакета в программе Rational Rose:

1. Дважды щелкните по изображению пакета на диаграмме классов.
2. Пакет откроется, и появится главная диаграмма классов.
3. Выберите нужный класс в списке браузера и перетащите его на диаграмму с помощью мыши. Для отображения стереотипа класса на диаграмме можно воспользоваться командой меню **Format** ⇒ **Stereotype display** (Формат ⇒ Показать стереотип).
4. Повторите предыдущий шаг для других классов, которые вы хотите поместить на диаграмму.

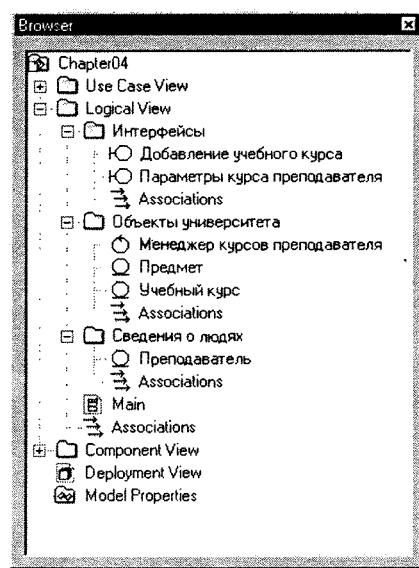


Рис. 4.11. Пакеты в браузере

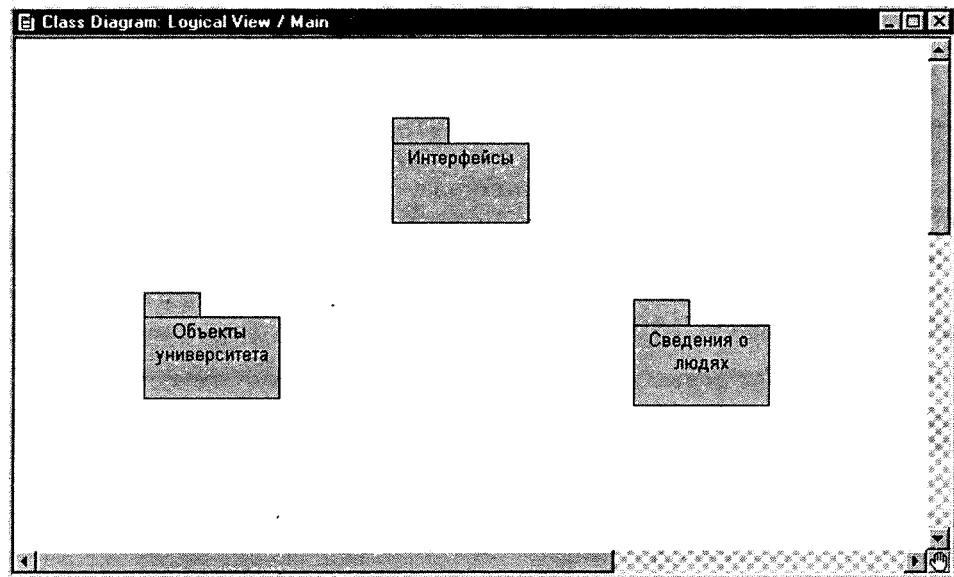


Рис. 4.12. Главная диаграмма классов

Главная диаграмма классов для пакета Объекты университета изображена на рис. 4.13. Заметьте, что класс учебный курс (CourseOffering) на ней отсутствует. Это класс реализации в пакете, и мы решили не показывать его на главной диаграмме. По мере добавления пакетов и классов в модель могут быть созданы дополнительные диаграммы.

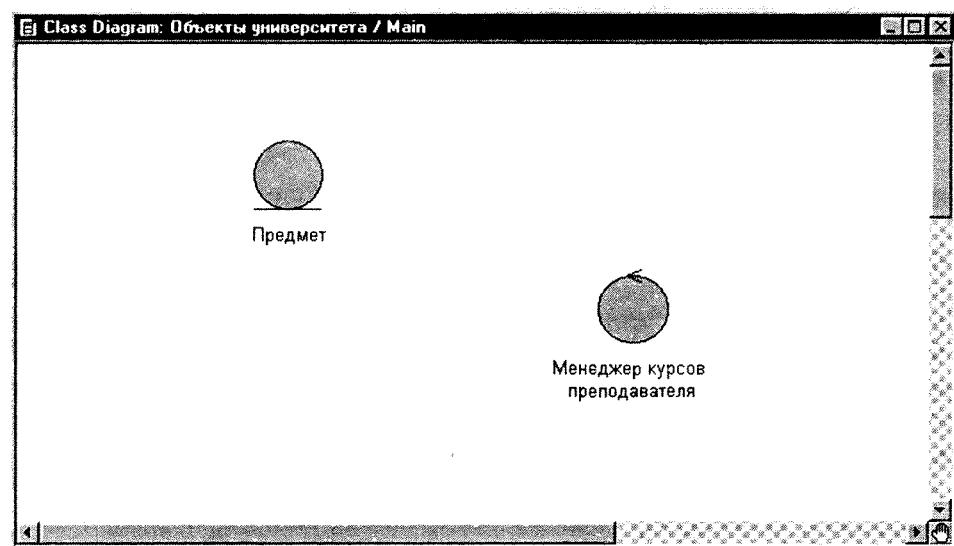


Рис. 4.13. Главная диаграмма классов пакета Объекты университета

Настройка видимости классов по умолчанию:

1. Выберите команду меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры).
2. Щелкните по вкладке **Diagram** (Диаграмма).
3. Установите флажок **Show Visibility** (Показать видимость) для отображения по умолчанию всех классов.

Установка видимости для выбранного класса:

1. Щелкните правой кнопкой мыши по одному из классов на диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Options** ⇒ **Show Visibility** (Параметры ⇒ Показать видимость).

Диаграмма классов, отражающая видимость пакетов, показана на рис. 4.14.

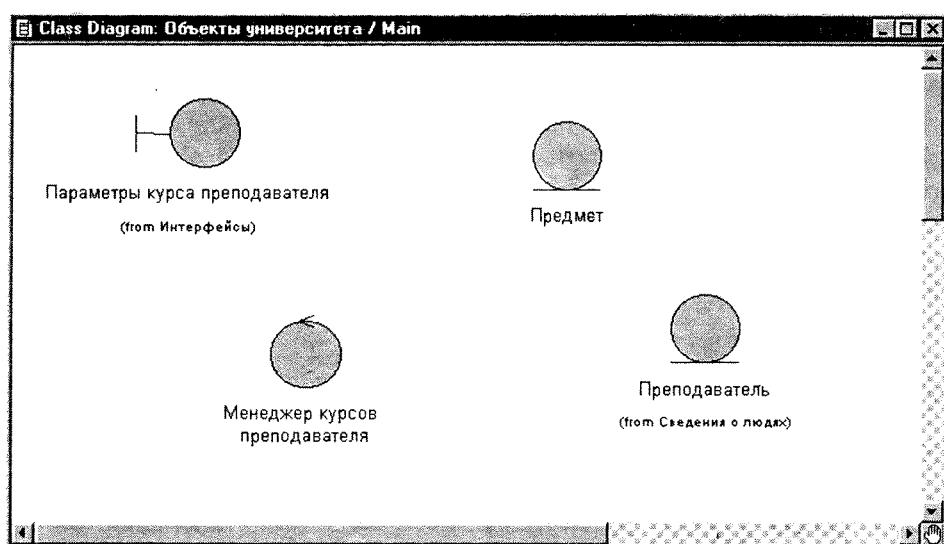


Рис. 4.14. Диаграмма классов, отражающая видимость пакетов

## Резюме

Объекты – это компьютерное представление сущностей (предметов реального мира или понятий, придуманных человеком). Объект – это концепция, абстракция или вещь с четко определенными границами и значением для системы. Каждый объект в системе имеет три характеристики: состояние, поведение и индивидуальность. Состояние объекта – одно из условий, в которых он может находиться. Поведение характеризует объект и показывает, как он реагирует на запросы других объектов. Индивидуальность означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта.

Класс – это описание группы объектов с общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами (ассоциативными или агрегационными) и семантикой. В языке UML классы изображаются в виде



разделенных прямоугольников. В секциях прямоугольника указываются имя, структура и поведение класса. После того как класс создан, его необходимо описать в документации. Документация предназначена для описания назначения класса, а не его структуры.

Стереотипы обеспечивают возможность создания новых типов элементов моделирования и должны основываться на элементах, входящих в метамодель языка UML. На этапе анализа выделяют три основных стереотипа для классов: класс-сущность, граничный класс и управляющий класс. Эти стереотипы используются для определения классов в разрабатываемой системе.

Пакет в логическом представлении модели – это набор классов и других связанных пакетов. Путем объединения классов в пакеты мы можем получить представление модели на более высоком уровне. Изучая содержимое пакета, мы получаем более детальное представление.

Диаграммы классов помогают графически изобразить некоторые или все классы системы. Диаграммы классов можно создать и в представлении модели прецедента. Они обычно прикрепляются к прецеденту и содержат представления классов, участвующих в их выполнении.



## Глава 5. Изучение взаимодействия объектов

### Реализация прецедентов

Диаграмма прецедентов представляет внешний вид системы. Выполнение прецедентов отображается с помощью потока событий. Сценарии используются для описания того, как реализуются прецеденты, взаимодействуя между группами объектов.

*Сценарий* (scenario) – это элемент прецедента. Он представляет собой одиничный проход по потоку событий для прецедентов. Сценарии помогают выделить объекты, классы и взаимодействия объектов, необходимые для исполнения единичного действия, определенного прецедентом. Сценарии описывают порядок того, как обязанности, возложенные на прецеденты, распределяются среди объектов и классов в системе. Сценарии говорят на языке конечных пользователей и экспертов и поэтому являются средством выражения их пожеланий по необходимому поведению системы для разработчиков.

Каждый прецедент – это сплетение первичных (нормальный поток для прецедента) и вторичных сценариев (логика ЧТО-ЕСЛИ в прецеденте). Это значит, что существует множество сценариев для системы – первичные и вторичные сценарии для всех прецедентов. На этапе анализа уже можно сказать, что определение первичного сценария для каждого выбранного прецедента будет достаточным. Когда вы обнаружите, что каждый новый сценарий повторяет большинство шагов из предыдущего, то вы добились цели. Данная фаза анализа должна завершаться по мере того, как разработчики продумают приблизительно 80% первичных сценариев и выборочно коснутся вторичных. Если проработать больше сценариев, результаты анализа, вероятно, окажутся хуже; если меньше – не будет достаточно понимания поведения системы, чтобы правильно оценить риски.

По методологии Rational Unified Process реализации прецедентов (use case realizations) отражаются в логическом представлении модели. Обратимся к концепции стереотипов, чтобы показать, что прецеденты, созданные в логическом представлении, являются реализациями прецедентов из представления use case. Другими словами, прецеденты в логическом представлении имеют те же имена, что и в представлении use case, а также стереотип, указывающий на реализацию. В языке UML реализация прецедентов изображается в виде пунктирного овала (см. рис. 5.1). Логическое представление прецедентов обычно отображается на диаграмме прецедентов (или наборе диаграмм), содержащейся в логическом представлении модели.



Название прецедента

Рис. 5.1. Нотация языка UML для реализации прецедента

Создание диаграммы прецедентов в логическом представлении модели в программе Rational Rose состоит из следующих шагов:

1. Щелкните правой кнопкой мыши по папке **Logical View** (Логическое представление) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Use Case** (Создать ⇒ Прецедент). В раздел логического представления модели будет добавлена новая диаграмма прецедентов с названием **New Diagram**.
3. Введите для новой диаграммы название **Realizations**.

Окно браузера с диаграммой прецедентов **Realizations** показано на рис. 5.2.

Последовательность создания реализаций прецедентов в программе Rational Rose:

1. Дважды щелкните по диаграмме прецедентов **Realizations** в списке браузера, чтобы открыть диаграмму.
2. Щелкните по кнопке **Use Case** (Прецедент) на панели инструментов.
3. Щелкните по диаграмме прецедентов. В диаграмму и список браузера будет добавлен новый прецедент.
4. Дважды щелкните по изображению прецедента. На экране появится диалоговое окно **Use Case Specification** (Параметры прецедента).
5. Введите название прецедента (такое же, как у модели) в поле ввода **Name** (Имя). Заметьте, что вы должны указать название в диалоговом окне параметров прецедента или в браузере, чтобы сообщить программе Rational Rose об использовании другого пространства имен (namespace). Если вы введете название прецедента непосредственно на диаграмме, программа Rational Rose будет считать, что это тот же прецедент, что и в представлении use case.
6. В открывшемся списке **Stereotype** (Стереотип) выберите стереотип **use-case realization**.
7. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.

Диаграмма прецедентов **Realizations** показана на рис. 5.3.

Связь между прецедентами в логическом и use case-представлении отражается путем добавления прецедентов из представления use case на диаграмму **Realizations** и соединения с их реализациями посредством односторонней ассоциативной связи с соответствующим стереотипом. (В языке UML используются стереотипные зависимости, но они пока не поддерживаются программой Rational Rose.)

На рис. 5.4 показана связь реализаций с представлением прецедентов на диаграмме функций **Realizations**.

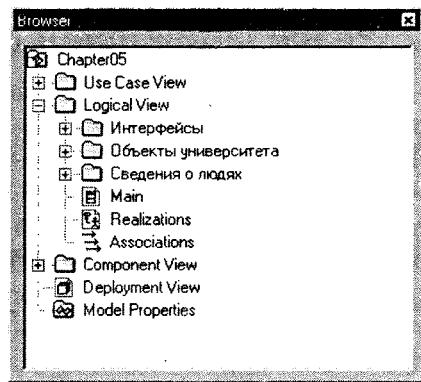


Рис. 5.2. Окно браузера с диаграммой реализаций прецедентов

## Документирование сценариев

61

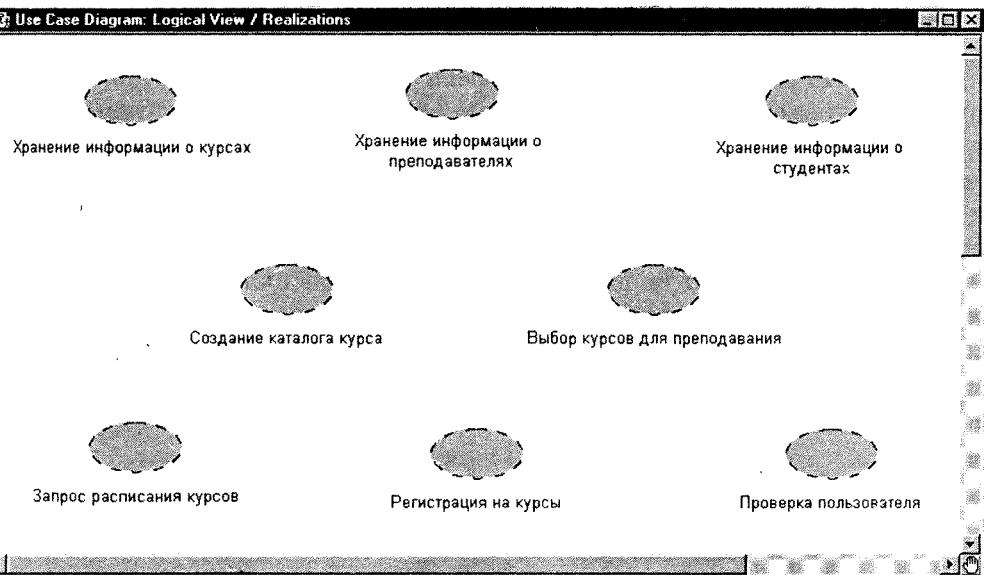


Рис. 5.3. Диаграмма реализаций прецедентов

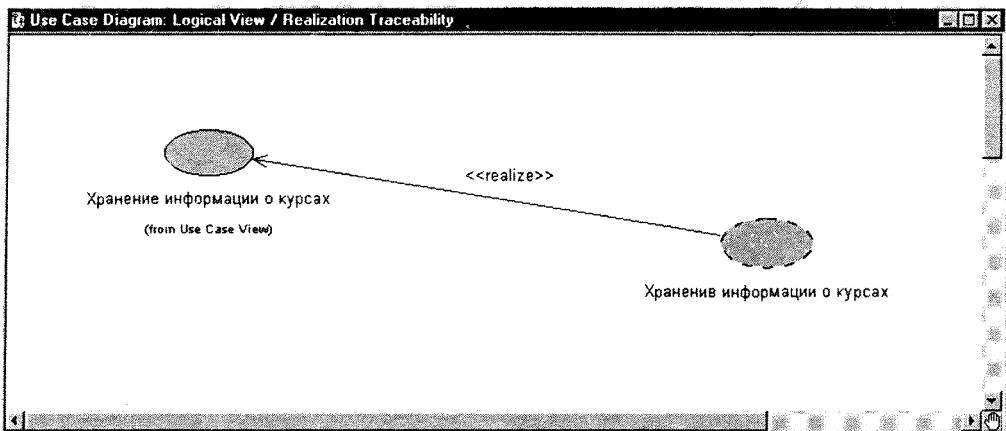


Рис. 5.4. Связь реализаций с представлением прецедентов

## Документирование сценариев

Поток событий для прецедента описывается словами, тогда как сценарии отображаются с помощью диаграмм взаимосвязи (interaction diagrams). Существует два типа диаграмм взаимосвязи – диаграммы последовательности действий (sequence diagrams) и диаграммы взаимодействий (collaboration diagrams). Каждая диаграмма является графическим представлением сценария.

## Диаграммы последовательности действий

*Диаграмма последовательности действий* (sequence diagram) отображает взаимодействие объектов, упорядоченное по времени. На ней показаны объекты и классы, используемые в сценарии, и последовательность сообщений, которыми обмениваются объекты, для выполнения сценария. Диаграммы последовательности действий обычно соответствуют реализациям прецедентов в логическом представлении системы.

В языке UML объект на диаграмме последовательности действий выглядит как прямоугольник, содержащий подчеркнутое название объекта. Название может состоять только из имени объекта, из имени объекта и его класса или только имени класса (анонимный объект). Эти три вида наименований объекта показаны на рис. 5.5.

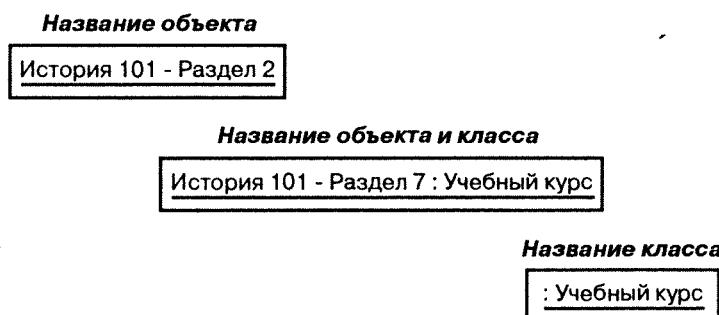


Рис. 5.5. Наименования объектов на диаграмме последовательности действий

Названия объектов могут быть конкретными (например, Алгебра 101, раздел 1) или общими (например, учебный курс). Часто анонимные объекты используются для представления любого объекта данного класса.

Каждый объект также имеет свою временную линию (timeline), изображаемую пунктиром под объектом. Сообщения, передаваемые между объектами, указываются стрелками, направленными от клиента (отправителя сообщения) к поставщику (получателю сообщения).

На рис. 5.6 представлена нотация языка UML для объектов и сообщений на диаграмме последовательности действий.

Для создания диаграммы последовательности действий в программе Rational Rose:

1. Щелкните правой кнопкой мыши по папке **Logical View** (Логическое представление) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Sequence Diagram** (Создать ⇒ Диаграмма последовательности действий). В список браузера будет добавлена новая диаграмма.
3. Введите ее имя.

## Диаграммы последовательности действий

63



Рис. 5.6. Нотация языка UML для объектов и сообщений на диаграмме последовательности действий

Окно браузера с диаграммой последовательности действий показано на рис. 5.7.

Чтобы создать объекты и сообщения на диаграмме последовательности действий в программе Rational Rose:

1. Дважды щелкните по диаграмме последовательности действий в списке браузера, чтобы открыть диаграмму.
2. Выберите из списка актера, щелкнув по нему мышью.
3. Перетащите актера на диаграмму последовательности действий.
4. Щелкните по кнопке **Object** (Объект) на панели инструментов.
5. Щелкните по диаграмме последовательности действий, чтобы добавить новый объект.
6. Введите имя объекта.

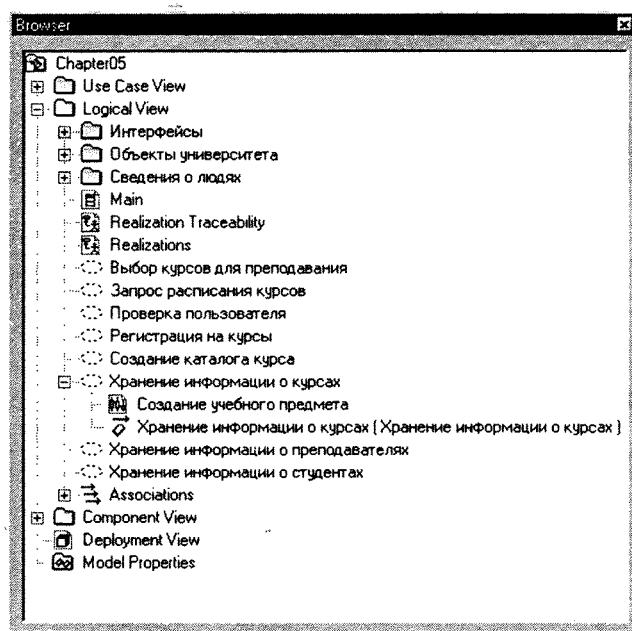


Рис. 5.7. Окно браузера с диаграммой последовательности действий

## Изучение взаимодействия объектов

7. Повторите предыдущие шаги для каждого объекта и актера в сценарии.
8. Щелкните по кнопке **Object Message** (Сообщение) на панели инструментов.
9. Щелкните по актеру или объекту- отправителю сообщения и проведите стрелку сообщения к актеру или объекту- получателю.
10. Введите название сообщения.
11. Повторите шаги с седьмого по девятый для каждого сообщения в сценарии.

Диаграмма последовательности действий для сценария создание учебного предмета (Create a Course) изображена на рис. 5.8.

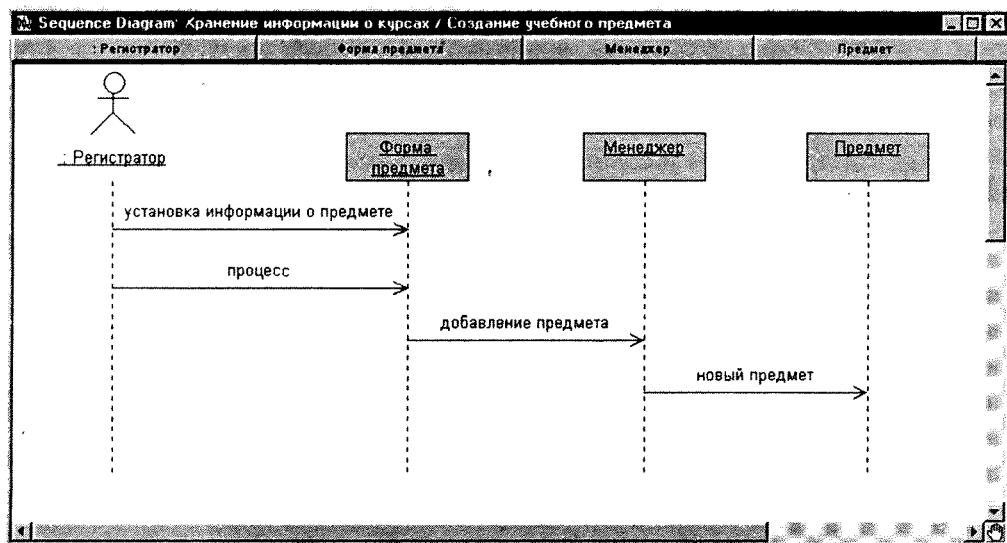


Рис. 5.8. Диаграмма последовательности действий

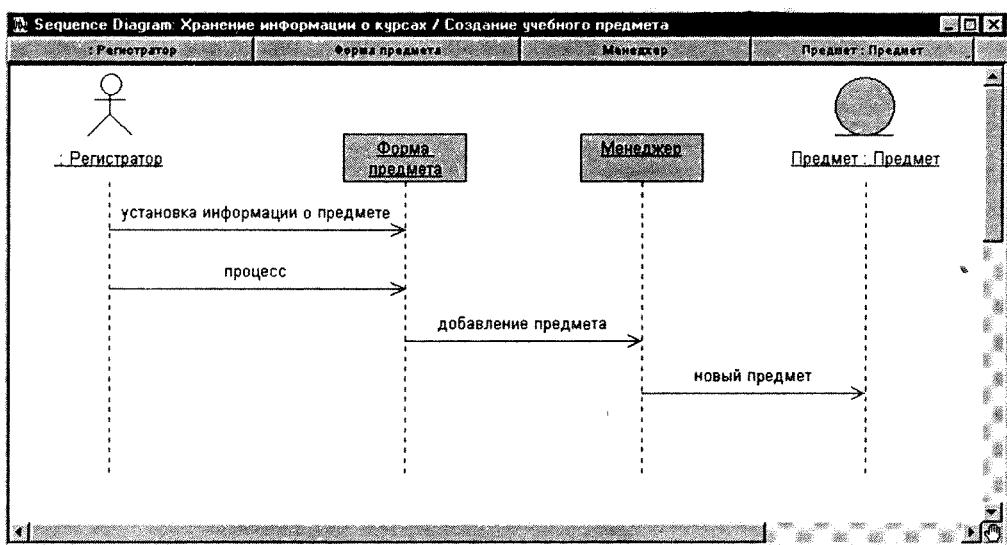


Рис. 5.9. Диаграмма последовательности действий с объектом, присвоенным классу

Присваивание объектов соответствующим классам на диаграмме последовательности действий в программе Rational Rose предусматривает выполнение следующих шагов:

1. В списке браузера выберите класс, щелкнув по нему мышью.
2. Перетащите класс на объект на диаграмме последовательности действий. Программа Rational Rose автоматически добавит имя класса с предшествующим знаком двоеточия к названию объекта. Если у объекта нет имени, название примет вид: : имя класса. Если у стереотипа данного класса есть значок, то он будет использован для изображения объекта на диаграмме.

Диаграмма последовательности действий с объектом предмет (a course), присвоенным классу предмет (Course), показана на рис. 5.9.

## Диаграммы последовательности действий и граничные классы

Граничные классы добавляются на диаграмму последовательности действий для того, чтобы показать взаимодействие с пользователем или другой системой. На стадии анализа назначение граничных классов на диаграмме заключается в описании требований к интерфейсу, но не в описании реализации интерфейса.

Реальные сообщения, поступающие от актера граничному классу, и информация об их последовательности зависят от структуры приложения и определяются на стадии проектирования. Они могут изменяться, по мере того как в систему добавляется информация о способах реализации.

## Сложность и диаграммы последовательности действий

Каждый раз, когда я веду занятия, у слушателей возникает вопрос: «Насколько сложной может быть диаграмма последовательности действий?» Я отвечаю всегда одинаково: «Сохраняйте ее простой». Прелест этой диаграммы в ее простоте – можно легко понять и увидеть объекты, взаимодействия объектов, сообщения между ними и функциональность, задаваемую сценариями.

Следующий вопрос выглядит так: «Что делать с условной логикой?» (с логикой ЕСЛИ-ТО-ИНАЧЕ, которая существует в реальном мире). Ответ на него также субъективен. Если логика проста и требует небольшого количества сообщений, я обычно добавляю ее к одной диаграмме и использую примечания и скрипты для указания выбора, который нужно сделать. С другой стороны, если логика ЕСЛИ-ТО-ИНАЧЕ требует сложных сообщений, я обычно рисую отдельные диаграммы: одну для случая ЕСЛИ, одну для ТО и одну для ИНАЧЕ.

Это делается для сохранения простоты диаграмм. Если нужно, их можно связать друг с другом. Это позволит пользователям перемещаться по набору диаграмм.

Для связывания диаграмм в программе Rational Rose:

1. Щелкните по кнопке **Note** (Сноска) на панели инструментов.
2. Щелкните по диаграмме, чтобы поместить на нее сноска.
3. Выберите в списке браузера диаграмму, которую нужно связать с текущей, и перетащите ее на сноsku.
4. Для перехода на связанную диаграмму необходимо дважды щелкнуть по сноске.

## Диаграммы взаимодействий

*Диаграмма взаимодействий* (collaboration diagram) – это альтернативный способ отображения сценариев. Такой тип диаграммы показывает взаимодействие объектов, организованное вокруг них, и их связи друг с другом. Диаграмма взаимодействий содержит:

- объекты, изображаемые в виде прямоугольников;
- связи между объектами, изображаемые в виде линий;
- сообщения в виде текста и стрелки, направленной от клиента к поставщику.

Нотация языка UML для объектов, связей и сообщений на диаграмме взаимодействий показана на рис. 5.10.

Последовательность создания диаграмм взаимодействий из диаграмм последовательности действий в программе Rational Rose:

1. Дважды щелкните по диаграмме последовательности действий в списке браузера, чтобы открыть диаграмму.
2. Выберите команду меню **Browse ⇒ Create collaboration diagram** (Промтотр ⇒ Создать диаграмму взаимодействий) или нажмите клавишу **F5**.
3. Расположите объекты и сообщения на диаграмме нужным образом.



Рис. 5.10. Нотация языка UML для объектов, связей и сообщений на диаграмме взаимодействий

Диаграмма взаимодействий показана на рис. 5.11.

Можно сначала создать диаграмму взаимодействий. В этом случае диаграмма последовательности действий может быть получена из нее. Для этого необходимо

## Диаграмма последовательности действий

67

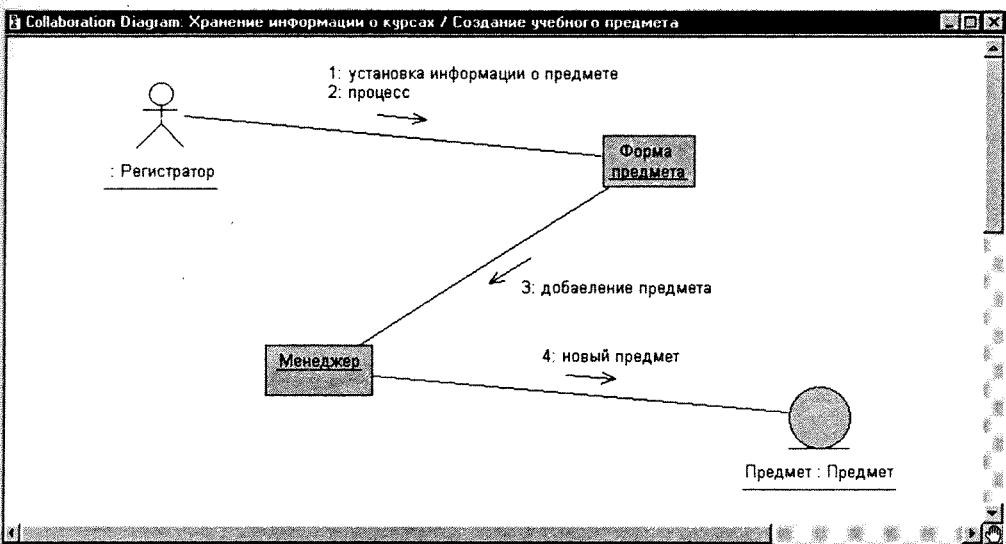


Рис. 5.11. Диаграмма взаимодействий

выбрать команду меню **Browse ⇒ Create Sequence Diagram** (Просмотр ⇒ Создать диаграмму последовательности действий) или нажать клавишу F5.

### Зачем нужны две разные диаграммы

Диаграмма последовательности действий используется для просмотра сценария во временном порядке: что происходит сначала, что происходит затем. Заказчики легко могут читать и понимать такие диаграммы. Поэтому они очень полезны на стадии анализа. Диаграмма взаимодействий представляет общую картину сценария, так как взаимодействия на ней организованы между связанными друг с другом объектами. Такой тип диаграмм чаще используется на этапе проектирования, когда планируется реализация отношений.

### Диаграмма последовательности действий для системы регистрации курсов

Продолжим анализ сценария добавление учебного курса (Add a Course Offering). Диаграмма показана на рис. 5.12.

Диаграммы классов могут быть также прикреплены к реализациям прецедентов. Они содержат представления классов, участвующих в выполнении прецедентов (participating classes).

Последовательность создания представления участвующих классов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по реализации прецедента в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class Diagram** (Создать ⇒ Диаграмма классов).

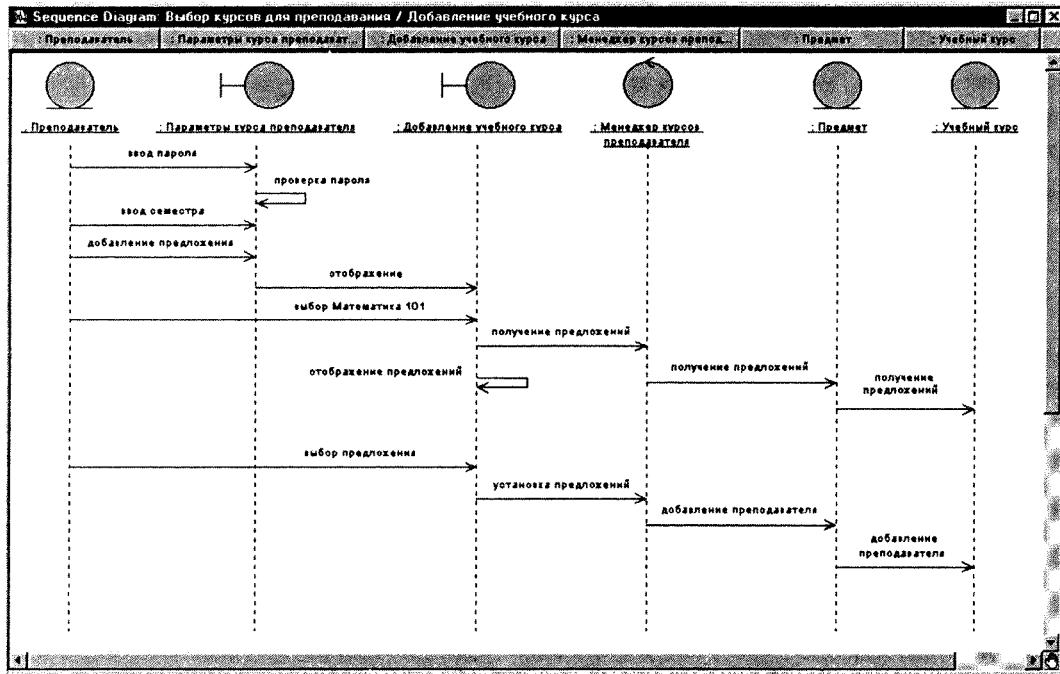


Рис. 5.12. Диаграмма последовательности действий для сценария добавление учебного курса

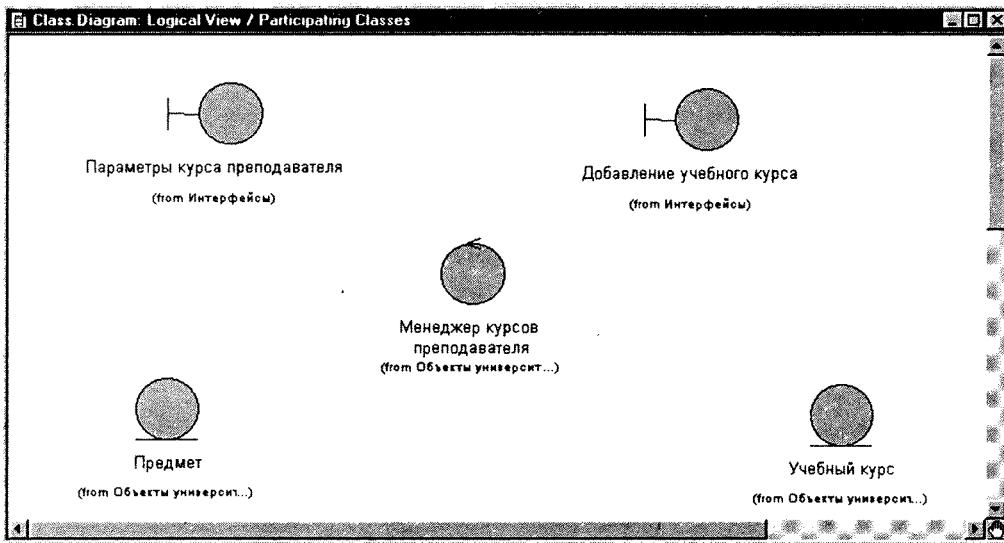


Рис. 5.13. Схема классов, участвующих в прецеденте

3. Введите имя новой диаграммы.
4. Дважды щелкните по новой диаграмме в списке браузера, чтобы открыть ее.
5. Выберите класс в логическом представлении модели и перетащите его на диаграмму с помощью мыши.
6. Аналогичным образом поместите на диаграмму другие нужные классы.

Участвующие классы для прецедента выбор предметов для обучения показаны на рис. 5.13.

## Резюме

Диаграмма прецедентов представляет внешний вид системы. Выполнение прецедентов отображается с помощью потока событий. Сценарии используются для описания того, как прецеденты реализуются в виде взаимодействия между группами объектов. Сценарий – это экземпляр прецедента. Он представляет собой одиночный проход по потоку событий для прецедента. Таким образом, каждый прецедент – это сплетение сценариев. Они помогают выделить объекты, классы и взаимодействия объектов, необходимые для исполнения единичного действия, определенного прецедентом.

Поток событий для прецедентов обычно описывается словами, тогда как сценарии – диаграммами взаимосвязи. Существует два типа диаграмм взаимосвязи – диаграммы последовательности действий (sequence diagrams) и диаграммы взаимодействий (collaboration diagrams). Каждая диаграмма – это графическое представление сценария.

Диаграмма последовательности действий отображает взаимодействие объектов, упорядоченное по времени. Диаграмма взаимодействий – это альтернативный способ отображения сценариев. Этот тип диаграммы показывает взаимодействие объектов, организованное вокруг самих объектов, и их связи друг с другом.



## Глава 6. Определение отношений

### Необходимость отношений

Система состоит из большого количества классов и объектов. Ее поведение обеспечивается взаимодействием объектов. Например, студент добавляется к курсу, когда на курс поступает сообщение добавить студента. В этом случае часто говорят, что объект посылает сообщение другому объекту. Отношения выполняют функцию проводников между объектами. Два типа отношений, которые можно выделить на этапе анализа, – это ассоциация и агрегация.

### Ассоциативные отношения

*Ассоциация* (association) – это двунаправленная семантическая связь между классами. Это не поток данных, определяемый в структурном анализе и проектировании, – данные могут поступать в обоих направлениях ассоциативной связи. Наличие ассоциаций между классами говорит о том, что объекты этих классов взаимосвязаны. Например, ассоциативные отношения между классами предмет (Course) и менеджер курсов преподавателя (ProfessorCourseManager) означают, что объекты класса предмет связаны с объектами класса менеджер курсов преподавателя. Количество связанных объектов зависит от мощности ассоциативных отношений. В языке UML ассоциативные отношения изображаются в виде линии, соединяющей связанные объекты, – см. рис. 6.1.

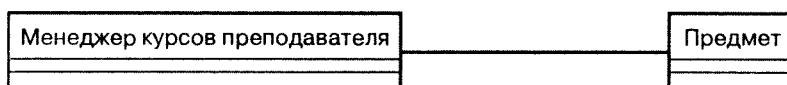


Рис. 6.1. Нотация языка UML для ассоциативного отношения

Последовательность создания ассоциативных отношений в программе Rational Rose:

1. На панели инструментов щелкните по кнопке **Association** (Ассоциация). Если она отсутствует, щелкните правой кнопкой мыши на панели инструментов и выберите команду **Customize** (Настройка) в появившемся контекстно-зависимом меню.
2. Щелкните по одному из классов на диаграмме классов.
3. Перетащите возникшую линию ассоциативной связи на второй класс.

Ассоциативное отношение между классами показано на рис. 6.2.

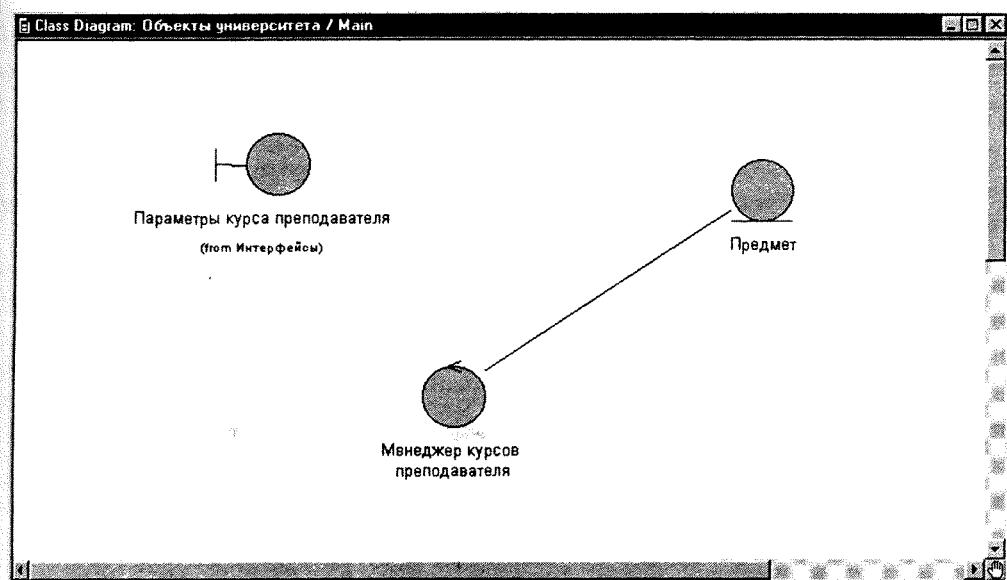


Рис. 6.2. Ассоциативное отношение

## Агрегационные отношения

*Агрегационное отношение* – это специальная форма ассоциации между целым и его частью или частями. Агрегация известна как отношение типа «часть от» или «содержит». В языке UML она изображается так же, как ассоциация, но с ромбом на конце линии связи, означающим класс-агрегат (целое), – см. рис 6.3.



Рис. 6.3. Нотация языка UML для агрегационного отношения

Чтобы определить, является ли ассоциативная связь агрегационной, воспользуйтесь следующими тестовыми вопросами:

1. Можно ли применить фразу «часть от», чтобы описать отношение?
2. Происходит ли автоматическое применение некоторых операций над целым к его частям (например, удаление предмета (Course) ведет к удалению всех относящихся к нему учебных курсов (CourseOffering))?
3. Существует ли выраженная асимметрия в отношении, когда один класс подчинен другому?

Например, предмет (математика 101) может читаться несколько раз в течение семестра. Каждый курс лекций по предмету представлен как учебный курс (то есть математика 101, раздел 1 или математика 101, раздел 2). Отношения

между предметом и учебным курсом моделируются как агрегация – предмет содержит несколько учебных курсов.

Для создания агрегационных отношений в программе Rational Rose:

1. На панели инструментов щелкните по кнопке **Aggregation** (Агрегация). Если она отсутствует, щелкните правой кнопкой мыши по панели инструментов и в появившемся контекстном меню выберите команду **Customize** (Настройка) для добавления кнопки.
2. На диаграмме классов щелкните по классу, выступающему в качестве целого, и перетащите возникшую линию агрегационной связи на класс, являющийся частью.

Агрегационное отношение между классами показано на рис. 6.4.

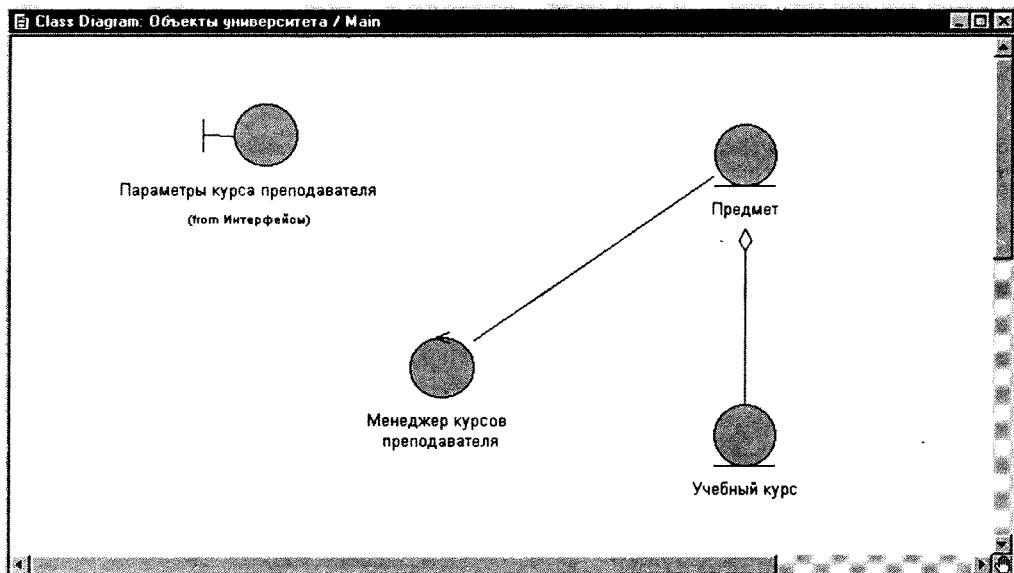


Рис. 6.4. Агрегационное отношение

## Ассоциация или агрегация

Если два класса жестко связаны отношением «целое-часть» – это типичное агрегационное отношение. «Использование агрегации иногда является условным решением. Часто далеко не очевидно, должна ли ассоциация моделироваться как агрегация. Однако, если вы аккуратно и последовательно рассуждали, небольшое различие между агрегацией и обычной ассоциацией не вызовет практических проблем»<sup>1</sup>.

Является ли отношение ассоциацией или агрегацией, часто зависит от предметной области. Какой тип отношений нужно выбрать для моделирования машины и шин? Если система предназначена для сервисного центра, и единственная причина, по которой вы рассматриваете шины, – их принадлежность к обслуживаемой

<sup>1</sup> Rumbaugh, James et al. *Object-Oriented Modeling and Design*. Upper Saddle River, NJ: Pentice Hall, 1991, p. 58.

машине, отношение должно быть агрегацией. Однако, если система предназначена для магазина шин, вы будете рассматривать шины независимо от автомобиля, следовательно, отношение должно быть ассоциативным.

## Именование отношений

Ассоциации можно присвоить имя. Обычно в названии ассоциации используется глагол или фраза с глаголом, отражающая смысл связи. Так как фраза с глаголом предполагает направление чтения, желательно, чтобы название корректно читалось слева направо или сверху вниз. Слова можно изменить, чтобы читать название связи в другом направлении, например: преподаватель читает курс, курс читается преподавателем.

Надо отметить, что присваивать ассоциации имя необязательно. Название дается в том случае, если оно придает большую ясность модели. Агрегационные отношения обычно не имеют названий, потому что читаются с использованием слов «имеет» или «содержит».

Чтобы указать названия отношений в программе Rational Rose:

1. На диаграмме классов выделите линию связи, щелкнув по ней мышью.
2. Введите название отношения (см. рис. 6.5).

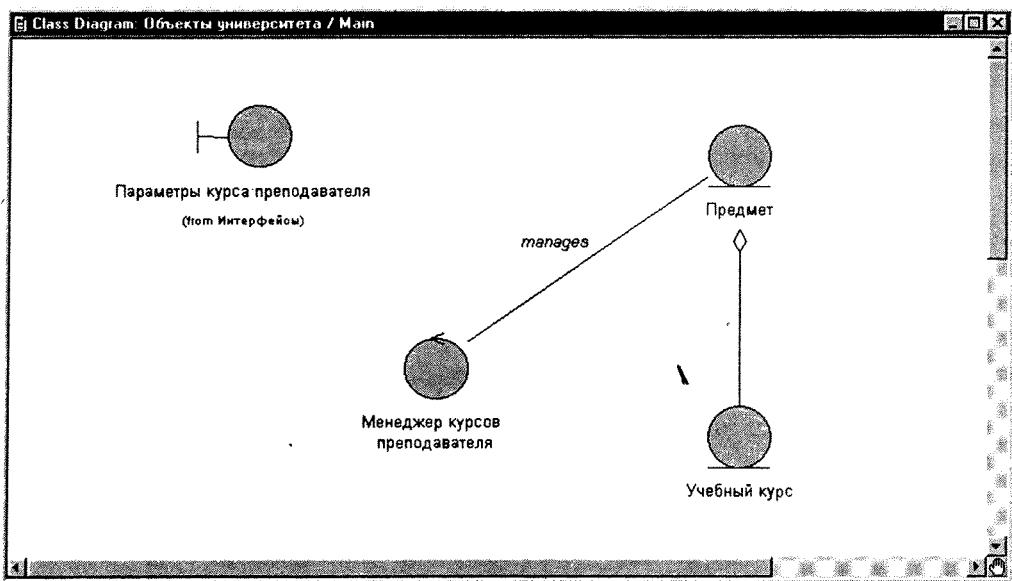


Рис. 6.5. Отношение с названием

## Именование ролей

Окончание линии ассоциации в месте, где она соединяется с классом, называется *ролью ассоциации*. Название роли может быть использовано вместо названия ассоциации. Для этой цели выбирают существительное, описывающее роль, в которой один класс выступает в связи с другим классом. Название роли помещается на

линии связи около класса, к которому оно применяется, – с одного или с обоих концов линии связи. Обычно нет необходимости использовать одновременно названия роли и отношения.

Для ввода названия роли в программе Rational Rose:

1. Щелкните правой кнопкой мыши по линии ассоциативной связи рядом с классом, к которому применяется роль.
2. В появившемся контекстно-зависимом меню выберите команду **Role Name** (Название роли).
3. Введите название роли.

На рис. 6.6 показана ассоциативная связь с назначением роли.

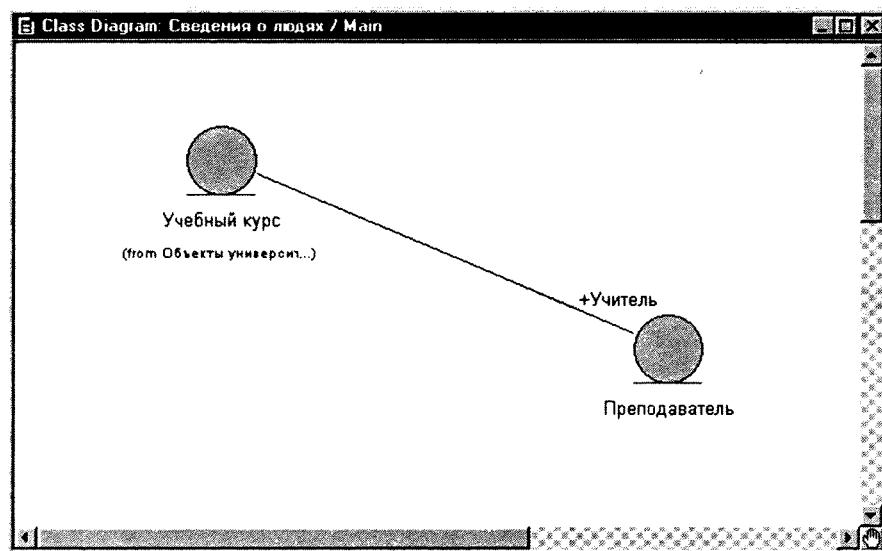


Рис. 6.6. Название роли

Отношение, изображенное на рис. 6.6, можно прочитать в обоих направлениях:

- преподаватель играет роль учителя для учебного курса;
- учебный курс имеет отношение к преподавателю, играющему роль учителя.

В вопросе присваивания названий отношениям или ролям стандартов нет. Однако многие разработчики предпочитают использовать названия ролей вместо названий ассоциаций, потому что первые лучше передают смысл связи. Это характерно для двунаправленных отношений, так как для них трудно подобрать глагольное выражение, которое бы читалось корректно в обоих направлениях. Какой, например, глагол можно применить в названии ассоциации на рис. 6.6? А при использовании названия роли значение понятно в обоих направлениях.

Названия ассоциативных отношений и ролей применяются только для придания ясности модели. Если существует отношение между компанией и сотрудником, можно подобрать глагол «работает» для названия ассоциации или существительные «сотрудник» и «работодатель» для названия ролей, чтобы показать

смысл связи. Если же дать классам имена сотрудник (employee) и работодатель (employer), необходимость в дополнительных названиях отпадет, так как смысл отношения понятен по названиям классов.

## Мощность отношений

**Мощность** (multiplicity) отношения указывается для классов и определяет допустимое количество объектов, участвующих в отношении с каждой стороны. Есть два индикатора мощности для каждого отношения ассоциации или агрегации – по одному с каждой стороны линии связи. Перечислим основные индикаторы мощности:

- 1 – ровно один;
- 0...\* – ноль или больше;
- 1...\* – один или больше;
- 0..1 – ноль или один;
- 5..8 – определенный диапазон (5, 6, 7 или 8);
- 4..7, 9 – комбинация (4, 5, 6, 7 или 9).

Чтобы определить мощность в программе Rational Rose:

1. Дважды щелкните по линии связи на диаграмме – откроется диалоговое окно **Specification** (Параметры).
2. Выберите вкладку **Detail** (Детально) для нужной роли.
3. Укажите требуемое значение мощности в поле **Cardinality** (Численное отношение).
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров отношения.

Индикаторы мощности показаны на рис. 6.7.

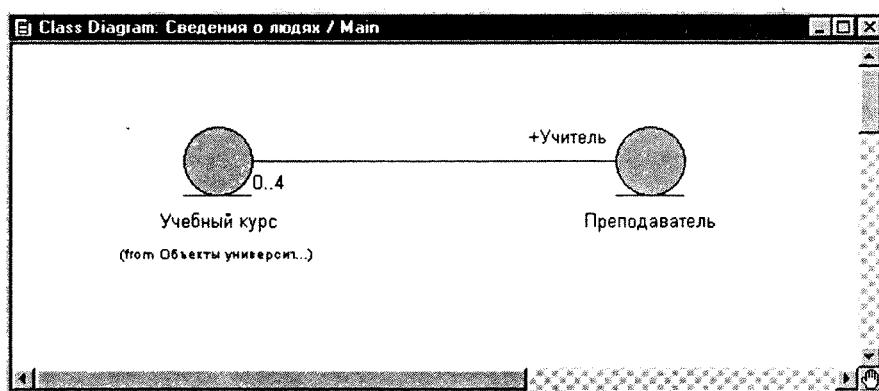


Рис. 6.7. Название роли

Диаграмму на рис. 6.7 можно объяснить так:

- один объект учебный курс связан с одним объектом преподаватель, который играет роль учителя. Например: математика 101, раздел 1

(объект учебный курс) имеет отношение к профессору Смиту (объект преподаватель);

- один объект преподаватель в роли учителя связан с объектами учебный курс в количестве от нуля до четырех. Например: профессор Смит (объект преподаватель) читает курсы математика 101, раздел 1; алгебра 200, раздел 2; дифференциальное исчисление 1, раздел 3 (объекты учебный курс). Так как мощность связи ограничена значениями от нуля до четырех, с объектом преподаватель может быть связано от нуля до четырех объектов учебный курс.

## Возвратные отношения

Несколько объектов, принадлежащих одному классу, могут взаимодействовать друг с другом. Такое взаимодействие показывается на диаграмме классов как **возвратная** (reflexive) ассоциация или агрегация. Для возвратных отношений обычно используется название роли, а не отношения.

Последовательность создания возвратного отношения в программе Rational Rose:

1. На панели инструментов щелкните по кнопке **Association** (Ассоциация) или **Aggregation** (Агрегация).
2. Щелкните по классу и проведите линии связи на свободное место диаграммы.
3. Отпустите кнопку мыши.
4. Щелкните по линии связи и перетащите ее обратно на тот же класс.
5. Введите название роли и мощность для каждого конца возвратной ассоциации или агрегации.

Возвратное отношение на рис. 6.8 можно интерпретировать следующим образом:

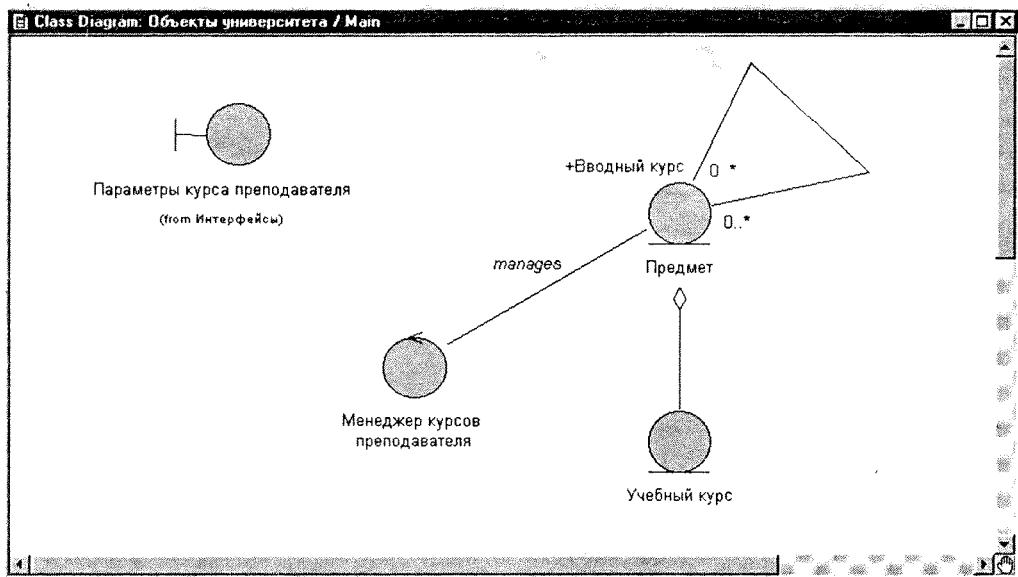


Рис. 6.8. Возвратное отношение

- один объект предмет, выступающий в качестве вводного курса (prerequisite), связан с нулем или более объектами предмет;
- один объект предмет связан с нулем или более объектами предмет, выполняющими функцию вводных курсов.

## Поиск отношений

Чтобы определить наличие отношений между двумя классами, изучают сценарии. Передача сообщений между объектами указывает на то, что последние взаимодействуют друг с другом. Ассоциации и агрегации обеспечивают путь для взаимодействия.

Отношения могут быть также выявлены на основе сигнатуры операций (см. главу 7).

### Отношения в системе регистрации учебных курсов

Взаимодействующие объекты и типы отношений, определенные для сценария добавить учебный курс, перечислены в табл. 6.1.

Таблица 6.1. Отношения между классами

Начальный класс	Конечный класс	Тип отношения
Параметры курса преподавателя	Добавление учебного курса	Агрегация
Добавление учебного курса	Менеджер курсов преподавателя	Ассоциация
Менеджер курсов преподавателя	Предмет	Ассоциация
Предмет	Учебный курс	Агрегация

Диаграмма классов с указанными отношениями изображена на рис. 6.9.

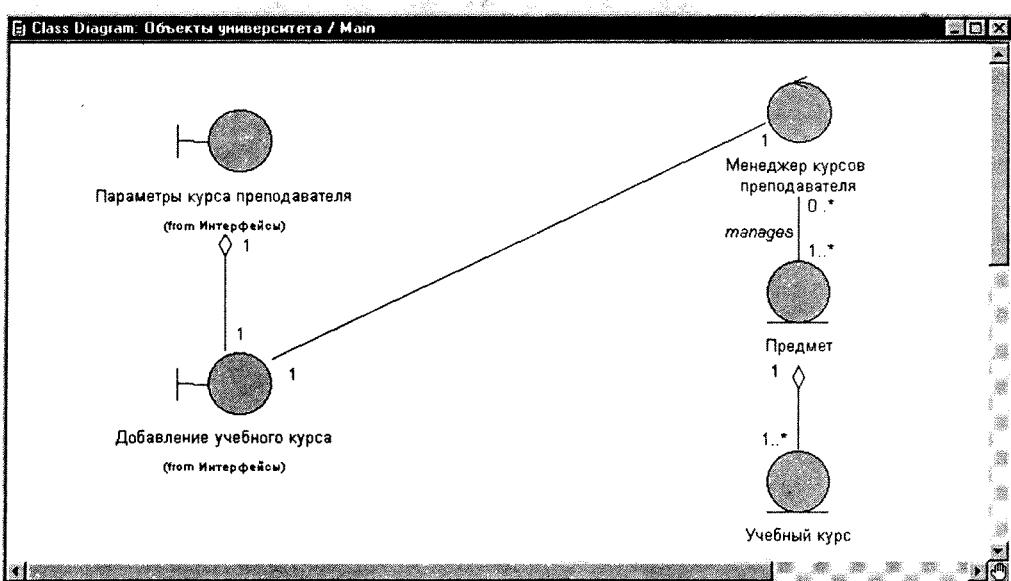


Рис. 6.9. Отношения в сценарии добавить учебный курс

## Отношения между пакетами

Отношения между пакетами также включают в модель. Такой тип связи является отношением зависимости и изображается в виде пунктирной стрелки, направленной к зависимому пакету (см. рис. 6.10). Если пакет А зависит от пакета В, значит, один или несколько классов в пакете А инициируют связь с одним или более общедоступными классами в пакете В. Пакет А в этом случае называется пакетом-клиентом (client package), а пакет В – пакетом-поставщиком (supplier package).

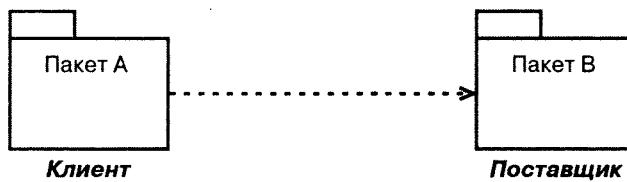


Рис. 6.10. Отношения между пакетами

Отношения между пакетами также выявляются путем изучения сценариев и отношений между классами системы. Так как это итеративный процесс, отношения могут изменяться в ходе анализа и проектирования.

### Отношения между пакетами в системе регистрации учебных курсов

В сценарии добавить учебный курс класс добавление учебного курса отправляет сообщение классу менеджер курсов преподавателя. Это указывает на наличие связи между пакетами Интерфейсы и Объекты университета. На

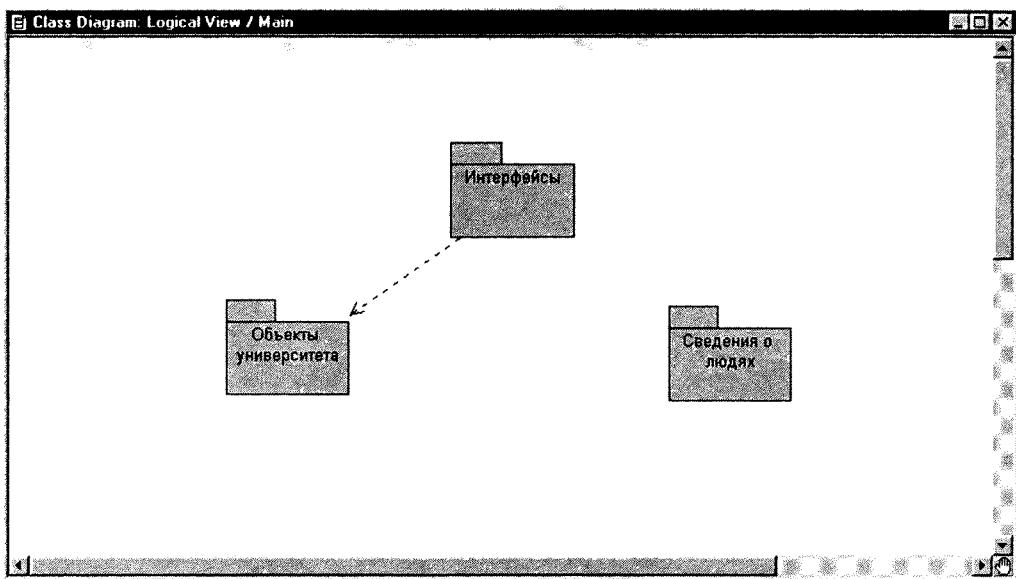


Рис. 6.11. Отношения между пакетами в системе регистрации учебных курсов

данном этапе мы не можем выделить какие-либо отношения с пакетом Сведения о людях.

Для создания отношений между пакетами в программе Rational Rose:

1. Щелкните по кнопке **Dependency Relationship** (Отношение зависимости) на панели инструментов.
2. Щелкните по пакету-клиенту и перетащите линию связи к пакету-поставщику.

Отношения между пакетами в системе регистрации учебных курсов показаны на рис. 6.11.

## Резюме

Отношения выступают в качестве проводника между объектами. Два типа объектных отношений, которые можно выделить на этапе анализа, – это ассоциации и агрегации. Ассоциацией называется двунаправленная семантическая связь между классами. Агрегация – это специальная форма ассоциации между целым и его частью или частями.

Ассоциации можно дать название. Обычно для этой цели используется глагол или фраза с глаголом, отражающая смысл связи. Вместо названия ассоциации может быть использована роль. Для ее названия выбирают существительное, описывающее роль, в которой один класс связан с другим классом.

Мощность определяет количество экземпляров класса, участвующих в отношении. Есть два индикатора мощности для каждого отношения ассоциации или агрегации – по одному с каждой стороны линии связи.

Несколько объектов, принадлежащих одному классу, могут взаимодействовать друг с другом. Такое взаимодействие изображается на диаграмме классов как возвратная ассоциация или агрегация.

Для выявления отношений между двумя классами изучают сценарии.

Пакеты могут быть связаны отношением зависимости. Если пакет А зависит от пакета В, значит, один или несколько классов в пакете А инициируют связь с одним или более общедоступными классами в пакете В.



## Глава 7. Добавление поведения и структуры

### Представление поведения и структуры

Класс реализует ряд обязанностей, от которых зависит поведение его объектов. Обязанности исполняются с помощью определенных для класса операций. Необходимо, чтобы операция выполняла только одну задачу и выполняла ее хорошо. Например, класс учебный курс (*CourseOffering*) должен добавлять и исключать студента. Для этой цели используются две операции: одна добавляет студента, другая – исключает. За всеми экземплярами класса закреплены соответствующие операции.

*Структура* (structure) объекта описывается атрибутами класса. Каждый атрибут – это поле данных, содержащееся в объекте класса. Объект, созданный на основе класса, наделен значениями всех атрибутов класса. Например, класс предмет (*course*) имеет следующие атрибуты: название (*name*), описание (*definition*) и количество учебных часов (*credit hours*). Следовательно, каждый объект предмет будет содержать значения перечисленных атрибутов. Они могут повторяться, так как в университете существуют учебные предметы с одинаковым количеством академических часов.

При формировании имен атрибутов и операций используется определенный стиль, благодаря которому достигается единообразие в описании классов и становится удобно работать с моделью и кодом.

Если какой-либо объект класса не наделен атрибутами или операциями, проверьте определение класса. Это может означать отсутствие целостности класса и необходимость его разделения. Предположим, что класс учебный курс (*CourseOffering*) имеет следующие атрибуты: номер курса (*offerNumber*), место занятий (*location*), время занятий (*timeOfDay*), факультет (*department*), количество курсов на факультете (*numberOfferingInDepartment*). Он может быть осведомлен о своем факультете, но информация о количестве других курсов на факультете ему не нужна.

Лучшая модель получилась бы при использовании класса учебный курс, связанного с классом факультет (*Department*). Это подтверждает общее правило о том, что класс должен представлять одну сущность.

### Создание операций

Сообщения на диаграммах взаимодействий обычно отображаются на соответствующие операции в классах-получателях. Однако в особых случаях сообщения

не становятся операциями. Если класс-получатель является граничным классом, представляющим графический интерфейс пользователя (GUI), сообщения отражают требования к интерфейсу. Такие сообщения реализуются обычно в виде элементов управления (кнопок и т.п.) и не отражаются на операции, так как требуемое поведение уже заложено в стандартных элементах управления. Например, актер преподаватель должен ввести пароль, чтобы запустить сценарий добавление учебного курса (*Add a Course Offering*). Это представлено в виде сообщения, направляемого граничному классу параметры курса преподавателя (*ProfessorCourseOptions*). Оно никогда не станет операцией в интерфейсном классе, а, скорее, будет реализовано в виде поля ввода в окне программы. Сообщения, поступающие актерам и от актеров, также требуют отдельного рассмотрения. Если актер является человеком, сообщение отражает действия человека, следовательно, должно быть реализовано в виде фрагмента руководства пользователя, а не в виде операции. В сценарии добавление учебного курса актер преподаватель имеет определенный пароль для доступа к системе – это обязательное требование, которое должно быть отражено в руководстве. Когда актер является внешней системой, создается отдельный класс, реализующий протокол взаимодействия с внешней системой. В этом случае сообщения отображаются на операции данного класса.

Для названия операции следует использовать только термины класса, выполняющего операцию, а не класса, запрашивающего выполнение операции. Например, операция добавления студента к учебному курсу называется добавить студента (*addStudent*). Кроме того, имена операций не должны отражать способ их выполнения, потому что он может измениться. К примеру, объект учебный курс может иметь не более десяти прикрепленных студентов.

Вам, вероятно, потребуется выяснить, сколько студентов прикреплено к объекту учебный курс в данный момент. Это значение вычисляется при подсчете связей между объектами учебный курс и студент. Если операция называется посчитать число студентов (*calculateNumberOfStudent*), следовательно, нужно использовать метод подсчета. Однако через год реализация может измениться и информация о количестве студентов будет храниться, например, в файле. Поэтому лучше назвать операцию получить число студентов (*getNumberOfStudent*). Это название не указывает на способ реализации операции.

Для отображения сообщений на операции в программе Rational Rose:

1. Присвойте объекты соответствующим классам, если вы не сделали этого ранее.
2. Щелкните правой кнопкой мыши по стрелке, отображающей сообщение.
3. В появившемся контекстно-зависимом меню выберите команду **New Operation** (Новая операция) – увидите диалоговое окно **Operation Specification** (Параметры операции).
4. Введите в нем имя операции.
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.

Если необходимая операция для данного класса уже существует, ее не нужно создавать заново – просто выберите в списке операций класса.

Диаграмма последовательности действий с операциями показана на рис. 7.1. Операции создаются независимо от диаграмм взаимодействий, так как не все сценарии изображаются в виде диаграмм. Это справедливо для операций, являющихся вспомогательными для других операций. Например, класс предмет, перед тем как зарегистрировать преподавателя, должен проверить, имеет ли тот право читать указанный курс. Для этой цели как нельзя лучше подойдет вспомогательная операция проверить преподавателя (`validateProfessor`).

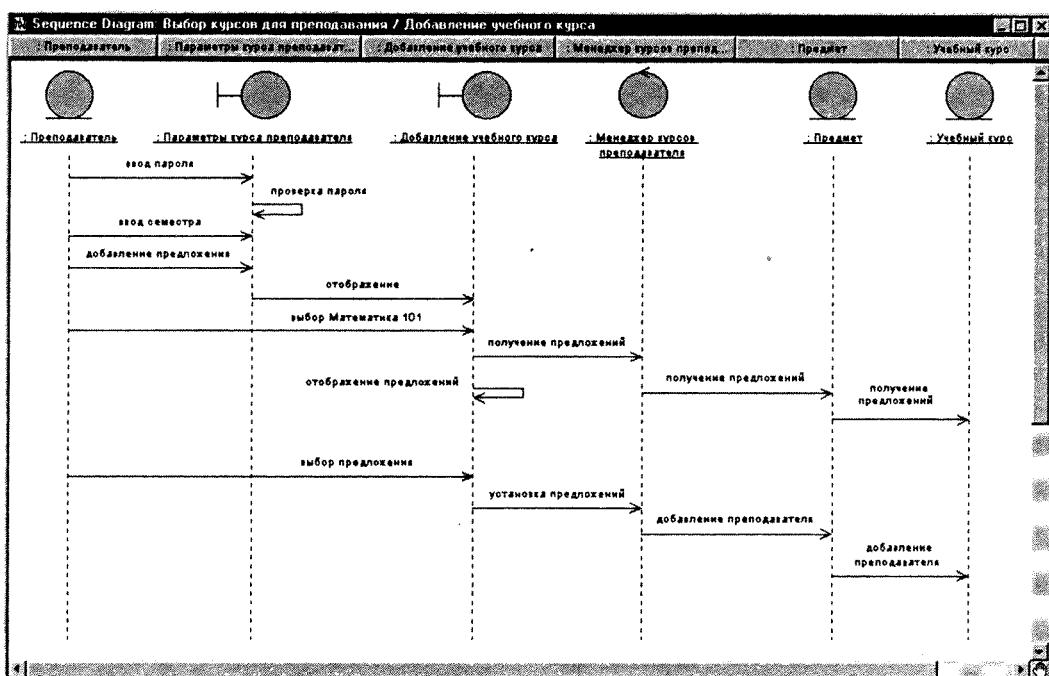


Рис. 7.1. Диаграмма последовательности действий с операциями

Чтобы создать операции в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Operation** (Создать ⇒ Операция).
3. Введите имя новой операции.

Операции для класса предмет (Course) показаны на рис. 7.2.

## Документирование операций

Каждая операция должна быть описана в документации, чтобы человек, изучающий модель, мог легко понять ее назначение. Описание должно отражать функциональность операции, а также содержать комментарии о входных параметрах и возвращаемом результате, если они имеются. Входные и возвращаемые параметры составляют *сигнатуру операции* (operation signature). Эта информация

может отсутствовать на начальном этапе и добавляться на последующих стадиях жизненного цикла, когда будут собраны достаточные сведения о классе.

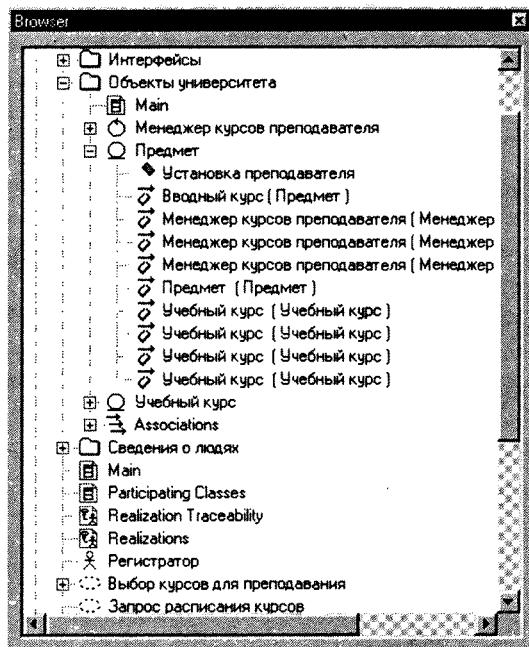


Рис. 7.2. Операции в списке браузера

Для описания операций в программе Rational Rose:

1. Щелкните в окне браузера по значку «+» слева от имени класса, чтобы раскрыть список его свойств.
2. Выберите нужную операцию, щелкнув по ней мышью.
3. Установите курсор в окне описания и введите описание операции.

Описание операции выбрать преподавателя (SetProfessor) класса предмет (Course) показано на рис. 7.3.

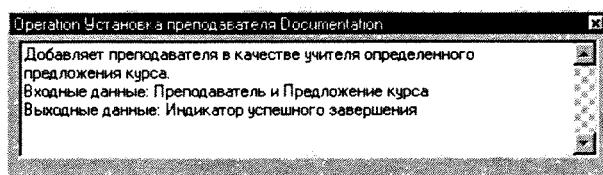


Рис. 7.3. Описание операции выбрать преподавателя

## Отношения и сигнатуры операций

Сигнатурой операции может обозначать отношение. Если класс, передаваемый как аргумент или возвращаемый операцией, является фундаментальным (fundamental) классом, как, например, строка (String), отношения обычно не выносятся на

диаграмму. Для нефундаментальных классов отношения отражаются на одной или нескольких диаграммах. Например, входными параметрами для операции зарегистрировать преподавателя (`setProfessor`) в классе предмет (`Course`) являются классы преподаватель (`Professor`) и учебный курс (`CourseOffering`). Это значит, что существуют отношения:

- между классами предмет и преподаватель;
- между классами предмет и учебный курс.

Отношения, основанные на сигнатурах, изначально моделировались как ассоциации, но в ходе проектирования системы они могут быть пересмотрены и представлены как отношения зависимости (dependency relationships). (Об уточнении и изменении отношений говорится в главе 12.) Взаимосвязи между пакетами также могут быть пересмотрены, по мере того как в модель включаются отношения, основанные на сигнатурах операций. К примеру, мы добавили в систему отношение между классами предмет и преподаватель. Значит, между пакетами Объекты университета и Сведения о людях существует отношение зависимости.

## Создание атрибутов

Большинство атрибутов класса выявляется при анализе предметной области, системных требований и описаний потоков событий, а также при составлении описания класса. Кроме того, хорошим источником для определения атрибутов является сама предметная область. Например, в требованиях к системе указано, что информация о названии предмета, его описании и количестве учебных часов

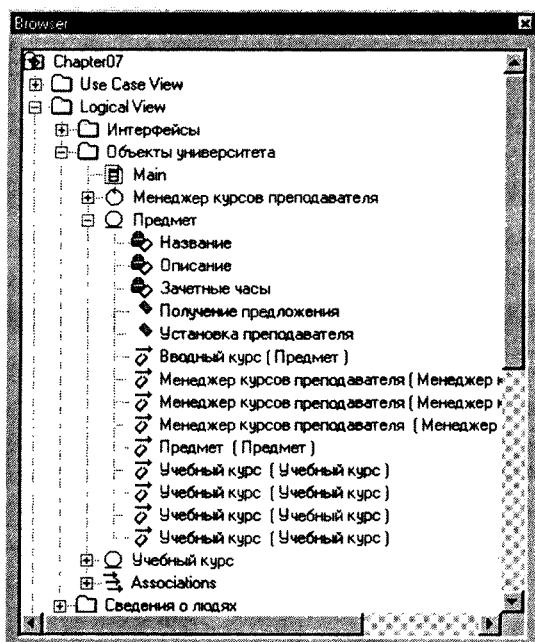


Рис. 7.4. Атрибуты в списке браузера

содержится в каталоге учебных курсов на семestr. Из этого следует, что название, описание и количество учебных часов – это атрибуты класса предмет.

Последовательность создания атрибутов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Attribute** (Создать ⇒ Атрибут).
3. Введите имя нового атрибута.

Атрибуты для класса предмет (Course) показаны на рис. 7.4.

## Документирование атрибутов

Определения атрибутов в документации должны быть краткими и четкими и содержать информацию о назначении атрибута, а не о его структуре. Приведу неудачный пример описания атрибута название класса предмет: «Символьная строка длиной до 15 знаков». Правильным будет следующий вариант: «Название учебного предмета, которое используется в университетских изданиях».

Для описания атрибутов в программе Rational Rose:

1. В окне браузера щелкните по значку «+» слева от имени класса, чтобы раскрыть список его свойств.
2. Выберите атрибут, щелкнув по нему мышью.
3. Установите курсор в окне описания и введите описание для атрибута класса.

Описание атрибута название (name) класса предмет (Course) показано на рис. 7.5.

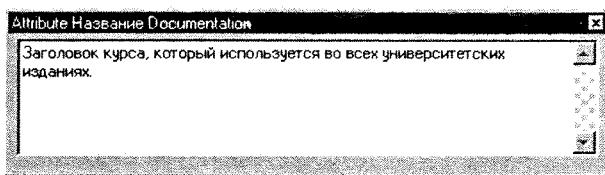


Рис. 7.5. Описание атрибута название

## Отображение атрибутов и операций

Атрибуты и операции можно показать на диаграмме классов. Чаще всего она создается именно для отражения структуры и поведения классов пакета. Отношения на эту диаграмму обычно не выносятся.

Последовательность создания диаграммы классов для отображения атрибутов и операций пакета:

1. Щелкните правой кнопкой мыши по пакету в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class Diagram** (Создать ⇒ Диаграмма классов). В список браузера будет добавлена диаграмма **New Diagram**.
3. Введите имя новой диаграммы.

Для добавления классов на диаграмму с помощью меню **Query** (Запрос):

1. Откройте диаграмму классов, дважды щелкнув по ней мышью в окне браузера.
2. Выберите команду меню **Query** ⇒ **Add Classes** (Запрос ⇒ Добавить классы).
3. Укажите нужный пакет.
4. Выберите классы с помощью мыши и щелкните по кнопке **>>>**, чтобы добавить классы на диаграмму. Для размещения на диаграмме всех классов щелкните по кнопке **All >>** (Все).

Последовательность фильтрации отношений в программе Rational Rose:

1. Откройте диаграмму, дважды щелкнув по ней мышью в окне браузера.
2. Выберите команду меню **Query** ⇒ **Filter Relationships** (Запрос ⇒ Фильтрация отношений).
3. Отметьте позицию **None** (Нет) переключателя **Type** (Тип), чтобы скрыть все отношения на диаграмме.
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Relations** (Отношения).

Чтобы отобразить определенные атрибуты или операции в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в окне диаграммы.
2. В появившемся контекстно-зависимом меню выберите команду **Options** ⇒ **Select Compartment Items** (Настройки ⇒ Выбрать элементы секции).
3. С помощью мыши укажите атрибуты и операции, которые требуется отобразить на диаграмме.
4. Щелкните по кнопке **>>>**.
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Edit Compartment** (Настройка секции).

Для отображения всех атрибутов и операций в программе Rational Rose выполните следующие действия:

1. Щелкните правой кнопкой мыши по классу в окне диаграммы.
2. В появившемся контекстно-зависимом меню выберите команду **Options** ⇒ **Show All Attributes** (Настройки ⇒ Показать все атрибуты).
3. Снова вызовите контекстно-зависимое меню для класса и выберите команду **Options** ⇒ **Show All Operations** (Настройки ⇒ Показать все операции).

Атрибуты и операции класса будут всегда отображаться на диаграммах, если установить флагки **Show All Attributes** (Показать все атрибуты) и **Show All Operations** (Показать все операции) в диалоговом окне настройки параметров программы, вызываемом командой меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры).

Чтобы отобразить стереотипы классов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в окне диаграммы.
2. В появившемся контекстно-зависимом меню выберите команду **Options** ⇒ **Stereotype Display** (Настройки ⇒ Отображение стереотипов), а затем в меню третьего уровня выберите один из вариантов отображения стереотипов: **None**

(Нет) – не выводить, **Label** (Название) – отображать названия в треугольных скобках, **Icon** (Значок) – отображать класс, используя значок стереотипа.

Диаграмма классов с названием «Атрибуты и операции» (Attributes and Operations) для пакета Объекты университета (UniversityArtifacts) показана на рис. 7.6. Для такого типа диаграмм я предпочитаю выводить стереотипы классов в виде названий.

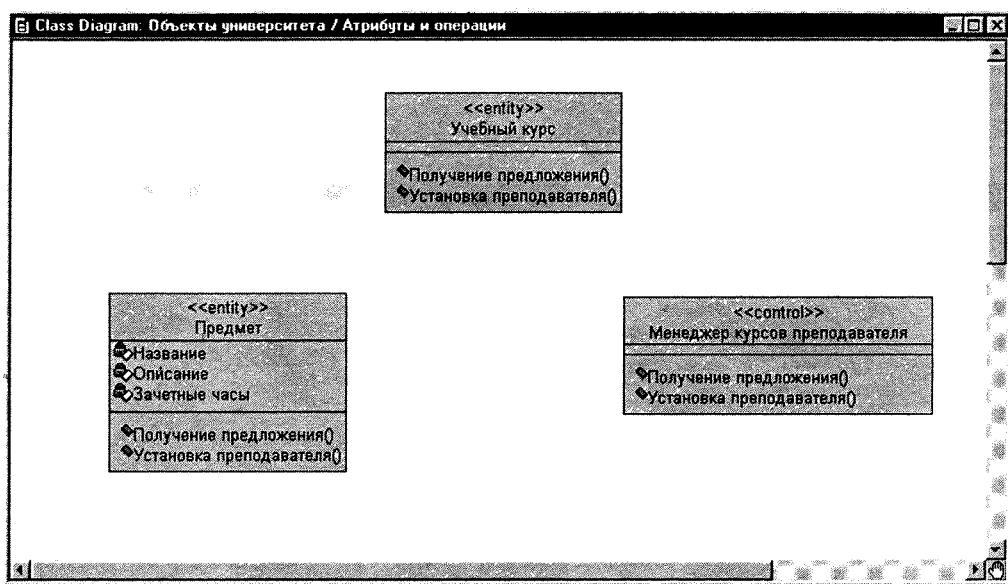


Рис. 7.6. Отображение атрибутов и операций на диаграмме классов

## Ассоциативные классы

Отношение может также иметь структуру и поведение. Это происходит в том случае, когда информация обращена к связи между объектами, а не к самому объекту.

Рассмотрим такой пример. Студент может посещать до четырех учебных курсов, а учебный курс может читаться нескольким студентам – от трех до десяти. Каждый студент получает оценку (grade) за учебный курс. Где должна храниться оценка? Она не принадлежит студенту, так как он наверняка получит различные оценки по разным предметам. Оценка не принадлежит и курсу, потому что студенты получают разные оценки за данный курс. Сведения об оценке принадлежат связи между студентом и учебным курсом. Они моделируются с помощью ассоциативного класса (association class), который ведет себя как и любой другой класс и также может иметь отношения. В нашем примере студент получает отчет об оценках (report card), куда включены связанные объекты оценка.

Для создания ассоциативных классов в программе Rational Rose:

1. Щелкните по кнопке **Class** (Класс) на панели инструментов.
2. Щелкните по диаграмме, чтобы поместить на нее класс.
3. Введите имя класса.
4. Добавьте необходимые атрибуты и операции для класса.
5. Щелкните по кнопке **Association Class** (Ассоциативный класс) на панели инструментов.
6. Щелкните по ассоциативному классу и проведите черту к линии связи между классами, соединяемыми ассоциативным классом.
7. Если требуется, добавьте дополнительные отношения к ассоциативному классу.

Ассоциативный класс оценка показан на рис. 7.7.

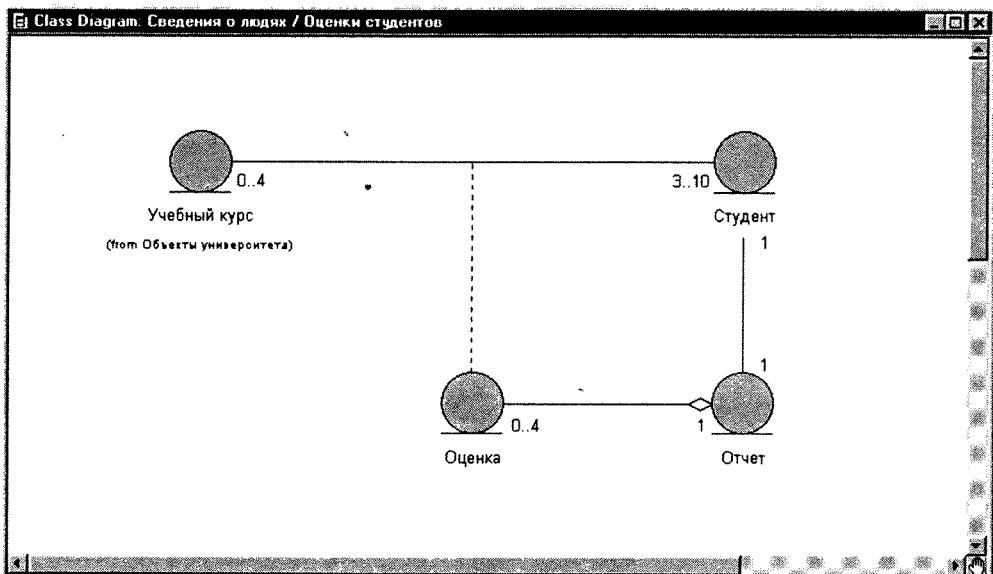


Рис. 7.7. Ассоциативный класс оценка

## Резюме

Класс выполняет ряд обязанностей, от которых зависит поведение его объектов. Обязанности исполняются с помощью конкретных операций. Структура объекта описывается атрибутами класса.

Каждый атрибут – это поле данных, содержащееся в объекте класса. Объект, полученный на основе класса, наделен значениями всех атрибутов класса. Атрибуты и операции, определенные для класса, – это основные значимые и функциональные элементы в разрабатываемом приложении.

Сообщения на диаграммах взаимодействий обычно отображаются на соответствующие операции в классах-получателях. Однако в некоторых случаях сообщения не становятся операциями, например: сообщения, поступающие

к актеру-человеку и от него, и сообщения для классов, представляющих пользовательский интерфейс.

Многие атрибуты класса выявляются при анализе предметной области, системных требований и описании потоков событий, а также при составлении описания класса. Кроме того, хорошим источником для определения атрибутов является сама предметная область.

Отношение может также иметь структуру и поведение. Это происходит в том случае, когда информация обращена к связи между объектами, а не к самому объекту. Структура и поведение отношений моделируются посредством ассоциативных классов.



## Глава 8. Изучение наследования

### Наследование

*Наследованием* (*inheritance*) называется такое отношение между классами, когда один класс использует часть структуры и/или поведения другого или нескольких классов. При наследовании создается иерархия абстракций, в которой подкласс (*subclass*) наследуется от одного или нескольких суперклассов (*super-class*). Наследование также называют иерархией типа «такой же, как» (*is-a*) или «такого вида, как» (*kind-of*). Подкласс наследует все атрибуты, операции и отношения, определенные в каждом его суперклассе. Значит, все атрибуты и операции, определенные на верхнем уровне иерархии, будут унаследованы классами на более низких ее уровнях. В подкласс могут быть добавлены дополнительные атрибуты и операции, применяемые только на данном уровне иерархии. Подкласс может содержать собственную реализацию унаследованной операции. Так как отношение наследования не является отношением между различными объектами, оно не имеет названия, не использует названия ролей и к нему не применяется понятие мощности.

На количество классов в иерархии наследования ограничений не существует. Однако на практике в программах, созданных с помощью C++, обычно используется от трех до пяти уровней, тогда как в приложениях, написанных на языке Smalltalk, – немного больше.

Наследование позволяет повторно использовать классы. Класс можно создать для одного приложения, после чего породить от него подкласс с расширенной функциональностью для использования в другом приложении.

Существует два способа определения наследования – обобщение и специализация. В любых разрабатываемых системах обычно используются оба метода.

### Обобщение

*Обобщение* (*generalization*) позволяет создавать суперклассы, объединяющие общие для нескольких классов структуру и поведение. Оно часто применяется на ранних стадиях анализа, когда набор созданных классов используется в основном для моделирования объектов реального мира. Классы проверяются на общность структуры (атрибутов) и поведения (операций). Например, оба класса студент и преподаватель имеют атрибуты имя, адрес и номер телефона.

При выяснении общих свойств, вероятно, придется поискать синонимы, так как названия атрибутов и операций, приведенные на разговорном языке, могут скрыть общность классов. Кроме того, проанализируйте атрибуты и поведение, которые на первый взгляд довольно специфичны, но на самом деле могут быть обобщены.

Например, класс студент имеет атрибут номер студента, а класс преподаватель – номер преподавателя. Если номера одинакового формата (например, четырехзначные числа), можно создать общий атрибут номер пользователя, заменяющий атрибуты номер студента и номер преподавателя. Если форматы отличаются, такие атрибуты должны храниться отдельно.

## Специализация

С помощью *специализации* (specialization) создаются подклассы, которые уточняют суперкласс – добавляют структуру и поведение. Такой метод наследования применяется, когда уже существует определенный класс. Подкласс создается, чтобы адаптировать поведение существующего класса. Например, в систему регистрации допускается добавить функцию, посредством которой почетные граждане обеспечивались бы бесплатными курсами. Новый подкласс почетный гражданин (SeniorCitizen) может быть добавлен в иерархию класса пользователь (RegistrationUser) для хранения данных, относящихся к почетным гражданам.

В подклассе операции могут быть перекрыты (overridden). Однако подкласс не должен ограничивать операции, определенные в его суперклассах, то есть не должен урезать их структуру и поведение.

Последовательность создания отношения наследования в программе Rational Rose:

1. Откройте диаграмму классов, на которой будет изображена иерархия наследования.
2. Щелкните по кнопке **Class** (Класс) на панели инструментов, а затем по диаграмме, чтобы поместить на нее класс.
3. Введите имя класса. Класс также может быть создан в браузере и перемещен на диаграмму.
4. Щелкните по кнопке **Generalization** (Обобщение) на панели инструментов.
5. Щелкните по подклассу и проведите линию связи к суперклассу.
6. Повторите последнее действие для других подклассов.

Отношение наследования показано на рис. 8.1.

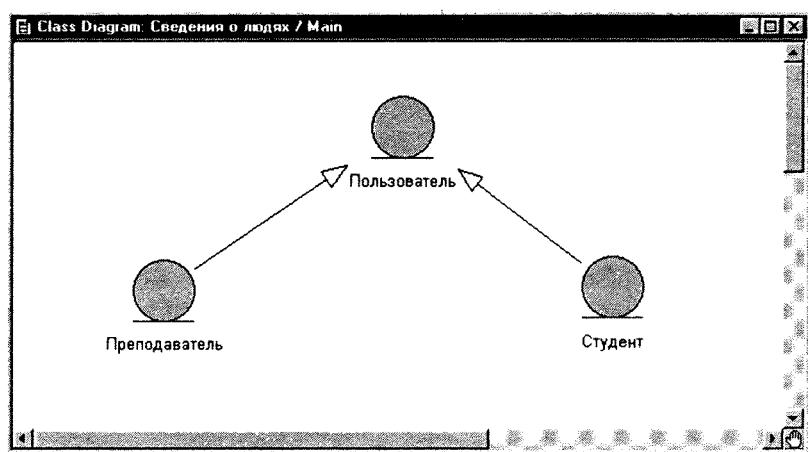


Рис. 8.1. Отношение наследования

## Дерево наследования

Основу для специализации (то есть цель создания подкласса) в отношении наследования называют *дискриминатором* (discriminator). Дискриминатор, как правило, наделен конечным набором значений и подклассов, которые создаются для каждого значения.

Например, одним из дискриминаторов для класса предмет (Course) является место обучения. Классы очный предмет (OnCampusCourse) и заочный предмет (OffSiteCourse) могут стать подклассами для класса предмет, созданными на основе этого дискриминатора. Отношения наследования для всех подклассов, полученных от одного дискриминатора, представляются в виде дерева. Другим подклассом предмет может стать класс обязательный предмет. Этот подкласс не будет частью дерева наследования, так как он принадлежит другому дискриминатору – типу предмета. Следует внимательно подходить к вопросу определения нескольких дискриминаторов для одного класса. Например, что произойдет, если обязательный предмет тоже очный? Является ли это примером множественного наследования? Не нужно ли здесь применить агрегацию? В ходе анализа и проектирования ответы на эти вопросы постепенно позволят получить законченную структуру модели.

Для создания дерева наследования в программе Rational Rose:

1. Откройте диаграмму классов, на которой будет изображена иерархия наследования.
2. Щелкните по кнопке **Class** на панели инструментов, а затем по диаграмме, чтобы поместить на нее класс.
3. Введите имя класса. Класс также может быть создан в браузере и перемещен на диаграмму.
4. Щелкните по кнопке **Generalization** на панели инструментов.
5. Щелкните по подклассу и проведите линию связи к суперклассу.
6. Для каждого подкласса, являющегося частью дерева наследования: щелкните по кнопке **Generalization** на панели инструментов, щелкните по подклассу и проведите линию обобщающей связи к значку наследования (в виде треугольника).

Дерево наследования может быть создано из двух отдельных обобщающих линий на диаграмме путем перетаскивания одной линии на другую.

Древовидное отношение наследования показано на рис. 8.2.

Я также предпочитаю отображать названия стереотипов на таких диаграммах, поскольку треугольник, изображающий наследование, может закрыть название класса при использовании значков.

После создания суперкласса атрибуты, операции и отношения размещают по возможности на самом высоком уровне иерархии. Какие же свойства необходимо перенести? Давайте посмотрим на иерархию с базовым классом пользователь. Атрибуты, операции и отношения для подклассов показаны на рис. 8.3. Так как атрибуты имя (name) и номер (IDNumber) одинакового формата, их можно с уверенностью перенести в суперкласс пользователь (RegistrationUser). Оба класса связаны с классом учебный курс (CourseOffering). Для этого отношения существуют два варианта:

## Дерево наследования

93

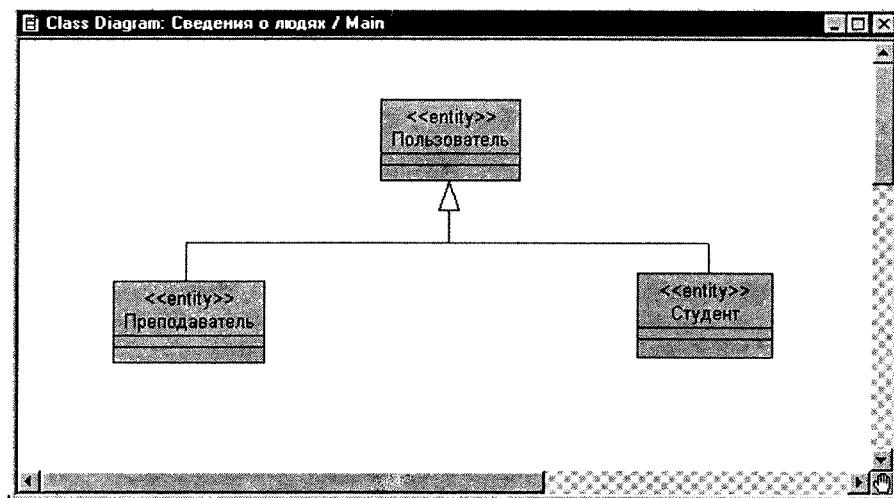


Рис. 8.2. Дерево наследования

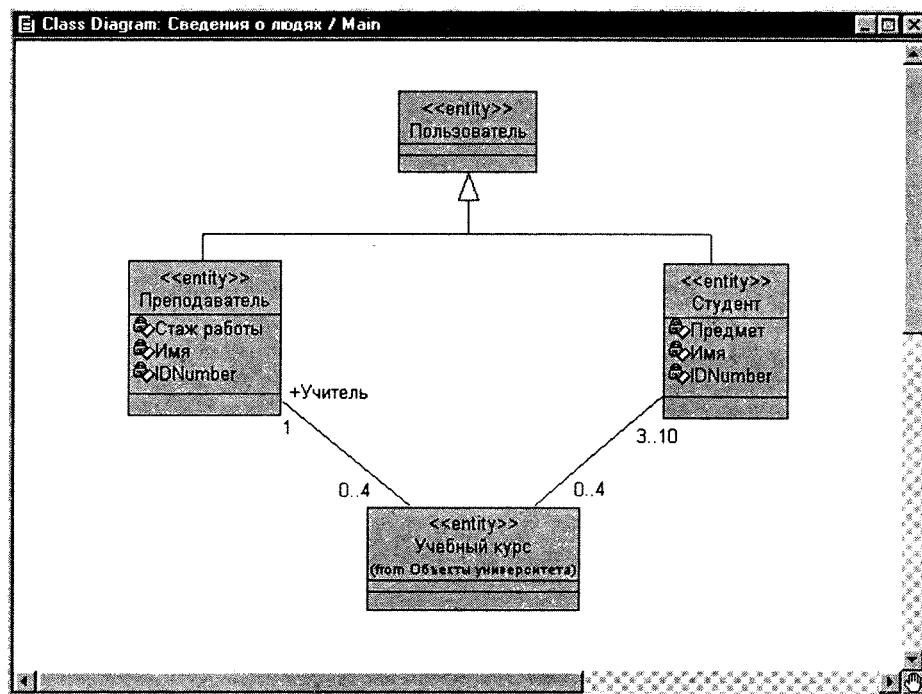


Рис. 8.3. Иерархия наследования для класса пользователь

- сохранить отношения на уровне подклассов;
- сделать отношение на уровне суперкласса со значением мощности, учитывающим объекты преподаватель и студент (один объект учебный курс связан с объектами пользователь в количестве от 4 до 11). Кроме того,

здесь накладывается дополнительное ограничение: один из объектов пользователь должен быть преподавателем.

Какой из этих вариантов является правильным? Оба. Какой из них лучше использовать? Зависит от ситуации. Если требуется, чтобы объект учебный курс «знал» всех студентов и преподавателя, то хранить единый список с такой информацией наверняка будет удобнее. В этом случае воспользуйтесь вторым вариантом. С другой стороны, если нужно знать только студентов или только преподавателей, подойдет первый вариант. Все зависит от требований к системе. В подобных ситуациях следует тщательно изучить функции и сценарии, чтобы определить нужное поведение.

Для перемещения атрибутов и операций в программе Rational Rose:

1. В окне браузера щелкните по значку «+» слева от имени подкласса, чтобы раскрыть список его свойств.
2. Выберите атрибут или операцию, которую нужно переместить.
3. Перетащите с помощью мыши атрибут или операцию на суперкласс.
4. Удалите данный атрибут или операцию из других подклассов.
5. Таким же образом переместите другие необходимые атрибуты из подклассов в суперкласс.

Иерархия наследования после переноса атрибутов показана на рис. 8.4.

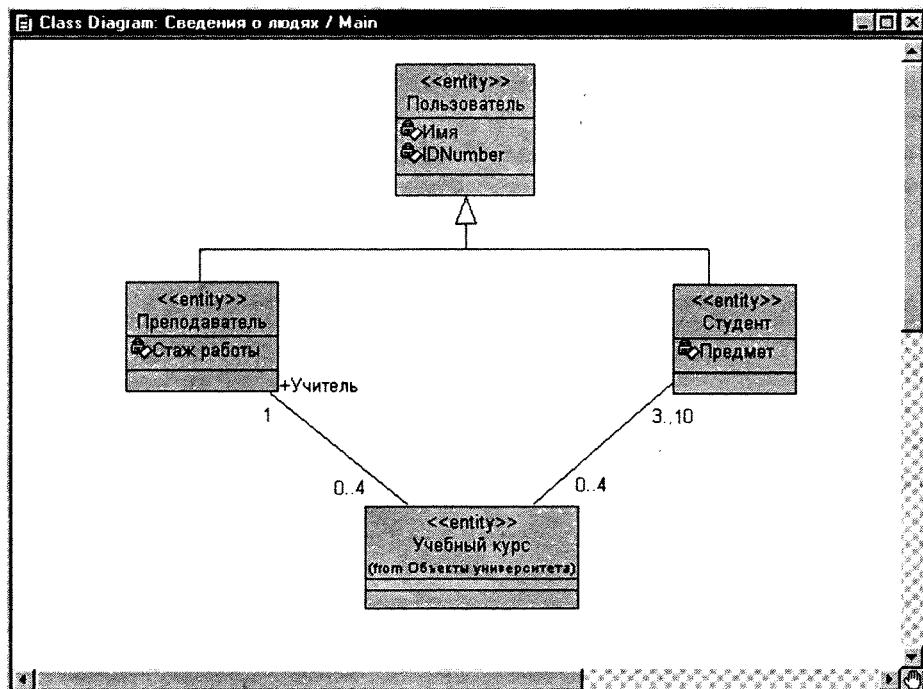


Рис. 8.4. Атрибуты и операции, перемещенные в суперкласс

## Одиночное и множественное наследование

При одиночном наследовании класс содержит единственный набор потомков, то есть одну цепочку суперклассов (например, легковая машина – это автомобиль, а автомобиль – средство передвижения). Множественное наследование включает более одной цепочки суперклассов (машина-амфибия – это автомобиль, автомобиль – средство передвижения, в то же время машина-амфибия – это лодка, а лодка – средство передвижения). При множественном наследовании возникает ряд проблем, в частности конфликт имен и несколько копий унаследованных свойств. Способ решения таких проблем выбирается в зависимости от языка программирования: виртуальные базовые классы в C++ или отсутствие поддержки множественного наследования в PowerBuilder. Множественное наследование может стать причиной запутанного и трудно сопровождаемого кода – чем больше суперклассов, тем труднее определить, что откуда взялось и что произойдет при внесении изменений. Вывод: используйте множественное наследование только при необходимости и с большой осторожностью.

## Наследование и агрегация

Наследование часто используется не по назначению. Существует мнение, что «чем больше я буду его использовать, тем лучше станет мой код». Это заблуждение. В действительности неправильное применение наследования может привести к проблемам. Например, студент может учиться очно или заочно. Создадим суперкласс **студент** (*Student*) и два подкласса – **студент очного отделения** (*FulltimeStudent*) и **студент заочного отделения** (*ParttimeStudent*). Во время работы такой структуры наверняка возникнут определенные проблемы. Что случится, если:

- студент очного отделения решит перейти на заочное? Это значит, что объекту придется сменить класс;
- будет добавлена еще одна размерность (например, студент, получающий стипендию и не получающий стипендию)? Здесь понадобятся новые подклассы для представления информации о стипендии, а также множественное наследование для поддержки всех комбинаций (студент очного отделения, получающий стипендию, студент заочного отделения, получающий стипендию и т.д.).

Наследование должно служить для отделения общности от специфики. Агрегация – для отражения комбинированных отношений. Часто оба типа отношений используются вместе. Класс **студент** имеет классификацию (агрегацию), которая, в свою очередь, делится на классы **студент-очник** и **студент-заочник** (наследование) – см. рис 8.5.

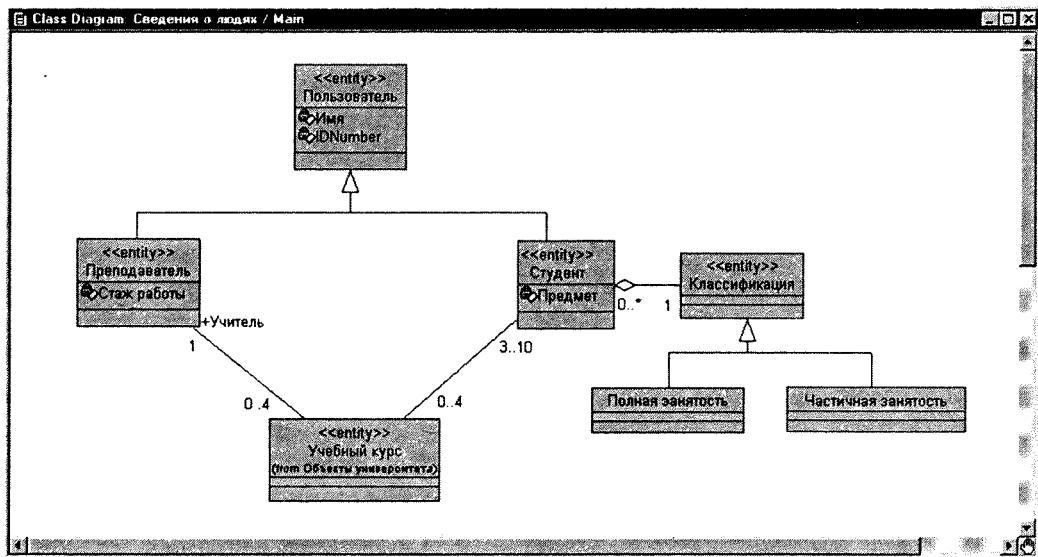


Рис. 8.5. Наследование в сравнении с агрегацией

## Резюме

Наследование позволяет создавать иерархию классов, когда общая структура и поведение разделяются между ними. Термин «суперкласс» характеризует класс, содержащий общую информацию. Классы-потомки называются подклассами. Подкласс наследует все атрибуты, операции и отношения, определенные во всех его суперклассах.

Есть два способа определения наследования в любой системе: обобщение и специализация. Обобщение обеспечивает возможность создания суперклассов, объединяющих общие для нескольких классов структуру и поведение. Специализация позволяет создавать подклассы, которые уточняют или дополняют структуру и поведение, определенные в суперклассе.



## Глава 9. Анализ поведения объекта

### Моделирование динамического поведения

Прецеденты и сценарии применяются для описания поведения системы, то есть взаимодействия объектов в ней. Иногда требуется рассмотреть поведение внутри самого объекта. Диаграмма состояний (statechart diagram) показывает положение одиночного объекта, события или сообщения, которые вызывают переход из одного состояния в другое, и действия, являющиеся результатом смены состояния.

Диаграмму состояний не нужно создавать для каждого класса в системе, только для классов с «особенным» динамическим поведением. Для определения динамических объектов в системе, то есть объектов, отсылающих и получающих большое количество сообщений, могут использоваться диаграммы взаимодействий. Диаграмма состояний также полезна для изучения поведения класса-агрегата и управляющего класса.

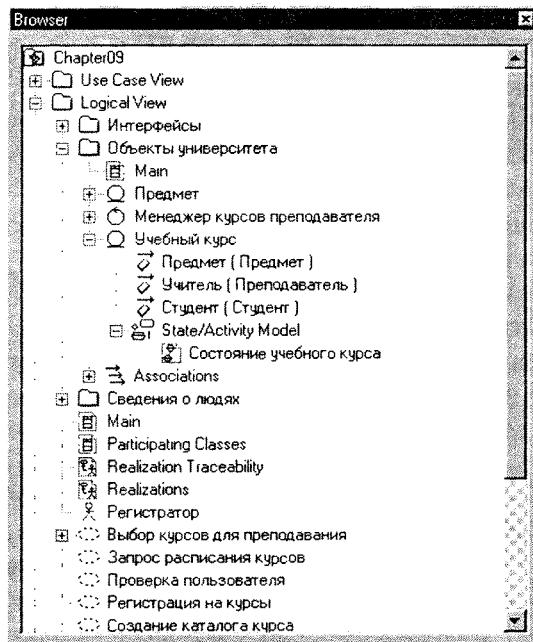


Рис. 9.1. Диаграмма состояний в браузере

Следует внимательно подойти к вопросам анализа и сосредоточиться на том, что представляет собой проблема, а не на том, как она будет решаться.

Для создания диаграммы состояний в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Statechart Diagram** (Создать ⇒ Диаграмма состояний). В список браузера будет добавлена диаграмма **New Diagram**.
3. Введите ее название.
4. Чтобы открыть диаграмму, щелкните по значку «+» слева от имени подкласса в окне браузера, потом по значку «+» слева от пункта **State/Activity Model** (Модель состояний и действий), а затем дважды по диаграмме состояний.

Диаграмма состояний в списке браузера для класса учебный курс (Course Offering) показана на рис. 9.1.

## Состояния

*Состояние* (state) – это некое положение в жизни объекта, при котором он удовлетворяет определенному условию, выполняет некоторое действие или ожидает события. Состояние объекта можно описать с помощью значений одного или нескольких атрибутов класса. Например, объект учебный курс может быть открыт (доступен для записи студентов) или закрыт (максимальное число студентов уже записано на курс). Состояние зависит от числа студентов, прикрепленных к объекту учебный курс. Кроме этого, оно может определяться наличием связи с другим объектом. Актер преподаватель может читать лекции или находиться в отпуске. Это зависит от наличия связи с объектом учебный курс. При анализе состояния объекта можно проверить мощность, выбранную для отношения с другим объектом. То есть, если нахождение в определенном состоянии зависит от наличия связи с другим объектом, значит, мощность отношения, относящаяся к связанному классу, должна включать нулевое значение (другими словами, отношение необязательно). Таким образом, состояния объекта определяются при изучении атрибутов и связей, указанных для него.

В языке UML состояние изображается в виде прямоугольника с закругленными углами – см. рис. 9.2.

Диаграмма состояний включает все сообщения, которые объект получает и отправляет. Сценарий – это одиночный проход по диаграмме состояний. Интервал между двумя сообщениями, отправляемыми объектом, обычно представляет состояние. Таким образом, для определения состояний объекта нужно изучить диаграмму последовательности действий (взгляните на интервалы между линиями сообщений, получаемых объектом). Если для каждой функции в системе регистрации учебных курсов были созданы диаграммы последовательности действий, то с помощью них можно обнаружить, что объекты класса учебный курс

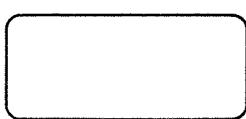


Рис. 9.2. Нотация языка UML для состояния

## Переходы между состояниями

99

(CourseOffering) могут находиться в одном из следующих состояний: инициализация (создан до регистрации, но студенты не прикреплены), открыт (доступен для записи студентов), закрыт (на курс записано максимальное число студентов), отменен (больше не читается).

Последовательность создания состояний в программе Rational Rose:

1. Щелкните по кнопке **State** (Состояние) на панели инструментов.
2. Щелкните по диаграмме, чтобы поместить на нее новое состояние.
3. Введите его название.

Состояния класса учебный курс показаны на рис. 9.3.

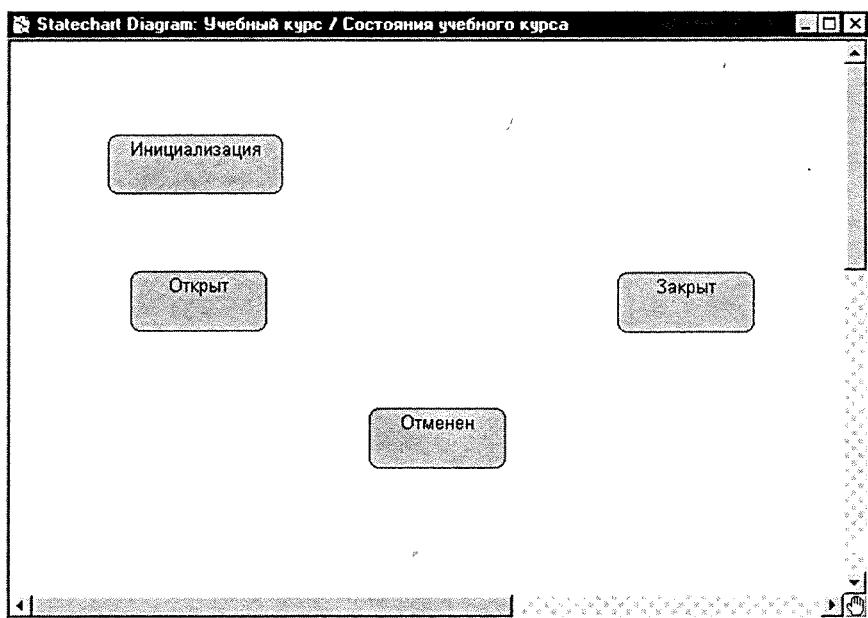


Рис. 9.3. Состояния

## Переходы между состояниями

Переходы между состояниями (state transitions) представляют собой смену исходного состояния последующим (которое может быть тем же, что и исходное). Переход может сопровождаться определенным действием.

Есть два способа выхода из состояния – автоматический и неавтоматический. Автоматическая смена состояния происходит, когда действие исходного состояния будет завершено – с переходом не связано каких-либо событий. Неавтоматический переход между состояниями вызывается определенным событием (от другого объекта или из внешней среды). Считается, что оба типа переходов выполняются за нулевое время и не могут быть прерваны. Переход между состояниями изображается в виде стрелки, направленной от исходного состояния к последующему.

Для создания переходов между состояниями в программе Rational Rose:

1. Щелкните по кнопке **State Transition** (Переход) на панели инструментов.
2. Щелкните по исходному состоянию диаграммы.
3. Проведите линию перехода к последующему состоянию.
4. Если требуется, введите название нового перехода.

Переходы между состояниями показаны на рис. 9.4.

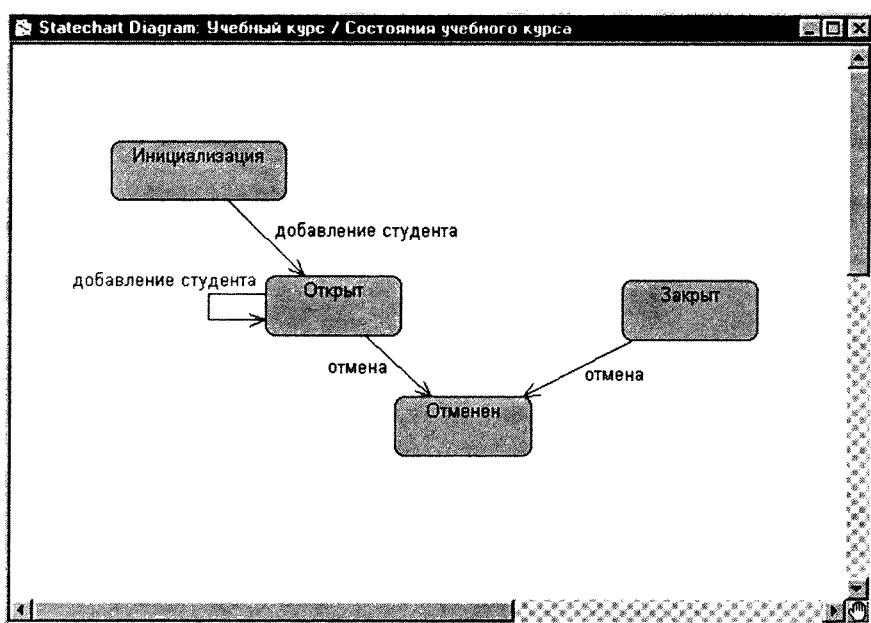


Рис. 9.4. Переходы между состояниями

## Особые состояния

Есть два особых состояния, присутствующих на диаграмме состояний, – *начальное* (start state) и *конечное* (stop state).

Каждая диаграмма должна иметь одно и только одно начальное состояние, так как объект может находиться в целостном состоянии сразу после создания. В языке UML начальное состояние изображается в виде маленького закрашенного круга (см. рис. 9.5).



Рис. 9.5. Нотация языка UML для конечного и начального состояний

Объект может иметь несколько конечных состояний, которые изображаются в виде закрашенного круга, обведенного дополнительной окружностью (см. рис. 9.5).

Чтобы создать начальное состояние в программе Rational Rose:

1. Щелкните по кнопке **Start** (Начальное состояние) на панели инструментов.
2. Щелкните по диаграмме, чтобы поместить на нее значок исходного состояния.
3. Щелкните по кнопке **State Transition** на панели инструментов.
4. Щелкните по начальному состоянию и проведите линию перехода к следующему состоянию.

Начальное состояние на диаграмме состояний показано на рис. 9.6.

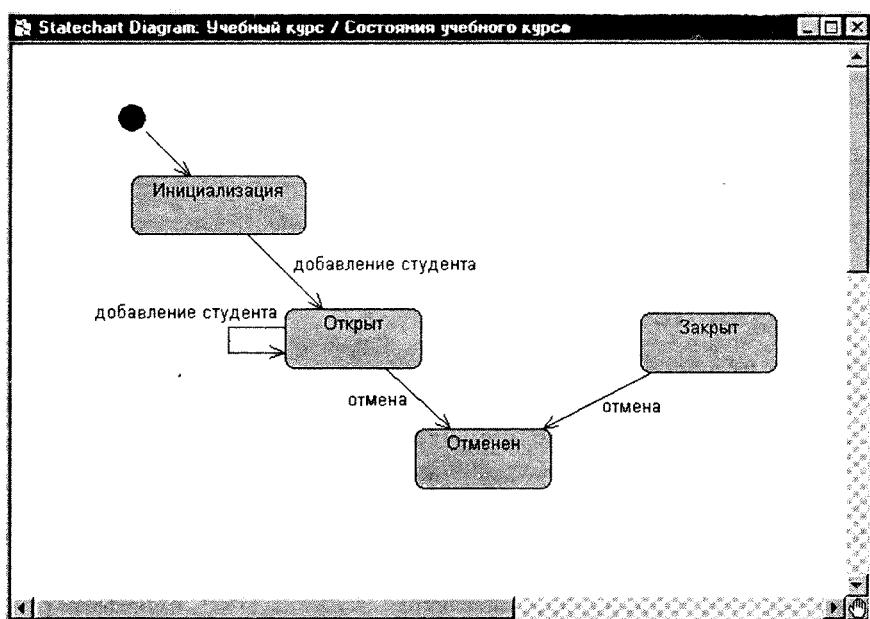


Рис. 9.6. Начальное состояние

Для создания конечного состояния в программе Rational Rose выполните следующие действия:

1. Щелкните по кнопке **Stop** (Конечное состояние) на панели инструментов.
2. Щелкните по диаграмме, чтобы поместить на нее значок конечного состояния.
3. Щелкните по кнопке **State Transition** на панели инструментов.
4. Щелкните по одному из состояний на диаграмме и проведите линию перехода к конечному состоянию.

Конечное состояние на диаграмме состояний показано на рис. 9.7.

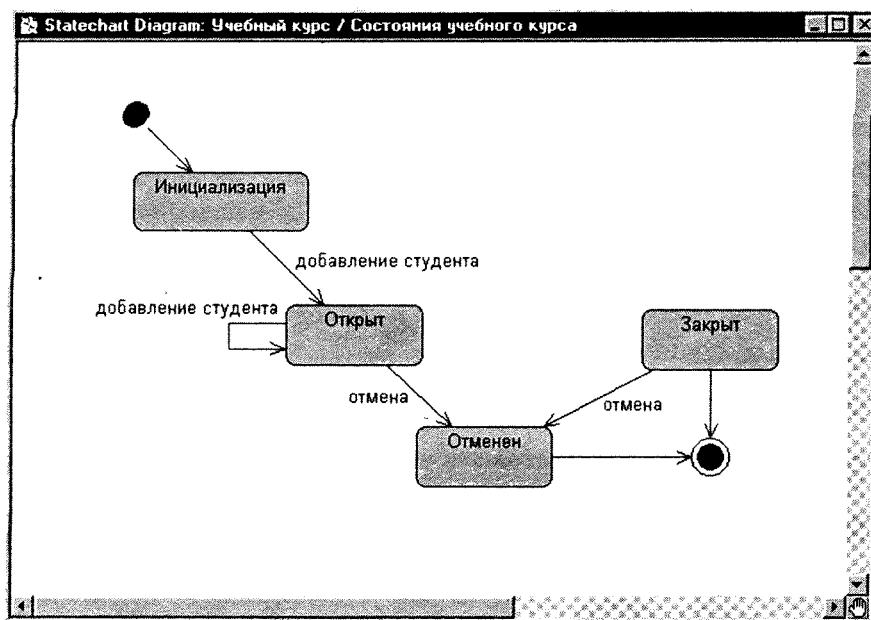


Рис. 9.7. Конечное состояние

## Параметры переходов

С переходом между состояниями может быть связано *условие* (guard condition) и/или определенное *действие* (action). Переход может также вызывать *событие* (event). Действие – это поведение, проявляющееся при возникновении перехода. Событие – сообщение, отправляемое другому объекту системы. Условие – булево выражение значений атрибутов, которое допускает переход, только если оно верно. И действие, и проверка условия представляют собой поведение объекта и обычно реализуются в виде операций. Часто такие операции являются *скрытыми* (private), то есть используются только самим объектом. В языке UML параметры перехода изображаются так, как показано на рис. 9.8.

Последовательность добавления параметров перехода в программе Rational Rose:

1. Щелкните правой кнопкой мыши по стрелке перехода на диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Specification** (Параметры), чтобы вызвать диалоговое окно параметров перехода.
3. Выберите вкладку **Detail** (Детально).
4. Укажите действие, условие и событие для перехода.
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров.

Параметры перехода на диаграмме состояний показаны на рис. 9.9.

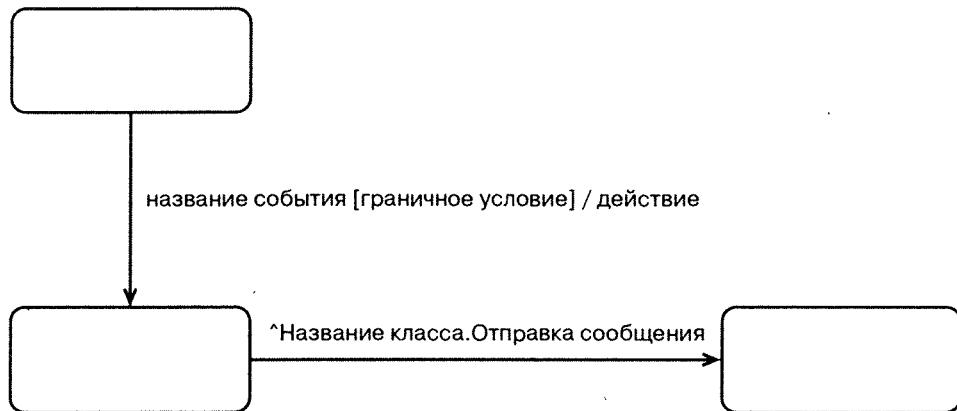


Рис. 9.8. Нотация языка UML для параметров переходов

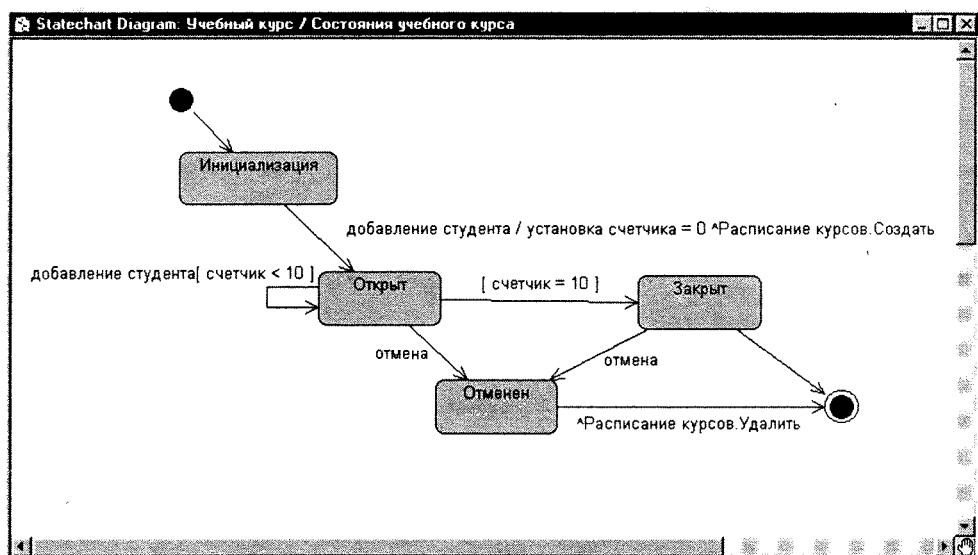


Рис. 9.9. Параметры переходов

## Параметры состояний

Действия, сопровождающие возможные переходы в определенное состояние, можно рассматривать как *входные действия* (entry action) для этого состояния. И наоборот, действия, сопровождающие переходы из данного состояния, являются для него *выходными* (exit action). Поведение, возникающее внутри состояния, называется *деятельностью* (activity). Деятельность начинается при входе в состояние и завершается или прерывается при переходе из него. Поведение может быть простым действием или событием, посылаемым другому объекту.

Как и в случае с действиями и проверками условий для перехода, поведение внутри состояния обычно реализуется в виде операций. В языке UML параметры состояний изображаются так, как показано на рис. 9.10.

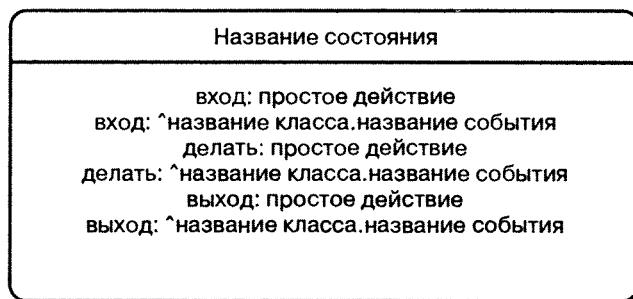


Рис. 9.10. Нотация языка UML для параметров состояний

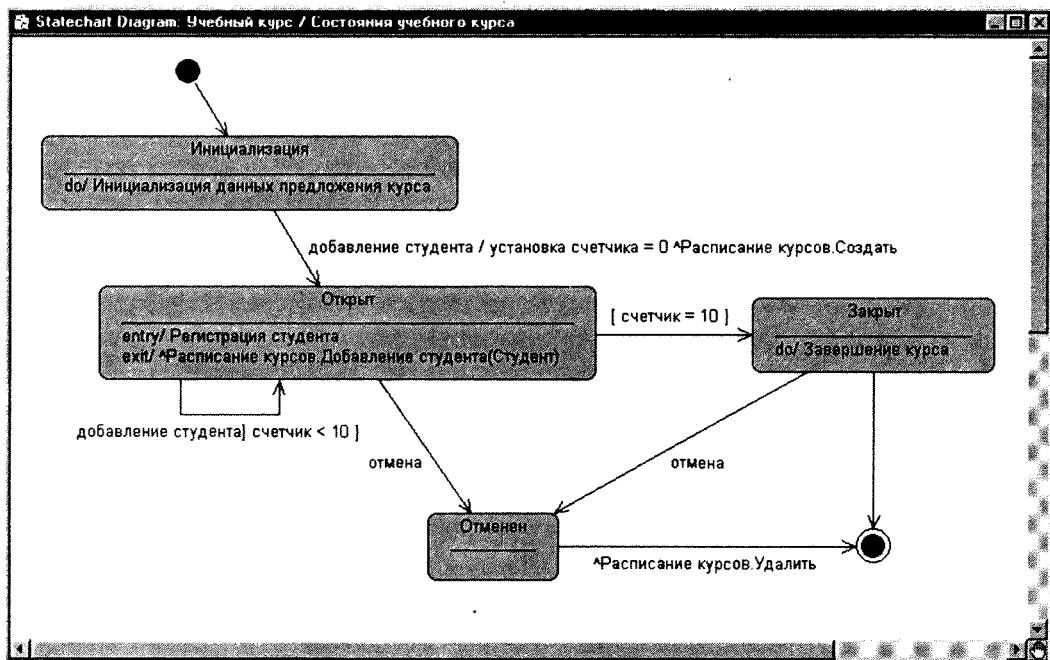


Рис. 9.11. Параметры состояний

Определение входных, выходных и внутренних действий для состояния в программе Rational Rose предусматривает выполнение следующих шагов:

1. Щелкните правой кнопкой мыши по изображению состояния на диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Параметры), чтобы вызвать диалоговое окно параметров состояния.
3. Выберите вкладку **Actions** (Действия).
4. Щелкните правой кнопкой мыши по списку **Action** (Действие).

5. В появившемся контекстно-зависимом меню выберите команду **Insert** (Добавить). В список будет добавлено новое действие.
6. Дважды щелкните по новому действию в списке, чтобы открыть диалоговое окно **Action Specification** (Параметры действия).
7. Укажите момент выполнения действия: **on entry** (при входе), **on exit** (при выходе), **on event** (при определенном событии).
8. Введите описание действия или события.
9. Укажите тип действия: **action** (действие) или **send event** (вызов события).
10. Если требуется, введите название действия или события.
11. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Action Specification**.
12. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **State Specification**.

Параметры состояний на диаграмме состояний показаны на рис. 9.11.

## Резюме

Классы, характеризующиеся выраженным динамическим поведением, анализируются с помощью диаграмм состояний. На таких диаграммах отображаются все состояния объекта, поступающие к объекту события и результирующие действия. Считается, что переходы между состояниями и сопровождающие их действия выполняются за нулевое время и не могут быть прерваны. Пребывание объекта в определенном состоянии и сопутствующая деятельность могут быть прерваны.



## Глава 10. Проверка модели

### Для чего нужна однородность

Согласно новому академическому словарю Вебстера слово «гомогенизировать» (homogenize) означает превращать в однородную массу, делать гомогенным, то есть однородным. По мере добавления новых прецедентов и сценариев необходимо добиваться, чтобы модель стала однородной. Это особенно актуально, когда несколько групп разработчиков создают различные части модели. Так как прецеденты и сценарии описываются обычными словами, для обозначения одной и той же вещи могут использоваться разные слова, смысл которых может трактоваться по-разному. На данном этапе возможно объединение, разделение или исключение классов из модели. Это процесс естественного развития и совершенствования модели в течение жизненного цикла проекта.

Однородность не возникает в определенной точке жизненного цикла – это длительный процесс. Проект, в котором синхронизация информационных фрагментов от разных групп разработчиков происходит на заключительном этапе, обречен на провал. Я пришла к выводу, что наиболее успешные коллективные проекты получаются, когда разработчики постоянно используют механизмы взаимодействия и согласования. Взаимодействие может быть простым (например, телефонный разговор) или формальным (плановые совещания) – все зависит от проекта и обсуждаемого вопроса. Очень важно, чтобы при этом группы разработчиков контактировали друг с другом.

### Объединение классов

Когда разные группы разработчиков создают различные сценарии, одному классу могут быть присвоены разные имена. Конфликты имен должны быть разрешены. Это обычно делается с помощью проходов по модели. Проверьте каждый класс вместе с его описанием. Также проверьте атрибуты и операции, определенные для классов, и поищите синонимы. Если вы обнаружили, что два класса выполняют одно и то же, выберите из них класс с именем, более близким к терминологии, используемой заказчиками.

Уделите особое внимание управляющим классам системы. Изначально для каждого прецедента предусматривается по одному управляющему классу. Однако управляющие классы со схожим поведением могут быть объединены. Изучите последовательную логику (sequencing logic) в управляющих классах системы. Если она идентична, управляющие классы можно объединить в один.

В системе регистрации учебных курсов имеется один управляющий класс для прецедента Сохранить информацию о курсах (Maintain Course Information) и один для прецедента Создать каталог курсов (Create Course Catalog). Каждый управляющий класс получает информацию от граничного класса и обрабатывает информацию о курсах. Вероятно, эти классы могут быть объединены, так как они имеют поведение и управляют одинаковой информацией.

## Разделение классов

Классы обязательно проверяются на предмет соответствия «золотому» правилу объектно-ориентированной технологии, которое утверждает, что класс должен выполнять одну задачу и выполнять ее хорошо. Например, класс информация о студенте (StudentInformation), содержащий сведения об актере студента, а также о курсах, которые тот закончил, выполняет слишком много функций. Его лучше представить в виде двух классов – информация о студенте и выписка (Transcript) – и ассоциативной связи между ними.

Часто атрибут класса имеет структуру и поведение внутри себя и должен быть выделен как отдельный класс. Например, рассмотрим факультет университета. Каждый учебный предмет предлагается определенным факультетом. Вначале эти сведения были представлены в модели как атрибут класса предмет (Course). Дальнейший анализ выявил необходимость получать данные о количестве студентов, обучающихся на факультете, количестве преподавателей, проводящих занятия на факультете, и количестве учебных предметов на каждом факультете. Таким образом, был создан отдельный класс факультет (Department). Первоначальный атрибут факультет в классе предмет был заменен ассоциативной связью между классами.

## Исключение классов

Иногда класс вообще может быть удален из модели. Это происходит в следующих случаях:

- когда класс не имеет какой-либо структуры или поведения;
- когда класс не участвует в выполнении какого-либо прецедента.

Тщательно проверяйте управляющие классы. Отсутствие последовательных действий может указывать на удаление управляющего класса. Особенно в случае, когда он просто проходной, то есть получает информацию от граничного класса и тут же передает ее в класс-сущность без какой-либо последовательной логики.

В системе регистрации учебных курсов мы изначально создали управляющий класс для прецедента выбор курсов для преподавания (Select Courses to Teach). Он позволяет преподавателю определить курсы для чтения в данном семестре. Для управляющего класса здесь не нужна последовательная логика – преподаватель вводит информацию с помощью формы пользовательского интерфейса, и класс преподаватель добавляется к выбранному курсу. В данном случае управляющий класс для прецедента может быть удален.

## Проверка целостности

Проверка целостности необходима потому, что статическое представление системы, показанное на диаграммах классов, и динамическое представление системы, изображенное на диаграммах прецедентов и диаграммах взаимодействий, разрабатываются параллельно. По причине одновременной разработки обоих представлений необходимо производить перекрестные проверки, чтобы убедиться, что в разных представлениях не сделаны разные допущения и не приняты различные решения.

Проверка целостности должна быть непрерывной частью всего жизненного цикла разрабатываемой системы.

Лучше всего, когда проверкой целостности занимается отдельная группа сотрудников (пять–шесть человек). В нее должны входить разные специалисты – аналитики, проектировщики, заказчики или их представители, эксперты по предметной области и специалисты по тестированию.

## Проход по сценарию

Основной метод проверки целостности – проход по сценариям с наибольшим риском, представленным на диаграммах взаимодействий. Так как каждое сообщение отражает поведение получающего класса, убедитесь в том, что каждое сообщение реализовано в виде операции на диаграмме классов. Проверьте, что два взаимодействующих объекта связаны между собой с помощью ассоциации или агрегации. Внимательно проследите за возвратными отношениями: их легко упустить из виду на этапе анализа. Возвратные отношения требуются, когда различные объекты одного класса взаимодействуют друг с другом в ходе выполнения сценария.

Убедитесь, что каждый класс, представленный на диаграмме классов, участвует хотя бы в одном сценарии. Проверьте, чтобы каждая операция, определенная для класса, была использована, по крайней мере, в одном сценарии или требовалась для завершенности модели. И наконец, проследите, чтобы каждый объект, включенный в диаграмму последовательности действий или диаграмму взаимодействий, принадлежал одному из классов на диаграмме классов.

## Отслеживание событий

Для каждого сообщения, изображенного на диаграммах взаимодействий, проверьте, чтобы операция в классе-отправителе отправляла событие, а операция в классе-получателе ожидала событие и могла его обработать. Убедитесь в наличии ассоциативной или агрегационной связи на диаграмме классов между классом-отправителем и классом-получателем. Если такая связь отсутствует, добавьте ее на диаграмму классов. Если для класса уже создана диаграмма состояний и переходов, то событие на ней должно быть представлено для класса-получателя. Необходимо, чтобы диаграмма состояний отражала все события, которые может получать класс.

## Просмотр документации

Каждый класс должен быть описан в документации! Проверьте уникальность имен классов и просмотрите все описания на предмет их полноты. Выясните, что все атрибуты и операции полностью документированы. На заключительном этапе убедитесь в соблюдении всех стандартов, форматов, спецификаций и правил оформления, определенных для проекта.

## Резюме

По мере добавления новых прецедентов и сценариев нужно добиваться однородности модели. Это особенно актуально, когда несколько групп разработчиков проектируют различные части модели. Классы проверяются на предмет необходимости:

- объединения двух или более классов;
- разделения класса;
- исключения класса из модели.

Проверка целостности происходит на протяжении всего жизненного цикла проекта. Она требуется для параллельной разработки нескольких представлений системы и для синхронизации фрагментов системы. Существует три основных способа проверки целостности: проход по сценарию, отслеживание событий и просмотр документации модели.



## Глава 11. Проектирование системной архитектуры

### Потребность в архитектуре

На протяжении многих лет я слышала разные определения программной архитектуры: от «программная архитектура – это то, чем занимаются специалисты по программной архитектуре» до «программная архитектура – это политика». Я пришла к выводу, что для программной архитектуры очень трудно подобрать определение. Ее можно представить как набор средств, используемых для указания стратегических решений по структуре и поведению системы, взаимодействию между элементами системы и физическому размещению системы.

«Создание качественного архитектурного базиса необходимо для успешной реализации объектно-ориентированных проектов. Некоторые разработчики пытаются пропустить эту фазу, либо спешат с выпуском продукта, либо не верят в преимущества архитектурных решений. В любом случае результат всегда плачевный – недостаточная проработка этого шага приводит к постепенному распаду проекта»<sup>1</sup>.

Развитие архитектуры – достаточно сложный вопрос. Архитектура системы развивается итеративно на стадии проработки. «Архитектура системы не возникает сразу. Она требует тщательного изучения прецедентов, создания прототипов для подтверждения основных концепций, создания архитектурного фундамента и других усилий в процессе задумки и проработки»<sup>2</sup>. Для проверки корректности решений на этапе проектирования моделируются работоспособные прототипы архитектуры. «Создание чего-то работоспособного очень важно, потому что позволяет группе специалистов проверять проектные предположения на практике»<sup>3</sup>.

### О разработчиках архитектуры

В каждом проекте должен участвовать главный специалист по архитектуре, у которого может быть несколько ассистентов. «В обязанности такого специалиста входит определение архитектуры программных продуктов, поддержка целостности архитектуры, оценка технических рисков проекта, определение порядка

<sup>1</sup> Booch, Grady. *Object Solutions*. Redwood City, CA: Addison-Wesley, 1995.

<sup>2</sup> Jacobson, Ivar. *The Objector Software Development Process*. Draft edition.

<sup>3</sup> Там же.

выпуска последовательных версий и планирование их содержания, проведение консультаций для групп проектировщиков, разработчиков, сборщиков и тестеров, а также помочь в определении будущей маркетинговой стратегии<sup>1</sup>.

## Представление архитектуры 4 + 1

Программная архитектура многомерна – она состоит из нескольких одновременно развивающихся представлений<sup>2</sup> (см. рис. 11.1).

Во всех последующих разделах этой главы рассказывается об элементах каждого представления и нотации языка UML для описания архитектурных решений.

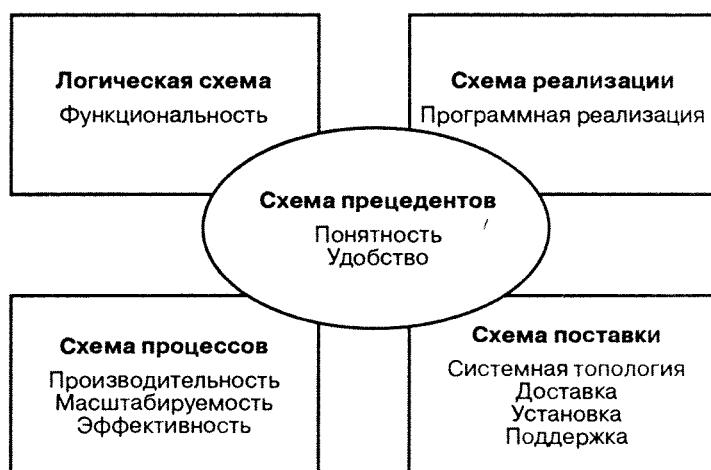


Рис. 11.1. Представление архитектуры 4 + 1

## Логическое представление

*Логическое представление* (logical view) архитектуры описывает функциональные требования к системе – то, что система должна обеспечивать для обслуживания пользователей. Логическая архитектура отображается на диаграмме классов, содержащей классы и отношения, которые представляют ключевые абстракции разрабатываемой системы.

Большинство нотаций языка UML содержится в самом логическом представлении архитектуры (классы, ассоциации, агрегации, обобщение, пакеты и др.). Оно вводится на фазе проработки при создании классов и пакетов, представляющих основные абстракции предметной области. Постепенно все больше классов и пакетов добавляется в модель для отражения решений, касающихся ключевых механизмов системы. Ключевой механизм – это решение относительно общих стандартов, правил и норм. Выбор ключевых механизмов системы часто называют

<sup>1</sup> Kruchten, Philippe. *Software Architecture and Iterative Development*. Santa Clara, CA: Rational Software Corporation, April 1994, p. 53.

<sup>2</sup> Там же.

*тактическим проектированием* (tactical design). «Плохое тактическое проектирование может уничтожить даже тщательно продуманную архитектуру, поэтому разработчики должны уменьшить этот риск, определив ключевые правила проекта»<sup>1</sup>. К некоторым ключевым механизмам относятся: язык разработки, хранение данных, удобный пользовательский интерфейс, обработка ошибок, механизмы взаимодействия, распределение и миграция объектов, сетевые средства.

На сегодняшний день существует много шаблонов, которые могут использоваться для реализации ключевых решений системы. Я настоятельно рекомендую изучить шаблоны, прежде чем предпринимать самостоятельные шаги.

Кроме того, концепции *связности* (cohesion), *замкнутости* (closure) и *повторного использования* (reuse) повлияют на ваш выбор. Роберт Мартин (Robert Martin) рассмотрел некоторые вопросы выбора пакетов системы в своей книге «Разработка объектно-ориентированных приложений на C++ с использованием методов Буча». Подходы и нотация Буча вполне применимы к методологии Rational Unified Process и языку UML. Отсюда вывод: язык UML может использоваться для отражения стратегических решений в системе при внесении пакетов и классов в модель для представления, реализации и документального описания этих решений.

### **Ключевые механизмы для задачи регистрации учебных курсов**

Поскольку многие разработчики владеют именно C++, а в дальнейшем систему планируется расширять для автоматизации других потребностей университета, то в качестве основного языка был выбран C++. Разработчики архитектуры выяснили, что для создания графического интерфейса пользователя (ГИП) потребуется определенный набор графических элементов управления, и в модель был добавлен пакет Элементы управления ГИП (GUI Controls). Стратегия хранения данных предполагает использование отдельного класса доступа к базе данных (теневого класса) для каждого информационного класса в системе. Существуют и другие стратегии, например с применением механизмов наследования. Но именно эта стратегия выбрана потому, что уже накоплен достаточный опыт в реализации такого метода хранения и он считается наименее рискованным. На текущем этапе в модель добавляется пакет для доступа к базе данных, содержащий необходимые теневые классы. Для обработки исключений решено использовать механизмы языка C++ *catch* и *throw*. Вместо обработчиков исключений в модель добавлен общий пакет Обработка ошибок (Error Handling). И наконец, в систему добавлен набор классов для реализации основных коммерческих операций Базовые средства (Foundation). Пакеты, представляющие ключевые решения для системы регистрации курсов, показаны на рис. 11.2.

Так как Обработка ошибок и Базовые средства используются всеми остальными пакетами системы, они являются глобальными пакетами (global packages).

<sup>1</sup> Solutions. Redwood City, CA: Addison-Wesley, 1995.

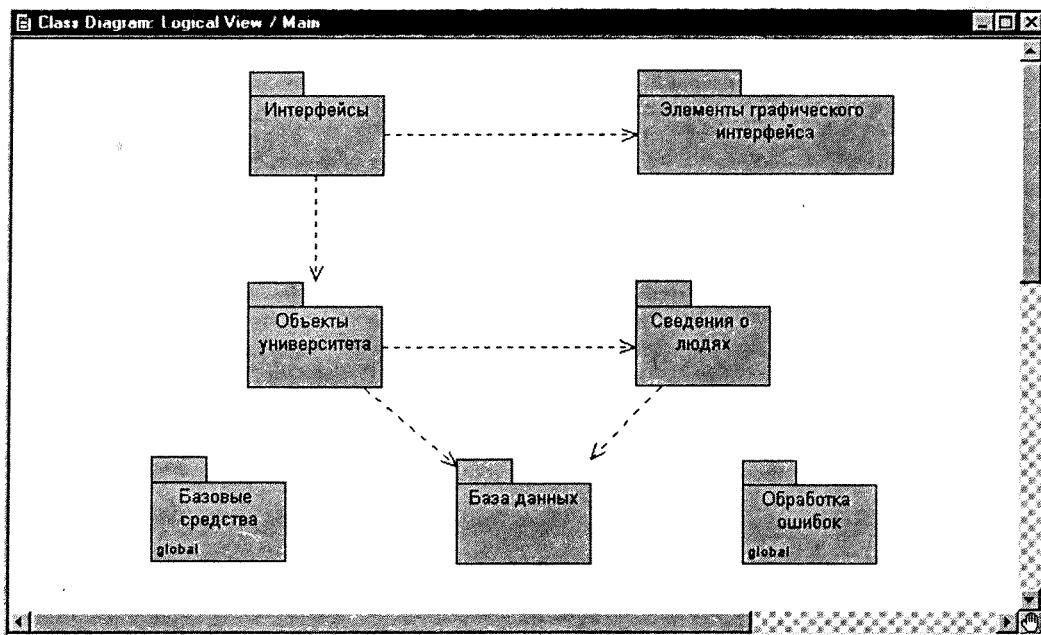


Рис. 11.2. Система регистрации учебных курсов

Выбор глобальных пакетов в программе Rational Rose состоит из следующих шагов:

1. Щелкните правой кнопкой мыши по пакету на диаграмме классов.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Параметры), чтобы вызвать диалоговое окно настройки параметров пакета.
3. Выберите вкладку **Detail** (Детально).
4. Установите флажок **Global** (Глобальный).
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров.

## Представление реализации

*Представление реализации* (implementation view) определяет реальную организацию программных модулей в среде разработки. Оно учитывает потребности в простоте разработки, управлении программными средствами, повторном использовании кода, а также языковых и инструментальных ограничениях.

Элементами моделирования в *представлении компонентов* (component view) являются пакеты, компоненты и связи между ними.

Пакет в данном представлении архитектуры – это физический раздел системы. Пакеты организованы в виде иерархии уровней или слоев, где каждый уровень имеет четко определенный интерфейс. На рис. 11.3 изображена типичная схема уровней системы.

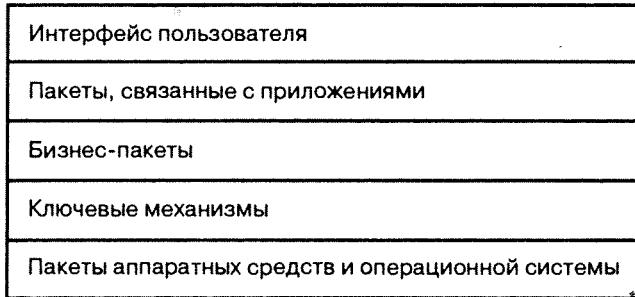


Рис. 11.3. Уровни системы

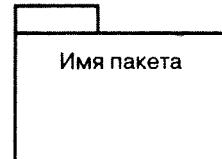


Рис. 11.4. Нотация языка UML для пакетов

Нотация языка UML для изображения пакетов в представлении компонентов напоминает изображение пакетов в логическом представлении – см. рис. 11.4.

Для создания пакетов в представлении компонентов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по разделу **Component View** (Представление компонентов) в окне браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Package** (Создать ⇒ Пакет). В список объектов браузера будет добавлен новый пакет **New Package**.
3. Введите нужное имя пакета.

Главная диаграмма компонентов обычно представляет определенные для системы пакеты.

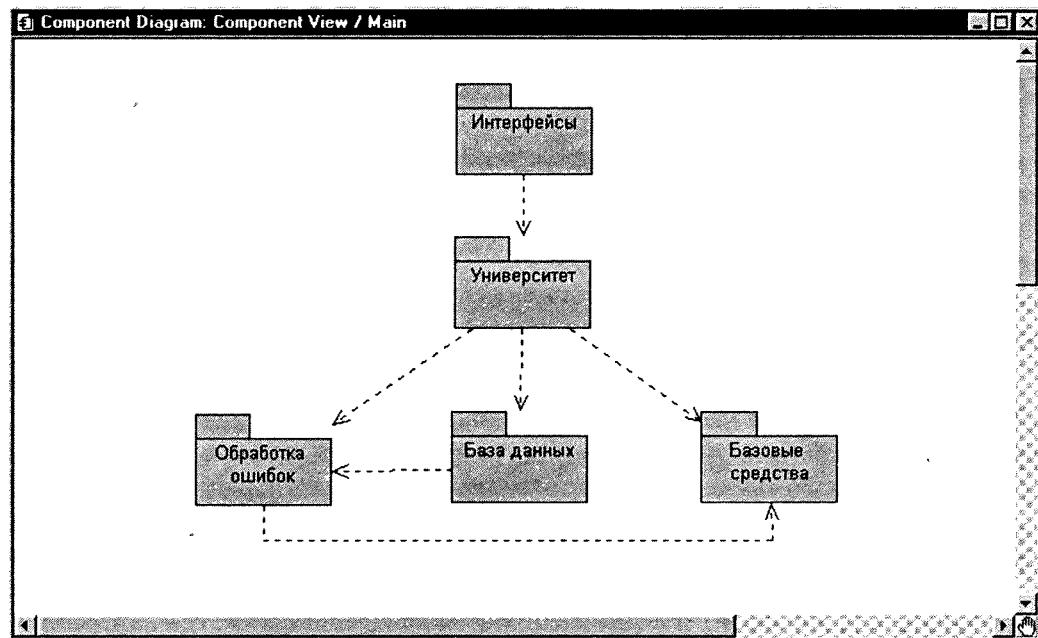


Рис. 11.5. Главная диаграмма компонентов

Чтобы получить главную диаграмму компонентов в программе Rational Rose:

1. Дважды щелкните по диаграмме **Main Diagram** (Главная диаграмма) в разделе **Component View** (Представление компонентов) в окне браузера, чтобы открыть диаграмму.
2. В списке браузера щелкните по пакету и перетащите его на диаграмму.
3. Повторите второй шаг для других пакетов, которые нужно поместить на диаграмму.
4. Чтобы добавить отношения зависимости, щелкните по кнопке **Dependency** (Отношение зависимости) на панели инструментов, затем по пакету-клиенту и проведите линию связи к пакету-поставщику.

Главная диаграмма компонентов для задачи регистрации учебных курсов показана на рис. 11.5.

### **Компоненты исходного кода**

В представлении компонентов модели компоненты исходного кода – это программные файлы, содержащиеся внутри пакетов. Тип файлов зависит от языка программирования (например, в C++ – файлы .h и .cpp, в Java – .java, в PowerBuilder – .pbl). Каждый компонент связан с каким-либо языком. Классы в логическом представлении отображаются на компоненты в представлении компонентов. Для C++ один класс отображается в один компонент. Однако иногда на один компонент может быть отображено больше одного класса. Это обычно происходит в том случае, когда между классами существует очень тесная связь. Например, контейнер и его итератор содержатся в одном .h- и одном .cpp-файле. Значит, класс-контейнер и класс-итератор будут отображаться на один компонент. Я также видела классы, которые используются как шаблон взаимодействия, отображаемый на один физический файл. Нотация языка UML для компонента показана на рис. 11.6.

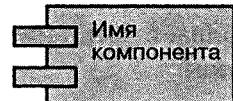


Рис. 11.6. Нотация языка UML для компонента

### **Программные компоненты в задаче регистрации учебных курсов**

Это относительно простая система, вот почему было принято решение обеспечить отображение один в один между классами и компонентами – каждый класс имеет собственный файл заголовка и .cpp-файл.

Для создания компонентов в программе Rational Rose:

1. Откройте диаграмму компонентов.
2. Щелкните по кнопке **Component** (Компонент) на панели инструментов.
3. Щелкните по диаграмме, чтобы поместить на нее компонент. Новый компонент также будет добавлен в список браузера.
4. Введите имя нового компонента.

Простая диаграмма компонентов показана на рис. 11.7.

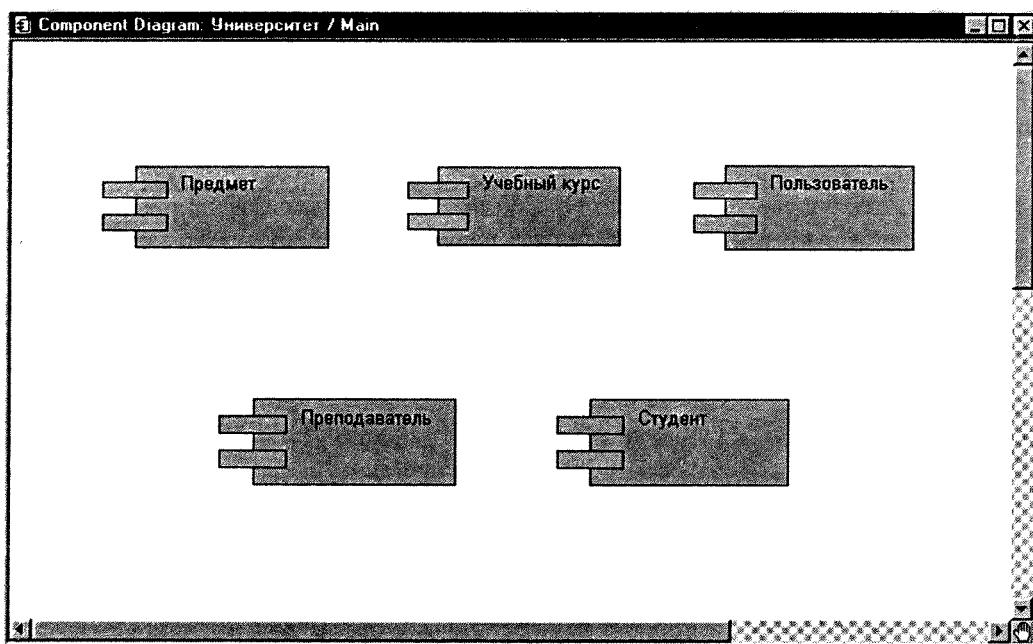


Рис. 11.7. Программные компоненты

Отображение классов на компоненты в программе Rational Rose предусматривает выполнение следующих действий:

1. Щелкните правой кнопкой мыши по компоненту в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. Щелкните по вкладке **Realize** (Реализация).
4. Щелкните правой кнопкой мыши по нужному классу в списке классов.
5. В появившемся контекстно-зависимом меню выберите команду **Assign** (Присвоить).
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

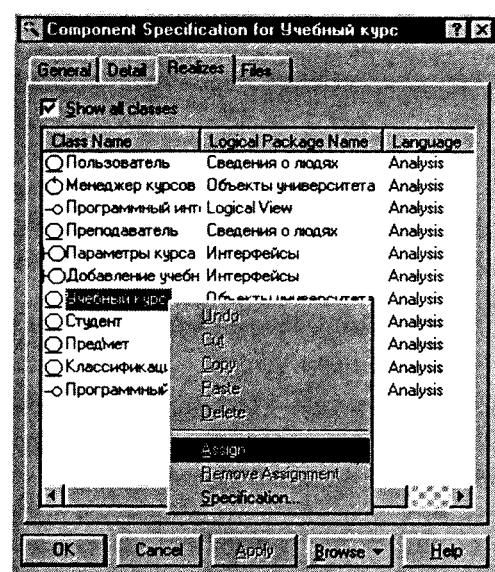


Рис. 11.8. Диалоговое окно настройки параметров для компонента учебный курс

Класс можно присвоить компоненту путем перетаскивания класса из окна браузера на изображение компонента в браузере или на диаграмме компонентов.

Диалоговое окно настройки параметров компонента учебный курс (Course Offering) показано на рис. 11.8.

## Представление процессов

*Представление процессов* (process view) отражает структуру программной реализации системы. Представление процессов учитывает такие потребности, как производительность, надежность, масштабируемость, целостность, управление системой и синхронизация. Компоненты также используются в этом представлении архитектуры. Для представления *программных* (run-time) и *исполняемых* (executable) компонентов системы создается диаграмма компонентов. Компоненты связаны отношением зависимости. Программные компоненты отображают классы на программные библиотеки, такие как Java-апплеты, элементы Active-X и динамические библиотеки. Исполняемые компоненты показывают интерфейсы и зависимости вызовов между исполняемыми модулями. Для демонстрации типа компонента может быть использован стереотип. Интерфейсные классы, созданные на этапе проектирования, изображаются с помощью *леденцовой нотации* (lollypop notation) – см. рис. 11.9.

В программе Rational Rose информация для представления процессов создается в виде диаграмм в представлении компонентов, содержащих либо программные, либо исполняемые компоненты. Диаграммы нужны для того, чтобы показать зависимости между компонентами различного типа в системе.

В системе регистрации учебных курсов созданы две динамические библиотеки (DLL) – для обработки информации о предметах и учебных курсах и для работы с базой данных. Такой подход был выбран исходя из возможных изменений в структуре курсов и в стратегии взаимодействия с базой данных.

При наличии изменений достаточно воспользоваться другой библиотекой. В системе есть три исполняемых модуля – один для регистратора, чтобы осуществлять ввод данных и управление информацией в системе; один для студента и один для преподавателя с целью получения доступа и использования системы. Между исполняемыми модулями нет никакого взаимодействия. Диаграмма компонентов для исполняемого модуля преподавателя (ProfessorOptions.exe) показана на рис. 11.10.

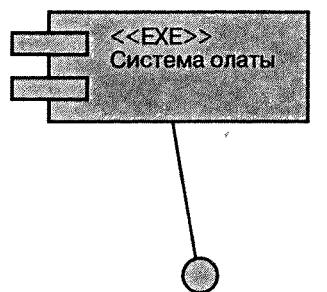


Рис. 11.9. Интерфейс компонента

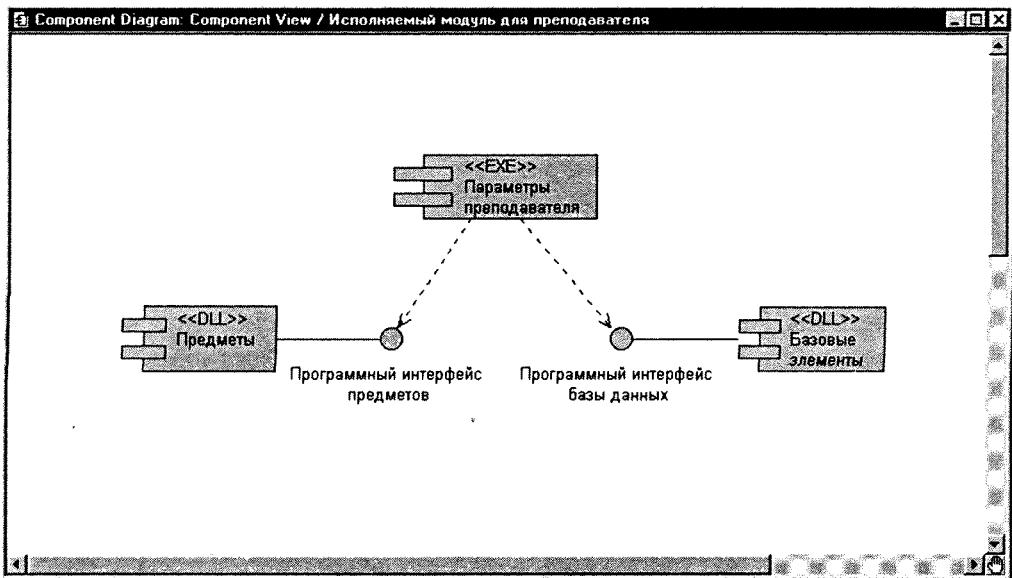


Рис. 11.10. Исполняемый модуль для преподавателя

## Представление средств внедрения

*Представление средств внедрения* (deployment view) отображает программные средства на узлы вычислительных систем (processing nodes). Оно показывает конфигурацию элементов обработки (processing elements) и работающих на них программных процессов. Представление средств внедрения учитывает такие потребности, как доступность системы, надежность, быстродействие и масштабируемость. Чтобы показать различные узлы вычислительных систем и связи между ними, создаются *диаграммы внедрения* (deployment diagrams). Такая диаграмма демонстрирует распределение компонентов по предприятию. Элементы обработки представлены в виде узлов вычислительных систем, которые соединены линиями, показывающими коммуникационные каналы между ними.

Программные процессы изображаются в виде текста, привязанного к узлу или группе узлов.

Такая диаграмма позволяет разработчикам архитектуры понять топологию системы и отобразить компоненты на исполняемые процессы. Здесь учитываются следующие вопросы: процессорная архитектура, скорость, емкость, пропускная способность каналов для взаимодействия процессов, физическое расположение аппаратных ресурсов, технология распределенной обработки.

### Диаграмма внедрения для системы регистрации учебных курсов

После изучения определенных для данной задачи компонентов, существующих аппаратных средств и оценки загруженности системы в период регистрации

на курсы разработчики архитектуры решили выделить пять вычислительных систем: одну – для запуска исполняемого модуля преподавателя, одну – для работы с базой данных и три – для регистрации студентов.

Последовательность создания диаграммы внедрения в программе Rational Rose:

1. Программа Rational Rose автоматически создает диаграмму внедрения. Чтобы открыть диаграмму, дважды щелкните по ней в окне браузера.
2. Чтобы создать узел, щелкните по кнопке **Processor** (Процессор) на панели инструментов, а затем по диаграмме.
3. Введите названия для нового узла вычислительной системы.
4. Для создания соединения между узлами щелкните по кнопке **Connection** (Соединение) на панели инструментов, а затем по одному из узлов на диаграмме внедрения и проведите линию связи к другому узлу.

Диаграмма внедрения для задачи регистрации учебных курсов показана на рис. 11.11.

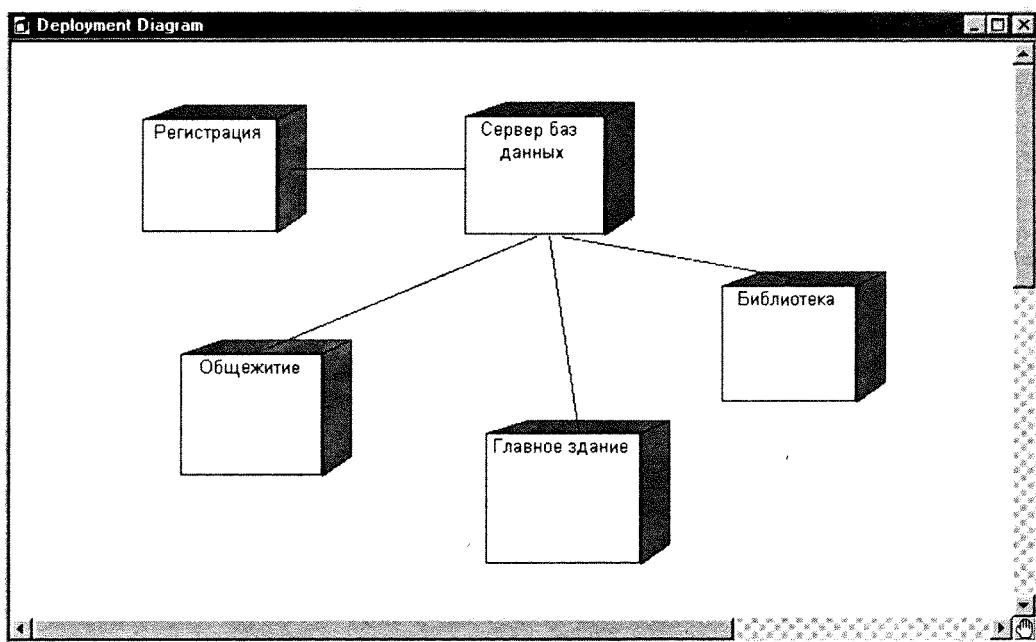


Рис. 11.11. Диаграмма внедрения

## Представление прецедентов

*Представление прецедентов* (use case view) архитектуры позволяет показать и проверить логическое представление, представления процессов, компонентов и средств внедрения. Диаграммы взаимодействий и диаграммы последовательности действий создаются для того, чтобы продемонстрировать, как различные элементы проектирования взаимодействуют для получения нужного поведения.

## Резюме

Программная архитектура состоит из нескольких одновременно развивающихся представлений: логического представления, представления процессов, представления компонентов и представления средств внедрения. Для их проверки разрабатываются сценарии. Хорошая архитектура состоит из четко определенных уровней абстракции, где существует разграничение между интерфейсом и реализацией каждого уровня. Ключевые механизмы отражают решения относительно общих стандартов, правил и норм. Чтобы показать архитектурную компоновку системы, создаются пакеты.

Архитектура также описывает физическую организацию системы. Для отражения физической реализации используется диаграмма компонентов. Диаграмма внедрения отображает конфигурацию аппаратных средств системы.



## Глава 12. Выпуск версий

### Процесс планирования версий

В *плане выпуска версий* (iteration release plan) представлены расписания для каждого шага развития системы. «Такой план должен определять серию архитектурных выпусков, постепенно расширяющихся по функциональности и в конечном счете охватывающих требования ко всей системе»<sup>1</sup>.

«В плане выпуска версий должны излагаться специфичные для версии цели:

- реализуемые возможности;
- уменьшаемые с данной версией риски;
- устранимые версией дефекты.

Критерии выхода:

- обновленные сведения о возможностях;
- обновленный план уменьшения рисков;
- документ, содержащий сведения о результатах выпуска версии;
- получение результатов тестирования продукта, включая список дефектов;
- план выпуска версий, содержащий измеримые вычисляемые критерии для оценки результатов следующей версии»<sup>2</sup>.

Сценарии, созданные на этапе анализа, являются основными входными данными для этой стадии разработки. Сценарии изучаются и сортируются согласно степени риска, важности для заказчика и потребности в первоочередной разработке определенных базовых сценариев. Эта задача наилучшим образом решается рабочей группой, в состав которой входят эксперт по предметной области, аналитик, специалист по архитектуре и специалисты по тестированию. «Сценарии должны быть сгруппированы так, чтобы для очередного выпуска они совместно обеспечивали реализацию значительной части поведения системы и указывали на необходимость рассмотрения следующего наибольшего риска»<sup>3</sup>.

После завершения очередной версии риски переоцениваются и план проекта при необходимости обновляется. «Для большинства проектов план определяет пять (плюс-минус два) промежуточных выпусков»<sup>4</sup>.

Процесс планирования версий представлен на рис. 12.1.

<sup>1</sup> Booch, Grady. *Object Solutions*. Redwood City, CA: Addison-Wesley, 1995.

<sup>2</sup> Kruchten, Philippe. *A Rational Development Process*. Rational Software Corporation. Available at [www.rational.com](http://www.rational.com).

<sup>3</sup> Booch, Grady. *Object Solutions*. Redwood City, CA: Addison-Wesley, 1995.

<sup>4</sup> Там же.

## Выпуск версий

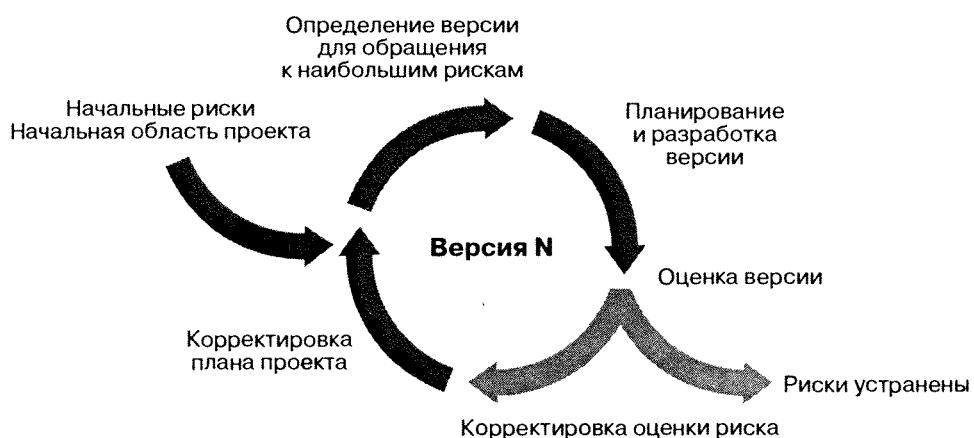


Рис. 12.1. Процесс планирования версий

### План выпуска версий для задачи регистрации учебных курсов

#### Версия 1:

- Сохранение сведений о преподавателе
- Выбор курсов для преподавания
- Создание учебного плана

#### Версия 2:

- Сохранение сведений о студенте
- Составление каталога

#### Версия 3:

- Регистрация на курсы
- Запрос списка учебных классов

Версия 1 содержит риск, связанный с базой данных. Сведения о курсах должны храниться в базе данных, которая доступна для всех. Сценарии Сохранение сведений о преподавателе и Выбор курсов для преподавания включены в данную версию, так как их необходимо завершить до составления каталога. С помощью второй версии добавляется функциональность, необходимая для регистрации студента (сведения о студентах должны быть помещены в базу данных, и каталог должен быть доступен студентам). Версия 3 завершает систему.

### Проектирование пользовательского интерфейса

На ранних стадиях жизненного цикла проекта для граничных классов системы были созданы пустые классы. Теперь они должны быть доработаны: нужно внести данные о количестве и расположении окон, а также создать обработчики событий, поступающих от пользователей. Для получения сведений о требованиях к пользовательскому интерфейсу применяются диаграммы последовательностей. Интерфейс пользователя позволяет получать все сообщения, поступающие от актера. Кроме того, он должен обеспечивать отправку всех сообщений, адресованных актеру.

## Проектирование пользовательского интерфейса для прецедента «Выбор курсов для преподавания»

После изучения сценариев, связанных с прецедентом Выбор курсов для преподавания (Select Courses to Teach), созданы окна для добавления, удаления и просмотра и определены следующие требования:

- преподаватель должен ввести пароль для входа в систему. Для ввода пароля создано отдельное окно;
- если пароль верен, на экране появляется окно **Параметры курса преподавателя** (ProfessorCourseOptions). Оно содержит поле для указания семестра и группу кнопок: **Добавить курс** (Add Course), **Удалить курс** (Delete Course), **Просмотр расписания** (Review Schedule), **Печать расписания** (Print Schedule).

В этой книге проектирование рассматривается на примере операции Добавить курс (Add Course) для данного сценария. Важно отметить, что проект системы не может основываться только на одном сценарии – он развивается по мере создания новых сценариев. Я выбрала один сценарий для иллюстрации нотаций Rational Unified Process и UML, которые могут быть использованы для представления различных решений при проектировании.

Операция Добавить курс подразумевает добавление преподавателя к учебному курсу в качестве учителя. Этот сценарий требует отдельного окна для ввода преподавателем необходимой информации. Я назвала это окно **Добавление** (Addition). Оно содержит следующие элементы:

- поле ввода **Название предмета** (Course name);
- поле ввода **Номер предмета** (Course number);
- список **Учебные курсы** (Course offerings);
- кнопку **OK**;
- кнопку **Отмена** (Cancel);
- кнопку **Выход** (Quit).

После того как преподаватель ввел название и номер предмета, система получит и отобразит список учебных курсов. Затем преподаватель может выбрать учебный курс. При щелчке по кнопке **OK** объекту предмет направляется сообщение.

Реализация пользовательского интерфейса зависит от выбранной библиотеки классов. Ее рассмотрение выходит за рамки данной книги. В этом разделе говорится о проектировании интерфейса. Как правило, он создается с помощью специальных инструментов для построения графических интерфейсов пользователя. В данном случае, после создания классов ГИП, могут быть использованы средства возвратного проектирования программы Rational Rose для добавления классов к модели.

## Добавление классов уровня проектирования

Классы обычно добавляются в модель, чтобы выяснить, как реализовать что-либо в системе. Например, в системе регистрации курсов определено, что преподаватель должен ввести пароль, который обязательно проверяется. Для реализации этой задачи в систему добавляется класс, проверяющий пароль. По мере добавления классов в систему они отображаются на соответствующих диаграммах.

Обновленная диаграмма классов для задачи регистрации учебных курсов изображена на рис. 12.2. В этой точке жизненного цикла я буду показывать только названия стереотипов, потому что в модель добавляется разная детальная информация.

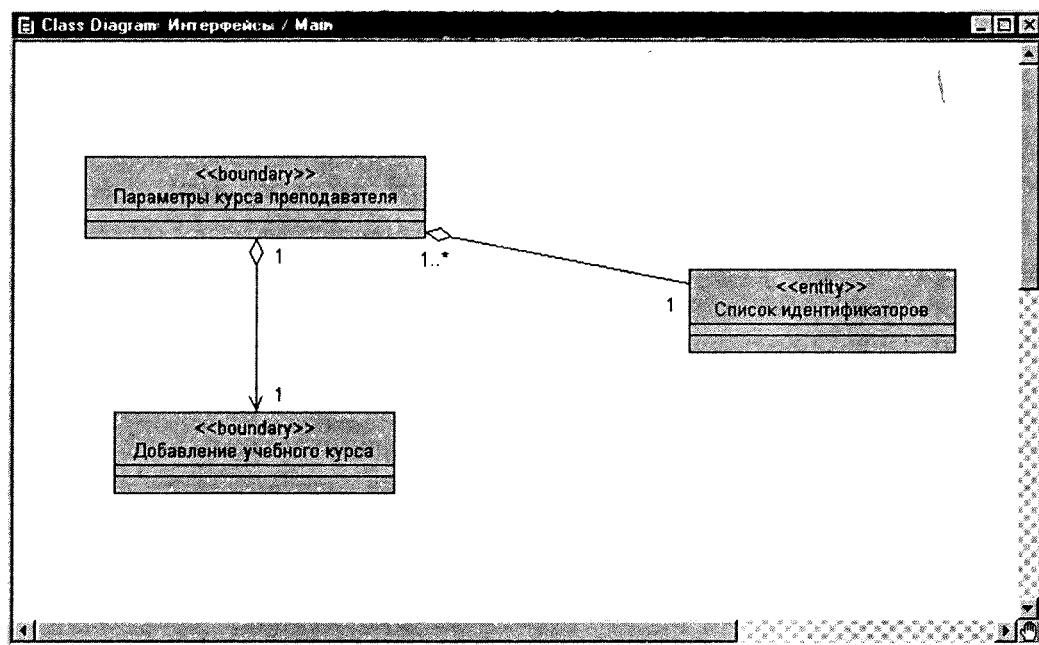


Рис. 12.2. Класс уровня проектирования

Вам необязательно поступать таким же образом. Чтобы стереотипы по умолчанию отображались только в виде названий, выберите в меню команду **Tools** ⇒ **Options** (Сервис ⇒ Параметры), затем вкладку **Diagram** (Диаграмма) и установите переключатель **Label** (Название) в группе переключателей **Stereotype display** (Отображение стереотипов).

## Использование шаблонов

*Шаблоны проектирования* (design patterns) содержат решения общих проблем при проектировании программ. Шаблоны проектирования используются в системе при решении вопросов «как». Говоря словами Грейди Буча, «шаблоны – это круто»<sup>1</sup>. Они позволяют повторно использовать удачные решения в области проектирования и архитектуры. Благодаря этому можно получить более простые в обслуживании системы и повысить производительность труда. Как и другие классы, создаваемые на этом этапе жизненного цикла, классы, составляющие шаблоны, добавляются в модель и на диаграмму классов. Например, шаблон абстрактный конструктор (Abstract Factory) может использоваться для получения объектов пользователь (RegistrationUser) различного типа. На сегодняшний

<sup>1</sup> Booch, Grady. *Best of Booch, SIGS Reference Library*. New York, NY, 1996, p. 167.

день издано много книг с описанием шаблонов проектирования. Одна из наиболее популярных – книга Е. Гаммы (E. Gamma) «Шаблоны проектирования: элементы многократно используемых объектно-ориентированных программ» (Design Patterns: Elements of Reusable Object-Oriented Software), выпущенная издательством Addison-Wesley в 1995 году.

## Проектирование отношений

На этапе проектирования отношений должны быть приняты решения относительно следующих вопросов: направленность (navigation), содержание (containment), уточнение (refinement) и реализация мощности (multiplicity implementation).

### Направленность

Ассоциации и агрегации являются двунаправленными отношениями. Во время проектирования ассоциативные связи проверяются, чтобы выяснить, действительно ли нужна двунаправленность. По возможности отношения создаются односторонними (то есть проходящими в одну сторону) – их так легче реализовать и поддерживать.

Для указания направленности отношения в программе Rational Rose:

1. Щелкните правой кнопкой мыши по линии ассоциативной или агрегационной связи.
2. В появившемся контекстно-зависимом меню выберите команду **Navigation** (Направленность), чтобы изменить направленность отношения.

Некоторые односторонние ассоциации показаны на рис. 12.3.

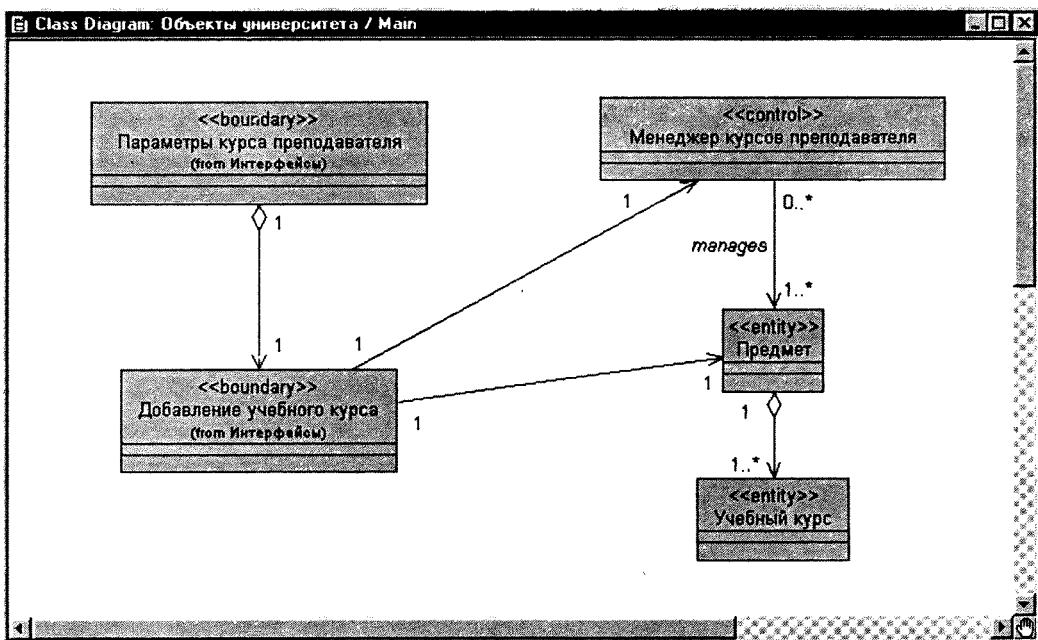


Рис. 12.3. Направленность отношений

## Содержание

В модели также необходимо определить тип содержания в агрегационном отношении. Содержание может быть реализовано по значению или по ссылке. Первое предполагает эксклюзивное владение для содержащего класса (агрегата) и изображается в виде закрашенного ромба. Второе не предполагает эксклюзивного владения и изображается в виде незакрашенного ромба.

Для указания типа агрегационного содержания в программе Rational Rose:

1. Дважды щелкните по линии агрегационной связи, чтобы открыть диалоговое окно настройки параметров отношения.
2. Выберите вкладку **Detail** (Детально) для роли, представляющей «целое» в агрегации.
3. Установите нужный тип содержания в группе переключателей **Containment** (Содержание).
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.

Отношение с содержанием по значению (класс параметры курса преподавателя (*ProfessorCourseOptions*) содержит класс добавление учебного курса (*AddACourseOffering*)) и отношение с содержанием по ссылке (отношение класса параметры курса преподавателя (*ProfessorCourseOptions*) к классу список доступных идентификаторов (*ValidIDList*)) показаны на рис. 12.4.

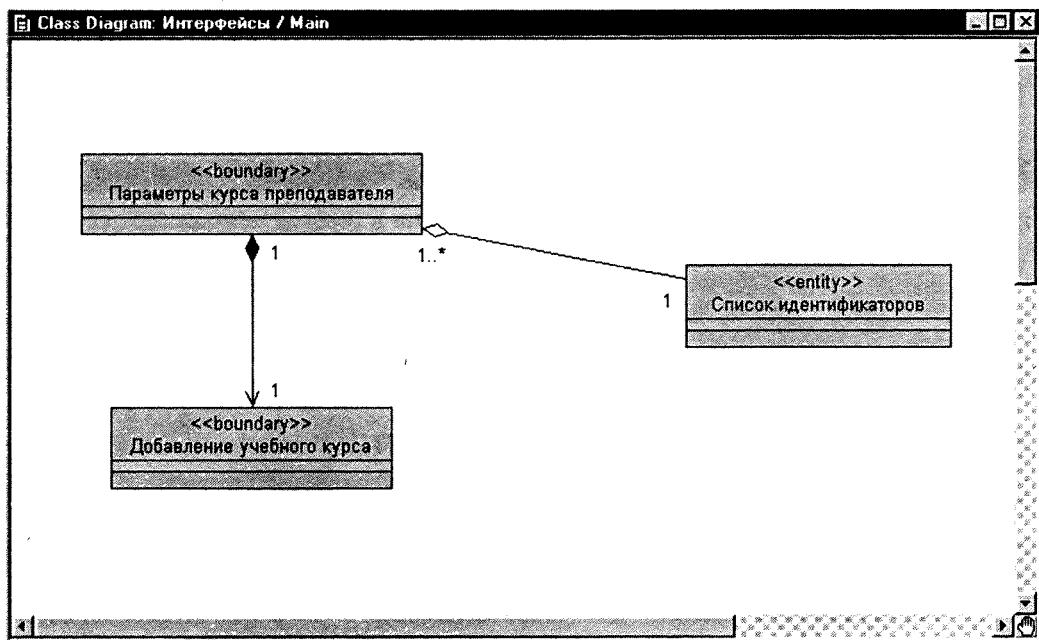


Рис. 12.4. Содержание

## Уточнение

На данном этапе ассоциативные отношения могут быть преобразованы в *отношения зависимости* (dependency relationship). Они означают следующее: объект,

запросивший услугу (клиент) у другого объекта (поставщика услуги), не имеет представления о расположении объекта-поставщика.

Ему необходимо сообщить, где находится объект-поставщик. Обычно объект-поставщик передается как параметр в один из методов класса-клиента или объявляется локально в области действия метода класса-клиента. Отношения зависимости изображаются пунктирной стрелкой, направленной от клиента к поставщику.

Последовательность создания отношений зависимости в программе Rational Rose:

1. Щелкните по кнопке **Dependency Relationship** (Отношение зависимости) на панели инструментов.
2. Щелкните по классу, выступающему в качестве клиента.
3. Перетащите линию связи к классу-поставщику.

Отношение зависимости между классами учебный курс (CourseOffering) и учебный курс БД (DBCourseOffering) показано на рис. 12.5.

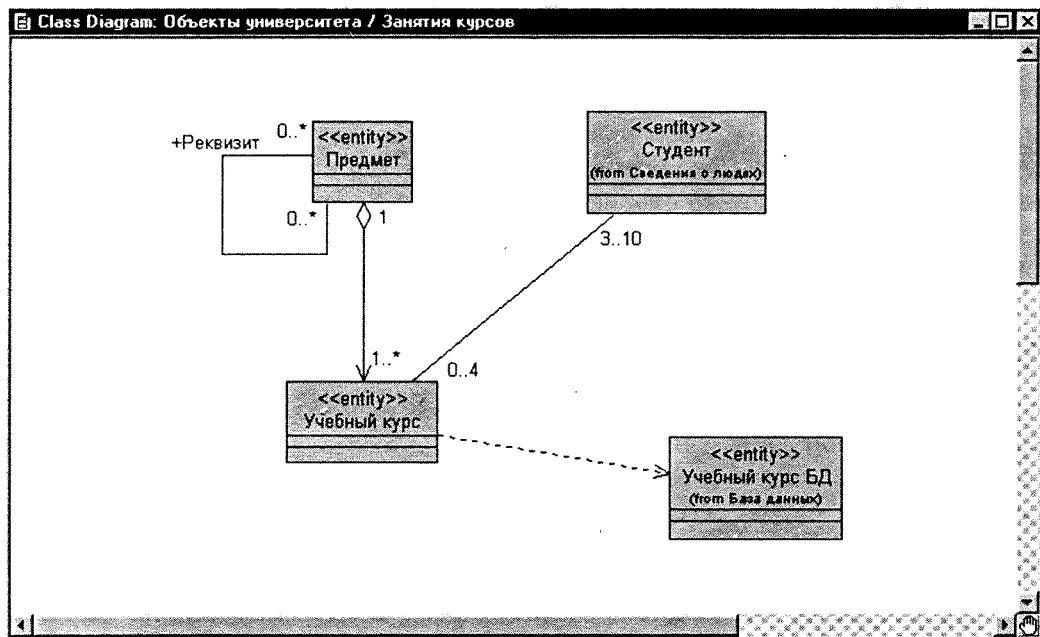


Рис. 12.5. Отношение зависимости

### Реализация мощности отношений

Мощность отношения со значением, равным единице, реализуется в виде встроенного объекта, ссылки или указателя. Мощность со значением больше единицы обычно реализуется с использованием класса-контейнера (то есть множества или списка). В этом случае список также может быть либо встроенным объектом, либо указателем на класс-контейнер. Решение об обновлении модели (чтобы показать все используемые классы-контейнеры) зависит от самого проекта.

Я обычно не показываю все контейнеры, чтобы не усложнять диаграмму. Вместо этого я устанавливаю параметры генерации кода в программе Rational Rose для выбора правильного контейнера.

### **Проектирование отношений в задаче регистрации учебных курсов**

Для сценария Добавление курса для преподавания (Add a Course to Teach) должны быть спроектированы следующие отношения:

- отношение классов параметры курса преподавателя (ProfessorCourse Options) и добавление учебного курса (AddACourseOffering);
- отношение классов параметры курса преподавателя и список доступных идентификаторов (ValidIDList);
- отношение классов добавление учебного курса и предмет (Course);
- отношение классов предмет и учебный курс (CourseOffering);
- отношение классов учебный курс и преподаватель (Professor);
- отношение классов учебный курс и учебный курс БД (DBCourseOffering).

### **Отношение классов «параметры курса преподавателя» и «добавление учебного курса»**

Как отмечалось ранее, реальный состав класса добавление учебного курса зависит от выбранных элементов пользовательского интерфейса. Проектировщики решили использовать отношение агрегации с содержанием по значению между классами, потому что время жизни окон взаимозависимо. Отношение направлено от класса добавление учебного курса к классу предмет.

### **Отношение классов «параметры курса преподавателя» и «список доступных идентификаторов»**

Класс список доступных идентификаторов используется во многих окнах для проверки правильности указанного имени пользователя. Проектировщики решили использовать одностороннюю ассоциацию от класса параметры курса преподавателя к классу список доступных идентификаторов.

### **Отношение классов «добавление «учебного курса» и «предмет»**

Это также класс пользовательского интерфейса. Проектировщики решили использовать отношение, направленное от класса добавление учебного курса к классу предмет.

### **Отношение классов «предмет» и «учебный курс»**

Это отношение направлено в одну сторону – от класса предмет к классу учебный курс. Кроме того, это агрегационное отношение с содержанием по значению, так как все взаимодействия с классом учебный курс проходят через класс предмет.

В данном проектном решении предполагается, что классы предмет и учебный курс взаимозависимы. В соответствии со сценарием так и должно быть. Но давайте рассмотрим получение выписки для студента. Если класс выписка

(Transcript) – это всего лишь указатель на учебные курсы, которые прослушал студент, то такая схема не подойдет. Классы предмет и учебный курс в этом случае будут существовать независимо (и агрегация должна быть реализована по ссылке). Таким образом, всегда есть отношения, которые могут видоизменяться при появлении новых сценариев.

### Отношение классов «учебный курс» и «преподаватель»

Проектное решение, основанное исключительно на данном сценарии, показывает, что отношение между классами учебный курс и преподаватель должно быть отношением зависимости. Это видно из того, что объект преподаватель является параметром операции добавить преподавателя (addProfessor) класса учебный курс. Однако есть и другой сценарий для этого прецедента – просмотр расписания (Review schedule). Здесь объект преподаватель должен «знать» связанные с ним объекты учебный курс. Следовательно, отношения зависимости не будет. Кроме того, сценарий Создание каталога (Create catalog) требует, чтобы каждый класс учебный курс был проинформирован о назначенному преподавателе. На основе этих сведений можно заключить, что отношение не меняется – это двунаправленная ассоциация.

### Отношение классов «учебный курс» и «учебный курс» БД

Это отношение может стать отношением зависимости, потому что объект учебный курс БД передается объекту учебный курс как параметр при вызове операции сохранения.

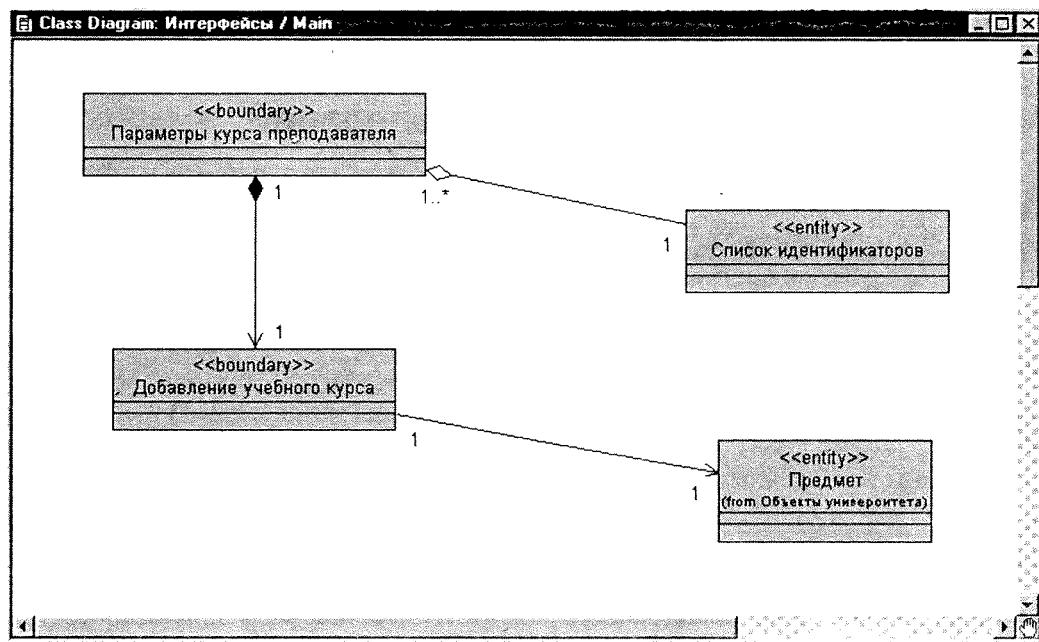


Рис. 12.6. Обновленная главная диаграмма классов для пакета Интерфейс

Обновленные диаграммы классов, выполненные при проектировании решения для отношений, показаны на рис. 12.6 и 12.7. Эти решения, возможно, претерпят изменения при добавлении прецедентов и сценариев.

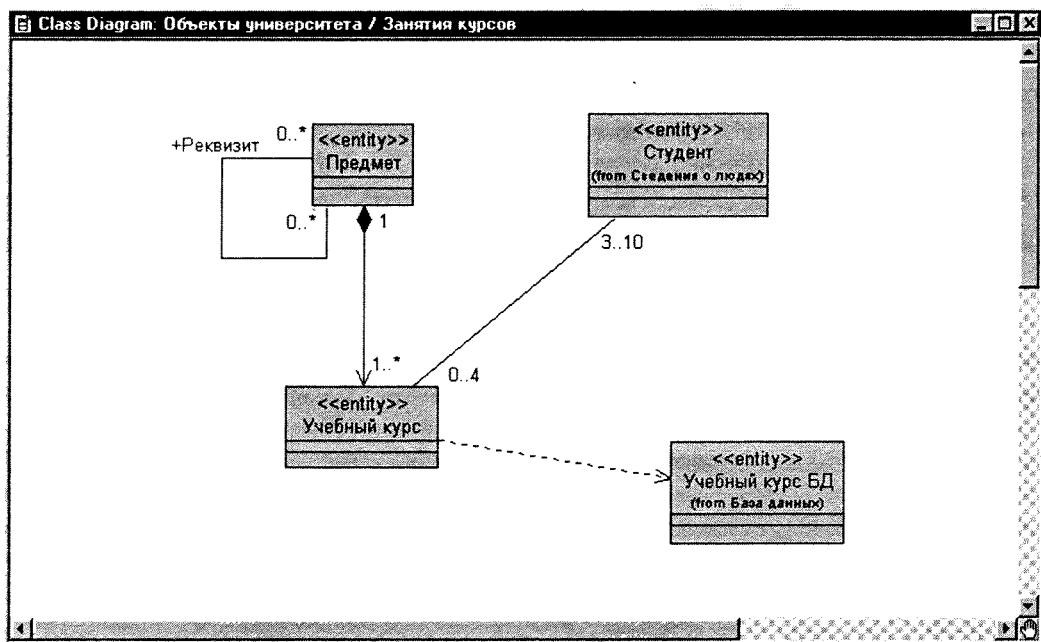


Рис. 12.7. Обновленная диаграмма классов

## Проектирование атрибутов и операций

Во время анализа достаточно указать только имена для атрибутов и операций. На этапе проектирования для атрибутов необходимо указать типы данных и начальные значения, а для операций – сигнатуры. Тип данных для атрибута может определяться языком программирования. Это может быть, например, целочисленный (integer) тип либо более сложный – строка (string) из библиотеки классов. Если требуется инициализация атрибута начальным значением при создании объекта класса, то эти сведения также помещаются в класс. В терминах методологии сигнатура операции включает список параметров (parameter list) и возвращаемый класс (return class). Для параметров и возвращаемого класса также должны быть указаны типы данных. Для атрибутов и операций необходимо определить тип доступа: общедоступный (public), защищенный (protected) или скрытый (private). Атрибуты обычно скрыты, тогда как операции могут быть скрытыми или общедоступными. Если класс является частью иерархии наследования, атрибуты и операции могут быть объявлены как защищенные, чтобы к ним получили доступ классы-потомки.

Чтобы установить типы данных и начальных значений атрибутов в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. Выберите вкладку **Attributes** (Атрибуты).
4. Щелкните по полю ввода начального значения или типа данных для перехода в режим редактирования.
5. Введите нужный тип данных или начальное значение атрибута.

Последовательность определения сигнатур операций в программе Rational Rose:

1. Щелкните правой кнопкой мыши по классу в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры). Откроется диалоговое окно **Class Specification** (Параметры класса).
3. Выберите вкладку **Operations** (Операции).
4. Дважды щелкните по операции, чтобы вызвать диалоговое окно **Operation Specification** (Параметры операции).
5. Укажите возвращаемый класс на вкладке **General** (Общие).
6. Выберите вкладку **Detail** (Детально).
7. Щелкните правой кнопкой мыши по списку **Arguments** (Аргументы).
8. В появившемся контекстно-зависимом меню выберите команду **Insert** (Добавить). В список аргументов будет добавлен новый аргумент. Укажите для него имя, тип данных и значение по умолчанию.
9. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Operation Specification**.
10. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Class Specification**.
11. Чтобы получить сигнатуру операции на диаграмме классов, воспользуйтесь настройкой параметров отображения, выбрав команду меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры).
12. Чтобы вывести сигнатуру операции только для определенных классов, выделите нужные классы и выберите команду меню **Format** ⇒ **Show Operation Signature** (Формат ⇒ Показать сигнатуру операции).

Атрибуты и операции классов можно также указать непосредственно на диаграмме классов, выбрав нужный элемент и воспользовавшись следующим форматом:

атрибут : тип = начальное значение

операция (аргумент : тип = значение по умолчанию) : возвращаемый класс

Некоторые операции и атрибуты для задачи регистрации учебных курсов показаны на рис. 12.8.

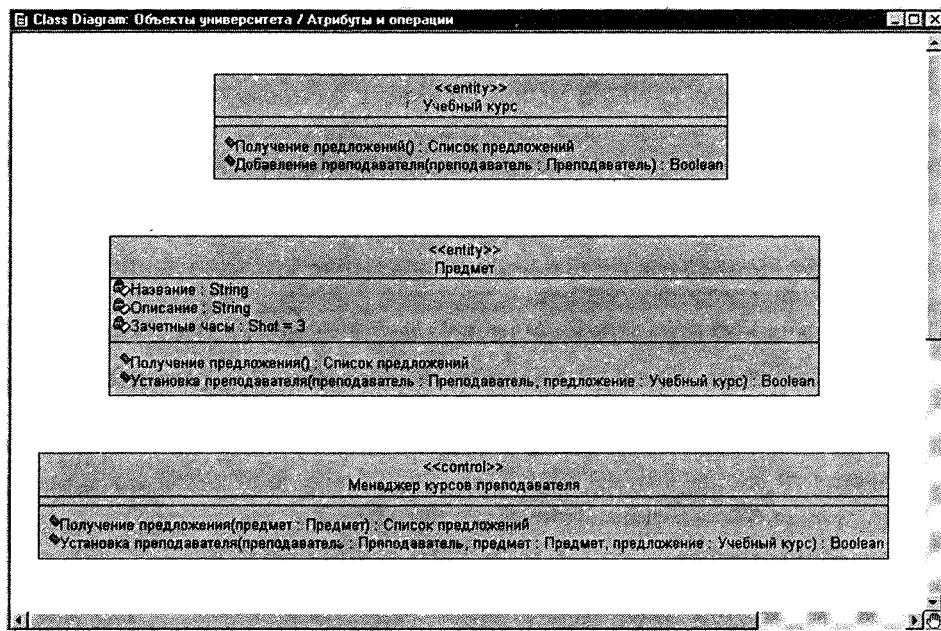


Рис. 12.8. Атрибуты и операции на уровне проектирования

## Проектирование наследования

Во время анализа были определены иерархии наследования для ключевых абстракций. На этапе проектирования эти иерархии дорабатываются, чтобы:

- повысить степень повторного использования;
- добавить классы уровня проектирования;
- добавить классы из выбранных библиотек.

Диаграммы, созданные в ходе анализа, просматриваются для выявления общности атрибутов, операций и отношений. Для вновь обнаруженных общих элементов определяются суперклассы. Это уменьшает общий объем кода и способствует тому, что одно и то же свойство не будет проявляться по-разному в различных классах, если оба класса наследуют его из общего суперкласса.

## Проектирование и генерация кода

Последний шаг на стадии проектирования версии – добавление методов, необходимых каждому классу C++, например конструкторов, деструкторов, копирующих конструкторов. Их можно добавить вручную, но это достаточно трудоемко. Поэтому средства генерации кода в программе Rational Rose позволяют добавлять методы такого типа.

Программа Rational Rose содержит разные средства для генерации кода. Код формируется на основе информации, полученной из диаграмм, спецификаций и параметров, указанных в свойствах генерации кода для всех элементов каждого типа. Подробное руководство по созданию кода в Rational Rose представлено в приложениях.

## Кодирование, тестирование и документирование версии

Один из заключительных шагов в построении версии – реализация содержания методов на выбранном языке программирования. В этом процессе используются диаграммы взаимодействий, потому что они отражают следующую информацию: кто сделал, что сделал – для кого и когда.

Тестирование представляет собой очень важную составляющую интерактивного и инкрементального жизненного цикла. В ходе анализа и проектирования появляются планы и процедуры тестирования. Для этих целей используются precedents, так как они описывают то, что система должна выполнять. Версию нужно протестировать на предмет выполнения задач, определенных в precedентах. Новые версии также объединяются с предыдущими версиями – вас не интересует готовность всей системы, чтобы собрать их вместе. Версия оценивается для того, чтобы выяснить, устраниет ли она связанные с ней риски. Все неустраниемые риски (а также вновь появившиеся) присваиваются следующей версии.

Решения, принятые относительно проектирования версий, отражаются в моделях версий. Эта информация используется для получения документации, которая должна составляться итеративным образом. Я обнаружила, что системы, требующие завершения проекта, чтобы приступить к документированию, редко имеют хорошую документацию (очень часто ее нет совсем).

## Использование возвратного проектирования для подготовки очередной версии

В ходе реализации текущей версии модель необходимо обновить, чтобы отразить в ней все изменения, выполненные в коде (добавленные методы или новые классы). Вместо обновления вручную можно воспользоваться *возвратным проектированием* (reverse engineering) программы Rational Rose, позволяющим получить модель на основе реализации. Эта информация может быть добавлена к уже существующей модели. Подробное руководство по средствам возвратного проектирования в программе Rational Rose 2000 представлено в приложениях.

## Резюме

План выпуска версий содержит расписания для каждого шага развития системы. Сценарии, созданные в процессе анализа, являются основными входными данными на этой стадии разработки. Сценарии изучаются и сортируются согласно степени риска, важности для заказчика и потребности в первоочередной разработке определенных базовых сценариев.

После выпуска очередной версии риски переоцениваются, и план проекта при необходимости обновляется.

На ранних стадиях жизненного цикла проекта для граничных классов системы создаются пустые классы, после чего они должны быть доработаны. В них нужно

внести данные о количестве и расположении окон, а также создать обработчики событий, поступающих от пользователей. Классы обычно добавляются в модель, чтобы была точная информация о том, как реализовать что-либо в системе. Шаблоны позволяют повторно использовать удачные решения в области проектирования и архитектуры. Это способствует созданию более простых в обслуживании систем и повышению производительности труда. Как и другие классы, создаваемые на этом этапе жизненного цикла, классы, составляющие шаблоны, добавляются в модель и на диаграмму классов. На этапе проектирования для отношений должны быть определены следующие параметры: направленность, содержание, уточнение и реализация мощности; для атрибутов – типы данных и начальные значения, для операций – сигнатуры. Диаграммы, полученные в ходе анализа, просматриваются для определения наследования. Операции в иерархии наследования изучаются для выявления полиморфизма. При его обнаружении операция объявляется виртуальной (*virtual*) или чисто виртуальной (*rige virtual*).

Последний шаг на стадии проектирования версии – добавление методов, необходимых каждому классу (например, конструкторов, деструкторов или копирующих конструкторов, если выбран язык программирования C++). В программе Rational Rose 2000 предусмотрены средства для формирования кода. Код генерируется на основе информации, полученной из диаграмм, спецификаций и параметров, указанных в свойствах генерации кода для всех элементов каждого типа.

На заключительном этапе реализуется содержание методов средствами выбранного языка программирования. Каждая версия должна быть протестирована и описана в документации, только после этого ее можно считать завершенной. Версия оценивается, и все неустранимые риски присваиваются следующей версии.



## **Приложение А. Генерация кода и возвратное проектирование для C++**

В этом приложении содержится подробная информация по генерации кода на языке C++ и возвратному проектированию.

### ***Этапы генерации кода***

1. Создание необходимого набора параметров.
2. Создание компонентов для тела пакета на диаграмме компонентов.
3. Назначение языка C++ компонентам.
4. Связывание классов с компонентами.
5. Привязка наборов параметров к элементам моделирования.
6. Выбор компонентов и генерация кода.
7. Оценка ошибок при генерации кода.

### ***Этапы возвратного проектирования***

1. Создание проекта.
2. Добавление заголовка проекта.
3. Добавление связанных библиотек и базовых проектов.
4. Установка типа файлов и их анализ.
5. Оценка ошибок.
6. Настройка параметров экспорта и экспорт в Rational Rose.
7. Обновление модели в Rational Rose.

## **Генерация кода**

### ***Этап 1. Создание необходимого набора параметров***

Для класса, роли, атрибута, операции и проекта в целом существуют параметры генерации кода. К параметрам, применяемым ко всему проекту, относятся имя файла, имя основного контейнера и место генерации кода. Параметры для класса определяют генерацию конструктора, деструктора, копирующего конструктора, операторов сравнения и методов установки/получения данных (get/set methods). Параметры для роли управляют созданием методов установки/получения данных, видимостью методов и определяют используемый класс-контейнер. Параметры операции задают тип операции (общая, виртуальная, абстрактная, статическая, дружественная) и позволяют ей стать константой. Наборы параметров могут

редактироваться. Также могут создаваться новые наборы параметров, чтобы указать особенности C++, требующиеся в проекте.

Для каждого класса создаются два файла: файл заголовка (.h) и файл спецификаций (.cpp).

Обычно определением параметров генерации кода, которые использует вся команда разработчиков, занимается несколько человек. Это позволяет каждому разработчику получать нужный код компонентов. Типичные наборы параметров могут быть такими: виртуальный деструктор, виртуальная операция, абстрактная операция, статическая операция, функция, не возвращающая данные, опережающее определение.

Для создания наборов параметров в программе Rational Rose:

1. Выберите команду меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры).
2. Выберите вкладку **C++**.
3. Укажите нужный тип набора параметров в открывшемся списке **Type** (Тип).
4. Щелкните по кнопке **Clone** (Клонировать), чтобы открыть диалоговое окно **Clone Property Set** (Клонированный набор параметров).

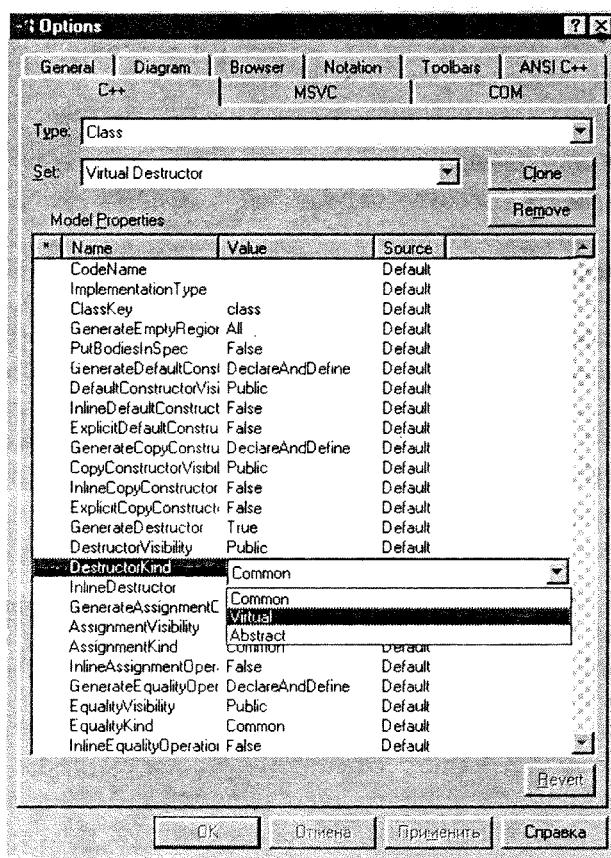


Рис. А. 1. Набор параметров виртуальный деструктор

5. Введите название нового набора параметров.
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.
7. В списке параметров щелкните по параметру, который требуется изменить.
8. Щелкните по значению параметра.
9. Введите или выберите из открывшегося списка новое значение параметра.
10. Аналогичным образом измените значения других параметров.
11. Щелкните по кнопке **Apply** (Применить), чтобы сохранить изменения.
12. Повторите выполненные действия для каждого нового набора параметров.
13. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Options** (Параметры).

Набор параметров виртуальный деструктор (Virtual Destructor) показан на рис. А.1.

### **Этап 2. Создание компонентов тела пакета на диаграмме компонентов**

Программа Rational Rose генерирует код на основе компонентов и их стереотипов, расположенных на диаграммах. Для компонентов без стереотипов создается h-файл, содержащий определение и декларацию класса. Для компонентов со стереотипом заголовок пакета (Package Specification) создается h-файл, включающий определение класса.

Если существует компонент со стереотипом тело пакета (Package Body), то для него создается файл .spp, содержащий декларацию класса.

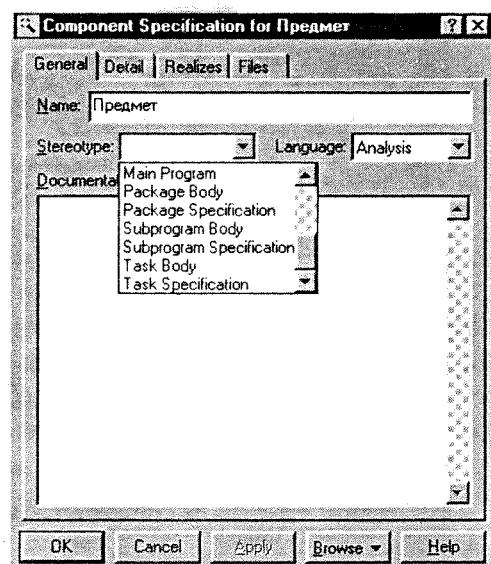
Последовательность указания стереотипов для компонентов в программе Rational Rose:

1. Дважды щелкните по диаграмме компонентов, чтобы открыть ее.
2. Щелкните правой кнопкой мыши по компоненту на диаграмме.
3. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
4. Выберите или введите нужный стереотип в открывшемся списке **Stereotype** (Стереотип).
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Диалоговое окно настройки параметров компонента показано на рис. А.2.

Для создания заголовка и тела компонентов в программе Rational Rose:

1. Дважды щелкните по диаграмме компонентов, чтобы открыть ее.



*Рис. А.2. Диалоговое окно настройки параметров компонента*

2. Щелкните правой кнопкой мыши по компоненту на диаграмме.
3. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
4. Для файла заголовка выберите стереотип **Package Specification** (Заголовок пакета) в открывавшемся списке **Stereotype**.
5. Для тела компонента выберите стереотип **Package Body** (Тело пакета) в открывавшемся списке **Stereotype**.
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Обновленная диаграмма с компонентами для h- и cpp-файлов C++ показана на рис. А.3.

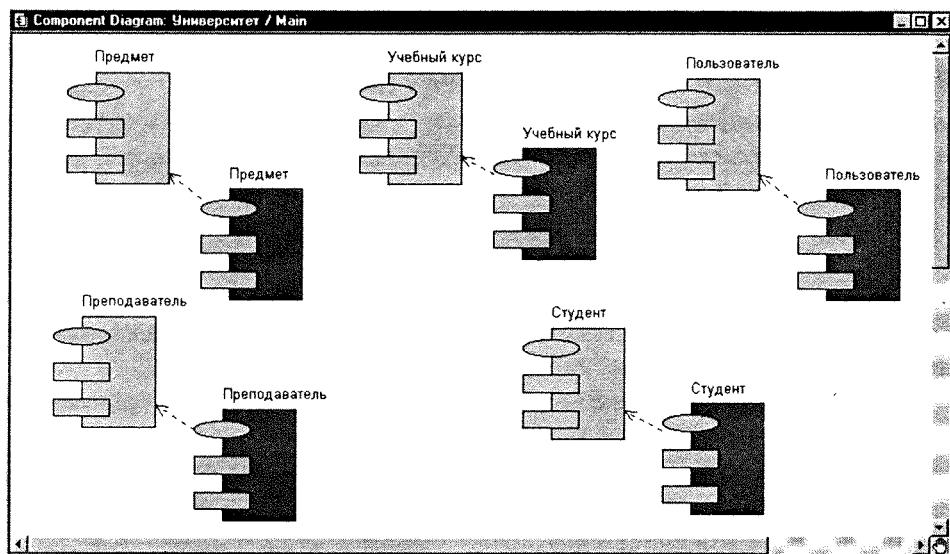


Рис. А.3. Обновленная диаграмма компонентов

### Этап 3. Назначение языка C++ компонентам

После создания компонентов для заголовка и тела им необходимо назначить язык C++. Если для модели по умолчанию выбран язык C++ (устанавливается на вкладке **Notation** (Нотация) диалогового окна настройки параметров, вызываемого командой меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры)), программа Rational Rose автоматически назначит его всем компонентам модели.

Последовательность назначения языка компоненту в программе Rational Rose:

1. Щелкните правой кнопкой мыши по компоненту в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. В открывавшемся списке **Language** (Язык) выберите **C++**.

4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Параметры компонента для класса предмет (Course) показаны на рис. А.4.

#### **Этап 4. Связывание классов с компонентами**

После создания компонентов устанавливается связь классов с компонентами, представляющими файлы заголовков.

Чтобы связать классы с компонентами в программе Rational Rose:

1. Дважды щелкните по диаграмме компонентов, содержащей компоненты для h- и cpp-файлов, чтобы открыть ее.
2. В списке браузера щелкните по классу и перетащите его на компонент, представляющий h-файл.

#### **Этап 5. Привязка наборов параметров**

##### **к элементам моделирования**

Каждый элемент моделирования (класс, атрибут или роль) изучается на предмет особенностей генерации кода. Если для элемента требуется набор параметров, отличный от используемого по умолчанию, то он привязывается к элементу моделирования.

Привязка набора параметров к выбранному элементу в программе Rational Rose предусматривает выполнение следующих действий:

1. Щелкните правой кнопкой мыши по элементу в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).

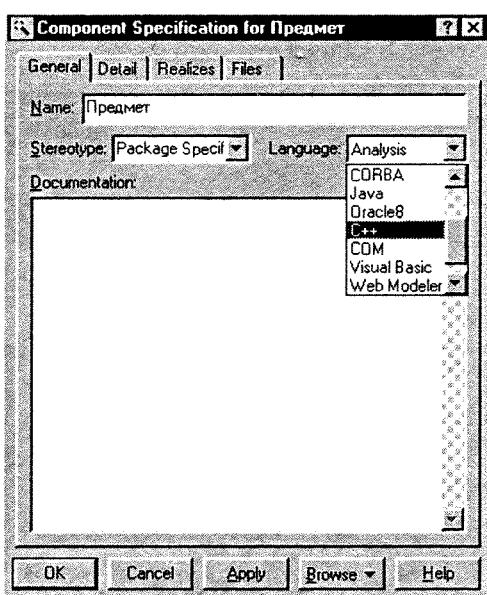


Рис. А.4. Назначение языка компоненту

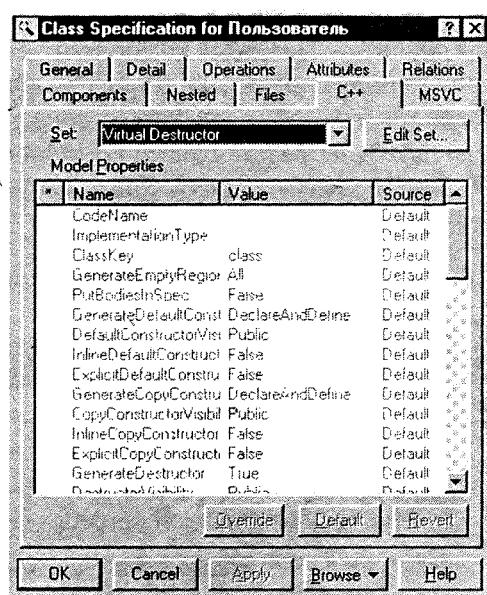


Рис. А.5. Привязка набора параметров

3. Выберите вкладку **C++**.
4. В открывающемся списке **Set** (Набор) укажите нужный набор параметров.
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров элемента.

Набор параметров виртуальный деструктор (*Virtual Destructor*) привязан к классу пользователь (*RegistrationUser*) – см. рис. А.5.

Поскольку для каждой комбинации элементов нельзя подобрать набор параметров, некоторые из них можно изменить. Это справедливо и в том случае, когда параметр является частью набора, используемого по умолчанию.

Для изменения параметра в программе Rational Rose:

1. Щелкните правой кнопкой мыши по элементу в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. Выберите вкладку **C++**.
4. В открывающемся списке **Set** (Набор) укажите нужный набор параметров.
5. В списке параметров щелкните по значению, которое требуется изменить.
6. Введите или выберите из открывающегося списка новое значение параметра.
7. Аналогичным образом измените значения других параметров.
8. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров элемента.

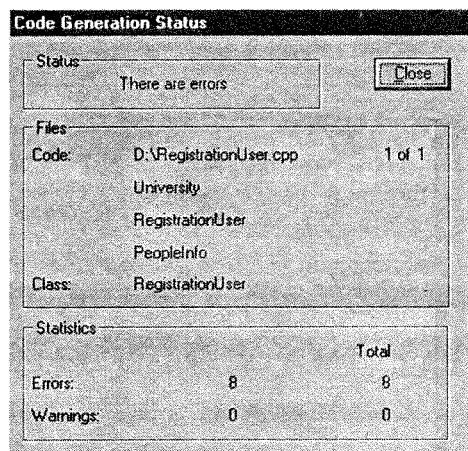
### **Этап 6. Выбор компонентов и генерация кода**

Код может быть сгенерирован для всего пакета, для компонента или набора компонентов. Название компонента используется в качестве имени файла, содержащего полученный код. Файл с кодом помещается в папку, соответствующую названию пакета в представлении компонентов.

Последовательность генерации кода в программе Rational Rose:

1. Выберите пакет, компонент или набор компонентов.
2. Выберите команду меню **Tools** ⇒ **C++** ⇒ **Code Generation** (Сервис ⇒ C++ ⇒ Генерация кода).
3. Процесс генерации кода будет отображаться в окне **Code Generation Status** (Состояние генерации кода).

Окно **Code Generation Status** показано на рис. А.6.



*Рис. А.6. Окно **Code Generation Status***

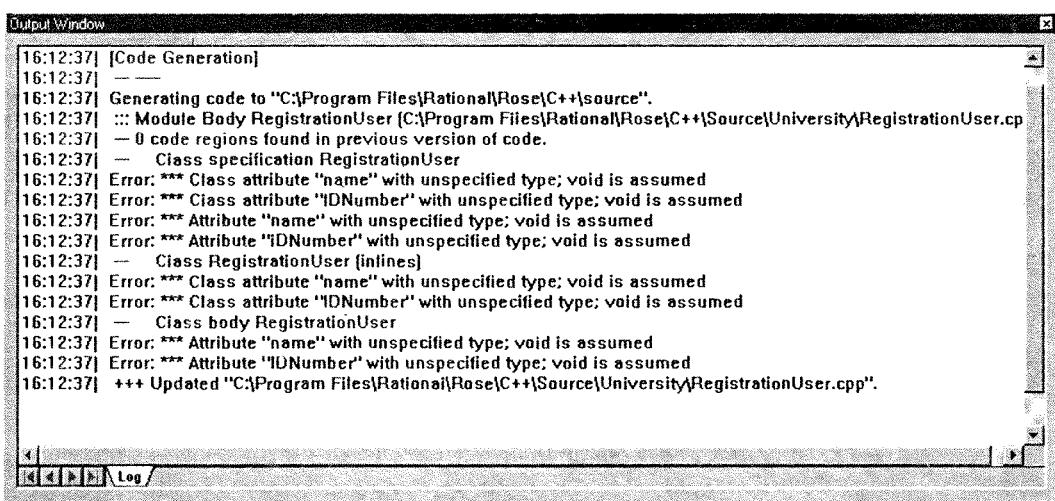
### **Этап 7. Оценка ошибок при генерации кода**

Программа Rational Rose выводит все ошибки и предупреждения в окне Log (Журнал). Если проектирование класса частично не завершено, то будут взяты значения по умолчанию, а в журнале появится предупреждение. Это особенно важно при использовании итеративного подхода в разработке, когда класс не реализуется полностью в одной версии.

Приведу примеры предупреждений и ошибок при генерации кода:

- Error: Missing attribute data type. Void is assumed.  
(Ошибка: Не указан тип данных атрибута. Подразумевается void.);
- Warning: Unspecified multiplicity/cardinality indicators. One is assumed.  
(Предупреждение: Не задан индикатор множественности. Подразумевается единица.);
- Warning: Missing operation return type. Void is assumed.  
(Предупреждение: Не указан тип возвращаемого значения для операции. Подразумевается void.).

Окно Log показано на рис. А.7.



*Рис. А.7. Окно Log в программе Rational Rose*

## **Возвратное проектирование с использованием анализатора кода C++**

### **Этап 1. Создание проекта**

Проект в анализаторе кода C++ (Rational Rose C++ Analyzer) содержит информацию, необходимую для получения элементов проектирования по исходному коду в файлах. В проекте анализатора указываются следующие сведения:

- заголовок (Caption) – информационное описание проекта;
- каталоги (Directories) – список каталогов, используемых анализатором. Каталоги, содержащие файлы с исходным кодом и сопутствующие файлы, должны быть включены в список каталогов;
- расширения (Extensions) – список расширений, распознаваемых анализатором;
- файлы (Files) – список файлов, которые требуется проанализировать;
- определенные символы (Defined Symbols) и неопределенные символы (Undefined Symbols) – список символов препроцессора и их расширений;
- категории (Categories) – список пакетов, которым могут присваиваться классы и пакеты;
- подсистемы (Subsystems) – список пакетов, которым могут присваиваться компоненты и пакеты;
- базы (Bases) – список базовых проектов, требующийся для разрешения ссылок в исходном коде;
- контекст типа 2 (Type 2 Context) – директивы препроцессора, требующиеся зависимым от контекста файлам исходного кода;
- параметры экспорта (Export Options) – список информации, которая экспортируется для создания или обновления модели в Rational Rose.

После создания проект сохраняется в файле с расширением .pjt.

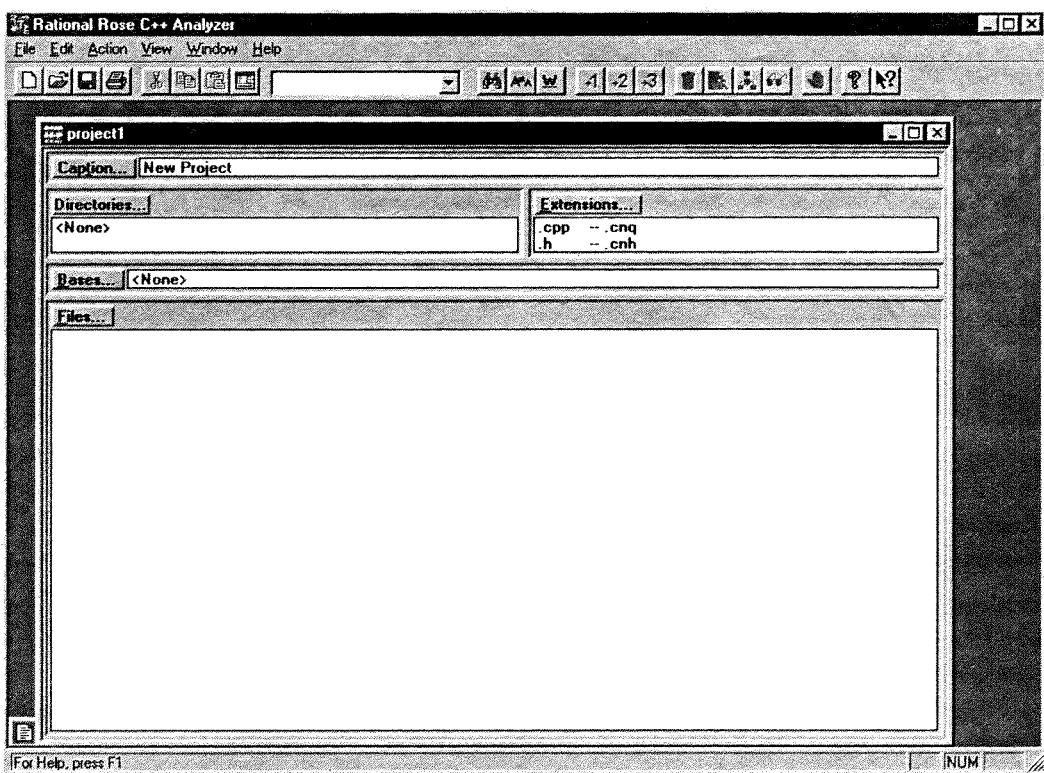


Рис. A.8. Окно проекта в анализаторе кода C++

Для создания проекта в анализаторе кода C++:

1. Выберите команду меню **Tools** ⇒ **C++** ⇒ **Reverse Engineering** (Сервис ⇒ C++ ⇒ Возвратное проектирование) для запуска анализатора кода C++.
2. Выберите команду меню **File** ⇒ **New** (Файл ⇒ Новый).

Окно проекта в анализаторе кода C++ показано на рис. А.8.

### **Этап 2. Добавление заголовка проекта**

Проект анализатора, так же как код, содержит описание. Каждый проект должен иметь заголовок – общие сведения о проекте (его название и назначение). Такая информация потребуется разработчикам для определения возможности повторного использования проекта.

Для добавления заголовка проекта в анализаторе кода C++:

1. Щелкните по кнопке **Caption** (Заголовок), чтобы открыть одноименное диалоговое окно (см. рис. А.9).
2. Введите в нем необходимую информацию.
3. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно.

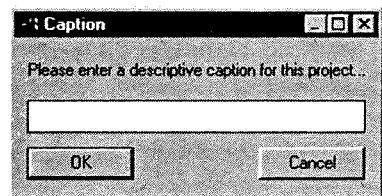


Рис. А.9. Диалоговое окно **Caption**

### **Этап 3. Добавление связанных библиотек и базовых проектов**

В список каталогов проекта включены каталоги, используемые анализатором. Каталоги, содержащие анализируемые и сопутствующие файлы, должны быть включены в список каталогов проекта.

Создание списка каталогов проекта в анализаторе кода C++ предусматривает выполнение следующих действий:

1. Щелкните по кнопке **Directories** (Каталоги), чтобы открыть диалоговое окно **Project Directory List** (Список каталогов проекта) – см. рис. А.10.
2. Выберите нужный каталог в списке **Directory Structure** (Структура каталогов).
3. Щелкните по кнопке **Add Current** (Добавить текущий), чтобы добавить в список каталогов текущий каталог.
4. Щелкните по кнопке **Add Subdirs** (Добавить подкаталоги), чтобы добавить в список каталогов текущий каталог и его непосредственные подкаталоги.
5. Щелкните по кнопке **Add Hierarchy** (Добавить иерархию), чтобы добавить в список каталогов текущий каталог и все вложенные подкаталоги.

Проект анализатора может содержать информацию из другого проекта, называемого базовым. Обычно он включает информацию о файлах заголовков для дополнительных библиотек, на основе которых построена программа.

Вместо того чтобы анализировать эту информацию в каждом проекте, где используются библиотеки, создается один базовый проект, который может использоваться в любом другом проекте. Базовый проект вносится в список базовых проектов.



Рис. А.10. Диалоговое окно **Project Directory List**

Если анализатор не находит нужного файла в каталогах из списка каталогов проекта, он ищет его в базовом проекте. Если базовых проектов несколько, они просматриваются в порядке включения в список.

Для добавления базового проекта в анализаторе кода C++:

1. Щелкните по кнопке **Base** (Базовый проект), чтобы открыть диалоговое окно **Base Project** (Базовый проект) – см. рис. А.11.

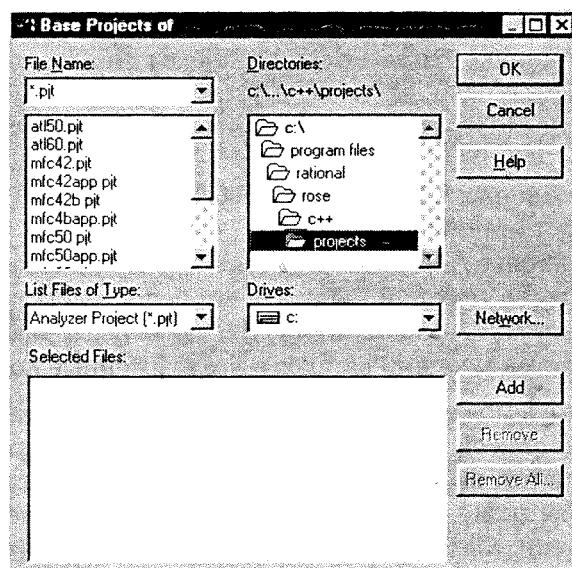


Рис. А.11. Диалоговое окно **Base Project**

2. Найдите каталог с нужным базовым проектом.
3. Щелкните по файлу проекта в списке файлов.
4. Щелкните по кнопке **Add** (Добавить), чтобы добавить проект в список базовых проектов.

#### **Этап 4. Установка типа файлов и анализ файлов**

Анализатор классифицирует файлы по трем типам – тип 1, тип 2 и тип 3. Когда файл добавляется в список файлов проекта, он относится к категории первого типа. Файлы этого типа являются семантически завершенными и независимыми от контекста. То есть файл включает список завершенных деклараций на языке C++ и либо содержит в себе всю необходимую информацию, либо получает информацию из директив `#include`. Файлы второго типа являются семантически завершенными, но зависимыми от контекста. То есть файл включает список завершенных деклараций на языке C++, но при этом содержит символы, определяемые по контексту, в который включен файл. Файлы третьего типа – семантически незавершенные и всегда обрабатываются в том случае, когда встречаются.

Изменение типа анализа в анализаторе кода C++ осуществляется следующим образом:

1. Из списка файлов выберите файл, щелкнув по нему мышью.
2. Выберите нужный тип файла в меню **Action** ⇒ **Set Type** (Действие ⇒ Установить тип).

Анализатор кода C++ может обрабатывать один файл или группу файлов. Он создает и хранит информацию об анализе в отдельном файле данных для каждого обработанного файла. Эти данные используются при очередном анализе исходного файла. После обработки статус файла в списке файлов обновляется. Файлу могут присваиваться следующие статусы:

- Неизвестный (Unknown): файл не обрабатывался анализатором;
- Устаревшие данные (Stale Data): файл содержит потенциально устаревшие данные;
- Проанализирован (Analyzed): файл успешно обработан анализатором.  
Этот статус присваивается только исходным файлам первого и второго типа;
- С циклическим кодом (CodeCycled): файл успешно обработан анализатором и содержит сведения, защищающие информацию в коде от перезаписи.  
Этот статус присваивается только исходным файлам первого и второго типа;
- Исключен (Excluded): это файл третьего типа, который анализируется каждый раз, когда встречается в другом файле;
- Содержит ошибки (Has Errors): при анализе файла обнаружены ошибки в исходном коде;
- Отсутствует (No Source): невозможно найти файл в файловой системе;
- Не проанализирован (Unanalyzed): для этого файла невозможно найти файл данных.

Чтобы проанализировать файлы в анализаторе кода C++:

1. Укажите тип для каждого анализируемого файла.
2. Выделите файлы в списке файлов.

3. Выберите команду меню **Action** ⇒ **Analyze** (Действие ⇒ Анализировать) для анализа файлов или команду **Action** ⇒ **CodeCycle** (Действие ⇒ Анализировать с циклическим кодом) для анализа с внесением сведений для Rational Rose.

Окно анализатора с информацией о состоянии анализа показано на рис. А.12.

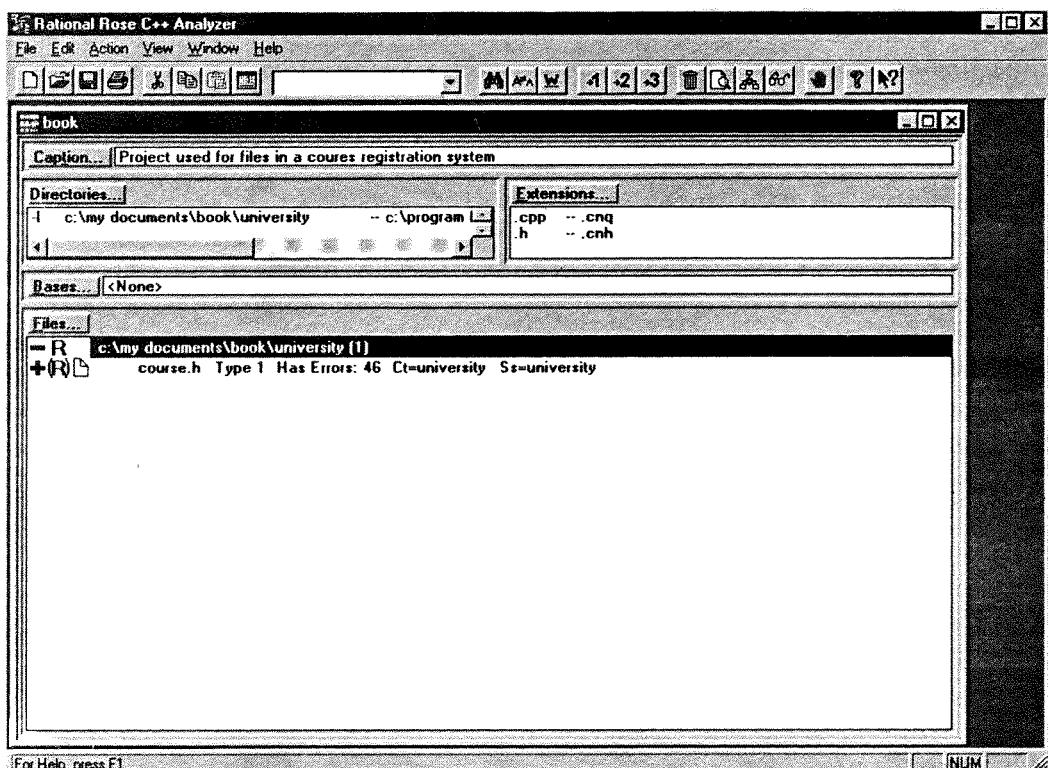


Рис. А.12. Статус анализа

### Этап 5. Оценка ошибок

Анализатор выводит все ошибки в окне **Log** (Журнал). Их также можно просмотреть, если дважды щелкнуть мышью по файлу в списке файлов. Каждую ошибку нужно оценить по степени важности. Приведу некоторые типичные ошибки:

- Неразрешенная ссылка (Unresolved reference): анализатор не смог найти исходный файл, на который указывает ссылка. Для устранения такой ошибки в список каталогов проекта необходимо добавить каталог с файлом, указанным в ссылке;
- Незнакомое расширение языка (Missing language extension): расширение языка не опознано анализатором. Для устранения этой ошибки расширение языка должно быть определено в проекте как символ;

- Контекстно-зависимый исходный файл (Context-sensitive source file): используется код из других каталогов, не включенный в данный файл. Для устранения ошибки измените файл на второй или третий тип.

Окно анализатора со списком ошибок, выявленных при анализе, показано на рис. А.13.

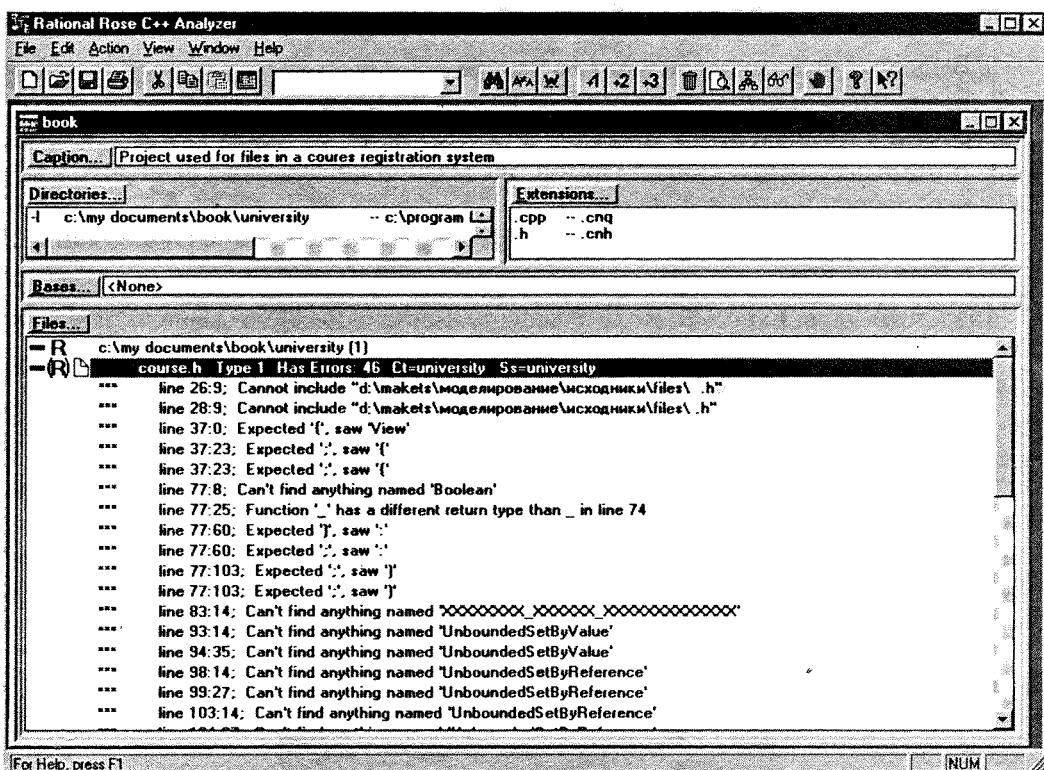


Рис. А.13. Ошибки анализа

### Этап 6. Настройка параметров экспорта и экспорт в Rational Rose

При указании параметров экспорта определяются элементы, которые должны быть смоделированы и отображены в экспортируемом файле. Например, класс может быть смоделирован и отображен, комментарии – добавлены, ассоциативные связи – смоделированы и отображены, отношения зависимости – смоделированы. Если элемент смоделирован и отображен, он будет виден в созданной или обновленной модели Rational Rose. Если элемент смоделирован, он может бытьображен средствами программы Rational Rose после создания или обновления модели.

В анализаторе сода C++ имеется несколько готовых наборов параметров экспорта:

- Двустороннее проектирование (RoundTrip): параметры экспорта, полезные для отработки проектирования в обе стороны. Создается файл с расширением .red.

- Первый взгляд (First Look): обобщенный взгляд на модель. Создается файл с расширением .mdl.
- Детальный анализ (Detailed Analysis): детальный взгляд на модель. Создается файл с расширением .mdl.

Вы можете использовать готовый набор параметров экспорта, изменить его или создать собственный.

Для экспорта параметров в анализаторе кода C++:

1. Выделите файлы для экспорта в списке файлов.
2. Выберите команду меню **Action** ⇒ **Export To Rose** (Действие ⇒ Экспортировать в Rational Rose).
3. В открывающемся списке **Option Set** (Набор параметров) выберите нужный набор параметров экспорта.
4. Щелкните по кнопке **OK** или **Overwrite** (Заменить) для экспорта данных в Rational Rose.

Диалоговое окно **Export To Rose** (Экспорт в Rational Rose) показано на рис. А.14.

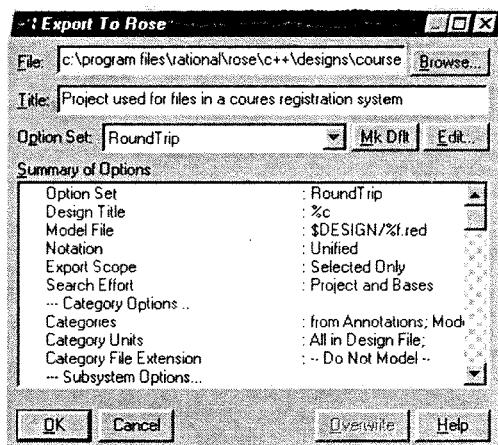


Рис. А.14. Диалоговое окно **Export To Rose**

### Этап 7. Обновление модели в Rational Rose

После создания анализатором файла .red он используется для обновления модели в Rational Rose. При этом в программе элементы модели заменяются элементами, полученными из исходного кода, а также добавляются новые элементы, не включенные ранее в модель.

Чтобы обновить модель в Rational Rose:

1. Откройте модель, которая будет обновляться.
2. Выберите команду меню **File** ⇒ **Update** (Файл ⇒ Обновление).
3. Найдите и выделите файл .red.
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Update Model From** (Обновить модель из).



## Приложение В. Генерация кода и возвратное проектирование для Visual C++ и Visual Basic

Данное приложение представляет собой подробное руководство по генерации кода для языков Visual C++ и Visual Basic и возвратному проектированию.

### Этапы генерации кода

1. Назначение языка Visual C++ или Visual Basic компонентам.
2. Связывание классов с компонентами.
3. Установка параметров генерации кода с помощью программы Model Assistant Tool.
4. Выбор компонентов и генерация кода с помощью мастера Code Update Tool.
5. Оценка ошибок при генерации кода.

### Этапы возвратного проектирования

1. Возвратное проектирование по коду Visual C++ или Visual Basic с помощью мастера Model Update Tool.
2. Оценка ошибок.

## Генерация кода

### Этап 1. Назначение языка Visual C++ или Visual Basic компонентам

Компонентам необходимо назначить язык, который устанавливается для всех связанных с компонентом классов.

Последовательность назначения языка компоненту в программе Rational Rose:

1. Щелкните правой кнопкой мыши по компоненту в списке браузера или по диаграмме.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).

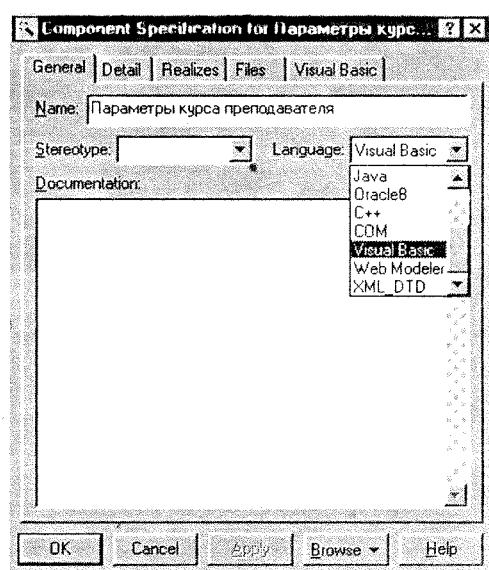


Рис. В. 1. Параметры компонента

3. В открывшемся списке **Language** (Язык) выберите нужный язык.
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Диалоговое окно настройки параметров компонента для класса параметры курса преподавателя (ProfessorCourseOptions) показано на рис. В.1.

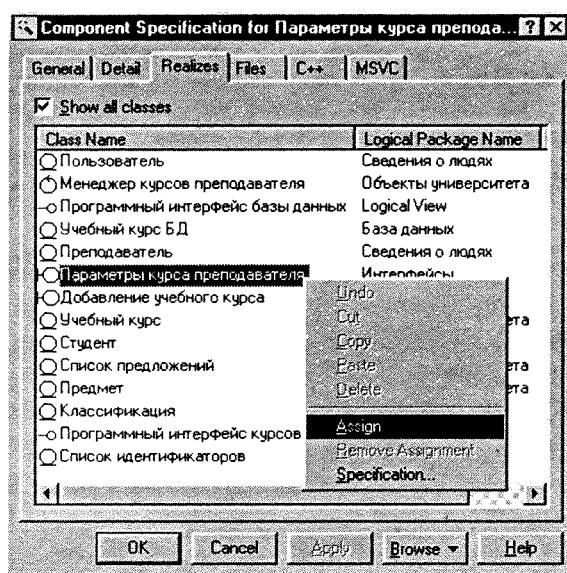
### **Этап 2. Связывание классов с компонентами**

После создания компонентов устанавливается связь классов с компонентами. Компоненты представляют проект на Visual C++ или Visual Basic.

Для связывания классов с компонентами в программе Rational Rose:

1. Щелкните правой кнопкой мыши по компоненту в списке браузера или по диаграмме компонентов.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification**.
3. Выберите вкладку **Realize** (Реализация).
4. Щелкните правой кнопкой мыши по классу, который требуется связать с компонентом, в списке классов.
5. В появившемся контекстно-зависимом меню выберите команду **Assign** (Присвоить).
6. Аналогичным образом свяжите с компонентом другие классы.

Параметры компонента для класса параметры курса преподавателя показаны на рис. В.2.



*Рис. В.2. Вкладка **Realize** диалогового окна настройки параметров компонента*

### Этап 3. Установка параметров генерации кода с помощью программы *Model Assistant Tool*

Инструмент Model Assistant Tool отображает элементы моделирования из программы Rational Rose на конструкции языков Visual C++ или Visual Basic. Для языка Visual Basic программа Model Assistant Tool способна создавать и определять константы, операторы, события, типы, перечисления, свойства, методы и параметры методов. Она позволяет создавать процедуры типа Get, Let и Set для свойств класса и ролей ассоциативной связи, а также определенный пользователем класс для наборов и списков. На языке Visual C++ программа Model Assistant Tool может создавать и определять операции для классов, такие как конструкторы, деструкторы и операции доступа для атрибутов и отношений.

В поле Preview (Предварительный просмотр) отображается код, который будет получен для выбранного элемента. Это позволяет увидеть, как повлияют на код настройки параметров генерации.

Программа Model Assistant Tool будет доступна при условии, что:

- языком, выбранным для модели по умолчанию, является Visual C++ или Visual Basic;
- класс связан с компонентом Visual C++ или Visual Basic.

Подробную информацию об инструменте Model Assistant Tool можно найти в файлах справки программы Rational Rose.

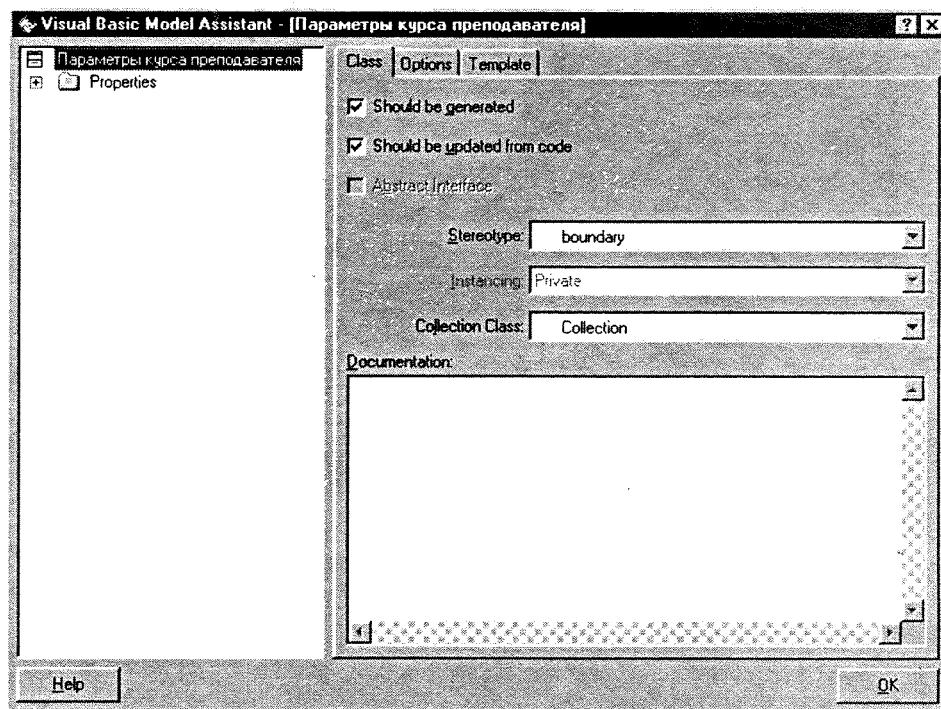


Рис. В.3. Окно программы *Model Assistant* для языка Visual Basic

Этапы запуска программы Model Assistant Tool:

1. Щелкните правой кнопкой мыши по нужному классу в списке браузера или по диаграмме классов.
2. В появившемся контекстно-зависимом меню выберите команду **Model Assistant**.

Окна программы Model Assistant Tool для класса с назначенным языком Visual Basic и Visual C++ показаны на рис. В.3 и В.4.

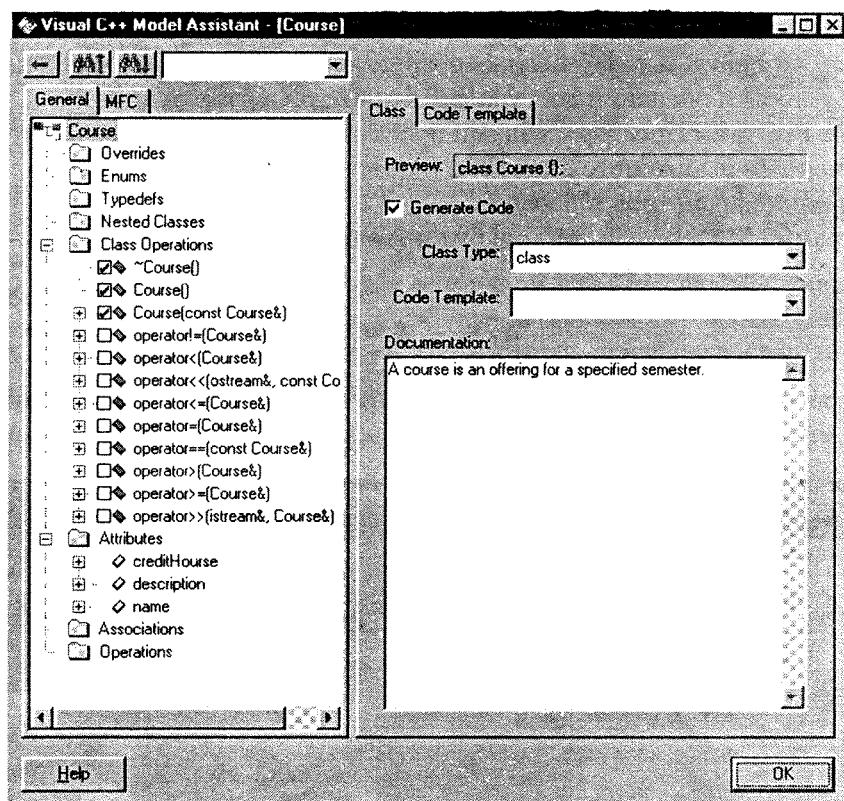


Рис. В.4. Окно программы Model Assistant для языка Visual C++

#### **Этап 4. Выбор компонентов и генерация кода с помощью мастера Code Update Tool**

Мастер Code Update Tool предназначен для генерации кода на языках Visual C++ или Visual Basic. Код может быть получен для всех компонентов пакета, одного компонента или набора компонентов.

Для запуска мастера Code Update Tool:

1. Щелкните правой кнопкой мыши по нужному компоненту в списке браузера или по диаграмме компонентов.

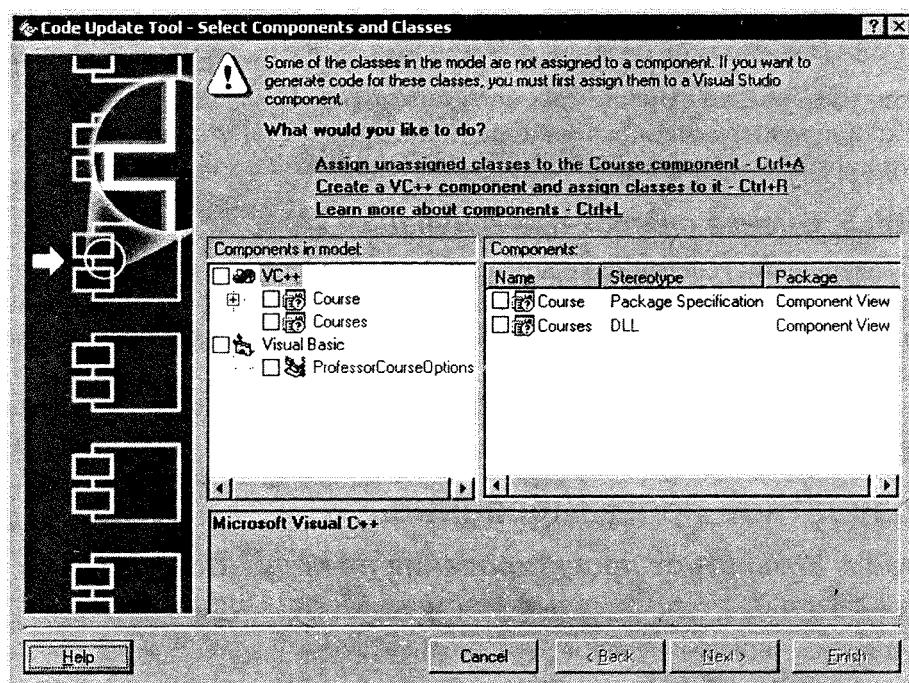


Рис. В.5. Мастер Code Update Tool

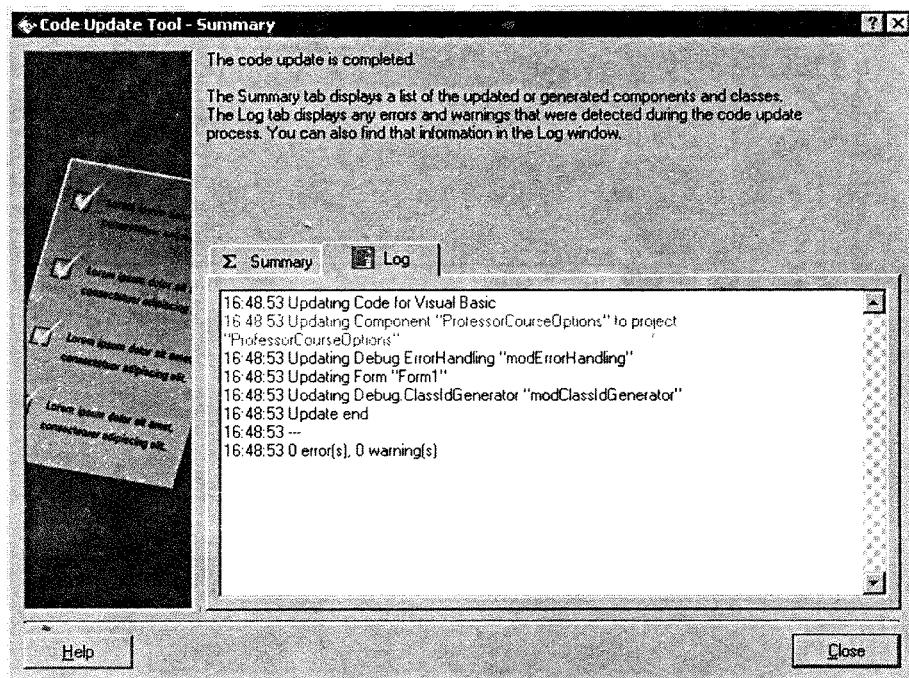


Рис. В.6. Итоговый отчет о генерации кода

2. В появившемся контекстно-зависимом меню выберите команду **Update Code** (Обновление кода).

Окно мастера Code Update Tool показано на рис. В.5.

Подробную информацию о мастере Code Update Tool можно найти в файлах справки программы Rational Rose.

### **Этап 5. Оценка ошибок при генерации кода**

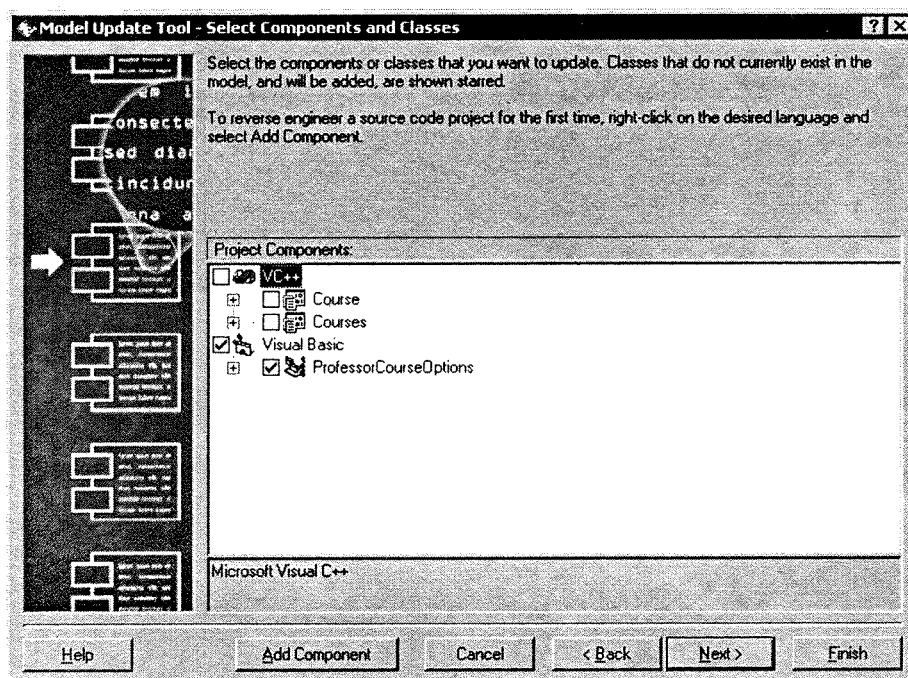
По завершении генерации кода мастер Code Update Tool выводит диалоговое окно с итоговым отчетом. Вкладка **Summary** (Сводная информация) содержит сведения о полученном коде, а все ошибки, возникшие при генерации, можно увидеть на вкладке **Log** (Журнал).

Диалоговое окно мастера Code Update Tool с итоговым отчетом показано на рис. В.6.

## **Возвратное проектирование**

### **Этап 1. Возвратное проектирование по коду Visual C++ или Visual Basic с помощью мастера Model Update Tool**

После внесения изменений и дополнений в код на Visual C++ или Visual Basic необходимо обновить модель, чтобы отразить в ней сделанные изменения. Это выполняется с помощью мастера Model Update Tool. Его можно использовать и для создания новой модели по исходному коду.



*Рис. В.7. Мастер Model Update Tool*

Для обновления или создания модели по исходному коду с помощью мастера Model Update Tool выполните следующие действия:

1. Выберите команду меню **Tools** ⇒ **Visual C++** ⇒ **Update Model from Code** (Сервис ⇒ Visual C++ ⇒ Обновить модель по коду) или команду **Tools** ⇒ **Visual Basic** ⇒ **Update Model from Code** (Сервис ⇒ Visual Basic ⇒ Обновить модель по коду).
2. Следуйте указаниям мастера.

Окно мастера Model Update Tool показано на рис. В.7.

### Этап 2. Оценка ошибок

По завершении возвратного проектирования мастер Model Update Tool отображает диалоговое окно с итоговым отчетом. Вкладка **Summary** содержит сведения об обновленной модели, а все ошибки, возникшие при обновлении, можно увидеть на вкладке **Log**.

Диалоговое окно мастера Model Update Tool с итоговым отчетом показано на рис. В.8.

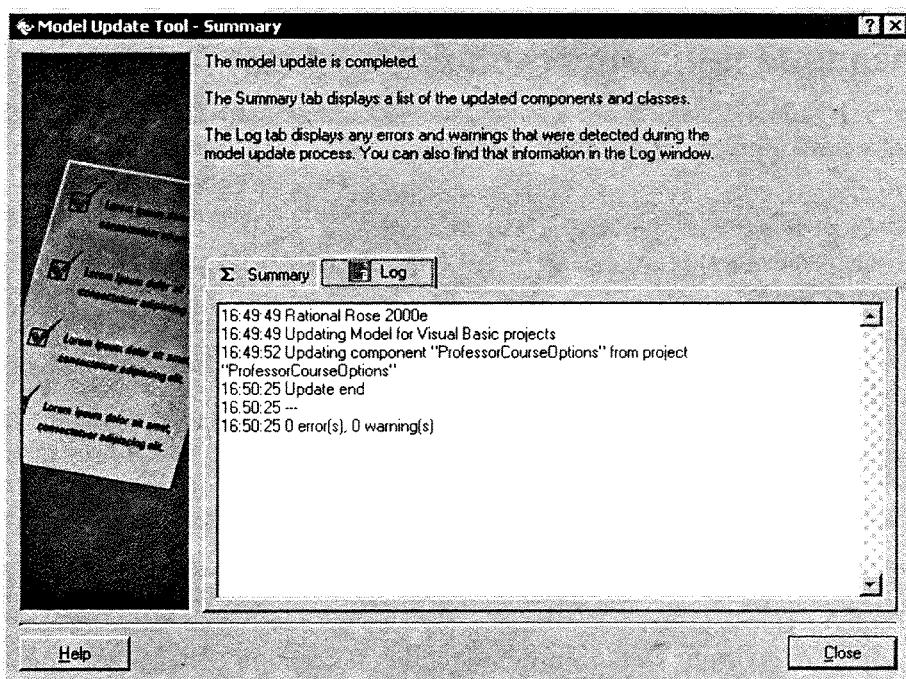


Рис. В.8. Итоговый отчет об обновлении модели



## Приложение С. Примеры программ на Visual Basic

Здесь содержится информация, необходимая для создания динамической библиотеки (DLL) на языке Visual Basic и использования ее в приложениях (подразумевается, что у вас есть версия Rational Rose Enterprise или Rational Rose Professional Visual Basic Edition). Если язык Visual Basic недоступен, сделайте следующее: выберите команду меню **Add-Ins** ⇒ **Add-In Manager** (Подключаемые модули ⇒ Менеджер подключаемых модулей), а затем модуль для Visual Basic и щелкните по кнопке **OK**. Поскольку обычно в Rational Rose операции можно выполнять различными способами, то в этом примере элементы будут создаваться по-разному.

### Создание динамической библиотеки ActiveX

Для установки Visual Basic в качестве языка, используемого по умолчанию, и отображения стереотипов в виде названий:

1. Выберите команду меню **Tools** ⇒ **Options** (Сервис ⇒ Параметры).
2. Выберите вкладку **Notation** (Нотация).
3. В открывающемся списке **Default Language** (Язык по умолчанию) укажите **Visual Basic**.
4. Выберите вкладку **Diagram** (Диаграмма).
5. Установите переключатель **Label** (Название) в группе переключателей **Stereotype display** (Отображение стереотипов).
6. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Options** (Параметры).

Последовательность установки параметров генерации кода для языка Visual Basic:

1. Выберите команду меню **Tools** ⇒ **Visual Basic** ⇒ **Options** (Сервис ⇒ Visual Basic ⇒ Параметры).
2. Сбросьте флагки **Generate debug code** (Генерация отладочной информации) и **Generate error-handling** (Генерация обработчиков ошибок).
3. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Visual Basic Properties** (Параметры Visual Basic).

Этапы создания класса Payclerk:

1. Щелкните по значку «+» слева от названия раздела **Logical View** (Логическое представление) в окне браузера.
2. Дважды щелкните по диаграмме **Main** (Главная диаграмма), чтобы открыть ее.

3. Щелкните по кнопке **Class** (Класс) на панели инструментов.
4. Щелкните по главной диаграмме, чтобы поместить на нее класс.
5. Наберите имя класса – **Payclerk**.

Чтобы ввести описание для класса Payclerk и добавить операцию:

1. Дважды щелкните по классу в списке браузера или по главной диаграмме классов – откроется диалоговое окно настройки параметров класса.
2. В поле **Documentation** (Описание) введите описание класса Payclerk – **Payclerk вычисляет недельный заработок**.
3. Выберите вкладку **Operations** (Операции).
4. Щелкните правой кнопкой мыши по списку операций.
5. В появившемся контекстно-зависимом меню выберите команду **Insert** (Добавить). В список будет добавлена новая операция.
6. Введите имя новой операции – **calcPay**.
7. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров класса.

Последовательность указания аргументов и типа возвращаемого значения для операции **calcPay**:

1. Щелкните по значку «+» слева от названия раздела **Logical View** в окне браузера.
2. Щелкните по значку «+» слева от названия класса Payclerk.
3. Дважды щелкните по операции **calcPay**, чтобы закрыть диалоговое окно настройки параметров операции.
4. В открывшемся списке **Return Class** (Возвращаемый класс) выберите **Currency**.
5. Выберите вкладку **Detail** (Детально).
6. Щелкните правой кнопкой мыши по списку аргументов.
7. В появившемся контекстно-зависимом меню выберите команду **Insert**. В список будет добавлен новый аргумент.
8. Введите имя нового аргумента – **rate**.
9. В открывшемся списке в колонке **Type** (Тип) выберите **Integer**.
10. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров операции.

Для создания интерфейсного класса **IPayroll**:

1. Щелкните правой кнопкой мыши по названию раздела **Logical View**.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Interface** (Создать ⇒ Интерфейс). В список браузера будет добавлен новый класс со стереотипом интерфейс (Interface).
3. Введите имя нового интерфейсного класса – **IPayroll**.

Добавление интерфейсного класса на главную диаграмму классов предусматривает выполнение следующих действий:

1. Дважды щелкните по диаграмме **Main** в разделе **Logical View** (Логическое представление), чтобы открыть ее.

2. В списке браузера выберите класс **IPayroll** и перетащите его с помощью мыши на главную диаграмму классов.

Последовательность добавления операции к классу **IPayroll**:

1. Щелкните правой кнопкой мыши по классу **IPayroll** на главной диаграмме классов.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Operation** (Создать ⇒ Операция). К классу будет добавлена новая операция.
3. Введите имя операции и тип возвращаемого значения – **calcPay(rate:Integer):Currency** (проверьте, что указали именно такой формат).

Для создания класса **Payroll**:

1. Щелкните правой кнопкой мыши по названию раздела **Logical View**.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class** (Создать ⇒ Класс). В список браузера будет добавлен новый класс.
3. Введите имя нового класса – **Payroll**.

Этапы добавления класса **Payroll** на главную диаграмму классов:

1. Дважды щелкните по диаграмме **Main** в разделе **Logical View**, чтобы открыть ее.
2. В списке браузера выберите класс **Payroll** и перетащите его с помощью мыши на главную диаграмму классов.

Чтобы создать отношения реализации между классами **IPayroll** и **Payroll**:

1. Щелкните по кнопке **Realize** (Реализация) на панели инструментов.
2. Щелкните по классу **Payroll** на главной диаграмме классов и проведите линию связи к классу **IPayroll**.

Для создания отношения между классами **Payroll** и **Payclerk**:

1. Щелкните по кнопке **Unidirectional Association** (Однонаправленная ассоциативная связь) на панели инструментов.
2. Щелкните по классу **Payroll** на главной диаграмме классов и проведите линию связи к классу **Payclerk**.

Чтобы указать название роли:

1. Щелкните правой кнопкой мыши по линии ассоциативной связи рядом с классом **Payclerk**.
2. В появившемся контекстно-зависимом меню выберите команду **Role Name** (Название роли).
3. Введите название роли – **myClerk**.

Этапы объявления ассоциации скрытой:

1. Щелкните правой кнопкой мыши по линии ассоциативной связи рядом с классом **Payclerk**.
2. В появившемся контекстно-зависимом меню выберите команду **Private** (Скрытый).

Для указания мощности:

1. Щелкните правой кнопкой мыши по линии ассоциативной связи рядом с классом **Payclerk**.

2. В появившемся контекстно-зависимом меню выберите команду **Multiplicity:1** (Мощность:1).

Установка параметров генерации кода для класса IPayroll с помощью программы Model Assistant Tool требует выполнения следующих действий:

1. В списке браузера щелкните правой кнопкой мыши по классу IPayroll или по диаграмме классов.
2. В появившемся контекстно-зависимом меню выберите команду **Model Assistant**.
3. Выберите класс IPayroll в списке программы Model Assistant.
4. В открывшемся списке **Instancing** (Способ создания экземпляров) выберите **MultiUse** (Многократное использование).
5. Щелкните по кнопке **OK**, чтобы закрыть окно **Model Assistant**.

Чтобы установить параметры генерации кода для класса Payroll с помощью программы Model Assistant Tool:

1. Щелкните правой кнопкой мыши по классу Payroll в списке браузера или по диаграмме классов.
2. В появившемся контекстно-зависимом меню выберите команду **Model Assistant**.
3. Выберите класс Payroll в списке программы Model Assistant.
4. В открывшемся списке **Instancing** (Способ создания экземпляров) выберите **MultiUse** (Многократное использование).
5. Щелкните по значку «+» слева от названия раздела **Implements Classes** (Реализует классы) в списке, чтобы открыть вложенный список.
6. Щелкните по значку «+» слева от названия класса IPayroll, чтобы открыть вложенный список.
7. Выберите операцию IPayroll\_calcPay.
8. Установите переключатель **Public** (Общедоступный) в группе элементов **Access** (Доступ).
9. Щелкните по значку «+» слева от названия раздела **Properties** (Свойства) в списке, чтобы открыть вложенный список.
10. Щелкните по значку «+» слева от названия роли myClerk, чтобы открыть вложенный список.
11. Выберите роль myClerk.
12. Установите флажок **New** (Новый).
13. Щелкните по кнопке **OK**, чтобы закрыть окно **Model Assistant**.

Для создания компонента PayrollCalculator:

1. Щелкните правой кнопкой мыши по названию раздела **Component View** (Представление компонентов) в браузере.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Component** (Создать ⇒ Компонент). В список браузера будет добавлен новый компонент.
3. Введите имя нового компонента – **PayrollCalculator**.



Последовательность указания стереотипа для компонента PayrollCalculator:

1. Щелкните правой кнопкой мыши по компоненту PayrollCalculator в браузере.
2. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
3. В открывшемся списке **Stereotype** (Стереотип) выберите стереотип ActiveX DLL.
4. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Этапы добавления компонента на главную диаграмму компонентов:

1. Дважды щелкните по диаграмме **Main** (Главная диаграмма) в разделе **Component View** (Представление компонентов), чтобы ее открыть.
2. В списке браузера выберите компонент PayrollCalculator и перетащите его с помощью мыши на главную диаграмму компонентов.

Для связывания классов с компонентом PayrollCalculator:

1. Дважды щелкните по компоненту PayrollCalculator в списке браузера или по диаграмме компонентов, чтобы открыть диалоговое окно настройки параметров компонента.
2. Выберите вкладку **Realize** (Реализация).
3. Выберите класс IPayroll, щелкнув по нему мышью.
4. Нажмите и удерживайте клавишу **Shift**.
5. Щелкните мышью по классу Payroll.
6. Щелкните мышью по классу Payclerk.
7. Щелкните правой кнопкой мыши по списку классов.
8. В появившемся контекстно-зависимом меню выберите команду **Assign** (Присвоить).
9. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.

Последовательность генерации кода на языке Visual Basic:

1. Выберите компонент PayrollCalculator на диаграмме компонентов, щелкнув по нему мышью.
2. Выберите команду меню **Tools** ⇒ **Visual Basic** ⇒ **Update code** (Сервис ⇒ Visual Basic ⇒ Обновить код) для запуска мастера Code Update Tool.
3. Щелкните по кнопке **Finish** (Готово) в окне мастера Code Update Tool.
4. На экране появится диалоговое окно сохранения модели. Введите PayrollCalculator в поле ввода **Filename** (Имя файла) и щелкните по кнопке **Save** (Сохранить).
5. Будет создан проект для Visual Basic с формой, названной по умолчанию **Form1**. Мастер Code Update Tool сообщит, что форма не входит в вашу модель, и запросит подтверждение на удаление ее из проекта. Установите флагок рядом с формой **Form1** в диалоговом окне **Synchronize** (Синхронизация) и щелкните по кнопке **OK** для продолжения генерации кода.

6. По завершении генерации кода появится диалоговое окно с итоговым отчетом. Щелкните по кнопке **Close** (Закрыть), чтобы закрыть окно мастера Code Update Tool.

Чтобы добавить код для метода класса Payclerk:

1. В программе Visual Basic перейдите к методу calcPay класса Payclerk.
2. Замените строку '## Your code goes here ...' на CalcPay = rate \* 40.
3. Перейдите к методу IPayroll\_calcPay класса IPayroll.
4. Замените строку '## Your code goes here ...' на IPayroll\_calcPay = myClerk(1).calcPay(rate).
5. Выберите команду меню **Project ⇒ PayrollCalculator Properties** (Проект ⇒ Свойства PayrollCalculator).
6. В открывшемся списке **Startup Object** (Начальный объект) выберите **(None)**.
7. Щелкните по кнопке **OK**, чтобы закрыть окно настройки параметров проекта.
8. Выберите команду меню **File ⇒ Make PayrollCalculator.dll** (Файл ⇒ Собрать PayrollCalculator.dll).
9. Щелкните по кнопке **OK** в диалоговом окне **Make Project** (Сборка проекта), чтобы сохранить созданную динамическую библиотеку.
10. Выберите команду меню **File ⇒ Exit** (Файл ⇒ Выход) для закрытия программы Visual Basic.

## Повторное использование библиотеки ActiveX

В этом разделе полученная библиотека будет использоваться для другого приложения с учетом всех этапов анализа и проектирования, рекомендованных в книге. Задача состоит в следующем. Вам нужно создать приложение, которое будет выводить зарплату служащего. Пользователь приложения – управляющий. Вы должны работать с программной библиотекой, предоставленной отделом оплаты труда (то есть созданной ранее DLL). Начнем с новой пустой модели.

Последовательность создания диаграммы прецедентов:

1. Щелкните по значку «+» слева от названия раздела **Use Case View** (Представление прецедентов) в окне браузера.
2. Дважды щелкните по диаграмме **Main** (Главная диаграмма) в разделе **Use Case View** (Представление прецедентов), чтобы открыть ее.
3. Щелкните по кнопке **Actor** (Актер) на панели инструментов, а затем по диаграмме, чтобы поместить на нее актера.
4. Введите имя нового актера – **Manager**.
5. Щелкните правой кнопкой мыши по изображению актера на диаграмме.
6. В появившемся контекстно-зависимом меню выберите команду **Options ⇒ Stereotype display ⇒ Icon** (Параметры ⇒ Отображение стереотипа ⇒ Значок).
7. Щелкните по кнопке **Use Case** (Прецедент) на панели инструментов, а затем по диаграмме, чтобы поместить на нее прецедент.

8. Введите имя нового прецедента – **Display Pay**.
9. Щелкните по кнопке **Unidirectional Association** (Однонаправленная ассоциативная связь) на панели инструментов.
10. Щелкните по актеру **Manager** на диаграмме и проведите линию связи к прецеденту **Display Pay**.

Для создания диаграммы реализации прецедентов:

1. Щелкните правой кнопкой мыши по названию раздела **Logical View** (Логическое представление) в браузере.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Use Case Diagram** (Создать ⇒ Диаграмма прецедентов). В список браузера будет добавлена новая диаграмма.
3. Введите ее название – **Realizations**.
4. Дважды щелкните по диаграмме **Realizations**, чтобы открыть ее.

Чтобы создать реализацию прецедента:

1. Щелкните по кнопке **Use Case** на панели инструментов, а затем по диаграмме, чтобы поместить на нее прецедент.
2. Дважды щелкните по прецеденту, чтобы открыть диалоговое окно настройки его параметров.
3. В поле ввода **Name** (Название) введите имя прецедента – **Display Pay**. Важно, чтобы вы указали название прецедента именно таким образом. В этом случае вы сообщите программе Rational Rose о необходимости использования другого пространства имен (namespace). Если ввести название прецедента непосредственно на диаграмме, программа сочтет, что это тот же прецедент **Display Pay**, что и в представлении прецедентов.
4. В открывшемся списке **Stereotype** (Стереотип) выберите стереотип **use-case-realization** (реализация прецедента).
5. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров.
6. На экране появится сообщение о том, что прецедент существует в нескольких пространствах имен. Это нормально. Щелкните по кнопке **OK**, чтобы закрыть сообщение.
7. Щелкните правой кнопкой мыши по прецеденту.
8. В появившемся контекстно-зависимом меню выберите команду **Stereotype ⇒ Icon** (Отображение стереотипа ⇒ Значок).

Этапы создания диаграммы последовательности действий:

1. Щелкните правой кнопкой мыши по реализации прецедента **Display Pay** в списке браузера.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Sequence Diagram** (Создать ⇒ Диаграмма последовательности действий).
3. Введите название новой диаграммы – **Display Pay for an Employee**.
4. Дважды щелкните по созданной диаграмме последовательности действий, чтобы открыть ее.

Для добавления объектов и сообщений на диаграмму последовательности действий:

1. В списке браузера выберите актера **Manager** и перетащите его с помощью мыши на диаграмму последовательности действий.
2. Щелкните по кнопке **Object** (Объект) на панели инструментов, а затем по диаграмме последовательности действий, чтобы поместить на нее объект.
3. Введите имя объекта – **aPayForm**.
4. Щелкните по кнопке **Object Message** (Сообщение) на панели инструментов.
5. Щелкните по пунктирной линии для актера и проведите линию сообщения к пунктирной линии для объекта **aPayForm**.
6. Ведите название сообщения – **display pay for joe**.
7. Щелкните по кнопке **Object** на панели инструментов, а затем по диаграмме последовательности действий, чтобы поместить на нее объект.
8. Введите имя объекта – **Joe**.
9. Щелкните по кнопке **Object Message** на панели инструментов.
10. Щелкните по пунктирной линии для объекта **aPayForm** и проведите линию сообщения к пунктирной линии для объекта **Joe**.
11. Ведите название сообщения – **get pay rate**.
12. Щелкните по кнопке **Object** на панели инструментов, а затем по диаграмме последовательности действий, чтобы поместить на нее объект.
13. Введите имя объекта – **aPayClerk**.
14. Щелкните по пунктирной линии для объекта **aPayForm** и проведите линию сообщения к пунктирной линии для объекта **aPayClerk**.
15. Ведите название сообщения – **calculate pay for this rate**.
16. Щелкните по кнопке **Message to Self** (Сообщение себе) на панели инструментов.
17. Щелкните по пунктирной линии для объекта **aPayForm**, чтобы добавить сообщение.
18. Ведите название сообщения – **display pay**.

Этапы создания классов **PayrollForm** и **Employee**:

1. Щелкните правой кнопкой мыши по названию раздела **Logical View** (Логическое представление) в браузере.
2. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class** (Создать ⇒ Класс).
3. Введите имя для нового класса – **PayrollForm**.
4. Щелкните правой кнопкой мыши по классу **PayrollForm** в браузере.
5. В появившемся контекстно-зависимом меню выберите команду **Open Specification** (Открыть параметры).
6. В открывшемся списке **Stereotype** выберите стереотип **Form** (Форма). Если вы не указали Visual Basic в качестве языка, используемого по умолчанию, данный стереотип будет недоступен. В таком случае можно либо пересоздать класс, установив перед этим язык, либо выбрать стереотип после привязки класса к компоненту с языком Visual Basic.

7. Щелкните по кнопке **OK**, чтобы закрыть окно настройки параметров класса.
8. Выберите класс **PayrollForm** в списке браузера и введите описание класса в окне **Documentation** (Описание) – **this form displays the weekly pay of an employee.**
9. Щелкните правой кнопкой мыши по названию раздела **Logical View** (Логическое представление) в браузере.
10. В появившемся контекстно-зависимом меню выберите команду **New ⇒ Class** (Создать ⇒ Класс).
11. Введите имя для нового класса – **Employee**.
12. Введите описание класса в окне **Documentation** (Описание) – **an Employee is someone who is paid by the Company.**

Для добавления динамической библиотеки к модели:

1. Щелкните по значку «+» слева от названия раздела **Component View** (Представление компонентов) в окне браузера.
2. Дважды щелкните по диаграмме компонентов **Main** (Главная диаграмма), чтобы открыть ее.
3. Откройте программу Проводник (Explorer) и найдите файл библиотеки **PayrollCalculator.dll**, созданный ранее.
4. Перетащите его на открытую диаграмму компонентов.
5. Выберите команду меню **Full Import** (Полный импорт).

Чтобы связать объекты с классами на диаграмме последовательности действий:

1. Если диаграмма последовательности действий **Display Pay for an Employee** закрыта, щелкните по ней правой кнопкой мыши в списке браузера, а в появившемся контекстно-зависимом меню выберите команду **Open** (Открыть).
2. Щелкните по значку «+» слева от названия раздела **Logical View** в окне браузера, чтобы открыть вложенный список.
3. Выберите класс **PayrollForm** и перетащите его на объект **aPayForm**.
4. Выберите класс **Employee** и перетащите его на объект **Joe**.
5. В окне браузера щелкните по значку «+» слева от названия раздела **COM** в разделе **Logical View**, чтобы открыть вложенный список.
6. В окне браузера щелкните по значку «+» слева от названия раздела **Payroll Calculator** в разделе **Logical View**, чтобы открыть вложенный список.
7. Выберите класс **Payroll** и перетащите его на объект **aPayClerk**.

Этапы связывания сообщений с операциями:

1. Сообщения, адресованные классу формы, не являются операциями. Они превратятся в поля формы, которые мы установим в программе Visual Basic. Щелкните правой кнопкой мыши по сообщению **get pay rate** и в появившемся контекстно-зависимом меню выберите команду **<new operation>** (Создать операцию). На экране увидите диалоговое окно настройки параметров операции.
2. В поле **Name** (Название) введите имя операции – **getRate**.

3. В открывшемся списке **Return Class** (Возвращаемый класс) выберите **Currency**. Как и раньше, если вы не указали Visual Basic в качестве языка, используемого по умолчанию, тип **Currency** будет недоступен.
4. Щелкните по кнопке **OK**, чтобы закрыть окно настройки параметров операции.
5. Щелкните правой кнопкой мыши по сообщению `calculate pay for this rate`.
6. В появившемся контекстно-зависимом меню выберите команду **IPayroll\_calcPay**.

Для создания диаграммы классов:

1. Щелкните по значку «+» слева от названия раздела **Logical View** в окне браузера, чтобы открыть вложенный список.
2. Дважды щелкните по диаграмме **Main** (Главная диаграмма), чтобы открыть ее.
3. Выберите команду меню **Query ⇒ Add Classes** (Запрос ⇒ Добавить классы). Разделом для добавления классов по умолчанию является **Logical View**.
4. Щелкните по кнопке **All >>** (Все), чтобы добавить на диаграмму классы **PayrollForm** и **Employee**.
5. В открывшемся списке **Package** (Пакет) выберите **PayrollCalculator**.
6. Щелкните по кнопке **All >>**, чтобы добавить на диаграмму класс **IPayroll**.
7. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно **Add Classes** (Добавить классы).
8. Щелкните по кнопке **Unidirectional Association** (Односторонняя ассоциативная связь) на панели инструментов, а затем по классу **PayrollForm** и проведите линию связи к классу **Employee**.
9. Щелкните правой кнопкой мыши по линии связи рядом с классом **Employee**.
10. В появившемся контекстно-зависимом меню выберите команду **Role Name** (Название роли) и введите название роли – **anEmployee**.
11. Щелкните правой кнопкой мыши по линии связи рядом с классом **Employee**.
12. В появившемся контекстно-зависимом меню выберите команду **Multiplicity:1** (Мощность:1).
13. Щелкните по кнопке **Unidirectional Association** на панели инструментов, затем по классу **PayrollForm** и проведите линию связи к классу **Payroll**. Укажите стереотип **<<coclass>>**.
14. Щелкните правой кнопкой мыши по линии связи рядом с классом **Payroll**.
15. В появившемся контекстно-зависимом меню выберите команду **Role Name** (Название роли) и введите название роли – **myPayClerk**.
16. Щелкните правой кнопкой мыши по линии связи рядом с классом **Payroll**.
17. В появившемся контекстно-зависимом меню выберите команду **Multiplicity:1**.

Последовательность установки параметров генерации кода с помощью программы Model Assistant Tool:

1. Щелкните правой кнопкой мыши по классу **PayrollForm**.
2. В появившемся контекстно-зависимом меню выберите команду **Model Assistant**.

3. Щелкните по значку «+» слева от свойства `anEmployee` в списке, чтобы открыть вложенный список.
4. Выберите элемент данных `anEmployee`.
5. Установите флагок **New** (Новый).
6. Щелкните по значку «+» слева от свойства `myPayClerk` в списке, чтобы открыть вложенный список.
7. Выберите элемент данных `myPayClerk`.
8. Установите флагок **New** (Новый).
9. Щелкните по кнопке **OK**, чтобы закрыть окно **Model Assistant**.

Этапы создания диаграммы компонентов:

1. Щелкните по значку «+» слева от названия раздела **Component View** (Представление компонентов) в окне браузера.
2. Дважды щелкните по диаграмме **Main** (Главная диаграмма), чтобы открыть ее.
3. Щелкните по кнопке **Package** (Пакет) на панели инструментов, а затем по диаграмме, чтобы поместить на нее пакет.
4. Введите название пакета – **Manager Options**.
5. В окне браузера щелкните по значку «+» слева от названия раздела **COM** в разделе **Component View** (Представление компонентов), чтобы открыть вложенный список.
6. В списке браузера выберите пакет `PayrollCalculator` в представлении компонентов и перетащите его на главную диаграмму компонентов.
7. Щелкните по кнопке **Dependency** (Отношение зависимости) на панели инструментов, затем по пакету `Manager Options` и перетащите линию связи к пакету `PayrollCalculator`.
8. Дважды щелкните по пакету `Manager Options` на диаграмме, чтобы открыть главную диаграмму компонентов пакета.
9. Щелкните по кнопке **Component** (Компонент) на панели инструментов, а затем на диаграмме, чтобы поместить на нее компонент.
10. Дважды щелкните по новому компоненту, чтобы открыть диалоговое окно настройки параметров компонента.
11. Введите имя компонента в поле ввода **Name** (Название) – `DisplayPay`.
12. Если вы не указали Visual Basic в качестве языка, используемого по умолчанию, выберите в открывшемся списке **Language** (Язык) язык Visual Basic.
13. В открывшемся списке **Stereotype** (Стереотип) выберите стереотип EXE.
14. Щелкните по кнопке **OK**, чтобы закрыть диалоговое окно настройки параметров компонента.
15. В списке браузера выберите класс `PayrollForm` в логическом представлении и перетащите его на компонент `DisplayPay`.
16. В списке браузера выберите класс `Employee` в логическом представлении и перетащите его на компонент `DisplayPay`.
17. В окне браузера щелкните по значку «+» слева от пакета `PayrollCalculator` в разделе **Component View** (Представление компонентов), чтобы открыть вложенный список.

18. В списке браузера выберите компонент PayrollCalculator и перетащите его на диаграмму компонентов.
19. Щелкните по кнопке **Dependency** (Отношение зависимости) на панели инструментов, а затем по компоненту DisplayPay и перетащите линию связи к компоненту PayrollCalculator.

Этапы генерации кода:

1. Выберите компонент DisplayPay, щелкнув по нему мышью.
2. Выберите команду меню **Tools** ⇒ **Visual Basic** ⇒ **Update code** (Сервис ⇒ Visual Basic ⇒ Обновить код) для запуска мастера Code Update Tool.
3. Щелкните по кнопке **Finish** (Готово) в окне мастера Code Update Tool.
4. Установите флажок рядом с формой **Form1** в диалоговом окне **Synchronize** (Синхронизация) и щелкните по кнопке **OK**, чтобы продолжить процесс генерации кода.
5. По завершении генерации кода появится диалоговое окно с итоговым отчетом. Щелкните по кнопке **Close** (Закрыть), чтобы закрыть окно мастера Code Update Tool.

Последовательность реализации методов на Visual Basic:

1. В программе Visual Basic найдите класс Employee.
2. Введите код реализации для метода `getRate`: `getRate = 10.`
3. Выберите класс PayrollForm.
4. Поместите на форму поле ввода.
5. Введите код реализации для метода `Form_Load`:

```
Dim theRate As Integer  
TheRate = anEmployee.getRate  
Text1.Text = myPayClerk.IPayroll_calcPay(theRate)
```

6. Выберите команду меню **Project** ⇒ **Display Pay Properties** (Проект ⇒ Свойства Display Pay).
7. В открывшемся списке **Startup Object** (Начальный объект) выберите класс PayrollForm.
8. Щелкните по кнопке **OK**, чтобы закрыть окно настройки параметров проекта.
9. Запустите исполняемый файл, и вы увидите в поле ввода число **400**.



## Глоссарий

**Автоматический переход** (Automatic transition) – переход между состояниями, который осуществляется по завершении деятельности внутри исходного состояния.

**Агрегация** (Aggregation) – более сильная форма ассоциации, при которой связь устанавливается между целым и его частью или частями.

**Актер** (Actor) – кто-то (или что-то) внешний по отношению к системе, кто должен взаимодействовать с разрабатываемой системой.

**Архитектура** (Architecture) – логическая и физическая структура системы, созданная на основе всех стратегических и тактических решений, принятых в ходе разработки.

**Ассоциативный класс** (Association class) – класс, содержащий информацию, которая относится к связи между двумя объектами, но ни к одному из объектов в отдельности.

**Ассоциация** (Association) – двунаправленная семантическая связь между двумя классами.

**Атрибут** (Attribute) – поле данных, содержащееся в объектах класса. Атрибуты составляют структуру класса.

**Базовый проект** (Base project) – проект с дополнительной информацией по отношению к основному программному проекту. Он обычно содержит сведения о файлах заголовков для вспомогательных библиотек классов.

**Библиотека классов** (Class library) – программная библиотека, содержащая классы, которые могут быть использованы другими разработчиками.

**Бизнес-цели** (Business goals) – список потребностей организации в приоритном порядке, помогающий выработать правильные решения и найти разумные компромиссы в процессе разработки.

**Визуальное моделирование** (Visual Modeling) – способ представления идей и проблем реального мира с помощью моделей.

**Действие** (Action) – поведение, которое сопровождает событие перехода между состояниями. Считается, что действие занимает нулевое время и не может быть прервано.

**Деятельность** (Activity) – поведение, возникающее внутри состояния. Деятельность может быть прервана событием перехода между состояниями.

**Диаграмма взаимодействий** (Collaboration diagram) – диаграмма, отражающая взаимодействие объектов, организованное вокруг самих объектов и связей между ними.

**Диаграмма внедрения** (Deployment diagram) – диаграмма, показывающая распределение процессов по узлам вычислительных систем в физической организации системы.

**Диаграмма классов** (Class diagram) – графическое представление некоторых или всех классов модели.

**Диаграмма компонентов** (Component diagram) – диаграмма, показывающая организацию программных компонентов и их зависимости, включая компоненты исходного кода, программные (run-time) и исполняемые (executable) компоненты.

**Диаграмма последовательности действий** (Sequence diagram) – диаграмма, отражающая взаимодействие объектов, упорядоченное во времени.

**Диаграмма прецедентов** (Use case diagram) – графическое представление актеров, прецедентов и взаимодействий между ними.

**Диаграмма состояний** (Statechart diagram) – диаграмма, отражающая набор состояний данного класса, события, вызывающие переходы между состояниями, и действия, выполняемые при смене состояний.

**Дизайн** (Design) – представление реализации системы.

**Задумка** (Inception) – определение концепции проекта.

**Итеративный и инкрементальный жизненный цикл** (Iterative and incremental life cycle) – создание серии архитектурных выпусков, развивающихся в законченную систему.

**Итерационный план** (Iteration plan) – расписание итеративных выпусков (releases), запланированных для системы.

**Класс** (Class) – описание группы объектов с общими свойствами (атрибутами), однотипным поведением (операциями), общими отношениями с другими объектами (ассоциативными или агрегационными) и общей семантикой.

**Ключевой механизм** (Key mechanism) – проектные решения, касающиеся отдельных частей архитектуры.

**Линия синхронизации** (Synchronization bar) – горизонтальная или вертикальная линия на диаграмме, указывающая, что определенные действия должны выполняться одновременно. Линии синхронизации также используются, чтобы показать объединение потоков событий.

**Модель** (Model) – абстракция, отражающая основу сложной проблемы или структуры и упрощающая работу с ней.

**Модель прецедентов** (Use case model) – совокупность актеров, прецедентов и диаграмм прецедентов системы.

**Наследование** (Inheritance) – отношение между классами, когда один класс использует часть структуры и/или поведения другого класса или нескольких классов.

**Обобщение** (Generalization) – процесс создания суперклассов, объединяющих общие для нескольких классов структуру и поведение.

**Объект** (Object) – концепция, абстракция или вещь с четко определенными границами и значением для системы.

**Операция** (Operation) – действия одного объекта над другим, направленные на вызов конкретной реакции. Операции определяют поведение класса.

**Определение требований к системе** (Requirement analysis) – описание задач системы.

**Охранное условие** (Guard) – условие, которое должно принять значение TRUE для осуществления определенного перехода.

**Переход между состояниями** (State transition) – переход от одного состояния к другому.

**Переходный период** (Transition) – поставка продукта пользователям (производство, распространение, обучение).

**Подкласс** (Subclass) – класс-потомок, унаследованный от одного или нескольких классов.

**Полиморфизм** (Polymorphism) – механизм, обеспечивающий возможность воздействовать на объекты, оперируя понятиями их суперклассов.

**Прецедент** (Use case) – представление бизнес-процессов системы. Модель диалога между актером и системой.

**Проверка целостности** (Consistency checking) – процесс сверки информации в статическом (диаграммы классов) и динамическом (диаграммы взаимодействий) представлении системы.

**Проработка** (Elaboration) – планирование необходимых действий и требуемых ресурсов; определение особенностей и проектирование архитектуры.

**Прототип проверки концепции** (Proof of concept prototype) – прототип, используемый для проверки начальных предположений и допущений, сформулированных для проблемной области.

**Раздел** (Partition) – пакеты, составляющие часть уровня абстракции.

**Реализация** (Implementation) – создание кода для получения действующей системы.

**Секция** (Swimlane) – участок диаграммы действий для выделения ответственных за действия. Секции часто соответствуют организационным единицам в бизнес-модели.

**Слой** (Layer) – набор пакетов на одном уровне абстракции.

**Создание** (Construction) – построение продукта с помощью серии инкрементальных итераций.

**Состояние** (State) – обобщенный результат поведения объекта; одно из условий, в которых может находиться объект.

**Специализация** (Specialization) – процесс создания подклассов, которые являются уточнением суперклассов и в которых добавляются, изменяются или скрываются структура и поведение.

**Сtereотип** (Stereotype) – новый тип элемента моделирования, расширяющий метамодель. Стереотипы должны основываться на элементах, являющихся частью метамодели языка UML.

**Суперкласс** (Superclass) – класс-предок, от которого унаследованы другие классы.

**Сценарий** (Scenario) – экземпляр precedента – единичный проход по потоку событий для precedента.

**Тестирование** (Test) – проверка всей системы.

**Точка принятия решения** (Decision point) – точка на диаграмме действий, в которой проверяется условие для выбора одного из возможных переходов.

**Унифицированный язык моделирования** (Unified Modeling Language – UML) – язык, используемый для определения, отображения и описания элементов объектно-ориентированной системы на стадии разработки.

**Управляемый модуль** (Controlled unit) – пакет, включенный в систему управления конфигурацией, который может быть загружен или сохранен независимо.

**Фоновая постановка задачи** (Background problem statement) – обобщенный материал, собранный перед началом работы над проектом. Обычно содержит описание предыдущей системы.