# Глава 5

# Классические методы анализа

В этой главе рассматриваются классические методы анализа требований, ориентированные на процедурную реализацию программных систем. Анализ требований служит мостом между неформальным описанием требований, выполняемым заказчиком, и проектированием системы. Методы анализа призваны формализовать обязанности системы, фактически их применение дает ответ на вопрос «Что должна делать будущая система?»

## Структурный анализ

Структурный анализ — один из формализованных методов анализа требований к ПО. Автор этого метода — Том Де Марко (1979) [47]. В этом методе программное изделие рассматривается как преобразователь информационного потока данных. Основной элемент структурного анализа — диаграмма потоков данных.

### Диаграммы потоков данных

Диаграмма потоков данных ПДД — графическое средство для изображения информационного потока и преобразований, которым подвергаются данные при движении от входа к выходу системы. Элементы диаграммы имеют вид, показанный на рис. 5.1.

Диаграмма может использоваться для представления программного изделия на любом уровне абстракции.

Пример системы взаимосвязанных диаграмм показан на рис. 5.2.

Диаграмма высшего (нулевого) уровня представляет систему как единый овал со стрелкой, ее называют основной или контекстной моделью. Контекстная модель используется для указания внешних связей программного изделия.

Для детализации (уточнения системы) вводится диаграмма 1-го уровня. Каждый из преобразователей этой диаграммы — подфункция общей системы. Таким образом, речь идет о замене преобразователя $F$ на целую систему преобразователей.

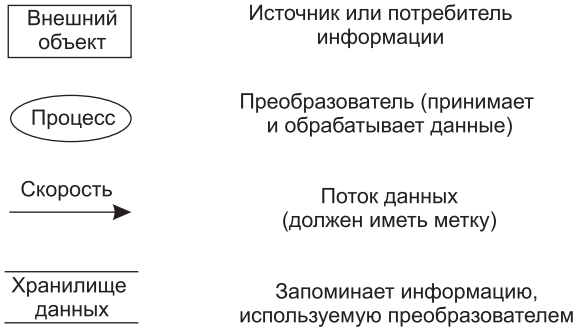


**Рис. 5.1.** Элементы диаграммы потоков данных

Дальнейшее уточнение (например, преобразователя *F3*) приводит к диаграмме 2-го уровня. Говорят, что ПДД1 разбивается на диаграммы 2-го уровня.

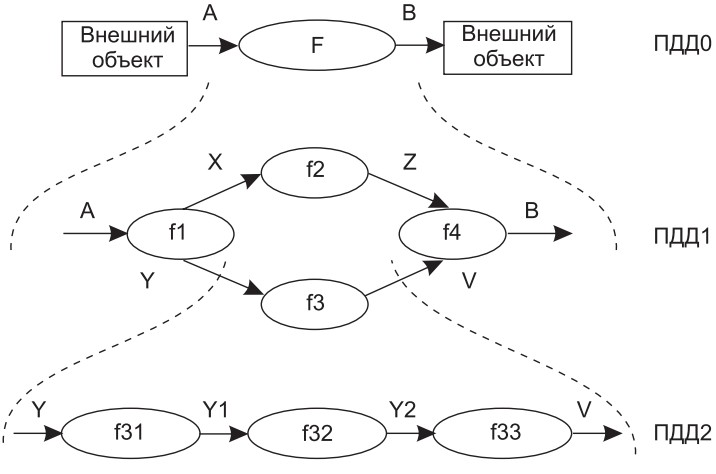


**Рис. 5.2.** Система взаимосвязанных диаграмм потоков данных

**ПРИМЕЧАНИЕ**

Важно сохранить непрерывность информационного потока и его согласованность. Это значит, что входы и выходы у каждого преобразователя на любом уровне должны оставаться прежними. В диаграмме отсутствуют точные указания на последовательность обработки. Точные указания откладываются до этапа проектирования.

Диаграмма потоков данных — это абстракция, граф. Для связи графа с проблемной областью (превращения в граф-модель) надо задать интерпретацию ее компонентов — дуг и вершин.

## Описание потоков данных и процессов

Базовые средства диаграммы не обеспечивают полного описания требований к программному изделию. Очевидно, что должны быть описаны стрелки — потоки данных,

и преобразователи — процессы. Для этих целей используются словарь требований (данных) и спецификации процессов.

*Словарь требований* (данных) содержит описания потоков данных и хранилищ данных. Словарь требований является неотъемлемым элементом любой CASE-утилиты автоматизации анализа. Структура словаря зависит от особенностей конкретной CASE-утилиты. Тем не менее можно выделить базисную информацию типового словаря требований.

Большинство *словарей* содержит следующую информацию.

1. *Имя* (основное имя элемента данных, хранилища или внешнего объекта).
2. *Прозвище* (Alias) — другие имена того же объекта.
3. *Где и как используется объект* — список процессов, которые используют данный элемент, с указанием способа использования (ввод в процесс, вывод из процесса, как внешний объект или как память).
4. *Описание содержания* — запись для представления содержания.
5. *Дополнительная информация* — дополнительные сведения о типах данных, допустимых значениях, ограничениях и т. д.

*Спецификация процесса* — это описание преобразователя. Спецификация поясняет: ввод данных в преобразователь, алгоритм обработки, характеристики производительности преобразователя, формируемые результаты.

Количество спецификаций равно количеству преобразователей диаграммы.

**ПРИМЕЧАНИЕ**

Исходными данными для создания диаграмм потоков данных и словарей требований являются словесные требования заказчика, которые обсуждались в предыдущей главе. Это положение распространяется на все методы анализа, рассматриваемые в данной главе.

## Расширения для систем реального времени

Как известно, программное изделие (ПИ) является дискретной моделью проблемной области, взаимодействующей с непрерывными процессами физического мира (рис. 5.3).


**Рис. 5.3.** Программное изделие как дискретная модель проблемной области

П. Вард и С. Меллор приспособили диаграммы потоков данных к следующим требованиям систем реального времени [99]:

1) информационный поток накапливается или формируется в непрерывном времени;
2) фиксируется управляющая информация. Считается, что она проходит через систему и связывается с управляющей обработкой;
3) допускается множественный запрос на одну и ту же обработку (из внешней среды).

Новые элементы имеют обозначения, показанные на рис. 5.4.

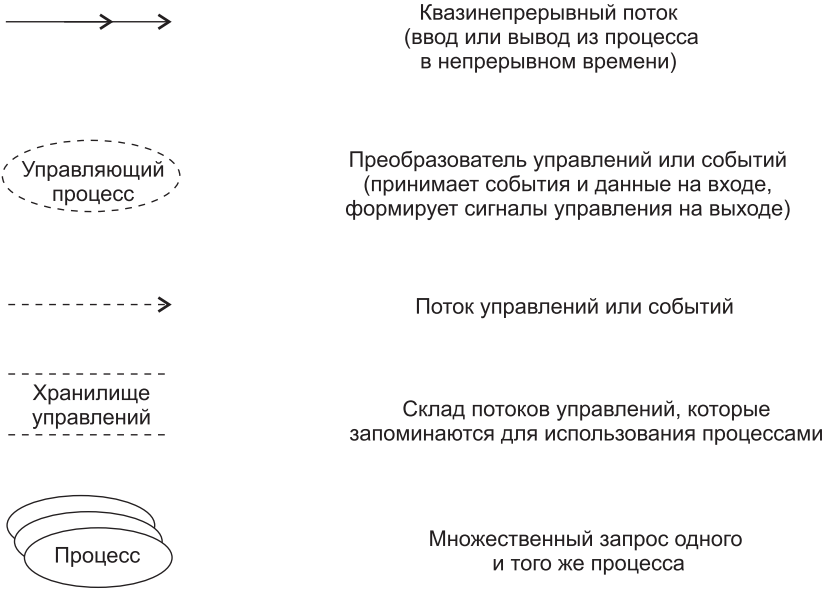


**Рис. 5.4.** Расширения диаграмм для систем реального времени

Приведем два примера использования новых элементов.

**Пример 5.1.** Использование потоков, непрерывных во времени.

На рис. 5.5 представлена модель анализа программного изделия для системы слежения за газовой турбиной.

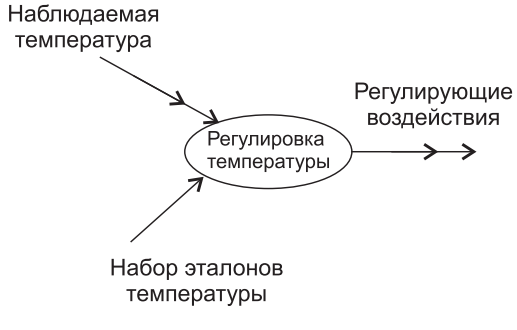


**Рис. 5.5.** Модель ПО для системы слежения за газовой турбиной

Видим, что здесь наблюдаемая температура измеряется непрерывно до тех пор, пока не будет найдено дискретное значение в наборе эталонов температуры. Преобразователь формирует регулирующие воздействия как непрерывный во времени вывод. Чем полезна эта модель?

Во-первых, инженер делает вывод, что для приема-передачи квазинепрерывных значений нужно использовать аналого-цифровую и цифро-аналоговую аппаратуру.

Во-вторых, необходимость организации высокоскоростного управления этой аппаратурой делает критичным требование к производительности системы.

**Пример 5.2.** Использование потоков управления.

Рассмотрим компьютерную систему, которая управляет роботом (рис. 5.6).

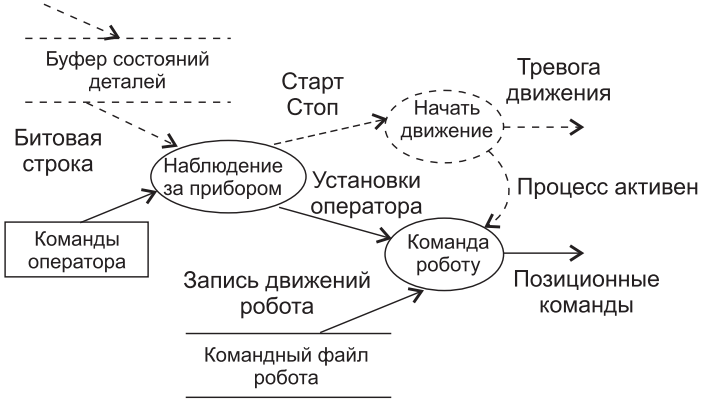


**Рис. 5.6.** Модель ПО для управления роботом

Установка в прибор деталей, собранных роботом, фиксируется установкой бита в буфере состояния деталей (он показывает присутствие или отсутствие каждой детали). Информация о событиях, запоминаемых в буфере, посылается в виде строки битов в преобразователь «Наблюдение за прибором». Преобразователь читает команды оператора только тогда, когда управляющая информация (битовая строка) показывает наличие всех деталей. Флаг события (Старт-Стоп) посылается в управляющий преобразователь «Начать движение», который руководит дальнейшей командной обработкой. Потоки данных посылаются в преобразователь команд роботу при наличии события «Процесс активен».

## Расширение возможностей управления

Д. Хетли и И. Пирбхаи сосредоточили внимание на аспектах управления программным продуктом [54]. Они выделили системные состояния и механизм перехода из одного состояния в другое. Д. Хетли и И. Пирбхаи предложили не вносить в ПДД элементы управления, такие как потоки управления и управляющие процессы. Вместо этого они ввели диаграммы управляющих потоков (УПД).

Диаграмма управляющих потоков содержит:

- обычные преобразователи (управляющие преобразователи исключены вообще);
- потоки управления и потоки событий (без потоков данных).

Вместо управляющих преобразователей в УПД используются указатели — ссылки на управляющую спецификацию УСПЕЦ. Как показано на рис. 5.7, ссылка изображается как косая пунктирная стрелка, указывающая на окно УСПЕЦ (вертикальную черту).


**Рис. 5.7.** Изображение ссылки на управляющую спецификацию

УСПЕЦ управляет преобразователями в ПДД на основе события, которое проходит в ее окно (по ссылке). Она предписывает включение конкретных преобразователей как результат конкретного события.

Иллюстрация модели программной системы, использующей описанные средства, приведена на рис. 5.8.

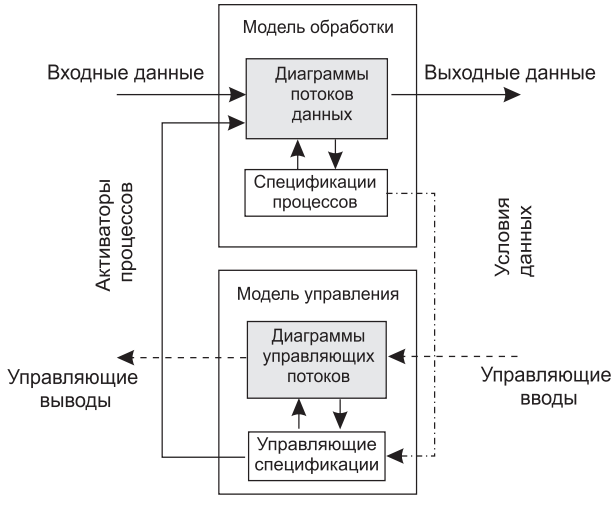


**Рис. 5.8.** Композиция модели обработки и управления

В модель обработки входит набор диаграмм потоков данных и набор спецификаций процессов. Модель управления образуют набор диаграмм управляющих потоков и набор управляющих спецификаций. Модель обработки подключается к модели управления с помощью активаторов процессов. Активаторы включают в конкретной ПДД конкретные преобразователи. Обратная связь модели обработки с моделью управления осуществляется с помощью условий данных. Условия данных формируются в ПДД (когда входные данные преобразуются в события).

## Модель системы регулирования давления космического корабля

Обсудим модель системы регулирования давления космического корабля, представленную на рис. 5.9.

Начнем с диаграммы потоков данных. Основной процесс в ПДД — Слежение и регулирование давления. На его входы поступают: измеренное Давление в кабине и Max давление. На выходе процесса — поток данных Изменение давления. Содержание процесса описывается в его спецификации ПСПЕЦ.

Спецификация процесса ПСПЕЦ может включать:

1) поясняющий текст (обязательно);
2) описание алгоритма обработки;
3) математические уравнения;
4) таблицы;
5) диаграммы.

Элементы со второго по пятый не обязательны.


**Рис. 5.9.** Модель системы регулирования давления космического корабля

С помощью ПСПЕЦ разработчик создает описание для каждого преобразователя, которое рассматривается как:

- первый шаг создания спецификации требований к программному изделию;
- руководство для проектирования программ, которые будут реализовывать процессы.

В нашем примере спецификация процесса имеет вид:

```
если Давление в кабине > max
        то Избыточное давление:=1;
        иначе Избыточное давление:=0;
алгоритм регулирования;
выч.Изменение давления;
конец если;
```

Таким образом, когда давление в кабине превышает максимум, генерируется управляющее событие Избыточное давление. Оно должно быть показано на диаграмме управляющих потоков УПД. Это событие входит в окно управляющей спецификации УСПЕЦ.

Управляющая спецификация моделирует поведение системы. Она содержит:

- таблицу активации процессов (ТАП);
- диаграмму переходов-состояний (ДПС).

Таблица активации процессов показывает, какие процессы будут вызываться (активироваться) в потоковой модели в результате конкретных событий.

ТАП включает три раздела — Входные события, Выходные события, Активация процессов. Логика работы ТАП такова: входное событие вызывает выходное событие, которое активирует конкретный процесс. Для нашей модели ТАП имеет вид, представленный в табл. 5.1.

**Таблица 5.1.** Таблица активации процессов

| **Входные события:** | | | |
|---|---|---|---|
| Включение системы | 1 | 0 | 0 |
| Избыточное давление | 0 | 1 | 0 |
| Норма | 0 | 0 | 1 |
| **Выходные события:** | | | |
| Тревога | 0 | 1 | 0 |
| Работа | 1 | 0 | 1 |
| **Активация процессов:** | | | |
| Слежение и регулирование давления | 1 | 0 | 1 |
| Уменьшение давления | 0 | 1 | 0 |

Видим, что в нашем примере входных событий три: два внешних события (Включение системы, Норма) и одно — условие данных (Избыточное давление). Работа ТАП инициируется входным событием, «втекающим» в окно УСПЕЦ. В результате ТАП вырабатывает выходное событие — активатор. В нашем примере активаторами являются события Работа и Тревога. Активатор «вытекает» из окна УСПЕЦ, запуская в УПД конкретный процесс.

Другой элемент УСПЕЦ — Диаграмма переходов-состояний. ДПС отражает состояния системы и показывает, как она переходит из одного состояния в другое.

ДПС для нашей модели показан на рис. 5.10.

Системные состояния показаны прямоугольниками. Стрелки показывают переходы между состояниями. Стрелки переходов подписывают следующим образом:

в числителе — событие, которое вызывает переход, в знаменателе — процесс, запускаемый как результат события. Изучая ДПС, разработчик может анализировать поведение модели и установить, нет ли «дыр» в определении поведения.

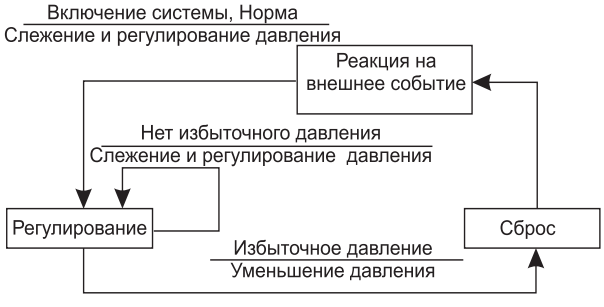


**Рис. 5.10.** Диаграмма переходов-состояний

## Методы анализа, ориентированные на структуры данных

Элементами предметной области для любой системы являются потоки, процессы и структуры данных. При структурном анализе активно работают только с потоками данных и процессами.

Методы, ориентированные на структуры данных, обеспечивают:

1. Определение ключевых информационных объектов и операций.
2. Определение иерархической структуры данных.
3. Компоновку структур данных из типовых конструкций — последовательности, выбора, повторения.
4. Последовательность шагов для превращения иерархической структуры данных в структуру программы.

Наиболее известны два метода: метод Варнье–Орра и метод Джексона.

В методе Варнье–Орра для представления структур применяют диаграммы Варнье [78]. Для построения диаграмм Варнье используют три базовых элемента: последовательность, выбор, повторение (рис. 5.11) [100].

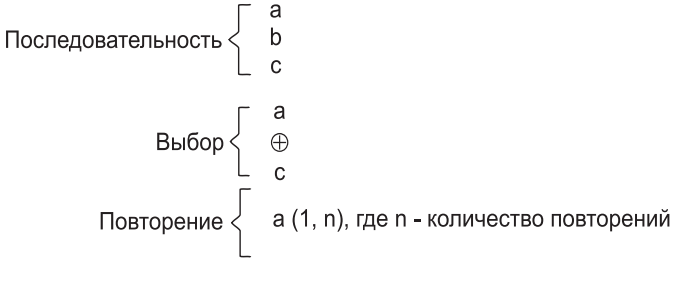


**Рис. 5.11.** Базовые элементы в диаграммах Варнье

Как показано на рис. 5.12, с помощью этих элементов можно строить информационные структуры с любым количеством уровней иерархии.

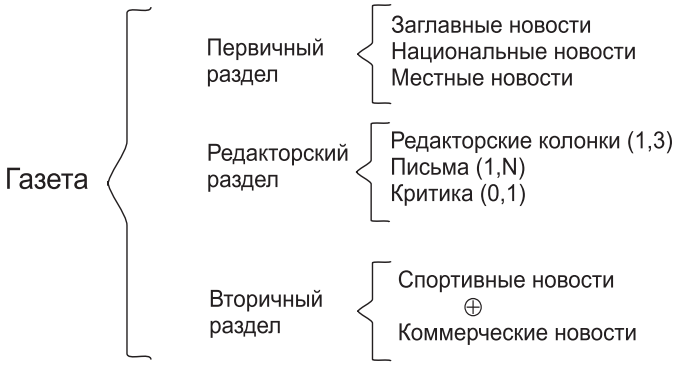


**Рис. 5.12.** Структура газеты в виде диаграммы Варнье

Как видим, для представления структуры газеты здесь используются три уровня иерархии.

## Метод анализа Джексона

Как и метод Варнье–Орра, метод Джексона появился в период революции структурного программирования. Фактически оба метода решали одинаковую задачу: распространить базовые структуры программирования (последовательность, выбор, повторение) на всю область разработки сложных программных систем. Именно поэтому основные выразительные средства этих методов оказались так похожи друг на друга.

### Методика Джексона

Метод Джексона (1975) включает 6 шагов [61]. Три шага выполняются на этапе анализа, а остальные — на этапе проектирования.

1. *Объект-действие*. Определяются объекты — источники или приемники информации и действия — события реального мира, воздействующие на объекты.
2. *Объект-структура*. Действия над объектами представляются диаграммами Джексона.
3. *Начальное моделирование*. Объекты и действия представляются как обрабатывающая модель. Определяются связи между моделью и реальным миром.
4. *Доопределение функций*. Выделяются и описываются сервисные функции.
5. *Учет системного времени*. Определяются и оцениваются характеристики планирования будущих процессов.
6. *Реализация*. Согласование с системной средой, разработка аппаратной платформы.

## Шаг объект-действие

Начинается с определения проблемы на естественном языке.

**Пример 5.3.** Разработать компьютерную систему для обслуживания университетских перевозок. Университет размещается на двух территориях. Для перемещения студентов используется один транспорт. Он перемещается между двумя фиксированными остановками. На каждой остановке имеется кнопка вызова.

При нажатии кнопки:

- если транспорт на остановке, то студенты заходят в него и перемещаются на другую остановку;
- если транспорт в пути, то студенты ждут прибытия на другую остановку, приема студентов и возврата на текущую остановку;
- если транспорт на другой остановке, то он ее покидает, прибывает на текущую остановку и принимает студентов, нажавших кнопку.

Транспорт должен стоять на остановке до появления запроса на обслуживание.

Описание исследуется для выделения объектов. Производится грамматический разбор. Возможны следующие кандидаты в объекты: территория, студенты, транспорт, остановка, кнопка. У нас нет нужды прямо использовать территорию, студентов, остановку — все они лежат вне области модели и отвергаются как возможные объекты. Таким образом, мы выбираем объекты транспорт и кнопка.

Для выделения действий исследуются все глаголы описания.

Кандидатами действий являются: перемещаться, прибывает, нажимать, принимать, покидать. Мы отвергаем перемещаться, принимать потому, что они относятся к студентам, а студенты не выделены как объект. Мы выбираем действия: прибывает, нажимать, покидать.

Заметим, что при выделении объектов и действий возможны ошибки. Например, отвергнув студентов, мы лишились возможности исследования загрузки транспорта. Впрочем, список объектов и действий может модифицироваться в ходе дальнейшего анализа.

## Шаг объект-структура

Структура объектов описывает последовательность действий над объектами (в условном времени).

Для представления структуры объектов Джексон предложил три типа структурных диаграмм. Они показаны на рис. 5.13. В первой диаграмме к объектам применяется такое действие, как последовательность, во второй — выбор, в третьей — повторение.

Рассмотрим объектную структуру для транспорта (рис. 5.14). Условимся, что начало и конец истории транспорта — у первой остановки. Действиями, влияющими на объект, являются Покинуть и Прибыть.

Диаграмма показывает, что транспорт начинает работу у остановки 1, тратит основное время на перемещение между остановками 1 и 2 и окончательно возвращается на остановку 1. Прибытие на остановку, следующее за отъездом с другой остановки, представляется как пара действий Прибыть($i$) и Покинуть($i$). Заметим, что диаграмму можно сопровождать комментариями, которые не могут прямо пред-

ставляться средствами метода. Например, «значение *i* в двух последовательных остановках должно быть разным».

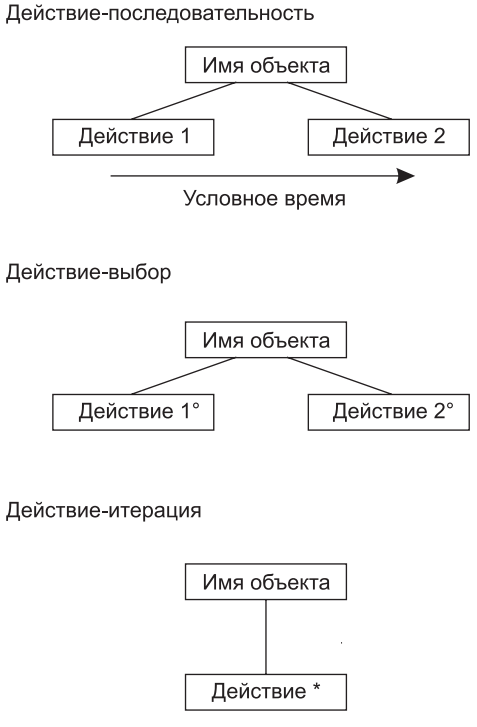


**Рис. 5.13.** Три типа структурных диаграмм Джексона

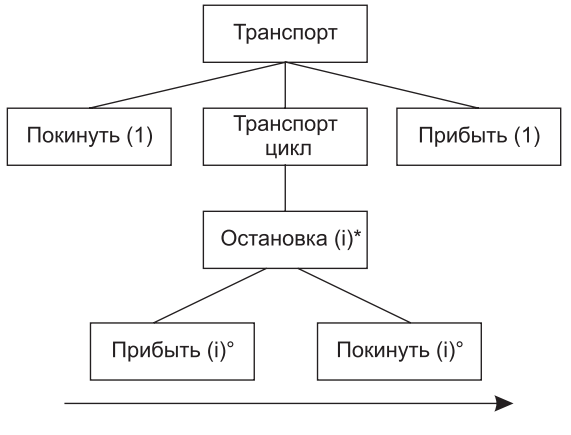


**Рис. 5.14.** Объектная структура для транспорта

Структурная диаграмма для объекта Кнопка показывает (рис. 5.15), что к нему многократно применяется действие Нажать.

В заключение заметим, что структурная диаграмма — время-ориентированное описание действий, выполняемых над объектом. Она создается для каждого объекта модели.


**Рис. 5.15.** Структурная диаграмма для объекта Кнопка

## Шаг начального моделирования

Начальное моделирование — это шаг к созданию описания системы как модели реального мира. Описание создается с помощью диаграммы системной спецификации.

Элементами диаграммы системной спецификации являются физические процессы (имеют суффикс 0) и их модели (имеют суффикс 1). Как показано на рис. 5.16, предусматриваются два вида соединений между физическими процессами и моделями.

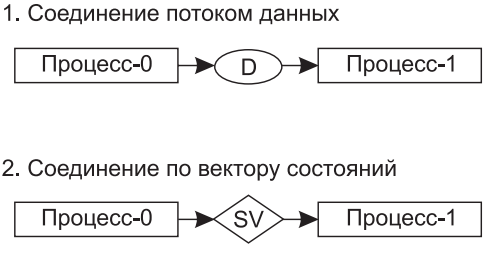


**Рис. 5.16.** Соединения между физическими процессами и их моделями

Соединение потоком данных производится, когда физический процесс передает, а модель принимает информационный поток. Полагают, что поток передается через буфер неограниченной емкости типа FIFO (обозначается овалом).

Соединение по вектору состояний происходит, когда модель наблюдает вектор состояния физического процесса. Вектор состояния обозначается ромбиком.

Диаграмма системной спецификации для системы обслуживания перевозок приведена на рис. 5.17.

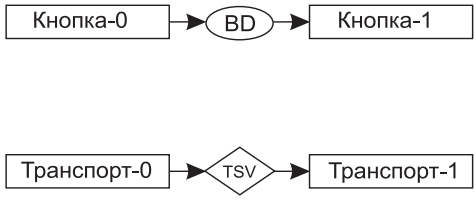


**Рис. 5.17.** Диаграмма системной спецификации для системы обслуживания перевозок

Для фиксации особенностей процессов-моделей Джексон предлагает специальное описание — структурный текст. Например, структурный текст для модели КНОПКА-1 имеет вид:

```
КНОПКА-1
    читать BD;
    НАЖАТЬ цикл ПОКА BD
        нажать;
        читать BD;
    конец НАЖАТЬ;
конец КНОПКА-1;
```

**ПРИМЕЧАНИЕ**

При нажатии кнопки формируется импульс, который может быть передан в модель как элемент данных, поэтому для кнопки выбрано соединение потоком данных.

Датчики, регистрирующие прибытие и убытие транспорта, не формируют импульса, они воздействуют на электронный переключатель. Состояние переключателя может быть оценено. Поэтому для транспорта выбрано соединение по вектору состояний.

Структура модели КНОПКА-1 отличается от структуры физического процесса КНОПКА-0 добавлением оператора для чтения буфера BD, который соединяет физический мир с моделью.

Прежде чем написать структурный текст для модели ТРАНСПОРТ-1, мы должны сделать ряд замечаний.

Во-первых, состояние транспорта будем отслеживать по переменным ПРИБЫЛ, УБЫЛ. Они отражают состояние электронного переключателя физического транспорта.

Во-вторых, для учета инерционности процессов в физическом транспорте в модель придется ввести дополнительные операции:

- ЖДАТЬ (ожидание в изменении состояния физического транспорта);
- ТРАНЗИТ (операция задержки в модели на перемещение транспорта между остановками).

С учетом замечаний структурная диаграмма модели примет вид, изображенный на рис. 5.18.

Соответственно, структурный текст модели записывается в форме:

```
ТРАНСПОРТ-1
    опрос TSV;
    ЖДАТЬ цикл ПОКА ПРИБЫЛ(1)
        опрос TSV;
    конец ЖДАТЬ;
    покинуть(1);
    ТРАНЗИТ цикл ПОКА УБЫЛ(1)
        опрос TSV;
    конец ТРАНЗИТ;
    ТРАНСПОРТ цикл
        ОСТАНОВКА
            прибыть(i);
            ЖДАТЬ цикл ПОКА ПРИБЫЛ(i)
                опрос TSV;
```

```
            конец ЖДАТЬ;
            покинуть(i);
            ТРАНЗИТ цикл ПОКА УБЫЛ(i)
                опрос TSV;
            конец ТРАНЗИТ;
        конец ОСТАНОВКА;
    конец ТРАНСПОРТ;
    прибыть(1);
конец ТРАНСПОРТ-1;
```

Он описывает полный цикл работы транспорта.

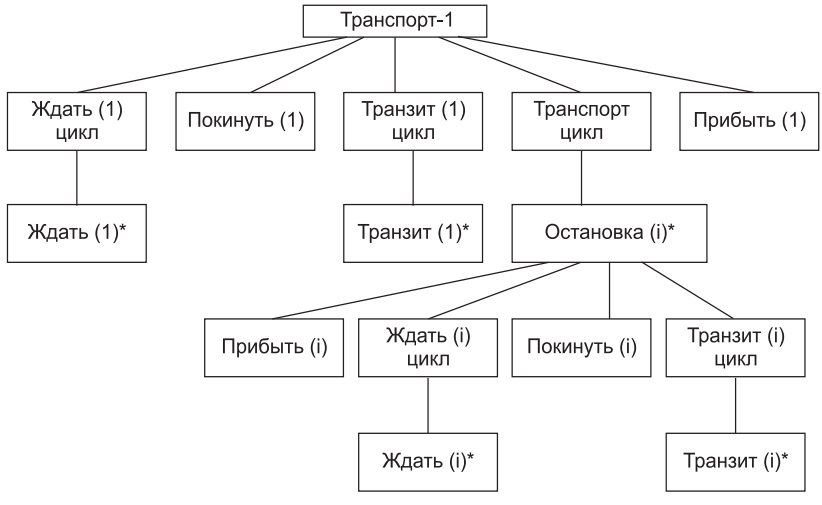


**Рис. 5.18.** Структурная диаграмма модели транспорта

## Контрольные вопросы и упражнения

1. Какие задачи решает аппарат анализа?
2. Что такое диаграмма потоков данных?
3. Чем отличается диаграмма потоков данных от блок-схемы алгоритма?
4. Какие элементы диаграммы потоков данных вы знаете?
5. Как формируется иерархия диаграмм потоков данных?
6. Какую задачу решает диаграмма потоков данных высшего (нулевого) уровня? Почему ее называют контекстной моделью?
7. Чем нагружены вершины диаграммы потоков данных?
8. Чем нагружены дуги диаграммы потоков данных?
9. Как организован словарь требований?
10. С помощью аппарата структурного анализа создайте модель системы управления летательного аппарата. В качестве исходных данных используйте требования заказчика из примера 4.1 предыдущей главы.

11. С чем связана необходимость расширения диаграмм потоков данных для систем реального времени? Какие средства расширения вы знаете?
12. С помощью расширений Варда и Меллора модифицируйте модель, построенную в пункте 10.
13. Как решается проблема расширения возможностей управления на базе диаграмм потоков данных?
14. Каковы особенности диаграммы управляющих потоков?
15. Поясните понятие активатора процесса.
16. Поясните понятие условия данных.
17. Поясните понятие управляющей спецификации.
18. Поясните понятие окна управляющей спецификации.
19. Как организована спецификация процесса?
20. Поясните назначение таблицы активации процессов.
21. Поясните организацию диаграммы переходов-состояний.
22. Примените расширения Хетли и Пирбхаи к усовершенствованию модели из пункта 12. Какие новые проблемы приходится теперь решать?
23. Какие задачи решают методы анализа, ориентированные на структуры данных?
24. Какие методы анализа, ориентированные на структуры данных, вы знаете?
25. Из каких базовых элементов состоят диаграммы Варнье?
26. Создайте с помощью диаграмм Варнье модель газеты вашего института.
27. Какие шаги выполняет метод Джексона на этапе анализа?
28. Какие типы структурных диаграмм Джексона вы знаете?
29. Как организовано в методе Джексона обнаружение объектов?
30. Что такое структура объектов Джексона?
31. Как создается структура объектов Джексона?
32. Поясните диаграмму системной спецификации Джексона.
33. Чем отличается соединение потоком данных от соединения по вектору состояний?
34. Какова задача структурного текста Джексона?
35. Усовершенствуйте модель из примера 5.3 данной главы для случая, когда университет размещается на трех территориях.

# Глава 7

# Классические методы проектирования

В этой главе рассматриваются классические методы проектирования, ориентированные на процедурную реализацию программных систем (ПС). Повторим, что эти методы появились в период революции структурного программирования. Учитывая, что на современном этапе программной инженерии процедурно-ориентированные ПС имеют преимущественно историческое значение, *конспективно* обсуждаются только два (наиболее популярных) метода: метод структурного проектирования и метод проектирования Майкла Джексона (этот Джексон не имеет никакого отношения к известному певцу). Зачем мы это делаем? Да чтобы знать исторические корни современных методов проектирования.

## Метод структурного проектирования

Этот метод поддерживает проектирование, ориентированное на потоки данных. Назначение метода — обеспечить систематический подход к созданию программной структуры, фундаменту архитектурного проектирования. В этом методе информация представляется как непрерывный поток, который подвергается серии преобразований по мере продвижения от входа к выходу. В качестве графического средства для отображения информационного потока используется диаграмма потока данных. Область применения — последовательная обработка не иерархических структур данных, которая характерна для управляющих приложений, методов численного анализа, управления процессами. Исходными данными для метода структурного проектирования являются компоненты модели анализа ПС, которая представляется иерархией диаграмм потоков данных [54, 76, 82, 99, 104]. Результат структурного проектирования — иерархическая структура ПС.

Шаги метода:

1) определение типа информационного потока;
2) выделение границ потока;
3) отображение диаграммы потока данных в начальную структуру системы;

4) определение иерархии управления (разложением на элементы);
5) уточнение полученной структуры (используются проектные эвристики, то есть продиктованные опытом рекомендации, ориентированные на повышение качества результата).

Действия на шаге 3 зависят от типа информационного потока в модели анализа.

## Типы информационных потоков

Различают два типа информационных потоков:

- поток преобразований;
- поток запросов.

Информация должна войти в ПС и выйти из нее в формате «внешнего мира». Примеры форматов информации внешнего мира: данные с клавиатуры, наборы для телефонной линии, рисунки для компьютерного графического дисплея. Для организации обработки входные данные из внешнего формата должны быть преобразованы во внутренний формат. Далее результаты обработки подвергаются обратному преобразованию, то есть возвращаются в формат «внешнего мира». Этим трем этапам соответствуют три элемента в *потоке преобразований*: Входящий поток, Преобразуемый поток и Выходящий поток (рис. 7.1).

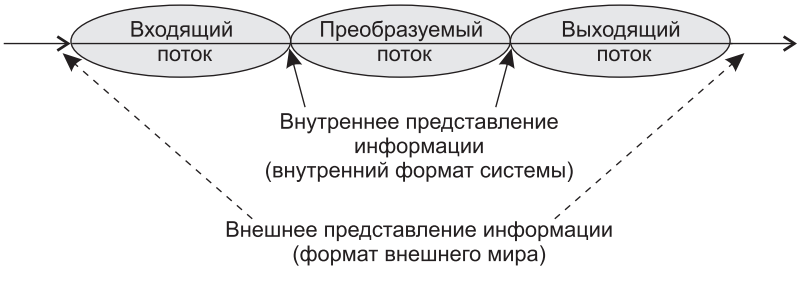


**Рис. 7.1.** Элементы потока преобразований

Информационные потоки могут также иметь особые элементы данных — запросы. Назначение элемента-запроса состоит в том, чтобы запустить поток данных по одному из нескольких путей. Потоки с элементами-запросами называют *потоками запросов*.

Анализ запроса и переключение потока данных на один из путей действий происходит в центре запросов. Центр запросов принимает входящий поток, выделяет

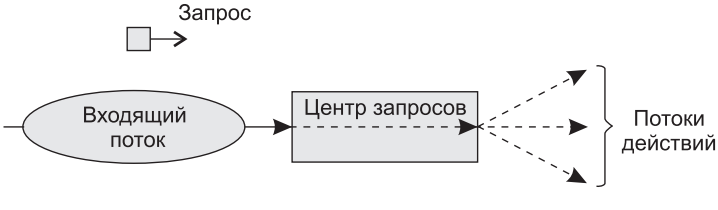


**Рис. 7.2.** Структура потока запроса

в нем запрос, проводит оценку запроса и по итогам оценки отправляет поток по одному из путей. Иными словами, центр запросов играет роль стрелочника.

Структуру потока запроса иллюстрирует рис. 7.2.

## Проектирование для потока данных типа «преобразование»

*Шаг 1. Проверка основной системной модели.* Модель включает: контекстную диаграмму ПДД0, словарь данных и спецификации процессов. Оценивается их согласованность с системной спецификацией.

*Шаг 2. Проверки и уточнения диаграмм потоков данных уровней 1 и 2.* Оценивается согласованность диаграмм, достаточность детализации преобразователей.

*Шаг 3. Определение типа основного потока диаграммы потоков данных.* Основной признак потока преобразований — отсутствие переключения по путям действий.

*Шаг 4. Определение границ входящего и выходящего потока, отделение центра преобразований.* Входящий поток — отрезок, на котором информация преобразуется из внешнего во внутренний формат представления. Выходящий поток обеспечивает обратное преобразование — из внутреннего формата во внешний. Границы входящего и выходящего потоков достаточно условны. Вариация одного преобразователя на границе слабо влияет на конечную структуру ПС.

*Шаг 5. Определение начальной структуры ПС.* Иерархическая структура ПС формируется нисходящим распространением управления. В иерархической структуре:

- модули верхнего уровня принимают решения;
- модули нижнего уровня выполняют работу по вводу, обработке и выводу;
- модули среднего уровня реализуют как функции управления, так и функции обработки.

Начальная структура ПС (для потока преобразования) стандартна и включает *главный контроллер* (находится на вершине структуры) и три подчиненных контроллера:

1. *Контроллер входящего потока* (контролирует получение входных данных).
2. *Контроллер преобразуемого потока* (управляет операциями над данными во внутреннем формате).
3. *Контроллер выходящего потока* (управляет получением выходных данных).

Данный минимальный набор модулей покрывает все функции управления, обеспечивает хорошую связность и слабое сцепление структуры.

Начальная структура ПС представлена на рис. 7.3.

*Шаг 6. Детализация структуры ПС.* Выполняется отображение преобразователей ПДД в модули структуры ПС. Отображение выполняется движением по ПДД от границ центра преобразования вдоль входящего и выходящего потоков. Входящий поток проходится от конца к началу, а выходящий поток — от начала к концу. В ходе движения преобразователи отображаются в модули подчиненных уровней структуры (рис. 7.4).

Центр преобразования ПДД отображается иначе (рис. 7.5). Каждый преобразователь отображается в модуль, непосредственно подчиненный контроллеру центра.

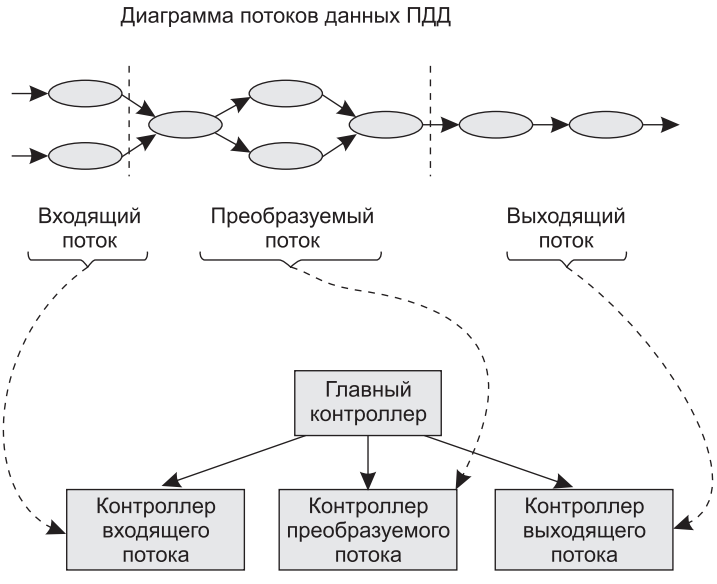


**Рис. 7.3.** Начальная структура ПС для потока «преобразование»

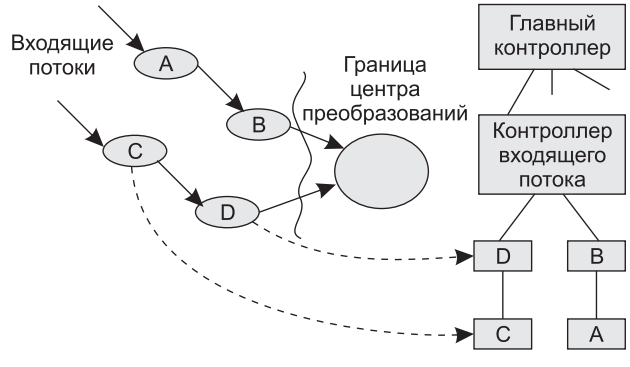


**Рис. 7.4.** Отображение преобразователей ПДД в модули структуры

Преобразуемый поток проходится слева направо.

Возможны следующие варианты отображения:

- 1 преобразователь отображается в 1 модуль;
- 2–3 преобразователя отображаются в 1 модуль;
- 1 преобразователь отображается в 2–3 модуля.

Для каждого модуля полученной структуры на базе спецификаций процессов модели анализа пишется сокращенное описание обработки. Описание включает: входную и выходную информацию модуля (интерфейсное описание); информацию, которая сохраняется модулем (данные, сохраняемые в локальной структуре данных); процедурное описание, которое указывает основные особенности решения и задачи;

короткое обсуждение ограничений и специальных свойств (файлы ввода-вывода, аппаратно-зависимые характеристики, специальные требования к параметрам времени).

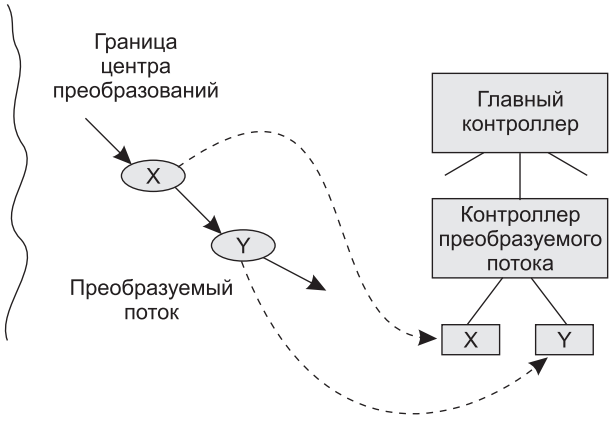


**Рис. 7.5.** Отображение центра преобразования ПДД

*Шаг 7. Уточнение иерархической структуры ПС.* Модули разделяются и объединяются для:

1) повышения связности и уменьшения сцепления;
2) упрощения реализации;
3) упрощения тестирования;
4) повышения удобства сопровождения.

Целью семи шагов является создание структурной организации системы, которую можно оценивать и уточнять. Модификации в это время требуют малой дополнительной работы, но оказывают глубокое воздействие на качество и сопровождаемость программного продукта.

## Проектирование для потока данных типа «запрос»

Во многих приложениях некоторые элементы данных (запросы) выполняют переключающую функцию, то есть функцию, управляющую переключением информационных потоков.

*Шаг 1. Проверка основной системной модели.* Модель включает: контекстную диаграмму ПДД0, словарь данных и спецификации процессов. Оценивается их согласованность с системной спецификацией.

*Шаг 2. Проверки и уточнения диаграмм потоков данных уровней 1 и 2.* Оценивается согласованность диаграмм, достаточность детализации преобразователей.

*Шаг 3. Определение типа основного потока диаграммы потоков данных.* Основной признак потоков запросов — явное переключение данных на один из путей действий.

*Шаг 4. Определение центра запросов и типа для каждого из потоков действия.* Если конкретный поток действия имеет тип «преобразование», то для него указываются границы входящего, преобразуемого и выходящего потоков.

*Шаг 5. Определение начальной структуры ПС.* В начальную структуру отображается та часть диаграммы потоков данных, в которой распространяется поток запросов. Начальная структура ПС для потока запросов стандартна и включает входящую ветвь и диспетчерскую ветвь.

Структура входящей ветви формируется так же, как и в предыдущей методике.

Диспетчерская ветвь включает диспетчер, находящийся на вершине ветви, и контроллеры потоков действия, подчиненные диспетчеру; их должно быть столько, сколько имеется потоков действий.

*Шаг 6. Детализация структуры ПС.* Производится отображение в структуру каждого потока действия. Каждый поток действия имеет свой тип. Могут встретиться поток-«преобразование» (отображается по предыдущей методике) и поток запросов. На рис. 7.6 приведен пример отображения потока действия 1. Подразумевается, что он является потоком преобразования.

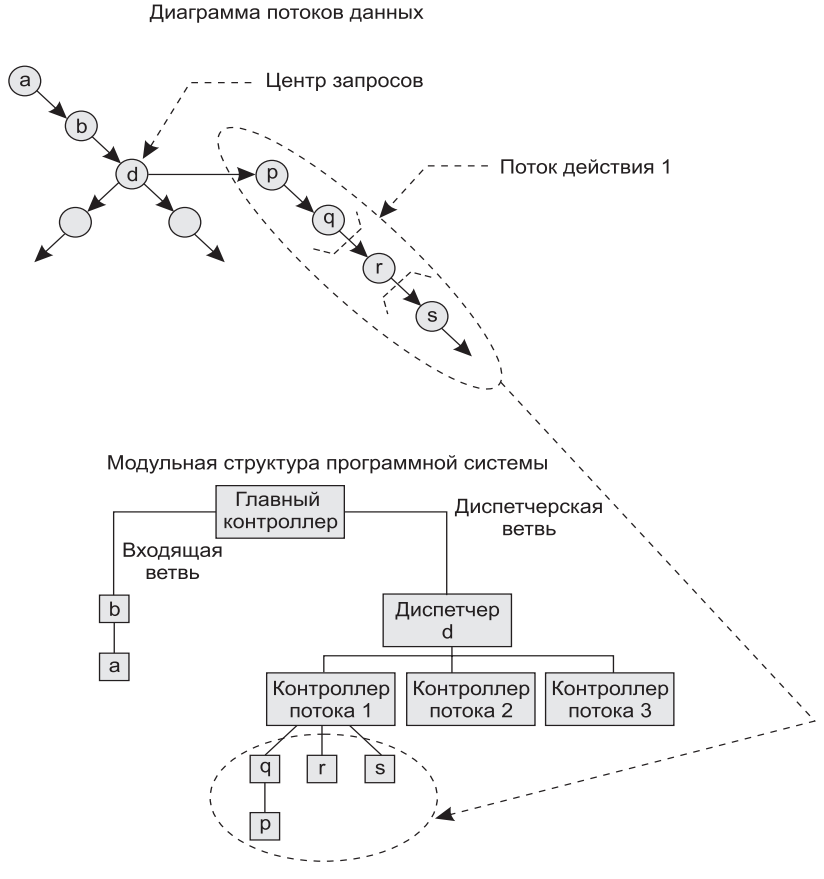


**Рис. 7.6.** Отображение в модульную структуру ПС потока действия 1

*Шаг 7. Уточнение иерархической структуры ПС.* Уточнение выполняется для повышения качества системы. Как и при предыдущей методике, критериями уточ-

нения служат: независимость модулей, эффективность реализации и тестирования, улучшение сопровождаемости.

# Метод проектирования Джексона

Для иллюстрации проектирования по этому методу продолжим пример с системой обслуживания перевозок, начатый в главе 5.

Метод Джексона включает шесть шагов [61]. Три первых шага относятся к этапу анализа. Это шаги: *объект — действие*, *объект — структура*, *начальное моделирование*. Их мы уже рассмотрели.

## Доопределение функций

Следующий шаг — доопределение функций. Этот шаг развивает диаграмму системной спецификации этапа анализа. Уточняются процессы-модели. В них вводятся дополнительные функции. Джексон выделяет три типа сервисных функций:

1. Встроенные функции (задаются командами, вставляемыми в структурный текст процесса-модели).
2. Функции впечатления (наблюдают вектор состояния процесса-модели и вырабатывают выходные результаты).
3. Функции диалога. Они решают следующие задачи:
   - наблюдают вектор состояния процесса-модели;
   - формируют и выводят поток данных, влияющий на действия в процессе-модели;
   - выполняют операции для выработки некоторых результатов.

Встроенную функцию введем в модель ТРАНСПОРТ-1. Предположим, что в модели есть панель с лампочкой, сигнализирующей о прибытии. Лампочка включается командой LON(i), а выключается командой LOFF(i). По мере перемещения транспорта между остановками формируется поток LAMP-команд. Модифицированный структурный текст модели ТРАНСПОРТ-1 принимает вид:

```
ТРАНСПОРТ-1
    LON(1);
    опрос SV;
    ЖДАТЬ цикл ПОКА ПРИБЫЛ(1)
        опрос SV;
    конец ЖДАТЬ;
    LOFF(1);
    покинуть(1);
    ТРАНЗИТ цикл ПОКА УБЫЛ(1)
        опрос SV;
    конец ТРАНЗИТ;
    ТРАНСПОРТ цикл
        ОСТАНОВКА;
            прибыть(i);
            LON(i);
            ЖДАТЬ цикл ПОКА ПРИБЫЛ(i)
                опрос SV;
            конец ЖДАТЬ;
```

```
            LOFF(i);
            покинуть(i);
            ТРАНЗИТ цикл ПОКА УБЫЛ(i)
                опрос SV;
            конец ТРАНЗИТ;
        конец ОСТАНОВКА;
    конец ТРАНСПОРТ;
    прибыть(1);
конец ТРАНСПОРТ-1;
```

Теперь введем функцию впечатления. В нашем примере она может формировать команды для мотора транспорта: START, STOP.

Условия выработки этих команд:

- Команда STOP формируется, когда датчики регистрируют прибытие транспорта на остановку.
- Команда START формируется, когда нажата кнопка для запроса транспорта и транспорт ждет на одной из остановок.

Видим, что для выработки команды STOP необходима информация только от модели транспорта. В свою очередь, для выработки команды START нужна информация как от модели КНОПКА-1, так и от модели ТРАНСПОРТ-1. В силу этого для реализации функции впечатления введем функциональный процесс М-УПРАВЛЕНИЕ. Он будет обрабатывать внешние данные и формировать команды START и STOP.

Ясно, что процесс М-УПРАВЛЕНИЕ должен иметь внешние связи с моделями ТРАНСПОРТ-1 и КНОПКА. Соединение с моделью КНОПКА организуем через вектор состояния BV. Соединение с моделью ТРАНСПОРТ-1 организуем через поток данных S1D.

Для обеспечения М-УПРАВЛЕНИЯ необходимой информацией опять надо изменить структурный текст модели ТРАНСПОРТ-1. В нем предусмотрим занесение сообщения Прибыл в буфер S1D:

```
ТРАНСПОРТ-1
    LON(1);
    опрос SV;
    ЖДАТЬ цикл ПОКА ПРИБЫЛ(1)
        опрос SV;
    конец ЖДАТЬ;
    LOFF(1);
    покинуть(1);
    ТРАНЗИТ цикл ПОКА УБЫЛ(1)
        опрос SV;
    конец ТРАНЗИТ;
    ТРАНСПОРТ цикл
        ОСТАНОВКА;
            прибыть(i);
            записать Прибыл в S1D;
            LON(i);
            ЖДАТЬ цикл ПОКА ПРИБЫЛ(i)
                опрос SV;
            конец ЖДАТЬ;
            LOFF(i);
            покинуть(i);
            ТРАНЗИТ цикл ПОКА УБЫЛ(i)
                опрос SV;
```

```
            конец ТРАНЗИТ;
        конец ОСТАНОВКА;
    конец ТРАНСПОРТ;
    прибыть(1);
    записать Прибыл в S1D;
конец ТРАНСПОРТ-1;
```

Очевидно, что при такой связи процессов необходимо гарантировать, что процесс ТРАНСПОРТ-1 выполняет операции опрос SV, а процесс М-УПРАВЛЕНИЕ читает сообщения Прибытия в S1D с частотой, достаточной для своевременной остановки транспорта. Временные ограничения, планирование и реализация должны рассматриваться в последующих шагах проектирования.

В заключение введем функцию диалога. Свяжем эту функцию с необходимостью развития модели КНОПКА-1. Следует различать первое нажатие на кнопку (оно формирует запрос на поездку) и последующие нажатия на кнопку (до того, как поездка действительно началась).

Диаграмма дополнительного процесса КНОПКА-2, в котором учтено это уточнение, показана на рис. 7.7.

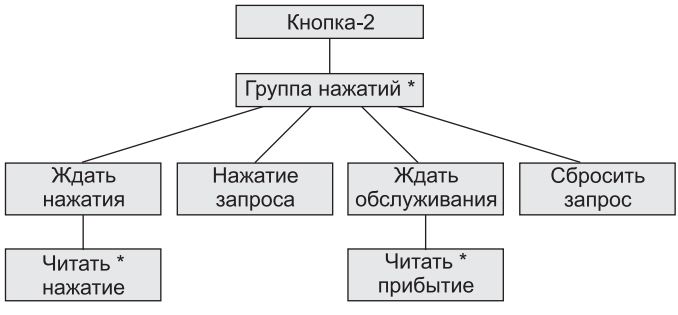


**Рис. 7.7.** Диаграмма дополнительного процесса КНОПКА-2

Внешние связи модели КНОПКА-2 должны включать:

- одно соединение с моделью КНОПКА-1 — организуется через поток данных B1D (для приема сообщения о нажатии кнопки);
- два соединения с процессом М-УПРАВЛЕНИЕ — одно организуется через поток данных MBD (для приема сообщения о прибытии транспорта), другое организуется через вектор состояния BV (для передачи состояния переключателя Запрос).

Таким образом, КНОПКА-2 читает два буфера данных, заполняемых процессами КНОПКА-1 и М-УПРАВЛЕНИЕ, и формирует состояние внутреннего электронного переключателя Запрос. Она реализует функцию диалога.

Структурный текст модели КНОПКА-2 может иметь следующий вид:

```
КНОПКА-2
    Запрос := НЕТ;
    читать B1D;
    ГрНАЖ цикл
        ЖдатьНАЖ цикл ПОКА Не НАЖАТА
            читать B1D;
        конец ЖдатьНАЖ;
```

```
        Запрос := ДА;
        читать MBD;
        ЖдатьОБСЛУЖ цикл ПОКА Не ПРИБЫЛ
            читать MBD;
        конец ЖдатьОБСЛУЖ;
        Запрос := НЕТ;
        читать B1D;
    конец ГрНАЖ;
конец КНОПКА-2;
```

Диаграмма системной спецификации, отражающая все изменения, представлена на рис. 7.8.

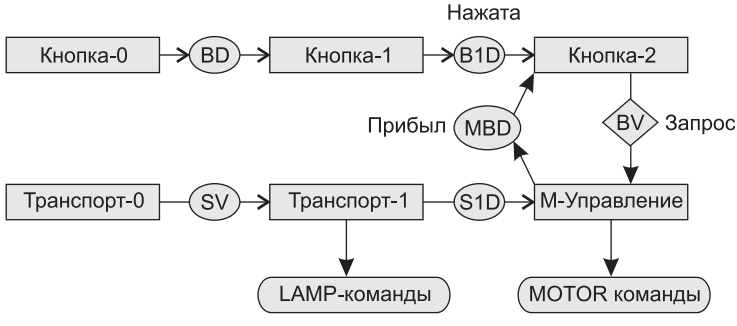


**Рис. 7.8.** Полная диаграмма системной спецификации

Встроенная в ТРАНСПОРТ-1 функция вырабатывает LAMP-команды, функция впечатления модели М-УПРАВЛЕНИЕ генерирует команды управления мотором, а модель КНОПКА-2 реализует функцию диалога (совместно с процессом М-УПРАВЛЕНИЕ).

## Учет системного времени

На шаге учета системного времени проектировщик определяет временные ограничения, накладываемые на систему, фиксирует дисциплину планирования. Дело в том, что на предыдущих шагах проектирования была получена система, составленная из последовательных процессов. Эти процессы связывали только потоки данных, передаваемые через буфер, и взаимные наблюдения векторов состояния. Теперь необходимо произвести дополнительную синхронизацию процессов во времени, учесть влияние внешней программно-аппаратной среды и совместно используемых системных ресурсов.

Временные ограничения для системы обслуживания перевозок, в частности, включают:

1) временной интервал на выработку команды STOP. Он должен выбираться путем анализа скорости транспорта и ограничения мощности;
2) время реакции на включение и выключение ламп панели.

Для рассмотренного примера нет необходимости вводить специальный механизм синхронизации. Однако при расширении может потребоваться некоторая синхронизация обмена данными.

## Контрольные вопросы и упражнения

1. В чем состоит суть метода структурного проектирования?
2. Какие различают типы информационных потоков?
3. Что такое входящий поток?
4. Что такое выходящий поток?
5. Что такое центр преобразования?
6. Как производится отображение входящего потока?
7. Как производится отображение выходящего потока?
8. Как производится отображение центра преобразования?
9. Какие задачи решают главный контроллер, контроллер входящего потока, контроллер выходящего потока и контроллер центра преобразования?
10. Поясните шаги метода структурного проектирования.
11. Что такое входящая ветвь?
12. Что такое диспетчерская ветвь?
13. Какие существуют различия в методике отображения потока преобразований и потока запросов?
14. Какие задачи уточнения иерархической структуры программной системы вы знаете?
15. С помощью аппарата структурного проектирования создайте проектную модель системы управления летательного аппарата. В качестве исходных данных используйте модель анализа, созданную в упражнении 10 главы 5.
16. Какие шаги предусматривает метод Джексона на этапе проектирования?
17. В чем состоит суть развития диаграммы системной спецификации Джексона?
18. Поясните понятие встроенной функции.
19. Поясните понятие функции впечатления.
20. Поясните понятие функции диалога.
21. В чем состоит учет системного времени (в методе Джексона)?
22. Сравните метод Джексона с методом структурного проектирования. Выделите их достоинства и недостатки.
23. На основе модели анализа, полученной в упражнении 35 главы 5, создайте по методу Джексона проектную модель для университета, размещаемого на трех территориях.

# Глава 11

# Особенности разработки баз данных

Неотъемлемой частью многочисленной категории программных систем являются базы данных. В этой главе обсуждаются: терминология баз данных, расширения языка UML, обеспечивающие моделирование баз данных, а также особенности разработки реляционных баз данных в объектно-ориентированном ПО.

## Основные понятия баз данных: модели данных

База данных (БД) — это совместно используемый набор устойчивых данных. База данных является единым и большим хранилищем данных. Это хранилище определяется один раз, а затем многократно (и совместно) используется различными пользователями или функциональными частями системы. Вместо отдельных файлов с избыточными данными все данные в БД собраны вместе, с минимальной долей избыточности. Обычно БД принадлежит не отдельному клиенту, а рассматривается как общий, разделяемый ресурс.

Термин «устойчивые» подчеркивает, что после помещения данных в БД удалить их можно только с помощью специального запроса, а не в результате побочного эффекта от действия некоторого клиента.

Хранимые в БД устойчивые данные имеют определенную логическую структуру, описываемую поддерживаемой моделью данных. В настоящее время применяют следующие модели данных:

- иерархическая;
- сетевая;
- реляционная;
- объектно-ориентированная;
- объектно-реляционная;
- полуструктурированная (XML-модель).

В иерархической модели структура данных представляется в виде дерева. Это значит, что каждая запись в БД может иметь сколь угодно потомков, но только одного родителя. Такая структура накладывает жесткие ограничения на организацию логических связей между данными.

В сетевой модели допускаются произвольные связи между данными. Недостатками являются высокая сложность структуры БД, трудности в ее понимании и обработке.

В реляционной модели (рис. 11.1) все данные находятся в таблицах. Связи между таблицами устанавливаются по совпадающим значениям в столбцах, имеющих одинаковые имена (ключевых столбцах).

| Номер заказчика | Фамилия | Город |
|---|---|---|
| 901 | Бендер | Шепетовка |
| 902 | Балаганов | Черноморск |
| 903 | Паниковский | Киев |

| Номер счета | Номер заказчика | Сумма |
|---|---|---|
| 1558 | 901 | 1000 |
| 1559 | 902 | 400 |
| 1560 | 903 | 600 |

**Рис. 11.1.** Структура реляционной БД

Достоинствами реляционной модели являются простота, гибкость, понятность, удобство реализации. Однако при увеличении числа таблиц заметно падает скорость работы с БД.

В объектно-ориентированной модели структура БД представляется в виде графа, узлами которого являются объекты. Здесь появляется возможность прозрачного отображения сложных взаимосвязей объектов, возможность «элегантного» применения всей мощи объектно-ориентированных механизмов (инкапсуляции, наследования, полиморфизма). Недостатками модели являются высокая понятийная сложность БД, неудобство обработки данных, низкая скорость выполнения запросов.

Объектно-реляционная модель предлагает гибридное решение, основанное на применении как реляционной, так и объектно-ориентированной модели. В данной модели структура БД представляется набором таблиц, но в самих таблицах хранятся объекты. При этом уменьшаются трудности, которые пришлось бы преодолеть на пути превращения чисто реляционной БД в чистую объектно-ориентированную БД.

Последняя модель, модель полуструктурированных данных, выполняет в БД особую задачу и обеспечивает максимальную гибкость. В этом случае данные модели самодостаточны: структура БД, описываемая набором вершин графа, определяется самими данными. Вершины графа соответствуют объектам и значениям их атрибутов, а дуги, обозначенные метками, соединяют объект со значениями его атрибутов и другими объектами. Метки дуг в модели играют двоякую роль. Представим, что из вершины $X$ в вершину $Y$ идет дуга с меткой $L$. Тогда возможны два варианта:

- вершину *X* можно рассматривать как объект или структуру, а вершину *Y* — как значение одного из атрибутов объекта или значение одного из полей структуры, причем имя атрибута (поля) равно *L*;
- обе вершины *X* и *Y* рассматриваются как объекты, между которыми есть связь с именем *L*.

Очевидно, что для задачи описания текстовых документов модель полуструктурированных данных реализует язык XML: вершины соответствуют фрагментам текста, а дуги — парам тегов.

## Организация реляционной базы данных

Простота и эффективность БД на основе реляционной модели по-прежнему обуславливает их доминирующее положение в программных приложениях. В настоящее время считается нормой использование реляционных БД в объектно-ориентированных программных системах. Судя по всему, это долгосрочная и устойчивая тенденция. Именно поэтому мы переходим к детальному рассмотрению реляционных БД.

Реляционная БД состоит из множества двумерных таблиц. В таблицах хранятся различные данные. Например, в составе БД могут быть таблицы заказчиков, товаров, счетов и т. д. Типовая структура таблицы реляционной БД представлена в табл. 11.1.

**Таблица 11.1.** Типовая структура таблицы реляционной БД

| | **Столбцы (атрибуты)** | | | |
|---|---|---|---|---|
| | **Фамилия** | **Город** | **Улица** | **Телефон** |
| **Строки (записи, кортежи)** | Бендер | Шепетовка | Рио | 290-00-87 |
| | Балаганов | Черноморск | Шмидта | 450-98-45 |
| | Паниковский | Киев | Крещатик | 750-12-34 |

Строки таблицы называют записями или кортежами. Столбцы называют атрибутами. На пересечении строки и столбца находится неделимое (атомарное) значение элемента данных. Набор допустимых значений атрибута (столбца) определяется его *доменом*. Домен может быть очень мал. Так, значениями атрибута *Размер* в таблице спортивных костюмов являются L, XL и XXL. И наоборот, домен атрибута *Фамилия* очень велик. В БД домен реализуется с помощью ограничения домена. Всякий раз при записи значения в БД проверяется его соответствие домену, зафиксированному для заданного атрибута. Таким образом, БД предохраняется от ввода недопустимых значений, например даты 32 мая (к искреннему сожалению барона Мюнхгаузена, благороднейшего и правдивейшего человека).

Виртуальным аналогом таблицы является *представление*, которое ведет себя с точки зрения клиента как обычная таблица, но не существует самостоятельно. Обычная таблица содержит данные. Представление же не содержит никаких данных, а только задает их источники (одну или несколько обычных таблиц, выбираемые строки, выбираемые столбцы). Фактически представление сохраняется в БД как запрос на создание определенного набора данных. Результат выполнения этого запроса является содержанием представления. При изменении данных в таблицах-источниках меняется и содержание представления.

Для выявления в таблице отдельной записи используют ключ. *Первичный ключ* (*Primary Key*, *PK*) имеет каждая таблица. Это столбец, однозначно определяющий каждую запись в таблице. В нашем примере в качестве *PK* может быть столбец *Фамилия*. Это правильно до появления, например, еще одного Бендера. Для обеспечения уникальности значения первичного ключа применяются две методики. Во-первых, может использоваться составной первичный ключ (*Composite Primary Key*), образуемый несколькими столбцами (естественными атрибутами) таблицы. Во-вторых, в качестве *PK* можно вводить в таблицу дополнительный столбец, не имеющий смысла с точки зрения предметной области. Его называют *суррогатным ключом*. Например, суррогатным ключом может быть *Номер заказчика* или *Номер заказа*.

Важную роль в реляционных БД играет еще один ключ — *Внешний ключ* (*Foreign Key*, *FK*). Внешний ключ — это столбец одной таблицы, который ссылается на первичный ключ другой таблицы. С помощью внешних ключей устанавливаются связи между различными таблицами БД (рис. 11.2). В этом примере показано, что таблицы счетов и заказчиков связаны ключом *Номер заказчика*. Если обратиться к таблице счетов, то *Номер счета* будет первичным ключом, а *Номер заказчика* — внешним ключом.

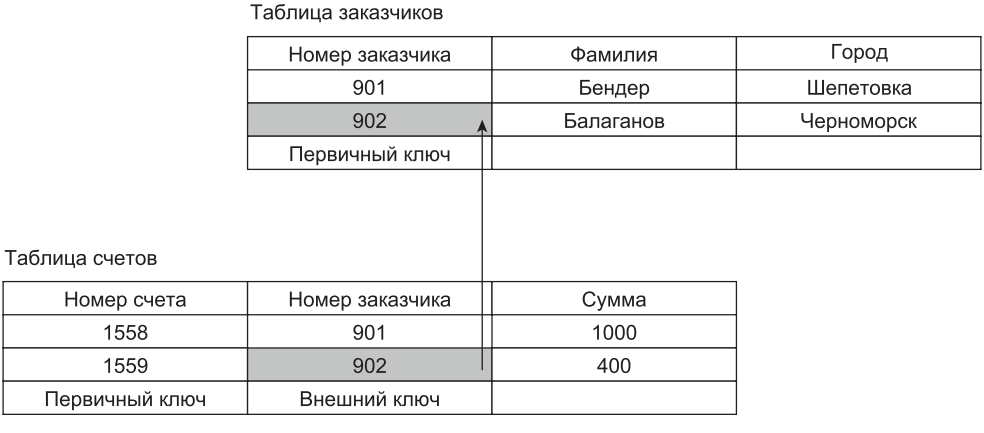


**Рис. 11.2.** Связь между таблицами БД на основе внешнего ключа

Для обеспечения целостности данных БД внешние ключи должны удовлетворять ограничению *ссылочной целостности*. Оно означает, что каждому значению внешнего ключа в одной таблице должно соответствовать значение существующего первичного ключа в другой таблице. Это важнейшее из всех ограничений, так как оно обеспечивает непротиворечивость перекрестных ссылок между таблицами. Если корректность значений *FK* не проверять, может нарушиться ссылочная целостность данных БД. Например, удаление строки из таблицы заказчиков может привести к тому, что в таблице заказов останутся записи о заказах, сделанных неизвестным теперь заказчиком (а кто же оплатит заказ?). Ограничения ссылочной целостности должны поддерживаться автоматически. Каждый раз при вводе или изменении данных БД средства управления проверяют ограничения и убеждаются в их соблюдении. Если ограничения нарушаются, изменение данных запрещается.

Кроме того, таблица может содержать вторичные ключи — индексы. Их используют как предметный указатель в книге. Чтобы найти в книге конкретный термин, не надо листать все страницы подряд — достаточно посмотреть в предметный указатель и найти нужный номер страницы. Например, можно создать индекс для столбца *Фамилия* (рис. 11.3). В результате сформируется небольшая таблица, в которой хранятся только фамилии и ссылки на позицию записи в основной таблице. Теперь для поиска записей не надо просматривать всю большую таблицу. В итоге получаем выигрыш в быстродействии. Правда, при добавлении и удалении записей (в основной таблице) таблицу индекса нужно создавать заново. Это замедляет операции.


**Рис. 11.3.** Ускорение доступа к таблице с помощью индекса

Оперативную обработку данных в реляционных БД выполняют *хранимые процедуры*. Разновидностью хранимых процедур являются *триггеры*. Триггер всегда связан с конкретной таблицей и вызывается автоматически при наступлении определенного события (например, вставки, удаления или обновления записи).

Обсудим отношения между таблицами. После формирования таблиц решают, как объединить их данные при извлечении из БД. Первым шагом является определение связей между таблицами. После этого возможно создание запросов, форм и отчетов, в которых выводятся данные из нескольких таблиц сразу. Например, чтобы напечатать счет, надо взять данные из разных таблиц и скомпоновать их.

Связь между таблицами устанавливает отношения между совпадающими значениями в ключевых полях. Рассмотрим разновидности отношений.

## Отношение «один-к-одному»

В этом случае каждой строке (записи) одной таблицы ставится в соответствие строка другой таблицы (рис. 11.4). Примером может быть отношение между таблицей сотрудников и таблицей их адресов.

Такое отношение встречается редко, так как соответствующие данные легко поместить в одну таблицу.

## Отношение «один-ко-многим»

Одной записи первой таблицы ставится в соответствие несколько записей во второй таблице (рис. 11.5). Каждая запись второй таблицы не может иметь более одной соответствующей записи в первой таблице.

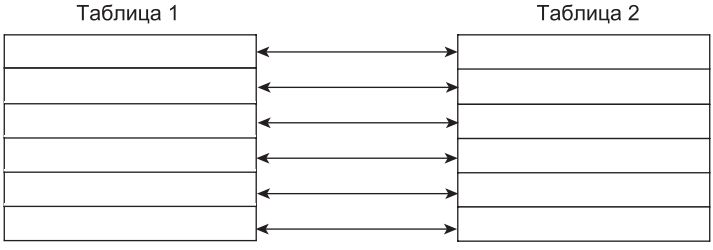


**Рис. 11.4.** Отношение «один-к-одному»

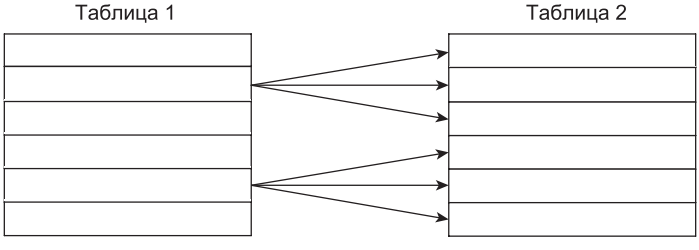


**Рис. 11.5.** Отношение «один-ко-многим»

Подобная разновидность отношения встречается наиболее часто.

## Отношение «многие-ко-многим»

Одной записи первой таблицы могут соответствовать несколько записей во второй таблице, а одной записи второй таблицы — несколько записей первой (рис. 11.6). Как правило, для организации таких отношений требуется вспомогательная таблица, которая состоит из первичных ключей двух основных таблиц.

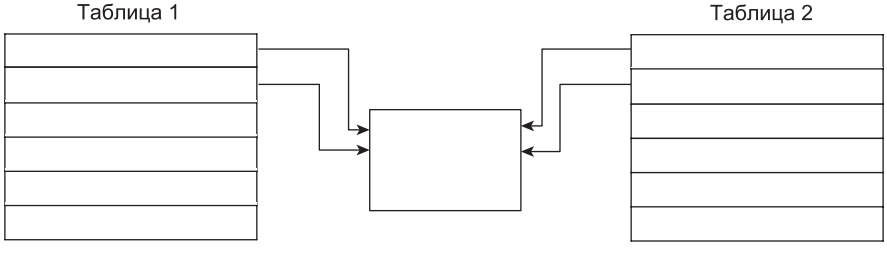


**Рис. 11.6.** Отношение «многие-ко-многим»

Такое отношение возникает между заказами и товарами. Один заказ может включать несколько наименований товара, а одно наименование товара входить в несколько заказов. Таким образом, должны существовать таблица заказов, таблица товаров и таблица с парами заказ-товар.

## Нормализация реляционных баз данных

Нормализация обеспечивает оптимизацию структуры БД. Она приводит к устранению избыточности в наборах данных. Выполняется нормализация БД последовательно, шаг за шагом. Правила нормализации оформлены в виде нормальных форм.

Первая нормальная форма (1НФ) требует, чтобы значения всех элементов данных в столбцах были атомарными. Пусть исходная таблица имеет вид, показанный в табл. 11.2.

**Таблица 11.2.** Исходная таблица продаж

| Заказчик | ИД продукта | Продукт | Количество | Цена | Всего |
|---|---|---|---|---|---|
| 1 | 111, 333, 555 | Лампа, нож, зонт | 4, 2, 3 | 1, 5, 50 | 4, 10, 150 |

Здесь каждая клетка содержит информацию о нескольких заказах одного клиента. Работать с такой таблицей очень сложно. Например, если клиент закажет дополнительный продукт, придется изменять практически всю таблицу. Нормализуем таблицу до 1НФ (табл. 11.3).

**Таблица 11.3.** Таблица продаж в 1НФ

| Заказчик | ИД продукта | Продукт | Количество | Цена | Всего |
|---|---|---|---|---|---|
| 1 | 111 | Лампа | 4 | 1 | 4 |
| 1 | 333 | Нож | 2 | 5 | 10 |
| 1 | 555 | Зонт | 3 | 50 | 150 |

Теперь легко получить отчет по каждому продукту. Однако если другому клиенту потребуется, например, лампа, придется все данные по этому продукту вводить заново.

Вторая нормальная форма (2НФ) требует, чтобы каждый неключевой столбец полностью зависел от первичного ключа. Перевод нашей таблицы в 2НФ приводит к появлению в БД двух таблиц (табл. 11.4 и табл. 11.5). Теперь столбцы *Продукт* и *Цена* выделены в отдельную таблицу, а таблица продаж преобразована в таблицу заказов. Отметим, что в таблице заказов остались только сведения, которые непосредственно связаны с заказами. Следовательно, вид товара учитывается только один раз.

**Таблица 11.4.** Таблица заказов: 2НФ базы данных (1-я часть)

| Заказчик | ИД продукта | Количество | Всего |
|---|---|---|---|
| 1 | 111 | 4 | 4 |
| 1 | 333 | 2 | 10 |
| 1 | 555 | 3 | 150 |

**Таблица 11.5.** Таблица артикулов: 2НФ базы данных (2-я часть)

| ИД продукта | Продукт | Цена |
|---|---|---|
| 111 | Лампа | 1 |
| 333 | Нож | 5 |
| 555 | Зонт | 50 |

Третья нормальная форма (3НФ) требует, чтобы все неключевые столбцы (атрибуты) были взаимно независимы и полностью зависели от первичного ключа. Зависимость существует, например, если значения одного столбца вычисляются по данным из других столбцов. В нашем примере для перевода в 3НФ таблицы заказов в ней надо исключить столбец *Всего* (его значения вычисляются по формуле *Цена×Количество*). Результат преобразования БД в 3НФ показан в табл. 11.6 и табл. 11.7.

**Таблица 11.6.** Таблица заказов: 3НФ базы данных (1-я часть)

| Заказчик | ИД продукта | Количество |
|---|---|---|
| 1 | 111 | 4 |
| 1 | 333 | 2 |
| 1 | 555 | 3 |

**Таблица 11.7.** Таблица артикулов: 3НФ базы данных (2-я часть)

| ИД продукта | Продукт | Цена |
|---|---|---|
| 111 | Лампа | 1 |
| 333 | Нож | 5 |
| 555 | Зонт | 50 |

Результатом проведения нормализации является оптимальная структура БД. В полученной БД имеется необходимое дублирование данных, но отсутствует избыточное.

**ПРИМЕЧАНИЕ**

На каждую из нормальных форм распространяется принцип вложенности: если БД находится в форме с номером $N$, то она находится и в форме с номером $N-1$.

## Расширение UML для моделирования баз данных

В настоящее время стандарт языка UML не имеет средств для моделирования баз данных. Наиболее известные подходы к решению этой проблемы предложены Р. Мюллером [11], корпорацией Rational [98, 70], Э. Нейбургом [12] и С. Амблером [21]. К сожалению, каждый из этих подходов использует свою нотацию моделирования, отличающуюся от нотаций других подходов. Очевидно, что унификация соответствующих обозначений — дело будущего. Мы же за основу расширения UML примем обозначения С. Амблера как наиболее простые и проработанные.

## Типы моделей данных

Известно, что при моделировании БД последовательно применяют три типа моделей:

- *Концептуальные модели данных*. Эти модели создают на первом шаге моделирования и используют для исследования понятий проблемной области с точки зрения заказчика. Основными элементами здесь являются бизнес-модели.
- *Логические модели данных*. Логические модели создают на втором шаге моделирования. Они фиксируют требования к системе с точки зрения разработчика и описывают логическую организацию системы с базой данных, реализующую эти требования (в терминах сущностей данных и отношений между сущностями). Основными элементами логических моделей являются диаграммы классов.
- *Физические модели данных*. Физические модели создают на третьем шаге моделирования. С их помощью проектируют внутреннюю схему базы данных, изображая таблицы данных, атрибуты (столбцы) таблиц и отношения между таблицами.

Тип модели должен быть обозначен или с помощью соответствующего стереотипа (табл. 11.8), или указан как текстовый комментарий в примечании UML. В случае физической модели разновидность механизма сохранения данных следует обозначить как один из стереотипов, перечисленных в табл. 11.9.

**Таблица 11.8.** Стереотипы для указания типа модели

| Стереотип | Тип модели |
|---|---|
| <<Концептуальная модель данных>> | Концептуальная модель данных |
| <<Логическая модель данных>> | Логическая модель данных |
| <<Физическая модель данных>> | Физическая модель данных |

**Таблица 11.9.** Стереотипы для различных устойчивых механизмов сохранения данных

| Стереотип | Разновидность механизма сохранения данных |
|---|---|
| <<Иерархическая БД>> | Иерархическая база данных |
| <<Сетевая БД>> | Сетевая база данных |
| <<Реляционная БД>> | Реляционная база данных |
| <<Объектно-ориентированная БД>> | Объектно-ориентированная база данных |
| <<Объектно-реляционная БД>> | Объектно-реляционная база данных |
| <<XML БД>> | База данных XML |

## Таблицы, сущности, представления и отношения

Для обозначения таблиц, сущностей и представлений используются прямоугольники классов. Эти прямоугольники имеют соответствующие стереотипы (табл. 11.10).

Приведем пример логической модели данных (рис. 11.7) и пример физической модели данных (рис. 11.8). Прямоугольники классов в концептуальных и логических моделях данных обозначают определения сущностей, для которых стереотип является дополнением. В физической же модели данных для реляционной БД предполагается, что любой прямоугольник класса без стереотипа — это таблица.

**Таблица 11.10.** Стереотипы для вершин в моделях данных

| Стереотип | Тип модели | Применение |
|---|---|---|
| <<Таблица>> | Физическая | Обычно в физической модели этот стереотип подразумевается по умолчанию |
| <<Ассоциативная таблица>> | Физическая | Применим к ассоциативным таблицам в физической модели для реляционной БД |
| <<Представление>> | Физическая | Применим при моделировании представления. Для вершины-представления указываются зависимости от таблиц-источников данных |
| <<Индекс>> | Физическая | Применим при моделировании индекса, который ускоряет доступ к таблице в реляционной БД. Принято указывать на зависимость индекса от индексируемого столбца таблицы |
| <<Сущность>> | Логическая, Концептуальная | Дополнительное обозначение, которое по умолчанию подразумевается типом модели |
| <<Хранимые процедуры>> | Физическая | Применим к классу, который содержит только сигнатуры операций для хранимых процедур БД |

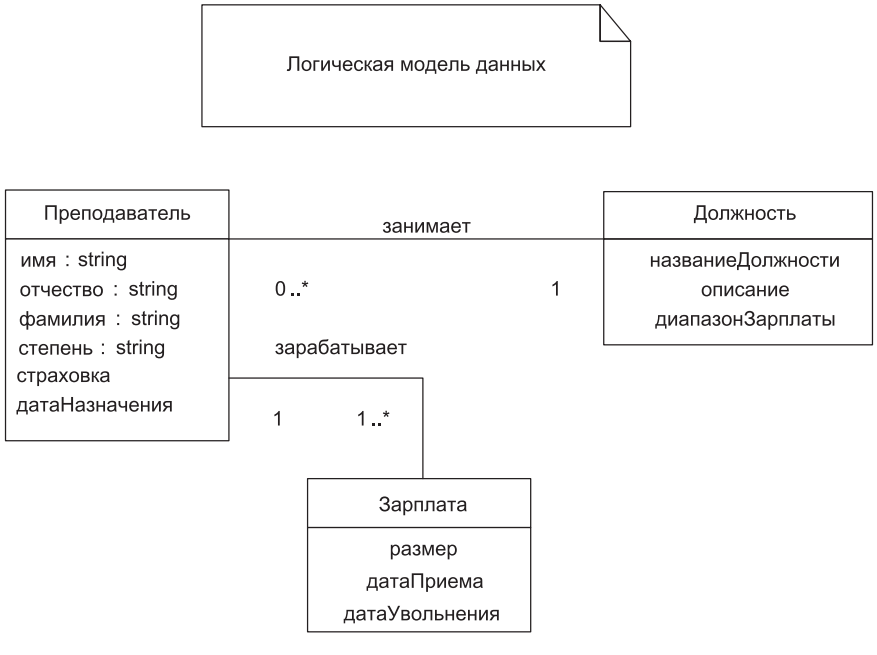


**Рис. 11.7.** Логическая модель данных

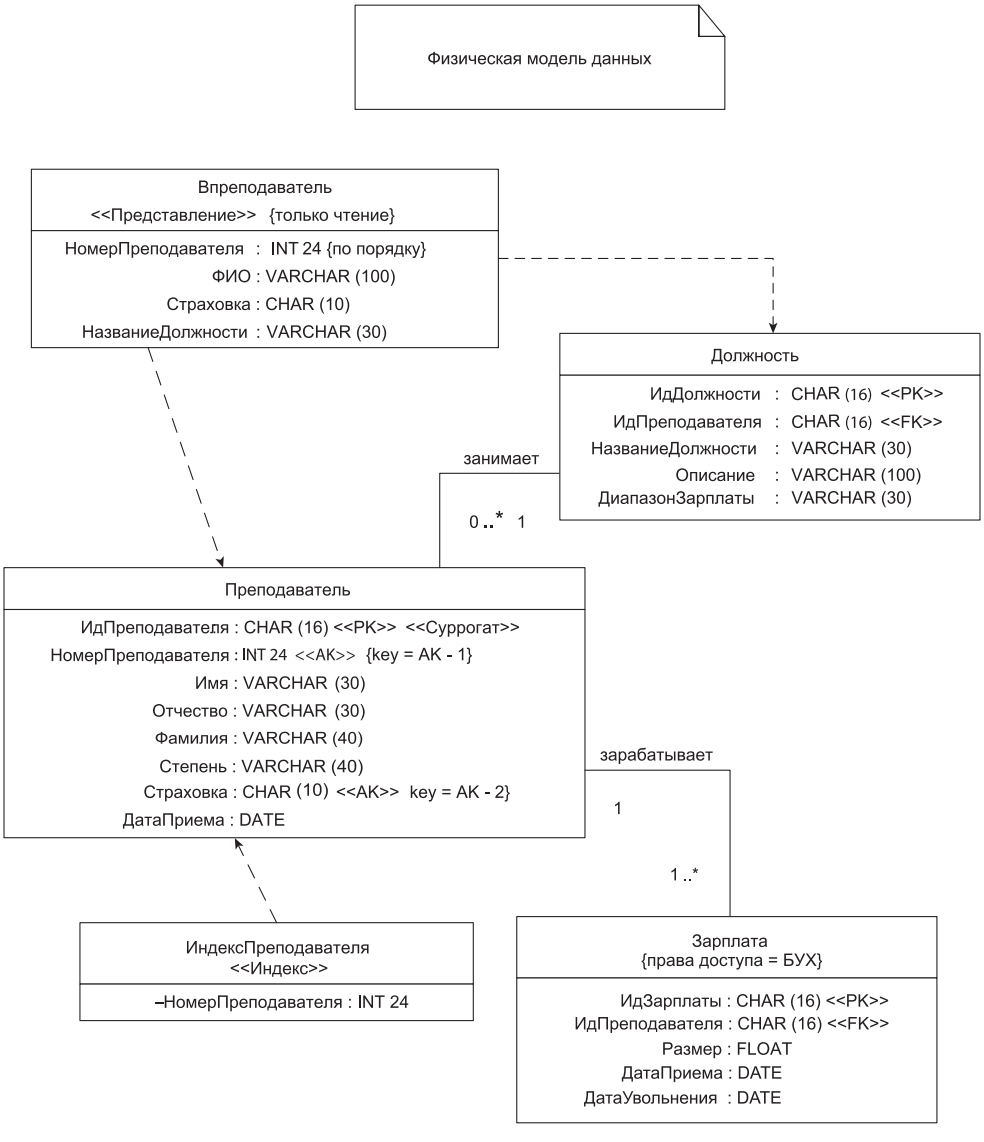


**Рис. 11.8.** Физическая модель данных

Отметим, что в физической модели (см. рис. 11.8) имеются три таблицы (Преподаватель, Зарплата, Должность), одно представление Впреподаватель и один индекс ИндексПреподавателя. Представление зависит от двух обычных таблиц. Индекс зависит от таблицы Преподаватель, в нем указано, что индексируется столбец таблицы НомерПреподавателя.

Атрибуты данных в концептуальных и логических моделях данных, так же как и столбцы (атрибуты) в физических моделях данных, моделируются с использованием

стандартного обозначения атрибута данных UML. Подразумевается, что атрибут сущности из логической модели автоматически преобразуется в соответствующий атрибут (столбец) таблицы из физической модели. Столбцы таблицы разделяют на ключевые или неключевые. В свою очередь, ключевой столбец может быть первичным ключом, внешним ключом или же комбинацией первичного и внешнего ключа. Ограничения на значения столбца указываются с помощью обычных ограничений UML. Подразумевается, что видимость столбцов всегда задается как публичная.

Отношения между вершинами моделей данных изображаются с помощью стандартных обозначений UML, используются разновидности отношений, применимые к диаграммам классов. Характеристика условий применения отношений приведена в табл. 11.11.

**Таблица 11.11.** Условия применения отношений в моделях данных

| Стереотип (название) | Визуальное представление | Тип модели | Условие применения |
|---|---|---|---|
| <<Наследник>> | Стрелка наследования | Любой | Одна сущность является наследником другой сущности |
| <<Агрегация>> | Стрелка с наконечником в форме полого ромба | Любой | Одна сущность является существующей самостоятельно частью другой сущности |
| <<Композиция>> | Стрелка с наконечником в форме закрашенного ромба | Любой | Одна сущность является частью другой сущности |
| <<Зависимость>> | Пунктирная линия с обычной стрелкой | Физическая | Указание зависимости представления или индекса от обычной таблицы |
| <<Identifying>> (Обязательная связь) | Стрелка композиции со стереотипом <<Identifying>> и мощностью 1 на обоих полюсах | Физическая | Связь между двумя таблицами, при которой дочерняя таблица не может существовать без родительской таблицы. В дочерней таблице должен присутствовать внешний ключ, являющийся первичным ключом родительской таблицы и определяющий связь между этими таблицами. Внешний ключ дочерней таблицы должен быть одновременно ее первичным ключом |
| <<Non-Identifying>> (Необязательная связь) | Линия ассоциации со стереотипом <<Non-Identifying>> и мощностью 0..1 хотя бы на одном полюсе | Физическая | Связь между двумя таблицами, при которой каждая таблица может существовать независимо от другой. В дочерней таблице должен присутствовать внешний ключ, являющийся первичным ключом родительской таблицы и определяющий связь между этими таблицами. Внешний ключ дочерней таблицы не должен являться ее первичным ключом |

## Ключи, ограничения, триггеры и хранимые процедуры

Напомним основные разновидности ключей:

- Первичный ключ — это столбец (или несколько столбцов), уникально идентифицирующий строку (запись) таблицы.
- Внешний ключ — это столбец таблицы, являющийся первичным ключом другой таблицы и обеспечивающий связь с этой таблицей. Внешний ключ может одновременно быть и первичным ключом собственной таблицы.

При формировании ключей следует учитывать следующие особенности:

- сущность может иметь несколько ключей-кандидатов, каждый из которых может быть составным;
- таблица может иметь первичный ключ и несколько альтернативных ключей, каждый из которых может быть составным;
- порядок, в котором столбцы появляются в ключах таблицы, может быть важен;
- традиционные модели данных обычно не позволяют определить, какой частью ключа является атрибут (столбец), а в документации эту информацию часто опускают.

Возможные стереотипы ключей перечислены в табл. 11.12.

**Таблица 11.12.** Стереотипы для ключей таблиц и сущностей

| Стереотип | Тип модели | Применение |
|---|---|---|
| <<CK>> | Концептуальная, Логическая | Указывает, что атрибут является для сущности частью ключа-кандидата |
| <<Уникальный идентификатор>> | Концептуальная, Логическая | Указывает, что атрибут является для сущности частью уникального идентификатора. Фактически это альтернатива к <<CK>> |
| <<PK>> | Физическая | Указывает, что столбец является для таблицы частью первичного ключа |
| <<AK>> | Физическая | Указывает, что столбец является для таблицы частью альтернативного ключа, известного также как вторичный ключ |
| <<Автосгенерированный>> | Физическая | Указывает, что столбец автоматически сгенерирован базой данных |
| <<Суррогат>> | Физическая | Указывает, что столбец является суррогатным ключом |
| <<Естественный>> | Любой | Указывает, что атрибут или столбец является частью естественного ключа |
| <<FK>> | Физическая | Указывает, что столбец является частью внешнего ключа, обеспечивающего связь с другой таблицей |

При детализации описания ключа можно использовать служебные идентификаторы, смысл которых поясняет табл. 11.13.

**Таблица 11.13.** Служебные идентификаторы для детализации описания ключей

| Идентификатор | Смысл | Примеры применения |
|---|---|---|
| key | Указывает, какому ключу-кандидату или альтернативному ключу принадлежит столбец (атрибут). Когда столбец является частью нескольких ключей, например является частью двух различных внешних ключей, тогда надо указать, к какому из них обращаетесь. Во втором примере столбец — это часть третьего альтернативного ключа | key = FK<br>key = AK-3 |
| order | Указывает порядок появления столбца в составном ключе. В примере столбец указан как четвертый элемент ключа | order = 4 |
| table | Указывает таблицу, к которой относится (обращается) внешний ключ. Это дополнительное обозначение, поскольку связь явно обозначается в самой модели | table = Клиент |

Приведем комплексный пример, иллюстрирующий указание в физической модели данных ключей, ограничений, триггеров и хранимых процедур (рис. 11.9).

Обсудим этот пример. Во-первых, здесь описаны следующие ключи:

- ИдЗаказа — первый элемент первичного ключа таблицы ЭлементЗаказа и первичный ключ таблицы Заказ. Кроме того, этот столбец является внешним ключом для связи таблицы ЭлементЗаказа с таблицей Заказ;
- столбец НаборЭлементовЗаказа — второй элемент первичного ключа таблицы ЭлементЗаказа;
- столбец ИдЗаказа — элемент нескольких ключей, поэтому нужно было указать соответствующую дополнительную информацию. Например, ИдЗаказа — второй элемент первого альтернативного ключа;
- столбец ИдЭлементаЗаказа — второй альтернативный ключ таблицы ЭлементЗаказа;
- столбец ИдЭлемента — первый элемент первого альтернативного ключа таблицы ЭлементЗаказа;
- столбец ИдЭлемента — является также внешним ключом к таблице Элемент;
- столбец ИдЗаказчика — внешний ключ к таблице Заказчик.

Во-вторых, здесь описаны следующие ограничения:

- для столбца ДатаЗаказа определено ограничение домена, указывая, что это должно быть позже 9-го июля 2003 года;
- для столбца ИдЗаказчика определено ограничение столбца, он не должен быть нулевым;
- для БД задано ограничение ссылочной целостности между двумя таблицами Заказ и ЭлементЗаказа, оно подписывает стрелку композиции. Видно, что при удалении заказа элементы заказа также должны быть удалены.

В-третьих, здесь показаны триггеры: используется обозначение для операции со стереотипом <<Триггер>>. Условия запуска процедур-триггеров отмечены ограничениями {после вставки} и {перед удалением}.

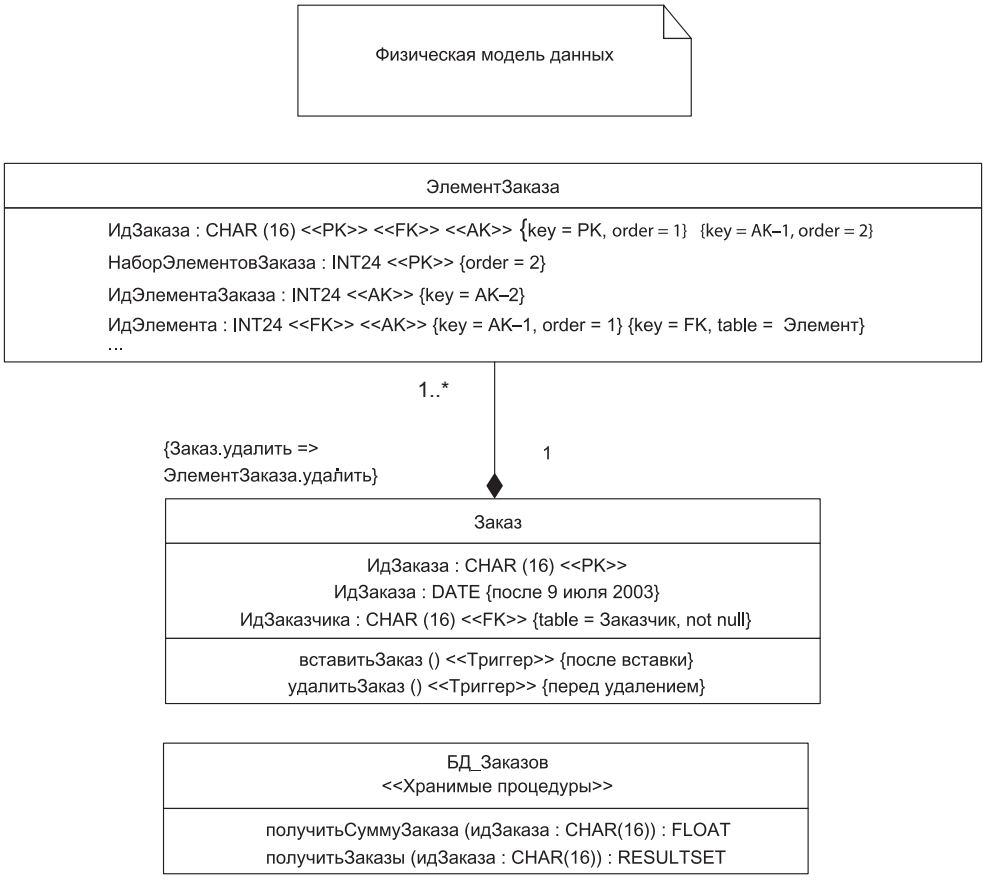


**Рис. 11.9.** Ключи, ограничения, триггеры и хранимые процедуры в физической модели данных

В-четвертых, на рис. 11.9 показаны хранимые процедуры: использован отдельный класс со стереотипом <<Хранимые процедуры>>. Этот класс перечисляет сигнатуры операций хранимых процедур с применением обозначений стандарта UML. Другой подход состоит в применении стереотипа <<Хранимая процедура>> к каждой сигнатуре отдельной операции. Отметим, имя этого класса должно совпадать с именем базы данных или с именем пакета в БД.

В заключение отметим, что в ранее приведенной физической модели (см. рис. 11.8) тоже были показаны ключи и ограничения. Например, было отмечено, что столбец ИдПреподавателя является первичным суррогатным ключом. На таблицу Зарплата наложено ограничение доступа: только сотрудникам из отдела БУХ разрешают обращаться к ее содержимому. Для представления Впреподаватель тоже введены ограничения: доступ разрешен только для чтения, а записи упорядочены по столбцу НомерПреподавателя.

## Особенности отображения атрибутов объектов и классов в реляционную базу данных

Между технологиями для разработки объектно-ориентированных программных систем с использованием БД, то есть между объектной и реляционной технологией, существует полное несоответствие. Для преодоления этого несоответствия необходимо разобраться в сути отображения объектов в реляционные базы данных и особенностях реализации этих отображений. Термин «отображение» будем применять для обозначения того, как объекты (классы) и их отношения отображаются в сохраняемые таблицы и отношения между ними в базе данных.

Начнем с атрибутов (элементов данных) класса. Атрибут может отобразиться в ноль или несколько столбцов в реляционной базе данных. При чем здесь ноль? Дело в том, что не все атрибуты надо запоминать в БД. Очевидно, что нет нужды сохранять неустойчивые атрибуты, то есть те, которые используются для временных вычислений. Например, объект класса Заказ может иметь атрибут средняяЦенаПокупки, которое необходимо в приложении, но не сохраняется в базе данных, потому что рассчитывается при необходимости. Кроме того, некоторые атрибуты объектов сами являются объектами, например объект класса Заказчик имеет в качестве атрибута объект класса Адрес. Этот факт отражается отношением между двумя классами, которое нужно отобразить; мало того, атрибуты самого класса Адрес тоже должны быть отображены. Важно отметить, что определение «атрибут отображается в ноль или несколько столбцов» по своей сути рекурсивно.

Самое простое отображение — это отображение отдельного атрибута в отдельный столбец. Оно становится предельно простым, когда их базовые типы совпадают, например, атрибут является числом, а столбец — числом с плавающей точкой, или тип атрибута — строка и тип столбца — строка.

Удобно считать, что классы прямо отображают в таблицы, в стиле «один-в-один», но это далеко не всегда возможно. За исключением очень простых баз данных, не удается получать классы в таблицах вида «один-к-одному»: вмешивается механизм наследования и необходимость его отображения. Однако, в качестве начального шага предпочтительно отображение одного класса в одну таблицу (изменения начальных отображений может потребовать настройка быстродействия).

Рассмотрим пример простого отображения логической модели, представленной диаграммой классов, в физическую модель данных (рис. 11.10). Пример иллюстрирует отображение атрибутов классов в столбцы таблиц БД. Например, атрибут датаВыполнения класса Заказ отображается в столбец ДатаВыполнения таблицы Заказ, а атрибут заказанноеКоличество класса ЭлементЗаказа отображается в столбец ЗаказанноеКоличество таблицы ЭлементЗаказа.

И все же в структуре классов и таблиц есть различия:

- В классе имеются два атрибута для фиксации налога (центрНалог, местнНалог), а в таблице только один столбец Налог. Очевидно, что при сохранении объекта значения двух атрибутов складываются и заносятся в столбец Налог таблицы. Когда же объект читается обратно в память, должны быть рассчитаны значения двух атрибутов (или должен использоваться ленивый подход инициализации, и значение каждого атрибута должно рассчитываться при первом обращении к нему). Подобные различия — хороший индикатор того, что структура таблицы

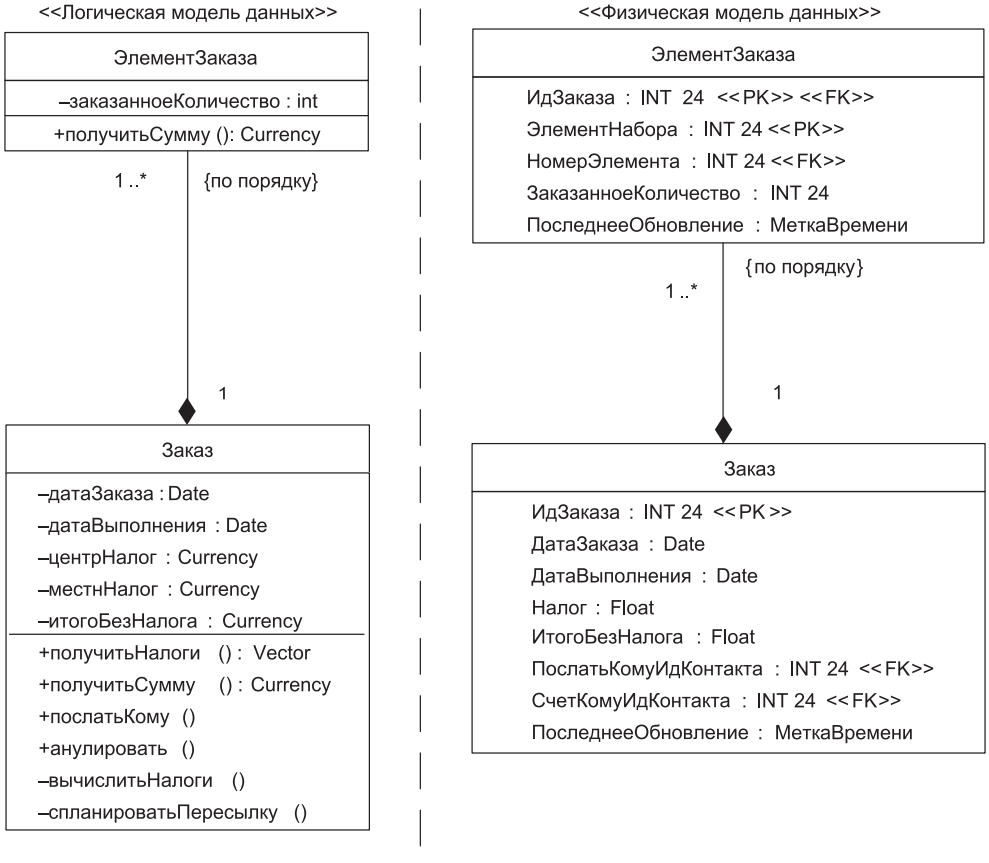


**Рис. 11.10.** Пример простого отображения атрибутов классов

должна быть реорганизована путем разбиения одного столбца для налога на два (структуры всех таблиц и связей между ними образуют схему базы данных).

- В отличие от классов в таблицах указаны ключи. Строки в таблицах однозначно определены первичными ключами, а отношения между строками устанавливаются с помощью внешних ключей. С другой стороны, отношения между классами реализуются через ссылки на классы, а не через внешние ключи. Как следствие, для полного сохранения в заказе объектов и их отношений объекты должны знать о значениях ключей, используемых в базе данных, чтобы идентифицировать их. Эту дополнительную информацию называют *теневой* (*скрытой*) *информацией*.
- В каждой модели используются различные типы данных. Атрибут итогоБезНалога класса Заказ имеет тип Currency, тогда как столбец ИтогоБезНалога таблицы Заказ имеет тип Float. При реализации этого отображения придется обеспечить двустороннее приведение значений (без потери информации).

## Теневая (скрытая) информация

Теневую информацию образуют любые данные, которые требуются объектам для их сохранения в БД и для обеспечения значений предметной области (иначе области бизнеса). Она обычно включает:

- информацию первичного ключа (особенно когда первичный ключ является ключом-суррогатом, не имеющим никакого смысла с точки зрения бизнеса);
- информацию для управления параллельным доступом (например, метки времени или инкрементные счетчики);
- булев флажок «Устойчив», указывающий наличие объекта в БД (равен *true*, если объект имеется в базе данных). С помощью флажка организуется выбор одного из двух операторов SQL для сохранения объекта (оператора обновления при истинном значении флажка и оператора вставки при ложном значении);
- информацию номеров версий.

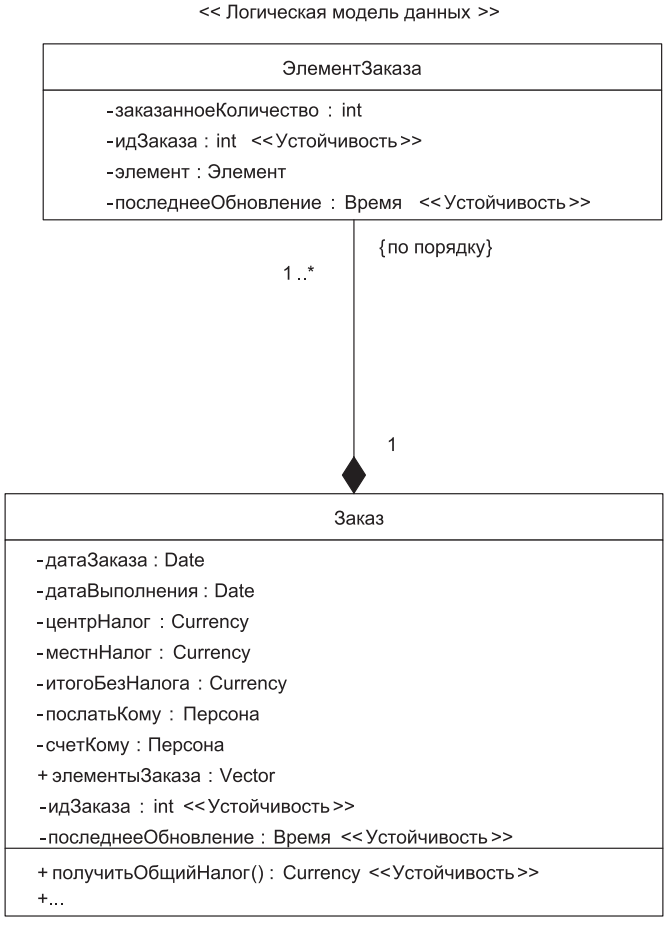


**Рис. 11.11.** Включение в диаграмму класса теневой информации

Например, в обсуждаемой физической модели для системы заказов (см. рис. 11.10) показано, что таблица Заказ использует столбец ИдЗаказа как первичный ключ. Кроме того, в этой таблице есть столбец ПоследнееОбновление, применяемый для управления параллельным доступом. Аналогов этих столбцов класс Заказ не имеет. Для правильного сохранения объекта класс Заказ должен реализовать теневые атрибуты, которые поддерживают эти величины.

Введем более детальную проектную модель для классов ЭлементЗаказа и Заказ (рис. 11.11). Здесь имеется целый ряд дополнений:

- новая диаграмма показывает теневые атрибуты, которые требуются классам для правильного сохранения в БД. Теневые атрибуты должны иметь видимость реализации и помечены стереотипом <<Устойчивость>>;
- показаны атрибуты «*строительных лесов*», требуемые для реализации отношений двух классов. Атрибуты «*строительных лесов*», такие как вектор элементыЗаказа в классе Заказ, также должны иметь видимость реализации;
- в класс Заказ добавлена операция получитьОбщийНалог(), обеспечивающая вычисление значения, требуемого для столбца Налог таблицы Заказ. Эта операция моделирует виртуальный атрибут Налог класса Заказ.

На диаграммах классов UML теневую информацию и «*строительные леса*» обычно не показывают. При этом полагают, что среда разработки ПО генерирует их автоматически.

## Метаданные отображения

Метаданные — это информация о данных. Метаданные документируют отображение классов в таблицы в простой текстовой форме. В табл. 11.14 изображены метаданные, описывающие отображение атрибутов, требуемое для сохранения классов Заказ и ЭлементЗаказа (см. рис. 11.11) в базе данных.

**Таблица 11.14.** Метаданные, описывающие отображение атрибутов

| Атрибут | Столбец |
|---|---|
| Заказ.идЗаказа | Заказ.ИдЗаказа |
| Заказ.датаЗаказа | Заказ.ДатаЗаказа |
| Заказ.датаВыполнения | Заказ.ДатаВыполнения |
| Заказ.получитьОбщийНалог() | Заказ.Налог |
| Заказ.итогоБезНалога | Заказ.ИтогоБезНалога |
| Заказ.послатьКому.идПерсоны | Заказ.ПослатьКомуИдКонтакта |
| Заказ.счетКому.идПерсоны | Заказ.СчетКомуИдКонтакта |
| Заказ.последнееОбновление | Заказ.ПоследнееОбновление |
| ЭлементЗаказа.идЗаказа | ЭлементЗаказа.ИдЗаказа |
| Заказ.элементыЗаказа.indexOf(элементЗаказа) | ЭлементЗаказа.ЭлементНабора |
| ЭлементЗаказа.элемент.номер | ЭлементЗаказа.НомерЭлемента |
| ЭлементЗаказа.заказанноеКоличество | ЭлементЗаказа.ЗаказанноеКоличество |
| ЭлементЗаказа.последнееОбновление | ЭлементЗаказа.ПоследнееОбновление |

В этой таблице обозначено:

- Заказ.датаЗаказа — атрибут датаЗаказа класса Заказ;
- Заказ.ДатаЗаказа — столбец ДатаЗаказа таблицы Заказ;
- Заказ.получитьОбщийНалог() — операция получитьОбщийНалог() класса Заказ;
- Заказ.счетКому.идПерсоны — атрибут идПерсоны экземпляра класса Персона, на который ссылается атрибут Заказ.счетКому;
- Заказ.элементыЗаказа.indexOf(элементЗаказа) — операция, возвращающая позицию указанного элемента заказа в векторе Заказ.элементыЗаказа.

Метаданные таблицы явно демонстрируют главное несоответствие между объектной и реляционной технологией: классы реализуют и поведение, и данные, а таблицы реляционной БД сохраняют только данные. В силу этого, при отображении атрибутов приходится также отображать в столбцы операции типа получитьОбщийНалог() и indexOf(). Иными словами, при этом операции «превращаются» в столбцы-данные. Достаточно часто требуется отображать в столбец две операции, представляющие отдельный атрибут: одну операцию для установки значения, например установитьИмя(), и одну операцию, возвращающую значение, например получитьИмя().

Другое действие происходит при отображении атрибута класса в ключевой столбец: данные «превращаются» в связи. Это происходит потому, что ключи реализуют отношения в реляционных базах данных.

## Отображение атрибутов уровня класса

Иногда класс реализует атрибут, значение которого применимо ко всем его экземплярам, а не только к отдельному объекту. Представим себе класс Заказчик, в котором есть атрибут номерСледующегоЗаказчика. Значение этого атрибута назначается новому объекту-заказчику. Для сохранения уникальности номеров объектов атрибут номерСледующегоЗаказчика должен работать только на уровне класса. Рассмотрим различные методики отображения атрибута, действующего только на уровне класса.

- *Каждый атрибут отображается в таблицу из одного столбца и одной строки*. Преимуществом такого похода является простота и быстрый доступ. Недостаток: возможно получение большого количества маленьких таблиц.
- *Атрибуты каждого класса отображаются в отдельную таблицу из нескольких столбцов и одной строки*. В этом случае для каждого нового атрибута уровня класса заводится новый столбец. Преимущество: простота и быстрый доступ. Недостаток: возможно получение большого количества маленьких таблиц (хотя и меньшего, чем в первом подходе).
- *Атрибуты всех классов отображаются в одну таблицу из нескольких столбцов и одной строки*. Эта таблица содержала бы отдельный столбец для каждого атрибута каждого класса. Преимущество: требуется минимальное число таблиц. Недостаток: потенциальные проблемы с параллелизмом, если многие классы одновременно обратятся к данным. Возможное решение состоит в том, чтобы ввести две таблицы: одну для атрибутов, которые доступны только для чтения, другую — для атрибутов, доступных по чтению-записи.

- *Атрибуты всех классов отображаются в две многострочные таблицы.* Одна таблица хранит все переменные классов, другая — все константы классов. Каждому атрибуту уровня класса отводится своя строка, содержащая три столбца: для имени класса (как элемент первичного ключа), для имени атрибута (как другой элемент первичного ключа) и для значения атрибута. Преимущества: вводится минимальное число таблиц; уменьшаются проблемы параллелизма. Недостаток: требуются разнообразные преобразования типов данных.

## Отображение деревьев наследования в реляционную базу данных

С одной стороны, деревья наследования широко применяются в объектно-ориентированном ПО. С другой стороны, реляционные базы данных не поддерживают механизм наследования. В силу этого возникает задача отображения дерева наследования классов в схему данных реляционной БД. Суть такого отображения состоит в применении специальных приемов сохранения унаследованных атрибутов объектов (при их размещении в реляционной БД). Рассмотрим известные методики для отображения наследования.

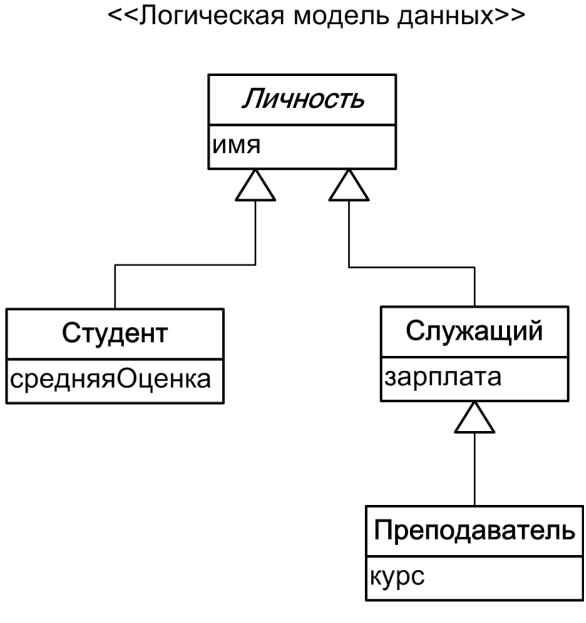


**Рис. 11.12.** Отображаемое дерево наследования классов

Отображать будем простую иерархию классов, включающую один абстрактный класс *Личность*, у которого имеются два конкретных класса-наследника Студент и Служащий (рис. 11.12). Напомним, что имя абстрактного класса записывается курсивом. Кроме того, примем, что у класса Служащий также есть наследник: класс

Преподаватель. Для упрощения можно считать, что каждый класс располагает только одним собственным (не унаследованным) атрибутом.

## Отображение дерева наследования в единственную таблицу

По этой методике атрибуты всех классов сохраняются в одной таблице. Например, результат отображения классов *Личность*, Студент и Служащий может быть представлен в виде таблицы Личность (рис. 11.13). Отметим, что правила хорошего тона требуют: в качестве имени таблицы следует использовать имя корневого класса дерева наследования.

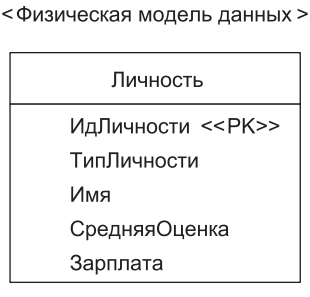


**Рис. 11.13.** Отображение дерева наследования в единственную таблицу

Видим, что в таблицу добавлены два служебных столбца — ИдЛичности и ТипЛичности. Первый столбец рассматривается как первичный ключ таблицы (суррогатный ключ), он помечен стереотипом <<PK>>, а второй столбец содержит код, указывающий, является ли личность студентом, служащим или, возможно, и тем и другим.

Столбец ТипЛичности обязан определить тип объекта, сохраняемого в определенной строке таблицы. Например, значение «*служ*» означало бы, что личность является служащим, «*студ*» означало бы студента, а «*оба*» означало бы оба типа. Хотя этот подход прост, он может отказать по мере роста числа типов и их комбинаций. Например, при необходимости учета преподавателя придется добавить значение «*прел*». Теперь значение «*оба*», представляя только два типа, становится нелепостью. Кроме того, может потребоваться дополнительная комбинация с привлечением преподавателя, например, кто-то может быть и преподавателем, и студентом, так что понадобится код для этой комбинации. Более универсален подход, в котором столбец типа личности заменяется столбцами для булевых переменных этоСтудент, этоСлужащий и этоПреподаватель (рис. 11.14).

Здесь демонстрируется результат отображения всех четырех классов из дерева наследования.

*Достоинства методики отображения:*

- простота выполняемых действий;
- легкость добавления новых классов — достаточно добавлять новые столбцы для дополнительных атрибутов (данных);

- поддержка полиморфизма обеспечивается простым изменением типа строки;
- быстрый доступ к данным, потому что данные находятся в одной таблице;

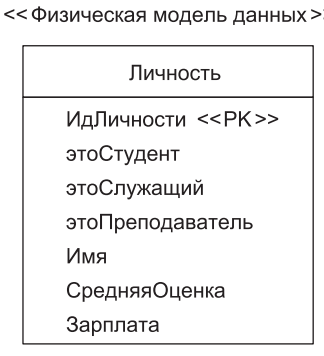


**Рис. 11.14.** Отображение дерева наследования в таблицу с булевыми переменными

- чрезвычайно проста генерация отчетов БД, так как все данные находятся в одной таблице.

*Недостатки методики отображения:*

- возрастает сцепление в дереве классов, потому что все классы сцеплены с одной и той же самой таблицей. Изменение в одном классе может затронуть таблицу, которая, в свою очередь, может затронуть другие классы дерева наследования;
- при существенном перекрытии между типами индикация типа усложняется;
- при больших деревьях наследования размеры таблицы растут очень быстро;
- пространство под базу данных расходуется не рационально.

*Область применения методики отображения:* целесообразно использовать для простых и (или) плоских деревьев наследования, в которых отсутствует или мало перекрытие между типами дерева.

## Отображение каждого конкретного класса в отдельную таблицу

По этой методике таблица создается для каждого конкретного класса. Каждая таблица включает и собственные атрибуты, реализованные классом, и его унаследованные атрибуты. Соответствующая физическая модель данных для нашего дерева наследования включает три таблицы (рис. 11.15).

Здесь каждому из конкретных классов Студент, Служащий и Преподаватель соответствует таблица, но нет таблицы для абстрактного класса *Личность*. Причина понятна, ведь на основе абстрактного класса объекты не создаются. Каждой таблице назначен ее собственный первичный ключ: ИдСтудента, ИдСлужащего и ИдПреподавателя соответственно.

*Достоинства методики отображения:*

- быстрый доступ к данным отдельного объекта;

- упрощается генерация отчетов БД, так как все требуемые данные об объекте (классе) сохраняются только в одной таблице.

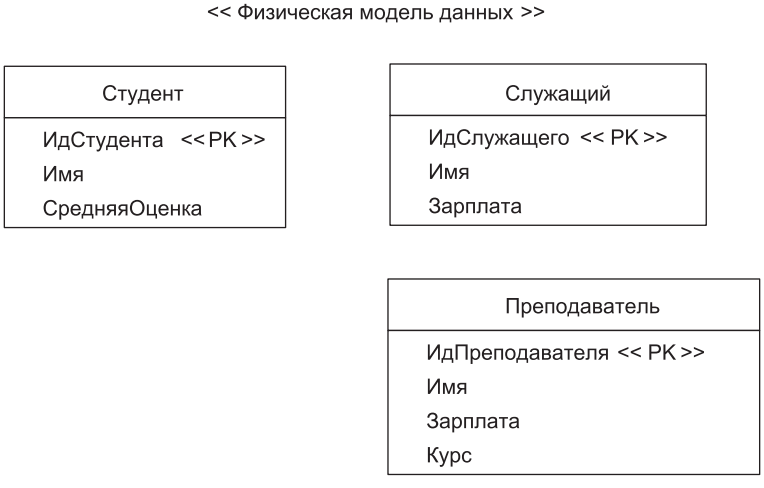


**Рис. 11.15.** Отображение конкретных классов в таблицы

*Недостатки методики отображения:*

- при изменении класса нужно изменить его таблицу и таблицу любого из его подклассов. Например, при добавлении в класс *Личность* атрибутов роста и веса следует добавить столбцы к таблицам Студент, Служащий и Преподаватель. Всякий раз, когда объект изменяет свою роль (возможно, студент поступил на службу), приходится копировать данные в соответствующую таблицу и назначать им новое значение столбца-идентификатора;
- трудно поддержать множественные роли и обеспечивать при этом целостность данных. Например, достаточно трудно решить вопрос о сохранении имени личности, которая является и студентом и служащим.

*Область применения методики отображения:* целесообразно использовать при редком изменении типов и (или) перекрытии между типами.

## Отображение каждого класса в отдельную таблицу

По этой методике таблица создается для каждого класса. Таблица включает не только столбцы для бизнес-атрибутов, но и столбцы для сохранения теневой информации. Пример физической модели данных для нашего дерева наследования включает четыре таблицы (рис. 11.16). Обратим внимание на связи между таблицами.

Данные для класса Студент теперь сохраняются в двух таблицах (Личность и Студент). Для получения этих данных надо обратиться к двум таблицам, то есть выполнить два отдельных чтения, по одному на каждую таблицу. Соответственно данные класса Преподаватель сохраняются в трех таблицах (Личность, Служащий и Преподаватель).

Обсудим применение ключей. Отметим, что ИдЛичности используется как первичный ключ во всех таблицах. Для таблиц Студент, Служащий и Преподаватель столбец ИдЛичности является и первичным ключом, и внешним ключом. В случае Студента ИдЛичности, его первичный ключ, как внешний ключ он используется

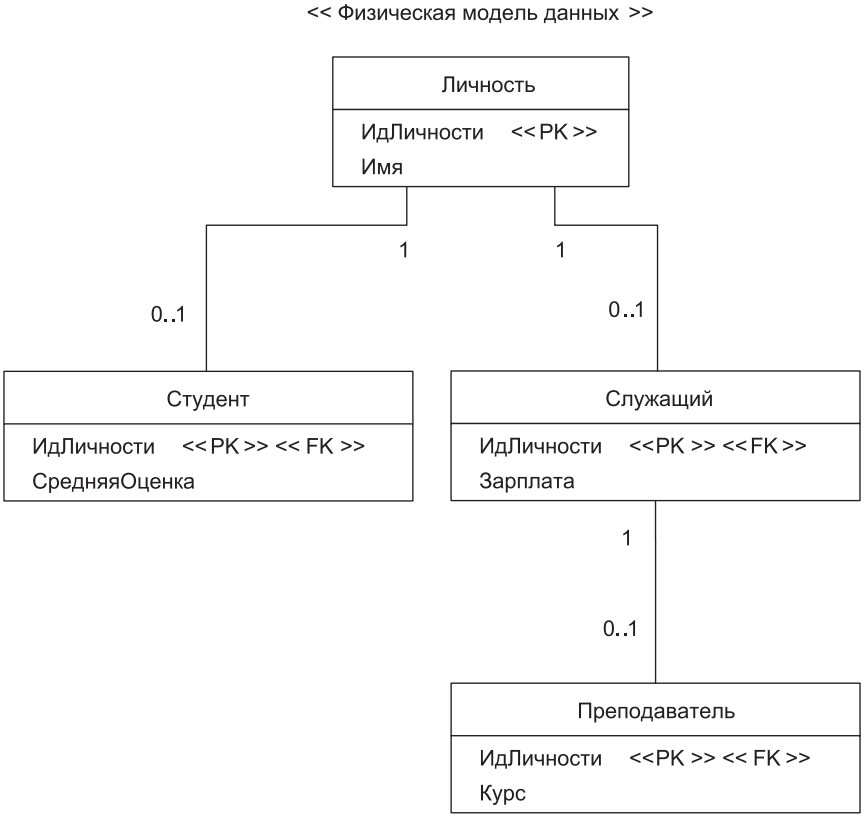


**Рис. 11.16.** Отображение каждого класса в отдельную таблицу

для поддержки отношения с таблицей Личность. Это обозначено применением двух стереотипов, <<PK>> и <<FK>>.

Весьма полезен такой прием, как добавление в таблицу Личность булевых столбцов (или столбцов типа личности), указывающих соответствующие подтипы личности. Несмотря на дополнительные затраты, это упрощает некоторые типы запросов к БД.

Более универсален подход, основанный на применении представлений — виртуальных таблиц, использующих в качестве источников данных сразу несколько обычных таблиц.

*Достоинства методики отображения*:

- простота понимания, в силу взаимно однозначного отображения;

- хорошая поддержка полиморфизма, так как для каждого типа (класса) имеются записи в соответствующих таблицах;
- легкость изменения суперклассов и добавления новых подклассов, поскольку надо лишь изменить (добавить) одну таблицу;
- рост объема данных прямо пропорционален росту числа объектов.

*Недостатки методики отображения:*

- в базе данных образуется много таблиц — по одной на каждый класс, плюс таблицы для поддержки отношений;
- возрастает время чтения и записи данных, поскольку требуется доступ к множеству таблиц. Эту проблему можно минимизировать при разумной организации базы данных, например при размещении разных таблиц в дереве наследования классов на различных физических дисках дисковода (предполагается, что головки дисковода работают независимо);
- усложняется генерация отчетов из базы данных (если вы не добавляете представления, моделирующие желаемые таблицы).

*Область применения методики отображения:* целесообразно использовать при существенном перекрытии между типами или при частом изменении типов.

## Отображение классов в универсальную табличную структуру

Четвертая методика для отображения дерева наследования в реляционную базу данных использует универсальный подход к отображению классов. Она соответствует подходу, управляемому метаданными. Этот подход не привязан к механизму наследования, он поддерживает все формы отображения: отображение атрибутов и самых разных отношений между классами и объектами. Обсудим универсальную схему базы данных, которая обеспечивает как сохранение атрибутов, так и деревьев наследования (рис. 11.17).

Значение отдельного атрибута сохраняется в таблице Значение, поэтому для сохранения объекта с семью обычными атрибутами в таблице должны присутствовать семь записей, по одной на каждый атрибут. Столбец Значение.ИдОбъекта сохраняет уникальный идентификатор для определенного объекта. Таблица ТипАтрибута содержит строки (записи) для основных типов данных: Data, String, Деньги, Integer и т. д. Эта информация требуется для преобразования значения объектного атрибута в тип Varchar при сохранении в некоторой позиции столбца Значение.Значение.

Рассмотрим пример отображения в эту схему отдельного класса. Для сохранения класса ЭлементЗаказа (см. рис. 11.11) в таблице Значение должны быть четыре записи:

- одна строка для сохранения количества заказанных элементов заказанноеКоличество;
- вторая строка для сохранения значения идЗаказа;
- третья строка для сохранения значения элемент, которое описывает заказываемый элемент;

- четвертая строка для сохранения значения последнееОбновление, которое указывает время последнего обновления информации об элементе.

Таблица Класс включила бы одну строку для класса ЭлементЗаказа, а таблица Атрибут — четыре строки для атрибутов класса, сохраняемых в базе данных (по одной строке на каждый атрибут).

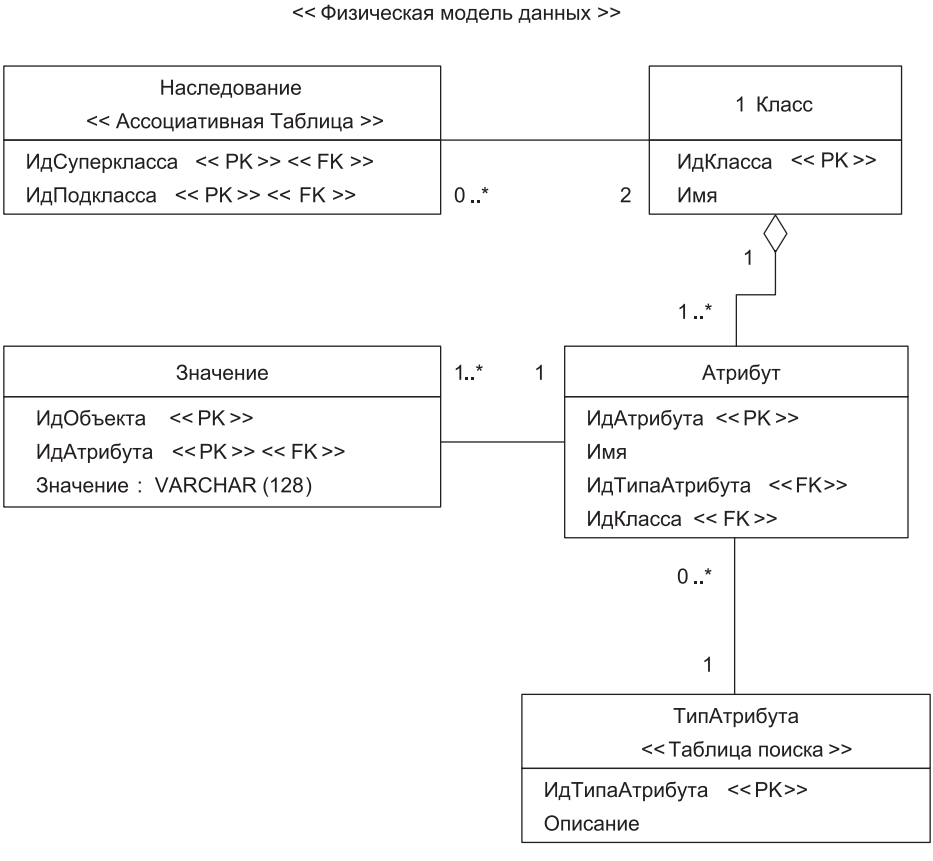


**Рис. 11.17.** Универсальная схема базы данных для сохранения объектов

Обсудим еще один пример — пример отображения в эту схему дерева наследования между Личностью и Студентом (см. рис. 11.12). Отображение наследования обеспечивает таблица Наследование совместно с таблицей Класс. Каждый класс должен быть представлен строкой в таблице Класс. Кроме того, должна быть строка в таблице Наследование, причем значение ее столбца Наследование.ИдСуперкласса должно ссылаться на строку в таблице Класс, представляющую Личность, а значение столбца Наследование.ИдПодкласса должно ссылаться на другую строку в таблице Класс, представляющую Студента. Чтобы отображать остальную часть дерева наследования, в таблице Наследование потребуется по одной строке для каждого отношения наследования.

*Достоинства методики отображения*:

- очень эффективна, если доступ к базе данных реализуется отдельным инкапсулированным модулем-пакетом;
- может быть расширена для обеспечения отображения других отношений между объектами;
- дает возможность быстрого изменения способа сохранения объектов — достаточно модифицировать метаданные, сохраненные в таблицах Класс, Наследование, Атрибут и ТипАтрибута.

*Недостатки методики отображения:*

- очень продвинутая методика, которая может быть трудна в реализации;
- работает только для малых объемов данных, поскольку для создания отдельного объекта приходится обращаться ко многим строкам базы данных;
- для поддержки метаданных требует создания отдельного приложения для администрирования;
- усложняется генерация отчетов, так как для получения данных о единственном объекте потребуется обращение к нескольким строкам базы данных.

*Область применения методики отображения:* целесообразно использовать в сложных приложениях, которые работают с малыми объемами данных. Можно применять в приложениях со специфичным доступом к данным, например, если данные предварительно загружаются в кэш-память.

## Отображение множественного наследования

При отображении множественного наследования можно использовать любую из трех рассмотренных ранее методик.

Положим, логическая модель задает дерево множественного наследования в семье (рис. 11.18).

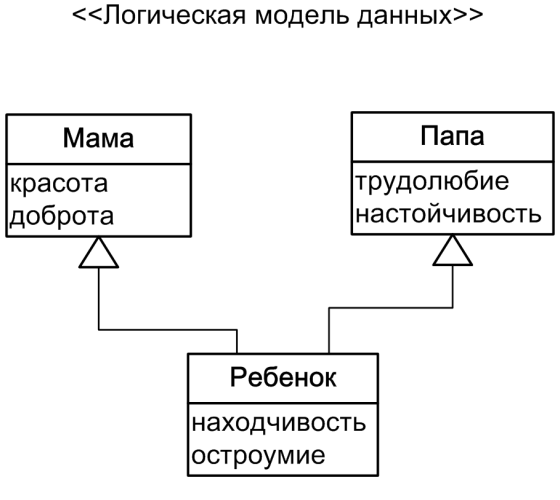


**Рис. 11.18.** Дерево множественного наследования в семье

Возможные варианты отображения этого дерева показаны на рис. 11.19–11.21.

Наибольшие трудности при отображении дерева в единственную таблицу возникли при выборе разумного имени таблицы, в этом случае мы остановились на имени Семья.

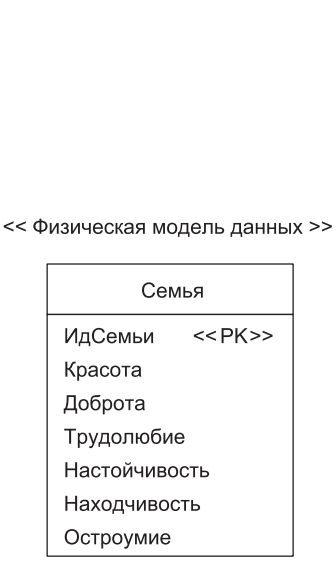


**Рис. 11.19.** Отображение дерева множественного наследования в одну таблицу

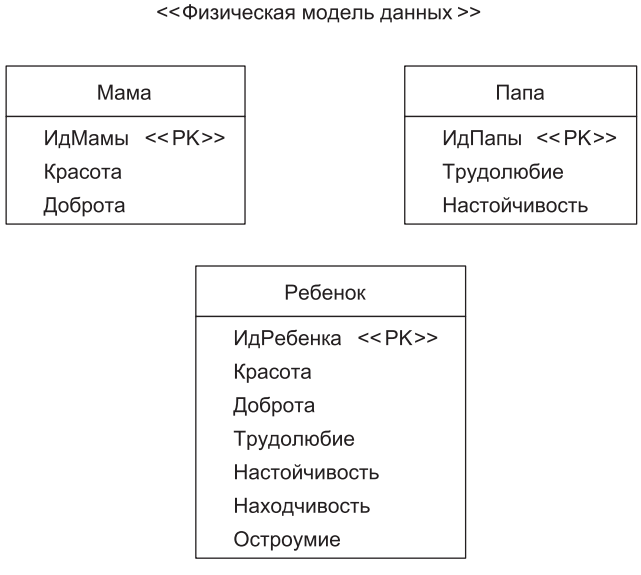


**Рис. 11.20.** Отображение конкретных классов при множественном наследовании

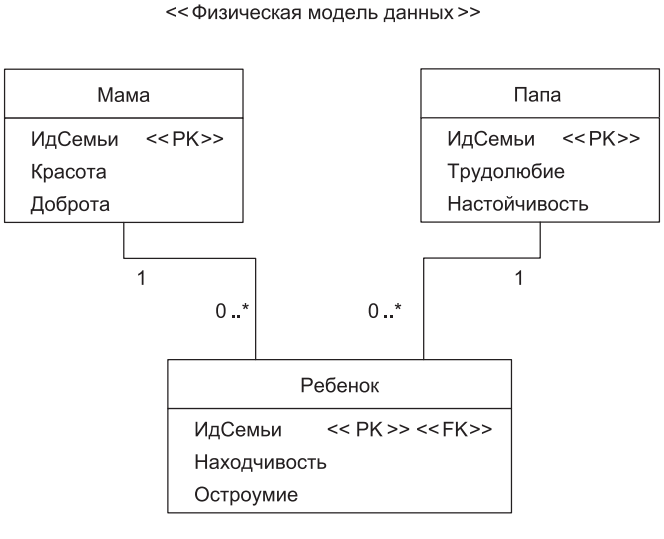


**Рис. 11.21.** Отображение каждого класса при множественном наследовании

# Объекты и базы данных: классификация и реализация отношений

Основной разновидностью отношений между объектами (классами), которые нужно отображать в схему БД, являются связи (ассоциации). Для иллюстрации обсуждаемых отношений используем простую логическую модель данных (рис. 11.22). Обратим внимание на тот факт, что атрибут курсы класса Преподаватель реализован как объект-контейнер типа HashSet и обеспечивает размещение внутри себя нескольких элементов. Аналогично реализованы атрибуты преподаватели в классах Кафедра и Курс.

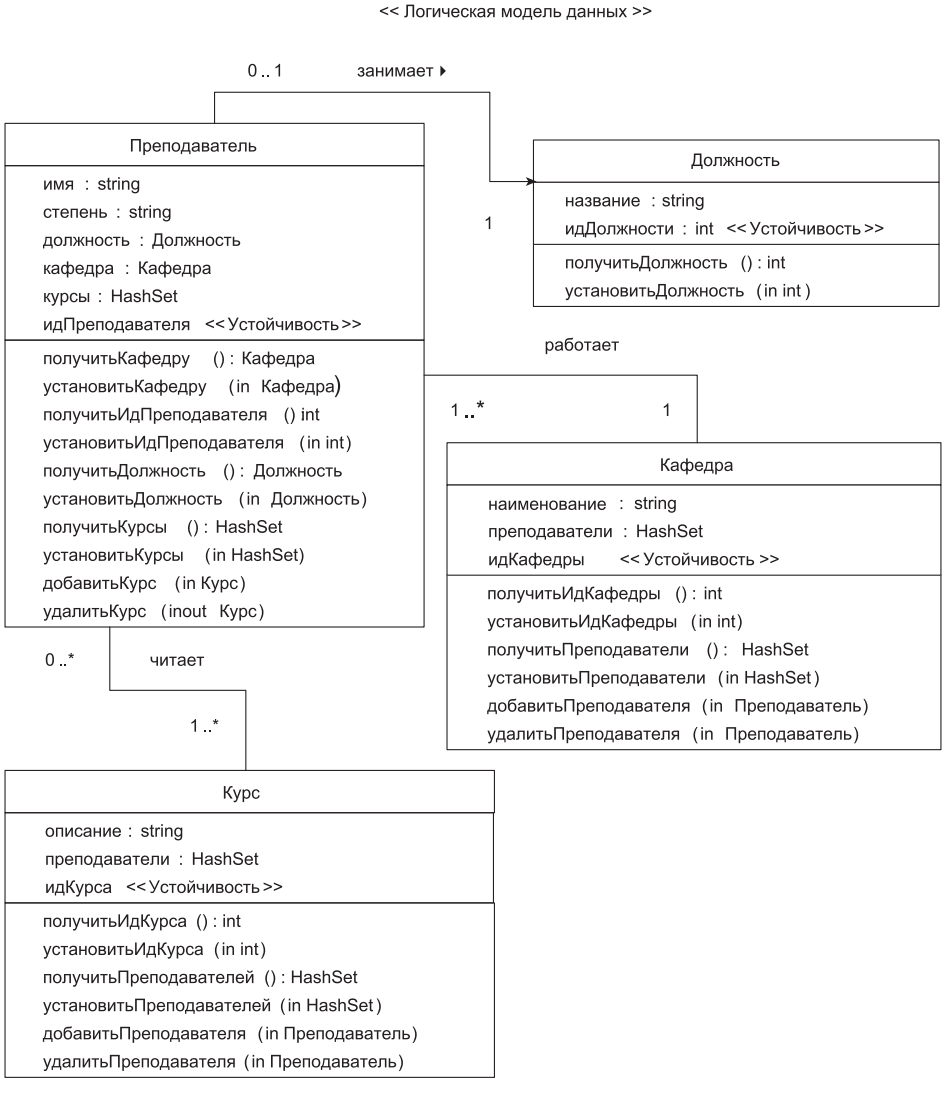


**Рис. 11.22.** Отношения между классами

С точки зрения отображения классифицируем отношения объектов по двум основаниям: множественности и направленности. По множественности различают три вида отношений:

- *взаимно-однозначные отношения «один-к-одному»*. Это отношения, где максимальное значение мощности равно единице. В нашей модели (рис. 11.22) таким является отношение занимает, существующее между Преподавателем и Должностью. Преподаватель занимает одну и только одну должность, а Должность занимается только одним преподавателем (некоторые должности остаются незанятыми);
- *отношения «один-ко-многим»*. В этих отношениях мощность на одном полюсе равна единице, а на другом — больше единицы. Такое отношение работает между Преподавателем и Кафедрой. Преподаватель работает на одной кафедре, а любая кафедра имеет одного или более работающих преподавателей;
- *отношения «многие-ко-многим»*. В этих отношениях максимальное значение мощности на обоих полюсах больше единицы, примером может служить отношение читает между Преподавателем и Курсом. Преподаватель читает один или несколько курсов, и каждый курс читается «нулем» или более преподавателей.

По направленности выделяют два вида отношений:

- *однонаправленные отношения*. Однонаправленные отношения имеют место, когда один объект знает об отношении с другим объектом, а другой объект не знает о первом объекте. Примером является отношение занимает между Преподавателем и Должностью (рис. 11.22), обозначенное обычной стрелкой. Объекты класса Преподаватель знают о *занимаемой* должности, но объекты класса Должность не знают, какой преподаватель их занимает. Однонаправленные отношения реализовать проще, чем двунаправленные отношения;
- *двунаправленные отношения*. Двунаправленные отношения существуют, когда объекты на обоих полюсах отношений знают друг о друге. Например, отношение работает между Преподавателем и Кафедрой является двунаправленным. Объекты класса Преподаватель знают, на какой кафедре они работают, а объекты класса Кафедра знают, какие преподаватели работают на них.

Между объектами возможны все шесть комбинаций отношений. Реляционные базы данных, напротив, не поддерживают однонаправленных отношений. Все отношения между таблицами являются двунаправленными.

## Реализация отношений между объектами

Отношение между объектами реализуется набором, состоящим из ссылок на объекты и операций. При единичной мощности (0.. 1, или 1) отношение реализуется ссылкой на объект, операцией-получателем и операцией-установщиком. Например, отношение «преподаватель работает на единственной кафедре» (см. рис. 11.22) реализуется классом Преподаватель *с* помощью набора, в состав которого входят:

- атрибут кафедра;
- операция получитьКафедру(), она возвращает значение атрибута кафедра;
- операция установитьКафедру(), она устанавливает значение атрибута кафедра.

Атрибуты и операции, требуемые для реализации отношения, обычно называют «*строительными лесами*».

При множественной мощности (*, или 0..*, или 1..*) отношение реализуется набором, включающим атрибут-контейнер (типа «коллекция» в языке Java) и операции для управления этим контейнером. Например, класс Кафедра реализует отношение «на кафедре работают преподаватели» с помощью набора, который включает:

- атрибут-контейнер преподаватели типа HashSet;
- операцию получитьПреподаватели() для получения значения атрибута преподаватели;
- операцию установитьПреподаватели() для установки значения атрибута преподаватели;
- операцию добавитьПреподавателя() для добавления преподавателя в атрибут-контейнер;
- операцию удалитьПреподавателя() для удаления преподавателя из атрибута-контейнера.

Когда отношение однонаправленное, набор реализуется только тем объектом, который знает о другом объекте. Например, в однонаправленном отношении между Преподавателем и Должностью только класс Преподаватель реализует набор для обеспечения ассоциации. Двунаправленные ассоциации, с другой стороны, реализуются обоими классами, как это сделано для отношения читает (вида «многие-ко-многим») между Преподавателем и Курсом.

## Реализация отношений в реляционных базах данных

Для иллюстрации отношений между таблицами используем простую физическую модель данных (рис. 11.23). Эта модель соответствует упрощенному варианту логической модели (см. рис. 11.22), в котором оставлено только по одному бизнес-атрибуту на класс. (На самом деле ситуация более тонкая, но пока оставим все нюансы в стороне.)

Отношения в реляционных базах данных поддерживаются с помощью внешних ключей. Внешний ключ — это столбец данных, который появляется в одной таблице и совпадает с первичным ключом (или его частью) другой таблицы. При отношении «один-к-одному» внешний ключ должен быть реализован одной из таблиц. Например, таблица Должность для реализации ассоциации включает столбец ИдПреподавателя, внешний ключ к таблице Преподавателя. Впрочем, вместо этого можно было реализовать столбец ИдДолжности в таблице Преподаватель.

Для обеспечения отношения «один-ко-многим» надо поместить во «многие таблицы» внешний ключ к «одной таблице». Например, для реализации отношения «преподаватели работают на кафедре» в таблицу Преподаватель включают столбец ИдКафедры (внешний ключ к таблице Кафедра). Следует заметить, что отношение «один-ко-многим» легко реализуется посредством отношения «многие-ко-многим» через ассоциативную таблицу.

Существует два подхода к обеспечению между таблицами ассоциаций «многие-ко-многим». Первый основан на многократной реализации в каждой таблице столбца внешнего ключа к другой таблице. Например, для обеспечения отношения «многие-

ко-многим» между Преподавателем и Курсом можно ввести три столбца ИдКурса в таблицу Преподаватель и пять столбцов ИдПреподавателя *в* таблицу Курс. Однако при этом возникает проблема, если преподаватель читает больше трех курсов или курс читается более чем пятью преподавателями.

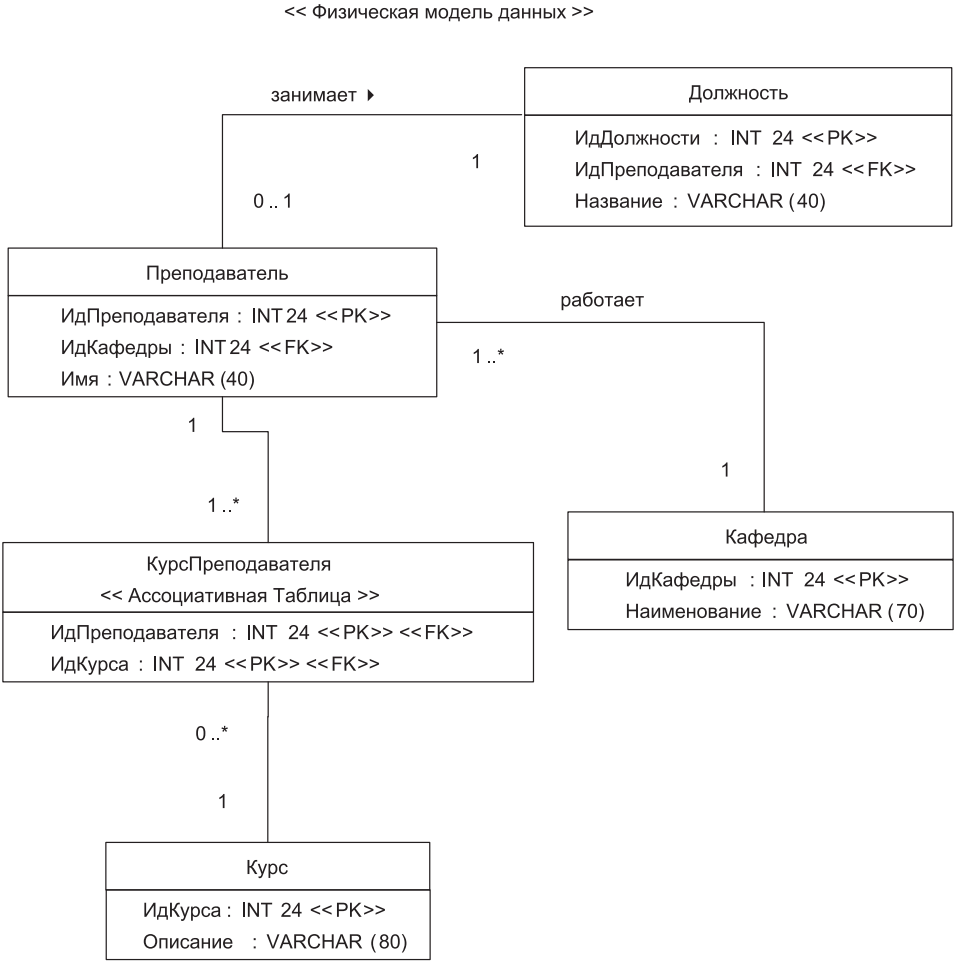


**Рис. 11.23.** Отношения между таблицами реляционной БД

Лучший подход состоит в применении ассоциативной таблицы, примером которой является таблица КурсПреподавателя (см. рис. 11.23). Ассоциативная таблица содержит комбинацию первичных ключей таблиц, которые она связывает. В этом случае можно назначить на один курс семьдесят преподавателей или предложить одному преподавателю тридцать курсов, это не имеет никакого значения. Основная уловка в том, что отношение «многие-ко-многим» преобразуется в два отношения «один-ко-многим», каждое из которых использует ассоциативную таблицу.

Поскольку для соединения таблиц используются внешние ключи, все отношения в реляционной базе данных являются двунаправленными. Это поясняет, почему не

имеет значения, в какой таблице реализуется отношение «один-к-одному», код для соединения двух таблиц фактически одинаков. Например, для физической модели (см. рис. 11.23) оператор SQL для соединения занимает выглядит так:

```
SELECT * FROM Должность, Преподаватель
WHERE Должность.ИдПреподавателя = Преподаватель.ИдПреподавателя
```

Если бы внешний ключ был реализован в таблице Преподаватель, то оператор SQL принял бы вид

```
SELECT * FROM Должность, Преподаватель
WHERE Должность.ИдДолжности = Преподаватель.ИдДолжности
```

Уменьшить усилия по отображению отношений может принцип непротиворечивости ключей в базе данных. На первом шаге следует отдавать предпочтение ключам, состоящим из единственного столбца. На следующем шаге можно использовать глобально уникальные суррогатные ключи.

## Отображение отношений объектов в реляционную базу данных

Отметим, что при отображении отношений объектов в базу данных надо сохранить заданную мощность. Поэтому отношение объектов «один-к-одному» отображают в отношение данных «один-к-одному», «один-ко-многим» отображают в «один-ко-многим», а отношение «многие-ко-многим» отображают во «многие-ко-многим». При этом следует избегать реализации отношения объектов «один-к-одному» с помощью отношения данных «один-ко-многим» или «многие-ко-многим». Техническая возможность такой подмены имеется, поскольку отношение данных «один-к-одному» является подмножеством отношения данных «один-ко-многим», а отношение «один-ко-многим» — подмножеством отношения «многие-ко-многим».

Прежде чем обсуждать отображение отношений, разберемся с отображением атрибутов для нашего иллюстративного примера. Рассмотрим метаданные отображения атрибутов нашей логической модели (см. рис. 11.22) в физическую модель данных (см. рис. 11.23). Метаданные представлены в табличной форме (табл. 11.15).

**Таблица 11.15.** Метаданные отображения атрибутов объектов в таблицы БД

| Атрибут | Столбец |
|---|---|
| Должность.название | Должность.Название |
| Должность.идДолжности | Должность.ИдДолжности |
| Преподаватель.имя | Преподаватель.Имя |
| Преподаватель.идПреподавателя | Преподаватель.ИдПреподавателя |
| Преподаватель.идПреподавателя | КурсПреподавателя.ИдПреподавателя |
| Кафедра.наименование | Кафедра.Наименование |
| Кафедра.идКафедры | Кафедра.ИдКафедры |
| Курс.описание | Курс.Описание |
| Курс.идКурса | Курс.ИдКурса |
| Курс.идКурса | КурсПреподавателя.ИдКурса |

Видим, что в таблицы отображаются только бизнес-атрибуты и атрибуты-прообразы первичных ключей. Атрибуты «*строительных лесов*», например Преподаватель.должность и Преподаватель.курсы, отображать не следует. Эти атрибуты представляются через теневую информацию, которая отображается в базу данных. Когда отношение читается из базы данных в память, значения, сохраненные в столбцах первичных ключей, будут сохранены в соответствующих теневых атрибутах объектов. Одновременно отношение между таблицами, представляемое столбцами первичных ключей, будет определяться установкой значений в атрибутах «*строительных лесов*» соответствующих объектов.

## Отображение отношений «один-к-одному»

Рассмотрим отношение «один-к-одному» между классом Преподаватель и классом Должность. Существует два варианта обеспечения между ними ссылочной целостности. В первом варианте всякий раз, когда объект класса Должность (или класса Служащий) читается в память, приложение автоматически проходит отношение занимает *и* автоматически читает соответствующий парный объект. Другой вариант состоит в ручном проходе отношения в программном коде, с использованием подхода ленивого чтения. В этом варианте парный объект читается только в тот момент, когда это потребуется приложению. Обсудим метаданные отображения отношений для нашего примера (табл. 11.16). Всего в примере применяются три отношения, каждому отношению соответствует пара строк метаданных.

**Таблица 11.16.** Метаданные отображения отношений

| Отношение объектов | Мощность | Автоматическое чтение | Столбцы | Атрибут «строительных лесов» |
|---|---|---|---|---|
| Преподаватель занимает Должность | Единица | Да | Должность.ИдПреподавателя | Преподаватель.должность |
| Должность занимается Преподавателем | Единица | Да | Должность.ИдПреподавателя | Преподаватель.должность |
| Преподаватель работает на Кафедре | Единица | Да | Преподаватель.ИдКафедры | Преподаватель.кафедра |
| На Кафедре имеет работу Преподаватель | Много | Нет | Преподаватель.ИдКафедры | Кафедра.преподаватели |
| Преподаватель читает Курс | Много | Нет | Преподаватель.ИдПреподавателя<br>КурсПреподавателя.ИдПреподавателя | Преподаватель.курсы |
| Курс читается Преподавателем | Много | Нет | Курс.ИдКурса<br>КурсПреподавателя.ИдКурса | Курс.преподаватели |

Но перейдем к нашему отношению занимает. Пошаговое чтение объекта класса Должность сводится к следующему:

1. Объект класса Должность читается из таблицы в память.
2. Автоматически проходится отношение занимает.
3. Значение в столбце Должность.ИдПреподавателя используется для идентификации отдельного преподавателя, которого надо прочитать в память.
4. В таблице Преподаватель ищется запись (строка) со значением ИдПреподавателя.
5. Объект класса Преподаватель (если он есть) читается и создается.
6. Для ссылки на объект класса Должность устанавливается значение атрибута Преподаватель.должность.

Соответственно шаги чтения объекта класса Преподаватель имеют следующий вид:

1. Объект класса Преподаватель читается в память.
2. Автоматически проходится отношение занимает.
3. Значение в столбце Должность.ИдПреподавателя используется для идентификации отдельной должности, которую надо прочитать в память.
4. В таблице Должность ищется строка с этим значением ИдПреподавателя.
5. Объект класса Должность читается и создается.
6. Для ссылки на объект класса Должность устанавливается значение атрибута Преподаватель.должность.

Есть одно «но», касающееся отображения в базу данных отношения «*занимает*». Хотя в логической модели направление этого отношения задано от Преподавателя *к* Должности, в базе данных оно реализовано от Должности к Преподавателю, поскольку внешний ключ размещен в таблице Должность. Это маленькое, но раздражающее несоответствие. В базе данных внешний ключ можно реализовать в любой таблице, причем совершенно безразлично, в какой. Если в будущем отношение «*занимает*» может превратиться в отношение «один-ко-многим», тогда такое размещение внешнего ключа обосновано и отражает потенциальное требование (поддержка преподавателя, занимающего несколько должностей). Однако для принятой логической модели (при отсутствии будущих требований по ее изменению) было бы корректнее реализовать внешний ключ в таблице Преподаватель.

Теперь рассмотрим, как объекты должны сохраняться в базе данных. Поскольку отношения должны автоматически проходиться и поддерживать ссылочную целостность, создается *транзакция*, то есть описание атомарного, неделимого действия. На следующем шаге для каждого объекта в транзакции добавляются SQL-операторы обновления. Каждый оператор обновления включает как бизнес-атрибуты, так и прообразы первичных ключей (см. табл. 11.15). Поскольку отношения между таблицами реализуются через внешние ключи и их значения модифицируются, отношения фактически сохраняются. Транзакция применяется к базе данных и исполняется.

## Отображение отношений «один-ко-многим»

Примером отношения «один-ко-многим» является отношение работает между классом Преподаватель и классом Кафедра (см. рис. 11.22). Преподаватель работает на одной кафедре, а в состав отдельной кафедры входит много работающих

преподавателей. Такое отношение должно автоматически проходиться от Преподавателя к Кафедре, но не в обратном направлении (см. табл. 11.16).

Когда объект класса Преподаватель читается из базы данных в память, отношение автоматически проходится для чтения объекта класса Кафедра, на которой он работает. Не следует иметь несколько копий одной и той же кафедры, например, если на кафедре работают 18 преподавателей, нужно, чтобы все преподаватели ссылались на один и тот же объект кафедры в памяти. Это значит, что требуется соответствующая методика, которая гарантирует только одну копию объекта в памяти (например, объект кафедры может сохраняться в кэш-памяти и выгружаться оттуда при необходимости). Итак, при наличии объекта класса Кафедра в памяти в атрибут Преподаватель.кафедра парного объекта заносится ссылка на него. Соответственно вызов операции Кафедра.добавитьПреподавателя() приводит к добавлению в контейнер кафедры объекта-преподавателя.

Сохранение отношений «один-ко-многим» в БД выполняется точно так же, как и для отношений «один-к-одному»: когда объекты сохраняются, сохраняются значения их первичных и внешних ключей, что приводит к автоматическому сохранению отношений.

В нашем иллюстративном примере используются такие внешние ключи (скажем, Преподаватель.ИдКафедры), которые указывают на первичные ключи других таблиц (в этом случае Кафедра.ИдКафедры). Это не обязательно, внешний ключ может ссылаться и на альтернативный ключ. Например, если бы таблица Преподаватель (см. рис. 11.23) включала столбец Страховка, тогда он мог бы быть альтернативным ключом для таблицы. В этом случае столбец Кафедра.ИдПреподавателя можно заменить на столбец Кафедра.Страховка.

## Отображение отношений «многие-ко-многим»

Для обеспечения отношения «многие-ко-многим» потребуется ассоциативная таблица, единственная цель которой состоит в поддержке отношений между двумя или более таблицами в реляционной базе данных. В логической модели нашего примера (см. рис. 11.22) есть отношение «многие-ко-многим» между Преподавателем и Курсом. Для его обеспечения в физическую модель данных (см. рис. 11.23) пришлось ввести ассоциативную таблицу КурсПреподавателя, реализующую отношение «многие-ко-многим» между таблицами Курс и Преподаватель. В реляционных базах данных столбцы, содержавшиеся в ассоциативной таблице, традиционно являются комбинацией ключей из таблиц, вовлеченных в отношения (в этом примере ключей ИдПреподавателя и ИдКурса). Название ассоциативной таблицы обычно является или комбинацией имен таблиц, которые она связывает, или именем ассоциации, которую она реализует. В данном случае был выбран первый вариант имени.

Важно разобраться с тем, как мощности отношения логической модели трансформируются в мощности связей между таблицами в физической модели. Правило такое: при введении ассоциативной таблицы кратности «переходят» к ней (сопоставьте рис. 11.22 и рис. 11.23). Для сохранения мощности оригинальных отношений на внешних полюсах связей ассоциативной таблицы (в базе данных) всегда вводится мощность 1. Оригинальное отношение указывало, что преподаватель читает один или более курсов и что курс читается нулем или более преподавателей. В базе дан-

ных все это сохраняется, вот только партнером Преподавателя и Курса становится ассоциативная таблица, обеспечивающая их связь.

Предположим, что объект преподавателя находится в памяти и требуется список всех читаемых им курсов. Для этого приложение должно выполнить следующие шаги:

1. Создать SQL-оператор Select, который присоединит друг к другу таблицы КурсПреподавателя и Курс, выбирая в таблице КурсПреподавателя записи с таким значением ИдПреподавателя, которое имеет нужный преподаватель.
2. Применить оператор Select к базе данных.
3. Собрать в объекты Курсов записи данных, представляющие эти курсы. В ходе этого проверяется: может объект класса Курс уже находится в памяти? Если да, тогда мы обновляем значения данных объекта (это вопрос параллелизма доступа к БД).
4. Для каждого объекта класса Курс вызывать операцию Преподаватель.добавитьКурс(). Так строится коллекция, наполняется контейнер объекта-преподавателя.

Аналогичный процесс обеспечивает формирование преподавателей, читающих данный курс. Для поддержки отношения, все еще с точки зрения объекта класса Преподаватель, надо выполнять следующие шаги:

1) начать транзакцию;
2) для любых изменяемых объектов-курсов добавить операторы Update;
3) для таблицы Курс добавить операторы Insert для создаваемых новых курсов;
4) для таблицы КурсПреподавателя добавить операторы Insert для новых курсов;
5) для таблицы Курс добавить операторы Delete для любых удаляемых курсов. Шаг пропускается, если индивидуальные удаления объектов уже произошли;
6) для таблицы КурсПреподавателя добавить операторы Delete для любых удаляемых курсов. Шаг пропускается, если индивидуальные удаления объектов уже произошли;
7) для таблицы КурсПреподавателя добавить операторы Delete для любых курсов, которые больше не читаются преподавателем;
8) запустить транзакцию.

Своеобразие отношений «многие-ко-многим» определяется необходимостью добавления ассоциативной таблицы. Два класса отображаются в три таблицы данных, чтобы поддержать такое отношение. Таким образом, для получения правильного результата нужна дополнительная работа.

## Отображение отношений композиции

До сих пор мы рассматривали отображение отношений ассоциации. Отображение в базу данных отношений агрегации и композиции имеет свои особенности. Мы уже обсуждали логическую модель с отношением композиции между классами Заказ и ЭлементЗаказа (см. рис. 11.10). На отношение было наложено ограничение *{по порядку}*, предупреждающее о том, что элементы в заказе должны появляться

в определенном порядке. При отображении отношения в базу данных пришлось добавить дополнительный столбец для отслеживания порядка. Это было показано в физической модели данных (см. рис. 11.10), здесь таблица ЭлементЗаказа включала столбец ЭлементЗаказа.ЭлементНабора, обеспечивающий сохранение информации о порядке. В этих условиях для обеспечения устойчивости сохранения и работы класса-агрегата Заказ необходимо принимать специальные меры.

*Данные следует читать в заданном порядке*. Атрибутом «*строительных лесов*», обеспечивающим отношение композиции, должен быть контейнер на основе упорядоченной коллекции. Этот контейнер должен обеспечивать упорядочивание и наращивание адресов при добавлении к Заказу новых элементов (объектов класса ЭлементЗаказа). В нашей логической модели (см. рис. 11.11) для создания контейнера элементыЗаказа используется класс Vector, класс из коллекций языка Java, который удовлетворяет этим требованиям. По мере чтения заказа и элементов заказа в память, контейнер-вектор должен заполняться в правильном порядке. Если значения в столбце ЭлементЗаказа.ЭлементНабора начинаются с единицы и увеличиваются на единицу, тогда можно просто использовать значение столбца как позицию для вставки элементов заказа в контейнер-коллекцию. Если это не так, тогда в оператор SQL, применяемый к базе данных, надо включить предложение ORDER BY. Это гарантирует, что строки будут появляться в надлежащем порядке.

*Порядковый номер не следует вводить в первичный ключ*. Положим, в памяти имеется заказ с тремя элементами, и они были сохранены в базе данных. Теперь надо вставить новый элемент заказа, между вторым и третьим элементами. В результате общее количество элементов заказа увеличивается до четырех. При этом в физической модели данных (см. рис. 11.10) придется перенумеровать порядковые номера каждого элемента заказа, который появляется после нового элемента, а затем переписать все эти элементы, даже если в элементе не изменилось ничего, кроме порядкового номера. Представим, что в это время другие таблицы (не показанные в модели) обращаются к строкам в таблице ЭлементЗаказа через внешние ключи, использующие столбец ЭлементНабора. (Ведь порядковый номер — столбец ЭлементНабора — является частью первичного ключа таблицы ЭлементЗаказа!) Ясно, что одновременная перенумерация элементов заказа будет, мягко говоря, затруднена. Лучшее решение состоит в назначении таблице ЭлементЗаказа другого первичного ключа, например столбца ИдЭлементаЗаказа (рис. 11.24).

*После реорганизации элементов заказа следует модифицировать порядковые номера элементов*. Всякий раз, когда выполняется перестройка элементов заказа (например, четвертый элемент переместился на место второго элемента заказа), следует модифицировать порядковые номера в базе данных. Эти изменения можно кэшировать в памяти до тех пор, пока не будет принято решение о перезаписи всего заказа. Правда, такое решение сопряжено с риском (правильная последовательность не будет сохранена в случае внезапного выключения питания).

*После удаления элемента заказа тоже следует модифицировать порядковые номера элементов*. После удаления третьего из четырех элементов заказа можно модифицировать порядковый номер того элемента, который теперь стал третьим элементом, или оставить все как есть. Порядковые номера все еще работают, их значениями являются 1, 2, 4, но их уже нельзя использовать как индикаторы позиции в контейнере-коллекции (если оставите дырку в третьей позиции).

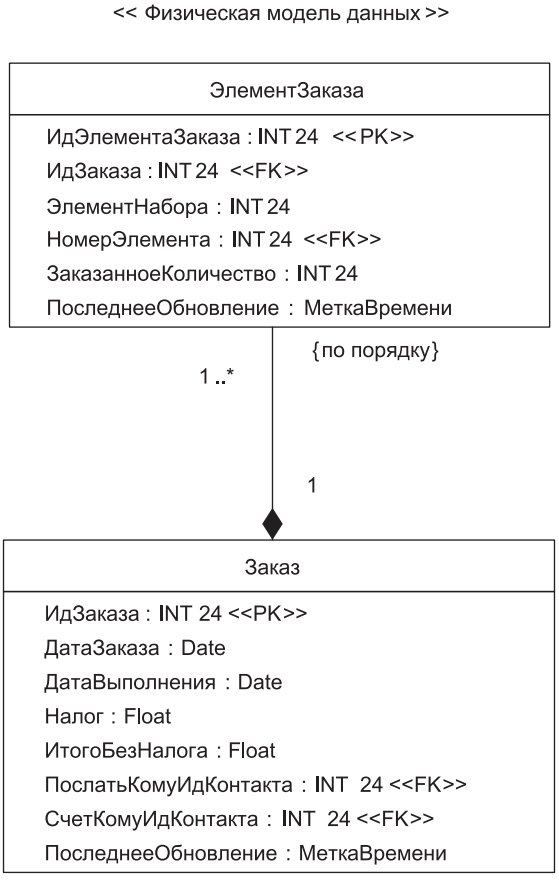


**Рис. 11.24.** Улучшение физической модели данных для системы заказов

*Порядковые номера целесообразно выбирать с дискретностью большей, чем единица*. Вместо того чтобы назначать порядковые номера строк 1, 2, 3, ... назначайте номера со значениями 10, 20, 30 и т. д. Тогда не потребуется модификация значения столбца ЭлементЗаказа.ЭлементНабора при каждой перестройке элементов заказа. Ведь при перемещении элемента между позициями 20 и 30 ему можно назначить порядковый номер 25. Конечно, по-прежнему придется модифицировать значения при попытке вставить элемент между позициями 27 и 28. Большие промежутки между значениями (например, 00, 50, 100, ...) делают такую необходимость более редкой, но не снимают проблему полностью.

## Отображение рекурсивных отношений

Рекурсивным (иначе рефлексивным) называют отношение классификатора (класса, сущности данных, таблицы) с самим собой. Представим логическую и физическую модели данных с рекурсивными отношениями (рис. 11.25). Ради простоты здесь не

показаны атрибуты классов. Видим, что в логической модели имеется рекурсивная ассоциация категории «один-ко-многим» и рекурсивная агрегация категории «многие-ко-многим». Рекурсивность ассоциации руководит фиксирует понятие, согласно которому разработчик может руководить другими разработчиками. Рекурсивность отношения агрегации, которое имеет с собой класс Группа, задает тот факт, что группа может быть частью одной или нескольких других групп.

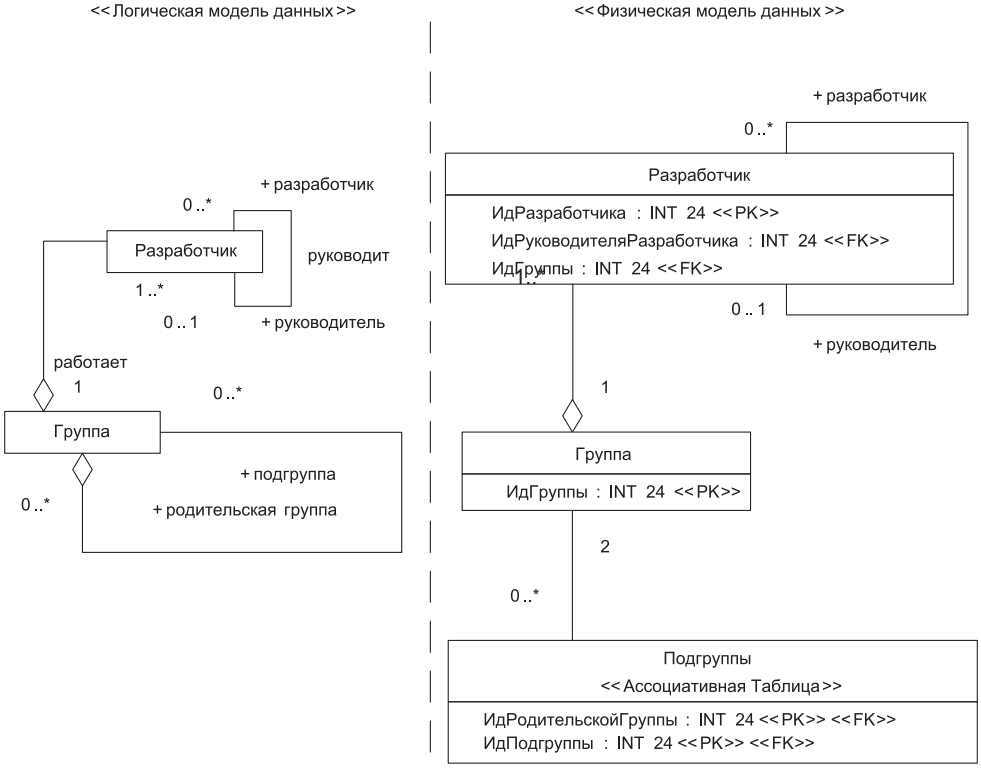


**Рис. 11.25.** Отображение рекурсивных отношений

Отметим, что отображение рекурсивных отношений подчиняется общим правилам. При отображении рекурсивной ассоциации столбец ИдРуководителяРазработчика ссылается на другую строку той же таблицы Разработчик: строку, в которой хранятся данные руководителя.

Соответственно рекурсивная агрегация «многие-ко-многим» реализуется в физической модели с помощью обычной ассоциативной таблицы Подгруппы. Разница лишь в том, что оба ее столбца являются внешними ключами в одной и той же таблице.

# Настройка быстродействия базы данных

Рассматривая различные аспекты разработки базы данных, мы ориентировались на программные системы, в которых совместно используются объектно-ориентированная и реляционная технологии (рис. 11.26).


**Рис. 11.26.** Объектно-ориентированная система, включающая реляционную БД

Увы, но между этими технологиями существует полное несоответствие. И вместе с тем индивидуальные достоинства технологий обязывают их жить в мире и согласии. Эти факты вынудили нас сконцентрироваться на вопросах прямого отображения содержания и отношений объектов в реляционную БД и их обратного отображения из БД.

Ясно, что с системной точки зрения база данных должна обеспечить требуемое быстродействие. При автономной работе настройка быстродействия БД сводится к действиям двух категорий:

- настройке быстродействия самой базы данных;
- настройке скорости доступа к данным базы.

Основной метод настройки быстродействия базы данных состоит в изменении ее схемы (структуры таблиц и связей между ними) путем денормализации. Другие методы включают:

- изменение типа столбцов ключей (числовой индекс более эффективен, чем символьный);
- сокращение количества столбцов, которые образуют составной ключ;
- введение табличных индексов для ускорения типовых запросов и отчетов.

Ускорение доступа к данным базы обычно опирается на такие методы:

- применение хранимых процедур, уплотняющих данные на сервере БД и сокращающих итоговый набор пересылок по сети;
- модификация SQL-запросов, учитывающая особенности БД;
- кластеризация данных для отражения типовых потребностей клиентов к доступу;
- уменьшение количества доступов за счет кэширования данных в приложении.

Объектно-ориентированное окружение реляционной БД вынуждает применять еще две категории действий: настройку отображений и ленивые чтения.

Мы изучили достаточно много способов отображения объектов в физические модели данных:

- четыре способа отображения атрибутов уровня класса;
- четыре способа для отображения дерева наследования классов;
- пять способов отображения отношений между объектами.

Каждый способ отображения имеет свои преимущества, недостатки и область применения. Выбирая конкретный способ для конкретных условий, можно улучшить скорость доступа к данным приложения. Положим, что применение подхода к отображению наследования «одна таблица на класс» слишком замедлило доступ к данным. В этом случае можно перейти к способу «одна таблица на все дерево наследования».

Правда, следует понимать, что смена способа отображения может потребовать изменения или схемы данных, или схемы объектов (вспомним про теневую информацию и «строительные леса»), или одновременного изменения обеих схем. Понятно, что в рамках единой системы таблицы влияют на объекты, а объекты — на таблицы. И все же: нельзя позволить схемам базы данных или физическим моделям данных управлять разработкой логических (объектных) моделей. Это фундаментальная ошибка. Конечно, надо изучать табличную организацию, учитывать ее как ограничения, но надо исключить ее негативное влияние на разработку системы.

Теперь обсудим ленивые чтения. Важное влияние на быстродействие оказывает такой аспект: должны ли все атрибуты автоматически читаться, когда объект извлекается из БД? Когда на это уходит много времени, целесообразно применение принципа «ленивого чтения». Основная идея: вместо автоматической пересылки атрибута по сети при чтении объекта его значение считывается только при необходимости. Для этого применяют операцию-получатель, цель которой состоит в обеспечении значения единственного атрибута. Эта операция выясняет: не был ли инициализирован атрибут и если не был, считывает значение атрибута из базы данных. Другое применение ленивого чтения — генерация отчетов при считывании объектов в результате поиска, когда требуется лишь малая часть данных объекта.

## Контрольные вопросы и упражнения

1. Дайте определение базы данных. Что означает в этом определении термин «устойчивые»?
2. Охарактеризуйте известные модели баз данных. Сформулируйте их достоинства и недостатки.
3. Поясните организацию таблицы реляционной базы данных, смысл ее понятий: запись, атрибут, домен. Как иначе называют запись и атрибут таблицы?
4. Что такое представление в реляционной базе данных? Чем отличается представление от обычной таблицы? Поясните достоинства и недостатки применения представлений.
5. Какую роль играют первичные ключи таблицы в реляционной базе данных? В чем разница между естественными и суррогатными ключами?
6. Поясните назначение внешнего ключа таблицы. Приведите примеры использования внешних ключей в гипотетической реляционной базе данных.

7. Какому ограничению должны удовлетворять внешние ключи? В чем смысл этого ограничения? Определите гипотетическую реляционную базу данных. Приведите примеры нарушения в ней ограничений на внешние ключи, охарактеризуйте последствия этих нарушений.
8. Дайте характеристику индексов в реляционной базе данных. В чем состоят достоинства и недостатки индексов?
9. Какие виды отношений между ключами таблиц вы знаете? Какие из них применяют часто (редко) и почему? Какие из отношений максимально увеличивают накладные расходы и почему?
10. Какую роль в реляционных базах данных играют хранимые процедуры и триггеры? Какая между ними разница?
11. Зачем проводится нормализация реляционных баз данных? Каков порядок ее проведения?
12. В чем суть первой нормальной формы реляционной базы данных?
13. В чем суть второй нормальной формы реляционной базы данных? Истинно ли утверждение «если база данных находится во второй нормальной форме, то одновременно она находится в первой нормальной форме»? Ответ обоснуйте.
14. Поясните суть третьей нормальной формы реляционной базы данных.
15. Какие типы моделей применяют при моделировании баз данных? Охарактеризуйте каждый тип модели, место моделей в общем процессе моделирования. Какие способы фиксации типа модели вы знаете? Приведите примеры моделей.
16. В чем состоит принцип обозначения таблиц, сущностей и представлений в языке UML? На ваш взгляд, в чем заключается главный недостаток обозначения таблиц? Ответ обоснуйте.
17. Как обозначаются в языке UML ключи, ограничения, триггеры и хранимые процедуры? Назовите достоинства и недостатки этих обозначений. Приведите примеры.
18. Какие средства отображения составных ключей в языке UML вы знаете? Приведите примеры описания таких ключей.
19. Поясните суть задачи отображения объектов в реляционную базу данных. Что при этом приходится отображать?
20. Как отображается атрибут объекта в реляционную базу данных? Назовите особенности этого отображения. Приведите примеры.
21. Возможно ли идентичное описание объекта (класса) в реляционной базе данных и вне этой базы? Ответ обоснуйте.
22. Охарактеризуйте состав и назначение теневой информации. В чем заключается ее скрытость? Сформируйте теневую информацию для конкретного приложения с базой данных.
23. На конкретном примере объясните назначение метаданных отображения. Как метаданные демонстрируют явное несоответствие между объектной и реляционной технологией?

24. В чем заключается специфика отображения атрибутов уровня класса? Дайте характеристику известных методик отображения таких атрибутов, акцентируя внимание на их достоинствах и недостатках.
25. Какова суть отображения дерева наследования в единственную таблицу? На конкретном примере охарактеризуйте достоинства, недостатки и область применения этой методики.
26. Поясните принцип отображения каждого конкретного класса в отдельную таблицу. На примере реального приложения охарактеризуйте достоинства, недостатки и область применения этого принципа.
27. В чем состоят особенности методики отображения каждого класса в отдельную таблицу? Какие способы улучшения организации таблиц, получаемых по этой методике, вы знаете? На примере конкретного приложения охарактеризуйте достоинства, недостатки и область применения этой методики.
28. Чем отличаются таблицы, получаемые по методике отображения каждого конкретного класса и методике отображения каждого класса?
29. Дайте развернутую характеристику отображения классов в универсальную табличную структуру. Каково его главное отличие от других методик отображения деревьев наследования? На примере нескольких реальных систем поясните достоинства, недостатки и область применения этой методики.
30. Создайте несколько вариантов небольшого объектно-ориентированного приложения с реляционной БД. В ходе разработки примените все известные методики отображения механизма наследования. Сравните полученные результаты.
31. Объясните специфику отображения множественного наследования по каждой из известных методик.
32. Какие существуют ограничения при отображении объектных отношений в реляционную БД? Каким образом сказываются их последствия?
33. Что называют *«строительными лесами»* при реализации отношений между объектами? Приведите примеры строительных лесов.
34. На конкретных примерах сравните строительные леса однозначного отношения и отношения с мощностью, большей единицы. Чем отличаются строительные леса однонаправленных и двунаправленных отношений?
35. Охарактеризуйте специфику реализации отношений в реляционных базах данных. Какие дополнительные средства нужны для обеспечения отношений «многие-ко-многим» и почему? Существует ли альтернативное решение?
36. Поясните, в чем заключается неоднозначность отображения в реляционную базу данных отношения «один-к-одному».
37. На примере двух связанных объектов опишите содержание процесса сохранения объекта в базе данных и процесса извлечения объекта из базы данных.
38. На примере объекта, связанного с тремя другими объектами, опишите процесс считывания объекта из базы данных со стороны «один» и процесс считывания объекта со стороны «много».

39. Постройте фрагмент приложения, в котором есть две группы объектов: первая группа включает три объекта, вторая группа — два объекта. Каждый из объектов первой группы связан с каждым из объектов второй группы, а каждый из объектов второй группы связан с каждым из объектов первой группы. Отобразите этот фрагмент в реляционную БД и опишите процесс считывания объекта из БД.
40. Укажите сходства и различия отображения отношений ассоциации и композиции (агрегации). Приведите конкретный пример отображения в БД объекта-агрегата, частями которого являются два объекта. Как должна быть организована запись этого фрагмента в БД и обновление данных отдельных элементов фрагмента?
41. Поясните специфику отображения в базу данных рекурсивных отношений между объектами. Как она проявляется для отношений «один-ко-многим» и «многие-ко-многим»?
42. Какие основные действия обеспечивают настройку быстродействия реляционной базы данных? Какие дополнительные действия можно выполнять, если база данных находится в объектно-ориентированном окружении? Объясните содержание этих действий.
43. Можно ли позволить схеме базы данных (или физической модели данных) управлять разработкой логической, объектной модели? Ответ обоснуйте.

# Глава 21

# Автоматизация разработки визуальной модели программной системы

В современных условиях создание сложных программных приложений невозможно без использования систем автоматизированной разработки ПО (CASE-систем). CASE-системы существенно сокращают сроки и затраты разработки, оказывая помощь инженеру в проведении рутинных операций, облегчая его работу на самых разных этапах жизненного цикла разработки. Наиболее развитой из CASE-систем в настоящее время принято считать IBM Rational Software Architect (RSA). В данной главе рассматривается порядок применения Rational Software Architect при формировании требований, анализе, проектировании и генерации программного кода.

## Общая характеристика системы IBM Rational Software Architect

IBM Rational Software Architect представляет собой интегрированное средство визуального моделирования объектно-ориентированных программных продуктов. Визуальное моделирование — процесс графического описания разрабатываемого программного обеспечения.

Ключевые функции RSA перечислены в табл. 21.1.

**Таблица 21.1.** Ключевые функции Rational Software Architect

| Функции | Преимущества |
|---|---|
| Поддержка UML 2 для формирования требований, анализа и проектирования с использованием диаграмм Use Case, классов, диаграмм последовательности, диаграмм коммуникации, диаграмм деятельности, конечных автоматов, компонентов и развертывания | UML 2.0 позволяет создавать все артефакты визуального моделирования в стандартной нотации, принятой различными заинтересованными лицами |

| Функции | Преимущества |
|---|---|
| Формирование для UML-проектов отчетов в форматах HTML, PDF и XML | Создание отчетов и документации, которые могут просматривать разработчики и другие заинтересованные лица |
| Использование механизма трансформаций для быстрого перехода от модели к коду и от кода к модели для Java/J2EE, WSDL, XSD, SOA, C/C++ и CORBA IDL | Автоматизация задач по генерации кода на основе моделей проекта. Трансформации можно настраивать в соответствии с шаблонами генерации кода, принятыми в организации |
| Редактирование UML-диаграмм и программного кода Java/J2EE, объектов баз данных | Графическая нотация UML упрощает разработку и осмысление кода для новых и существующих приложений |

Экран среды Rational Software Architect показан на рис. 21.1.

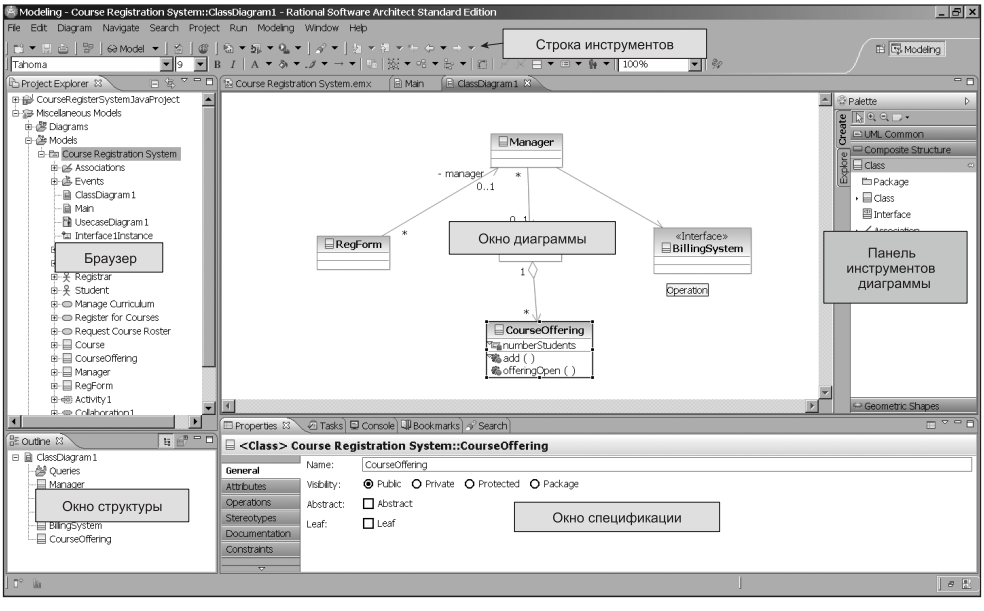


**Рис. 21.1.** Экран среды Rational Software Architect

В его составе выделим шесть элементов: строку инструментов, браузер, окно структуры, окно диаграммы, палитру инструментов диаграммы и окно спецификации.

Элементы строки инструментов (рис. 21.2) позволяют выполнять стандартные и специальные действия.

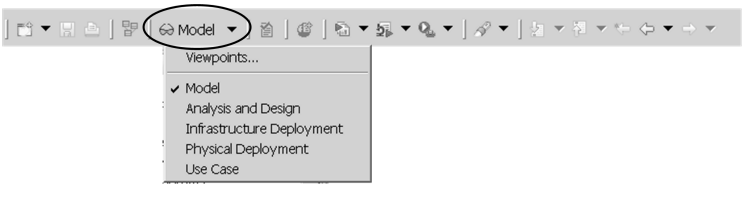


**Рис. 21.2.** Элементы строки инструментов Rational Software Architect

Обратим внимание на раскрывающийся список (Model Viewpoints), определяющий точку зрения на проект, которая меняет вид экрана RSA.

Браузер Rational Software Architect (Project Explorer) является инструментом иерархической навигации, позволяющим просматривать названия и пиктограммы, отображающие диаграммы и элементы визуальной модели (рис. 21.3). Знак плюс (+) рядом с папкой означает, что внутри папки находятся дополнительные элементы. Для «разворачивания» папки надо нажать на знак +. Если папка «развернута», то слева от нее появляется знак минус (–). Для «сворачивания» структуры папки нажимается знак минус. Браузер позволяет добавлять, перемещать, упорядочивать и сортировать элементы модели.

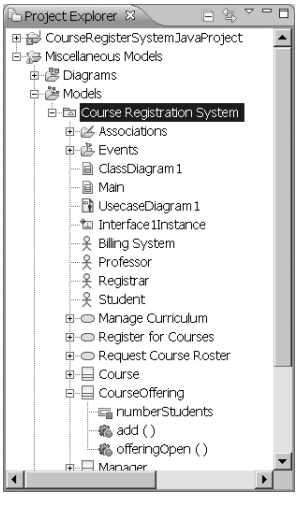


**Рис. 21.3.** Браузер Rational Software Architect

Окно структуры отображает структуру элемента модели, например диаграммы, открытой в данный момент в области окна диаграммы. Информация в окне (рис. 21.4) представляется в виде последовательности элементов.

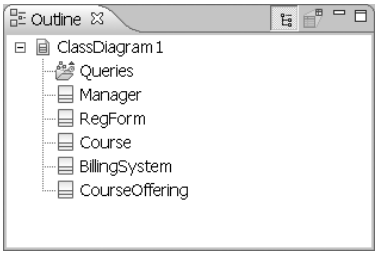


**Рис. 21.4.** Окно структуры в Rational Software Architect

Палитра инструментов диаграммы содержит средства, используемые при создании диаграммы. Содержание палитры меняется в зависимости от вида активной диаграммы.

В окне диаграммы можно создавать, отображать и изменять диаграмму на языке UML. В качестве примера на рис. 21.5 показано окно для диаграммы классов и палитра инструментов, используемых в этом типе диаграммы.

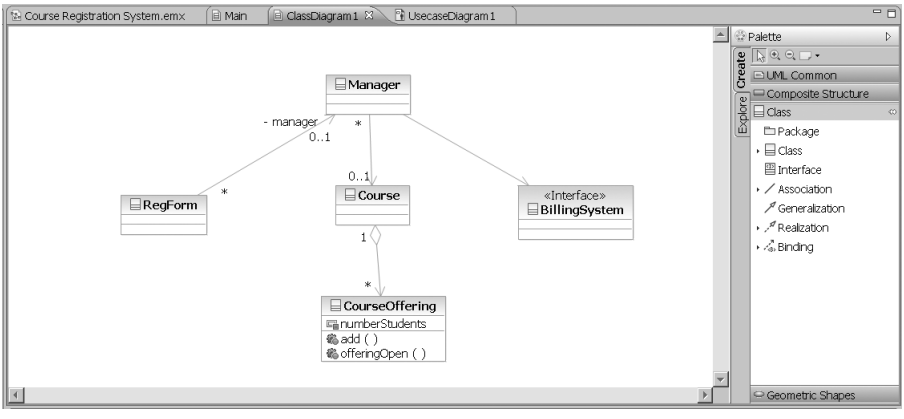


**Рис. 21.5.** Окно и палитра инструментов для диаграммы классов Rational Software Architect

Все модели, диаграммы, вершины и отношения диаграмм, равно как и остальные ресурсы проекта, имеют определенные свойства. Окно спецификации (рис. 21.6.) позволяет задавать и переопределять значения свойств. Все свойства в окне разбиты на категории. Состав этих категорий зависит от вида рассматриваемого объекта (модель, диаграмма, класс и т. д.), но в окне обязательно присутствуют следующие категории:

- General: Содержит основные характеристики объекта, например имя, уровень видимости и т. д.
- Documentation: Используется для описания (документирования) объекта.
- Advanced: Отображает страницу свойств в стиле Eclipse (базисной среды, использованной для создания RSA).

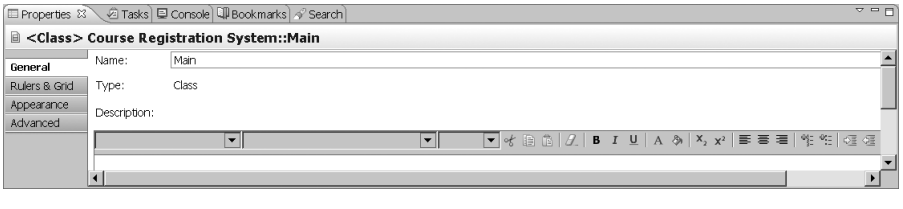


**Рис. 21.6.** Окно спецификации в Rational Software Architect

В качестве примера работы с Rational Software Architect рассмотрим построение модели университетской системы для регистрации учебных курсов. Эта система используется:

- профессором — для определения содержания читаемого курса;
- студентом — для выбора изучаемого курса;

- регистратором — для формирования учебного плана и расписания;
- учетной системой — для определения денежных затрат.

RSA предоставляет разнообразные варианты построения проектов и моделей. Воспользуемся одним из простейших вариантов. Для создания модели в главном меню системы RSA выберем пункт New — Other (рис. 21.7).

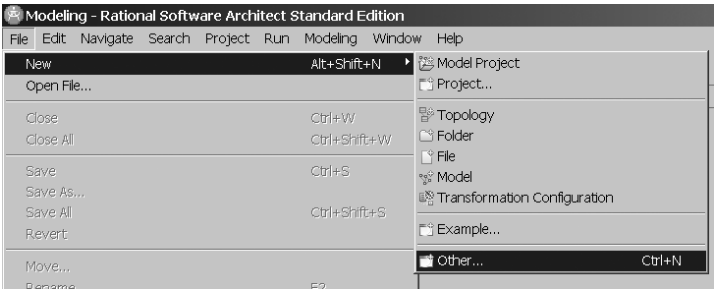


**Рис. 21.7.** Создание модели системы регистрации учебных курсов

На экране появится окно Select a wizard (Выберите программу-мастер), в котором выбираем пункт UML Model (рис. 21.8).

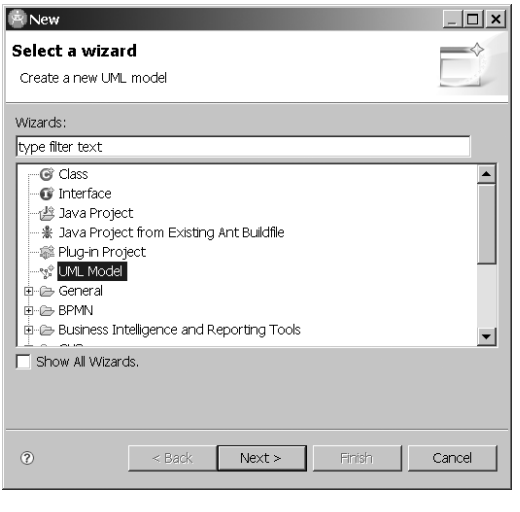


**Рис. 21.8.** Выбор вида модели

После щелчка по Next будет выведено окно Create Model (создание модели), где предлагается создать модель на базе стандартного шаблона или существующей модели. Выбираем вариант стандартного шаблона и щелкаем по Next (рис.21.9).

Это приводит к выводу следующего диалогового окна Create Model, позволяющего выбрать категорию, к которой относится создаваемая модель (раздел Categories), и один из стандартных шаблонов, предлагаемых для выбранной категории (раздел Templates). Для нашего примера выберем категорию Analysis and Design (анализ и про-

ектирование) и шаблон Blank Analysis Package (пустой пакет анализа). В окошке File name укажем название файла с нашей моделью Course Registration System (рис. 21.10).

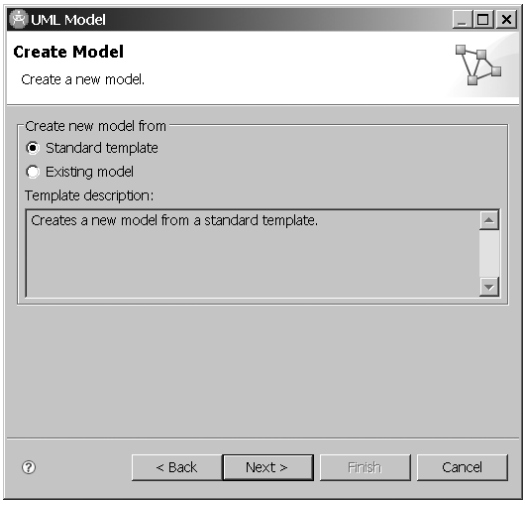


**Рис. 21.9.** Выбор стандартного шаблона в качестве основы для создания модели

Щелчком по Next переходим к следующему окну диалога. Пропускаем его (щелкаем по Next) и в следующем окне Model Capabilities (рис. 21.11), чтобы иметь

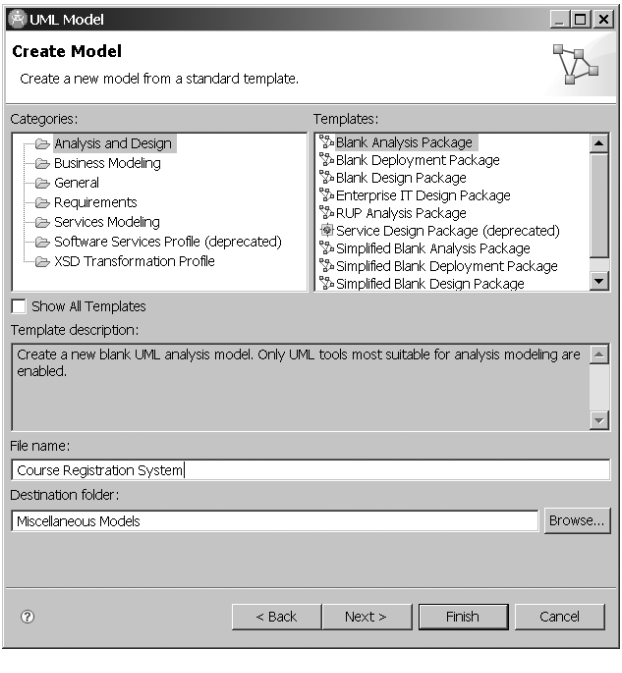


**Рис. 21.10.** Выбор шаблона для создания модели и определение ее названия

возможность максимального использования средств RSA, делаем щелчок по кнопке Enable All. Завершаем щелчком по кнопке Finish.

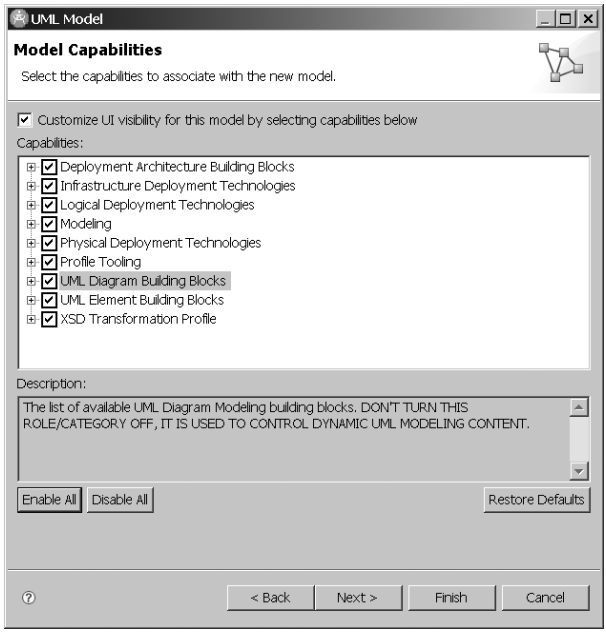


**Рис. 21.11.** Выбор возможностей RSA, доступных в новой модели

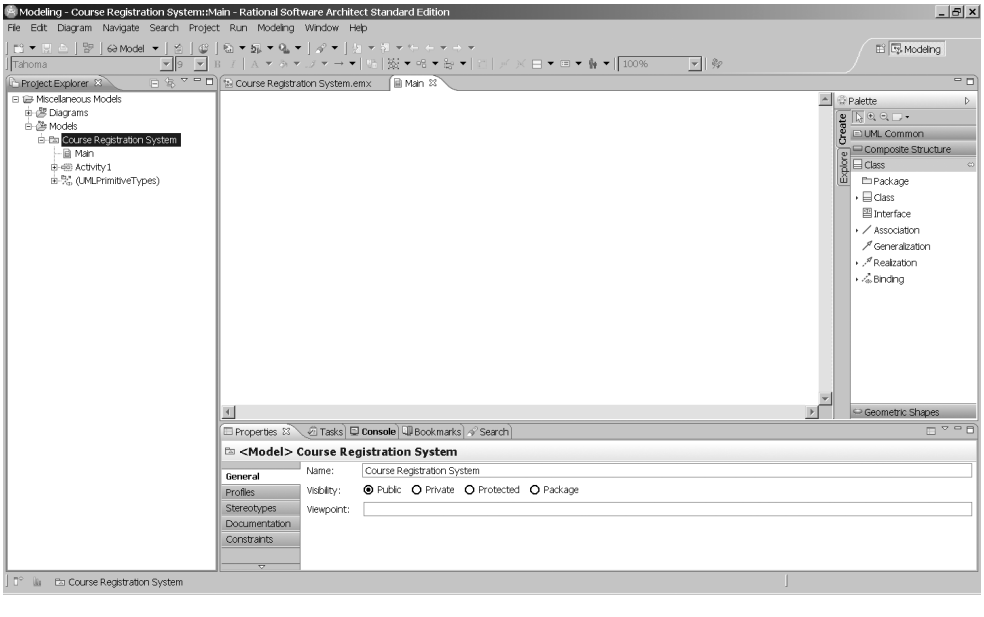


**Рис. 21.12.** Экран для модели системы регистрации учебных курсов

В результате перечисленных действий экран RSA приобретет вид, показанный на рис. 21.12. Центральную часть экрана занимает главное окно модели. В левой части экрана в браузере проектов (Project explorer) отображается дерево элементов модели.

Теперь приступим к построению модели.

## Создание диаграммы Use Case

Моделирование системы регистрации курсов начнем с создания диаграммы Use Case. Этот тип диаграммы представляется актерами, элементами Use Case и отношениями между ними. Для создания диаграммы Use Case:

1) в окне браузера раскроем пункт Diagrams (диаграммы), для чего щелкнем по значку + слева от названия этого пункта;
2) в раскрывшемся списке выбираем нашу модель (Course Registration System) и делаем щелчок правой кнопкой мышки;
3) в появляющемся меню и подменю выбираем пункт Add Diagram > Use Case Diagram.

Описанную последовательность иллюстрирует рис. 21.13.

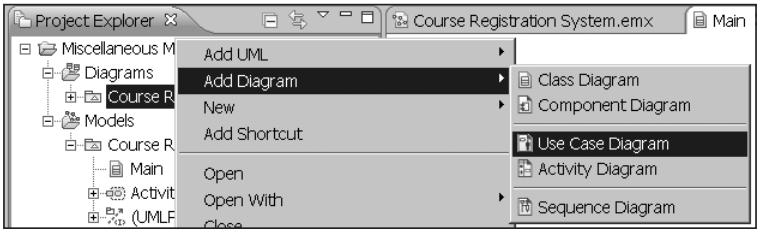


**Рис. 21.13.** Открытие окна диаграммы Use Case

На экране появится окно, в котором и будет создаваться диаграмма Use Case (рис. 21.14). Оставим ее название по умолчанию — UsecaseDiagram1. Справа от окна диаграммы располагается палитра инструментов (Palette), применяемых для создания диаграмм Use Case.

Первый шаг построения диаграммы состоит в определении актеров, фиксирующих роли внешних объектов, взаимодействующих с системой. В рассматриваемой предметной области будут фигурировать четыре актера — Student (студент), Professor (профессор), Registrar (регистратор) и Billing System (учетная система).

Для начала введем в диаграмму актера Student. Для этого выполним следующие действия:

1. В палитре инструментов Palette щелкнем по значку Actor.
2. Щелкнем в том месте диаграммы, куда мы собираемся добавить актера.
3. Пока пиктограмма актера остается выделенной, введем имя актера — Student.
4. Теперь опишем содержание актера Student. Такое описание можно сделать на странице Properties в окне, расположенном под диаграммой (изображение актера должно оставаться выделенным). В левой части страницы выбираем закладку Documentation, а в расположенное справа поле заносим текст описания:


**Рис. 21.14.** Окно диаграммы Use Case

«Студент — это человек, обучающийся в университете». В результате диаграмма Use Case примет вид, представленный на рис. 21.15.

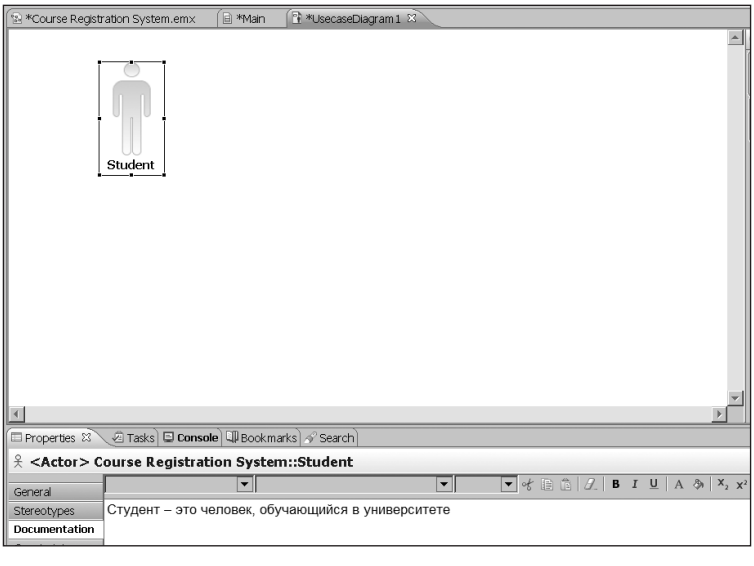


**Рис. 21.15.** Введение актера Student в диаграмму Use Case

Действуя аналогично, поместим в диаграмму трех других актеров (Professor, Registrar и Billing System — профессор, регистратор, учетная система). Для документирования этих актеров могут использоваться следующие тексты:

- «Профессор — это человек, который читает лекции в университете».
- «Регистратор — это человек, управляющий системой регистрации курсов».
- «Учетная система — это внешняя система, отвечающая за выписку счетов».

На этом этапе диаграмма имеет вид, показанный на рис. 21.16.

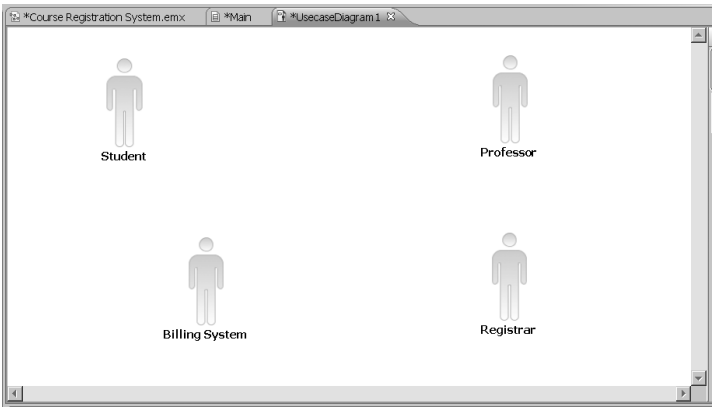


**Рис. 21.16.** Четыре актера в диаграмме Use Case

Далее для каждого актера нужно определить соответствующие элементы Use Case. Элемент Use Case представляет определенную часть функциональности, обеспечиваемой системой. Элементы Use Case можно идентифицировать путем рассмотрения каждого актера и его взаимодействия с системой. В нашей модели актер Student хочет регистрироваться на курсы (элемент Use Case Register for Courses). Актер Billing System получает информацию о регистрации. Актер Professor хочет запросить список курса (элемент Request Course Roster). Наконец, актер Registrar должен управлять учебным планом (элемент Manage Curriculum).

Процедура введения в диаграмму элементов Use Case и их описаний аналогична рассмотренной для актеров. Так, для элемента Register for Courses нужно:

1. На палитре инструментов щелкнуть по значку элемента Use Case.
2. Для добавления элемента щелкнуть в нужном месте диаграммы Use Case.
3. Пока элемент Use Case остается выделенным, ввести имя Register for Courses.
4. На странице Properties под окном диаграммы выбрать закладку Documentation и в расположенное справа поле внести текст описания: «Запускается студентом. Позволяет создавать, удалять, изменять и/или просматривать расписание студента в указанном семестре. Взаимодействует с учетной системой».

Диаграмма после этих операций показана на рис. 21.17.

Аналогичные действия выполняются для ввода других элементов Use Case (Request Course Roster, Manage Curriculum), при этом первоначальные описания элементов могут иметь следующий вид:

- «Запускается профессором. Позволяет выбирать курсы, которые он будет читать в указанном семестре, и получать расписание занятий» (для элемента Request Course Roster).
- «Запускается регистратором. Позволяет составлять каталог курсов на семестр, управлять информацией об учебных курсах, а также о студентах и преподавателях, работающих с системой» (для элемента Manage Curriculum).

Вид диаграммы после описанных действий представлен на рис. 21.18.

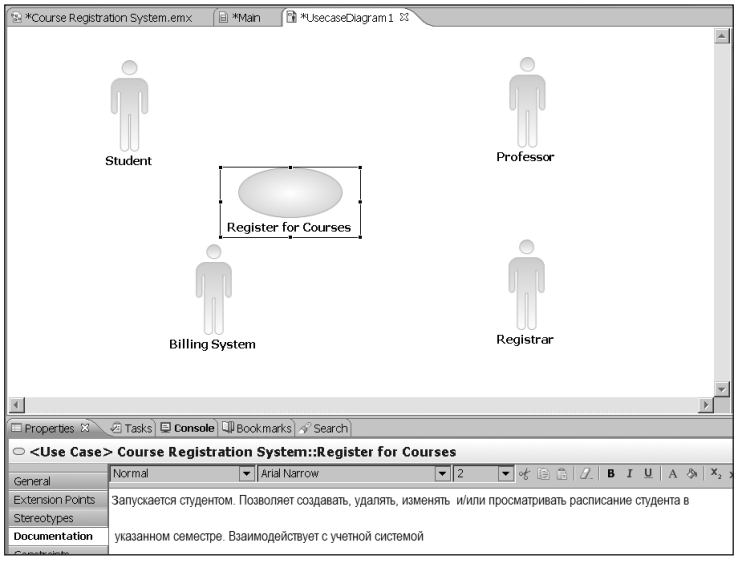


**Рис. 21.17.** Введение элемента Use Case Register for Courses в диаграмму Use Case

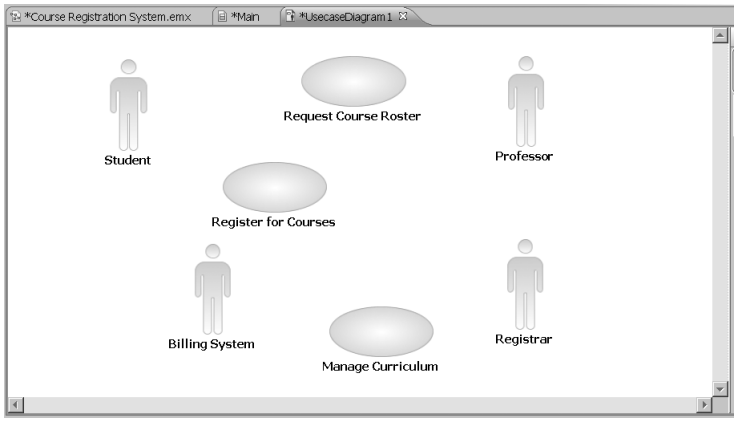


**Рис. 21.18.** Четыре актера и три элемента Use Case

Далее между актерами и элементами Use Case рисуются отношения. По умолчанию палитра инструментов предлагает ненаправленные ассоциации (значок Association). Направление взаимодействия показывают однонаправленные стрелки. С их помощью задается инициатор взаимодействия. Для включения режима направленных ассоциаций нужно щелкнуть по треугольнику слева от пиктограммы Association, после чего ниже появится значок Directed Association (рис. 21.19). Щелчок по этому значку включает режим направленных ассоциаций. Режим остается активным до возврата (оператором) режима ненаправленных ассоциаций.

В системе регистрации курсов актер Student инициирует элемент Use Case Register for Courses, который, в свою очередь, взаимодействует с актером Billing System. Актер Professor инициирует элемент Use Case Request Course Roster. Актер Registrar инициирует элемент Use Case Manage Curriculum (рис. 21.20).

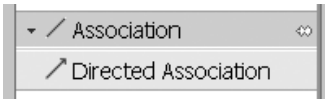


**Рис. 21.19.** Варианты ассоциаций

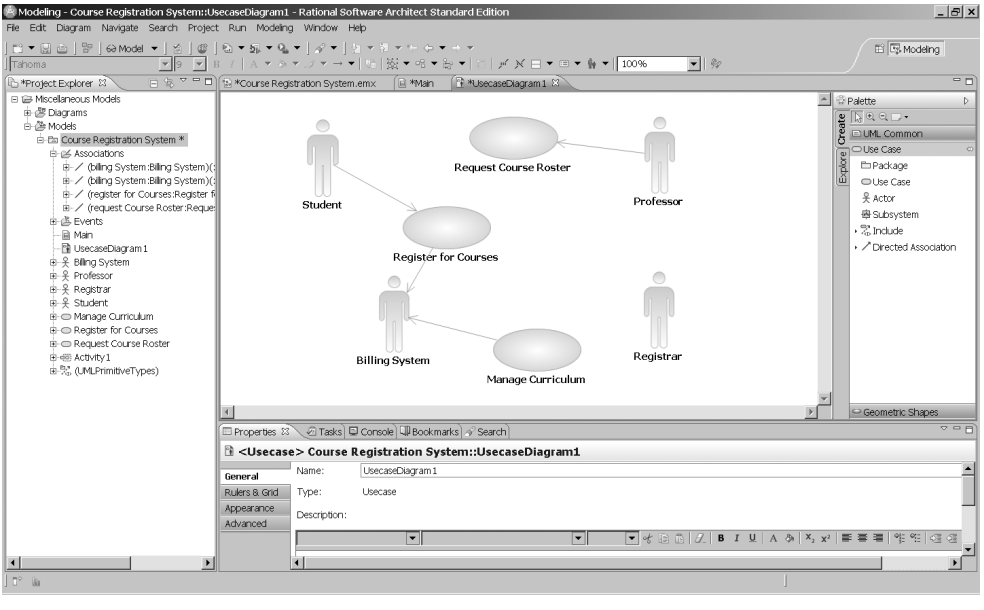


**Рис. 21.20.** Отношения между актерами и элементами Use Case

Для внесения отношений в диаграмму Use Case нужно выполнить следующие операции:

1. На палитре инструментов щелкнуть по значку однонаправленной ассоциации (Directed Association).
2. Установить курсор на изображение актера Student, нажать левую кнопку мыши и, удерживая ее, перетащить курсор на элемент Use Case Register for Courses, после чего отпустить кнопку.
3. На панели инструментов щелкнуть по значку однонаправленной ассоциации (Directed Association).
4. Установить курсор на изображение элемента Use Case Register for Courses, нажать левую кнопку мыши и, удерживая ее, перетащить курсор на актера Billing System, после чего отпустить кнопку.

5. Повторить аналогичные действия для ввода других ассоциаций (от актера Professor к элементу Use Case Request Course Roster и от актера Registrar к элементу Use Case Manage Curriculum).

**ПРИМЕЧАНИЕ**

В ходе установления ассоциации после отпускания кнопки мыши на изображении линии появляется окошко, в которое можно занести название ассоциации.

Обратите внимание, что все введенные актеры и элементы Use Case, а также установленные между ними ассоциации появляются и в браузере проекта (в левой части окна RSA).

## Создание диаграммы последовательности

Функциональность элемента Use Case отображается графически в диаграмме последовательности (Sequence diagram). Эта диаграмма показывает упорядоченный по времени поток сообщений между участниками взаимодействия, например при добавлении студента к списку слушателей определенного учебного курса. Иными словами, словесное описание действий в элементе Use Case заменяется набором графических фигур — прямоугольников и стрелок диаграммы последовательности. Для создания диаграммы последовательности нужно определить обобщенные объекты (роли), называемые участниками взаимодействия. Такие обобщенные объекты размещаются в верхней части диаграммы и изображаются прямоугольниками, в которых указаны имя объекта и класса, разделенные двоеточием. Каждый объект имеет собственную временную ось (lifeline) в виде пунктирной линии под его прямоугольником. Сообщения между объектами представляются стрелками, начинающимися на пунктирной линии под отправителем сообщения и завершающимися на пунктирной линии под получателем этого сообщения. Еще одна особенность диаграммы — это спецификация выполнения, изображаемая в виде прямоугольника, вытянутого вдоль временной оси. Верхняя грань этого прямоугольника определяет момент начала действия, порождаемого сообщением, а нижняя грань — момент окончания этого действия.

Рассмотрим процесс создания диаграммы последовательности Add a Course для элемента Use Case Register for Courses. Начальные действия при создании диаграммы последовательности напоминают те, что производились при создании диаграммы Use Case, но в данном случае выбираем пункт Sequence Diagram (рис. 21.21).

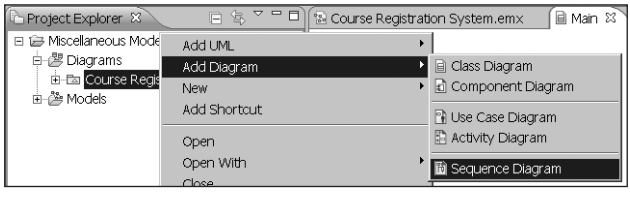


**Рис. 21.21.** Создание диаграммы последовательности

Появляющееся в браузере имя Interaction1 заменяем на Add a Course.

На экране возникает окно диаграммы последовательности, справа от которого располагается палитра инструментов. Заметим, что палитра теперь содержит иные инструменты, а именно те, которые используются при создании диаграмм последовательности.

Теперь мы будем добавлять в диаграмму такие обобщенные объекты и сообщения, которые реализуют необходимую функциональность. Сценарий, который мы собираемся формализовать с помощью диаграммы последовательности, уже существует — он является фрагментом текста, который содержит спецификация элемента Use Case Register for Courses.

Сценарий инициируется актером Student. Перетащим этого актера из браузера в диаграмму (рис. 21.22) и присвоим студенту имя John:

1. Нажмем левую кнопку мыши на значке актера Student в браузере и перетащим его в диаграмму последовательности, после чего отпустим кнопку.
2. Щелкаем по значку актера в диаграмме последовательности и вводим имя — John.


**Рис. 21.22.** Диаграмма последовательности с актером, инициирующим взаимодействие

В этом сценарии студент должен заполнить информацией (fill in info) регистрационную форму (registration form), после чего форма предъявляется на рассмотрение (submitted). Очевидно, что необходим обобщенный объект, который принимает информацию от студента. Создадим его, присвоим имя registration form и добавим в диаграмму два сообщения «fill in information» и «submit». С этой целью:

1) в палитре инструментов щелкнем по значку Lifeline;
2) щелкнем в окне диаграммы. На экране появляется меню выбора типа обобщенного объекта (рис. 21.23). Так как соответствующий класс мы еще не определяли, то выберем неопределенный тип (Unspecified Type);
3) в появляющийся прямоугольник, пока он остается выделенным, введем имя registration form;
4) непосредственно под значком Lifeline на палитре инструментов располагается пиктограмма выбора типа сообщения. По умолчанию предлагается синхронное

сообщение. Положим, нас этот тип не устраивает, поэтому щелкаем по треугольнику слева. В результате откроется меню возможных вариантов (рис. 21.24), в котором выбираем нужный тип (Asynchronous Message);

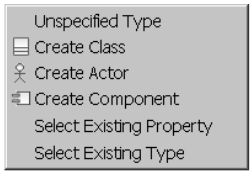


**Рис. 21.23.** Меню выбора типа обобщенного объекта

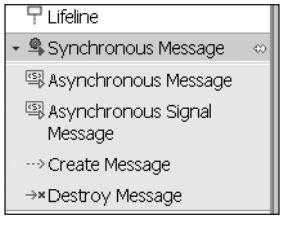


**Рис. 21.24.** Меню выбора типа сообщения

5) нажмем левую кнопку мыши на пунктирной линии под актером John и, удерживая кнопку нажатой, перетащим стрелку на пунктирную линию под прямоугольником registration form;
6) пока стрелка остается выделенной, введем имя сообщения — fill in information;
7) повторяем шаги 4–6 для создания сообщения submit.

После перечисленных действий диаграмма последовательности приобретет вид, представленный на рис. 21.25.


**Рис. 21.25.** Диаграмма последовательности после добавления registration form

Очевидно, что registration form является промежуточным звеном в цепи передачи информации. Будем считать, что John хочет записаться на курс math 101. Следующее звено — manager, которому registration form должна послать соответствующее сообщение. Чтобы отразить это на диаграмме (рис. 21.26), нужно добавить обобщенный объект manager и передаваемое ему сообщение:

1. На палитре инструментов щелкнем по значку Lifeline.
2. Для добавления обобщенного объекта щелкнем в нужном месте диаграммы.
3. Пока обобщенный объект остается выделенным, введем имя manager.
4. На палитре инструментов щелкнем по значку сообщения (Asynchronous Message).
5. Щелкнем по пунктирной линии под registration form и перетащим стрелку на пунктирную линию под прямоугольником manager.
6. Пока стрелка остается выделенной, введем имя сообщения — add John to math 101.


**Рис. 21.26.** Диаграмма последовательности после добавления manager

Дальнейшее построение диаграммы выполняется аналогичными действиями, поэтому опишем лишь последующие шаги сценария и отобразим диаграмму последовательности после каждого шага.

Обобщенный объект manager взаимодействует как с регистрационной формой, так и с набором учебных курсов. Для обслуживания нашего студента должен существовать обобщенный объект math 101, которому manager обязан передать, что John должен быть добавлен к курсу. Это реализуется с помощью сообщения add John (рис. 21.27).

Обобщенный объект math 101 не принимает самостоятельных решений о возможности добавления студентов. Этим занимается обобщенный объект класса Предложение курса, который назовем section 1. Участник взаимодействия math 101 обращается к section 1 с предложением добавить студента John, используя для этого два сообщения: accepting students? и add John (рис. 21.28).

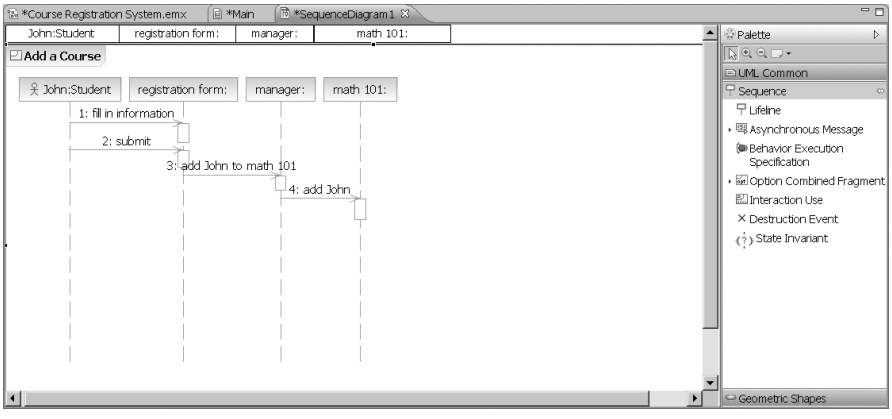


**Рис. 21.27.** Диаграмма последовательности после добавления math 101

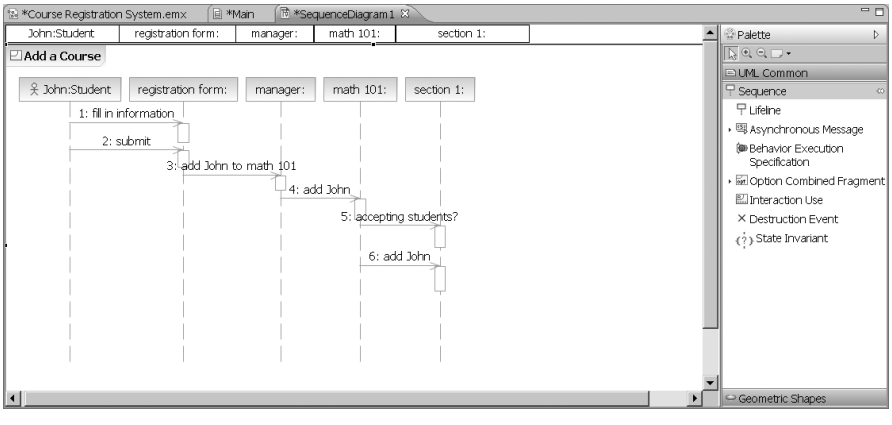


**Рис. 21.28.** Диаграмма последовательности после добавления section 1

Вопросами оплаты учебы занимается учетная система, а уведомляет ее о необходимости выписки счета manager. После того как последний удостоверился, что студенту John предоставляется возможность изучать курс math 101 (на диаграмме рис. 21.29 это не показано), он уведомляет учетную систему, представленную на диаграмме обобщенным объектом bill (billing system). В результате диаграмма последовательности Add a Course обретет вид, приведенный на рис. 21.29.

На этом предварительный этап создания диаграммы последовательности будем считать завершенным. Дальнейшую коррекцию диаграммы произведем после определения классов, их атрибутов и операций.

## Создание диаграммы классов

Объекты из диаграмм последовательности группируются в классы. Основываясь на нашей диаграмме последовательности, мы можем идентифицировать следующие объекты и классы:

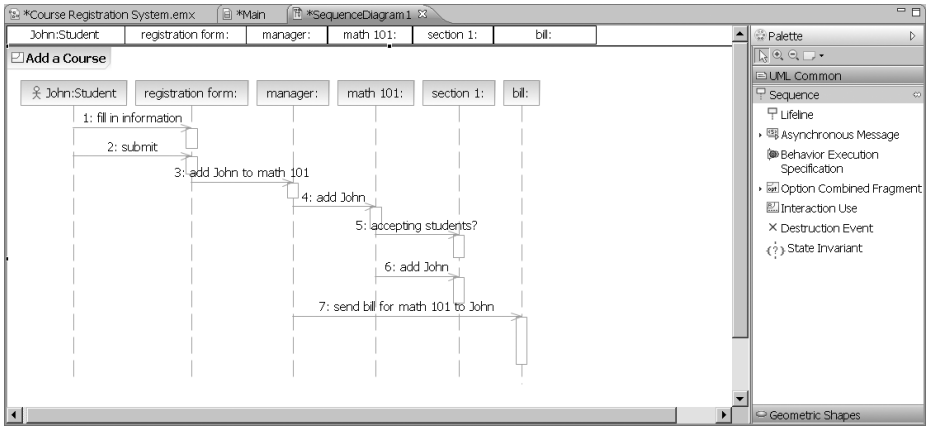


**Рис. 21.29.** Диаграмма последовательности после добавления bill

- registration form является обобщенным объектом класса RegForm;
- manager является обобщенным объектом класса Manager;
- math 101 является обобщенным объектом класса Course;
- section 1 является обобщенным объектом класса CourseOffering;
- bill является интерфейсом к внешней учетной системе, поэтому мы будем использовать имя BillingSystem как имя его класса.

Начнем с создания этих классов.

1. В браузере сделаем щелчок правой кнопкой мыши по названию нашей модели, а в открывшемся на пункте Add UML подменю выберем позицию Class (рис. 21.30).

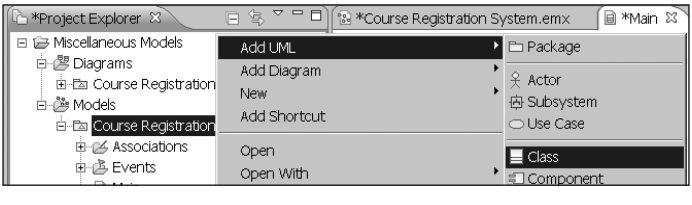


**Рис. 21.30.** Создание нового класса

2. Пока значок класса, появившийся в браузере, остается выделенным, введем имя класса RegForm.
3. Повторим предыдущие шаги для добавления других классов: Manager, Course, CourseOffering и BillingSystem.

В браузере появляются значки и названия созданных классов (рис. 21.31).

После создания классов они описываются (документируются). Например, у класса может быть такое описание: «CourseOffering — это предлагаемый университетом курс». Порядок добавления описания:

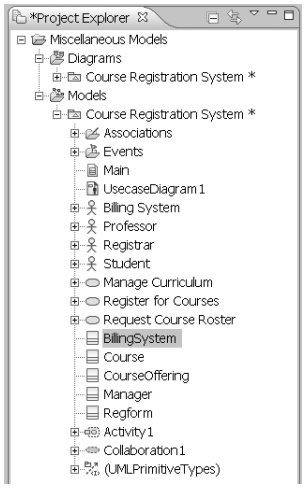


**Рис. 21.31.** Представление созданных классов в браузере

1. В браузере выделяется класс CourseOffering.
2. В окне свойств класса (оно располагается в нижней части экрана) выбирается страница Properties, а в ней — закладка Documentation.
3. В поле для текста описания вносится требуемый текст (рис. 21.32).

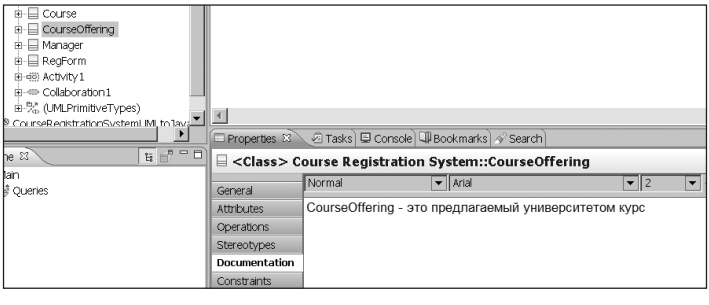


**Рис. 21.32.** Пример документирования класса

Следующий шаг — построение диаграммы классов. Открытие диаграммы классов производится аналогично открытию двух предыдущих диаграмм (рис. 21.33).

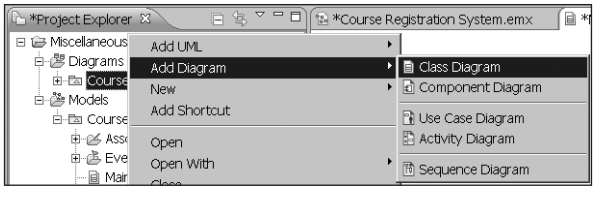


**Рис. 21.33.** Создание диаграммы классов

На экране появится окно диаграммы классов, а слева от него — соответствующая палитра инструментов. Теперь перенесем классы из браузера в диаграмму классов:

1. Удерживая нажатой клавишу Ctrl, последовательно отметим в браузере требуемые классы, после чего перетащим их в окно диаграммы.
2. Переупорядочим классы в диаграмме (выделяя конкретный класс и перетаскивая его на новое место).

**ПРИМЕЧАНИЕ**

Классы можно добавлять в диаграмму перетаскиванием их из окна браузера по одному.

На данном этапе диаграмма классов имеет вид, показанный на рис. 21.34.

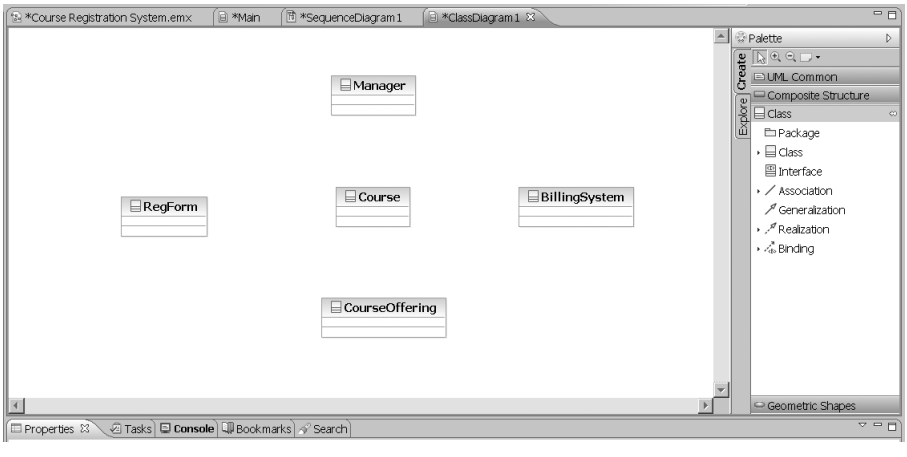


**Рис. 21.34.** Окно и палитра инструментов диаграммы классов

Для создания новых типов моделирующих элементов в UML используется понятие стереотипа. С помощью стереотипа можно «нагрузить» элемент новым смыслом. Используем предопределенный стереотип Interface для класса BillingSystem, так как этот класс определяет только интерфейс к внешней учетной системе (billing system).

1. Выделим на диаграмме класс BillingSystem.
2. На странице Properties выбираем закладку Stereotypes и в поле Keywords заносим слово-стереотип Interface.

Обратим внимание на то, что новый стереотип класса отразится в виде соответствующей надписи на изображении класса в диаграмме (рис. 21.35).

Для определения взаимодействия объектов нужно указать отношения между соответствующими классами. Для того чтобы увидеть, как объекты должны «разговаривать» друг с другом, исследуются диаграммы последовательности. Если объекты должны «разговаривать», то должен быть путь для коммуникации между их классами. Двумя типами структурных отношений являются ассоциации и агрегации.

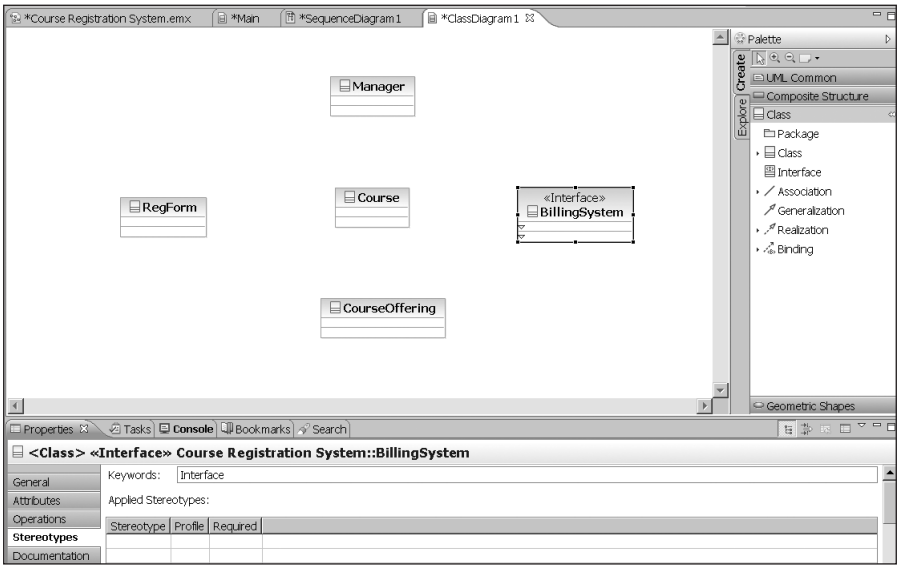


**Рис. 21.35.** Класс Billing System после введения стереотипа Interface

Ассоциация определяет соединение между классами. Исследуя диаграмму последовательности Add a Course, мы можем определить существование следующих ассоциаций: от RegForm к Manager, от Manager к Course и от Manager к BillingSystem (рис. 21.36).

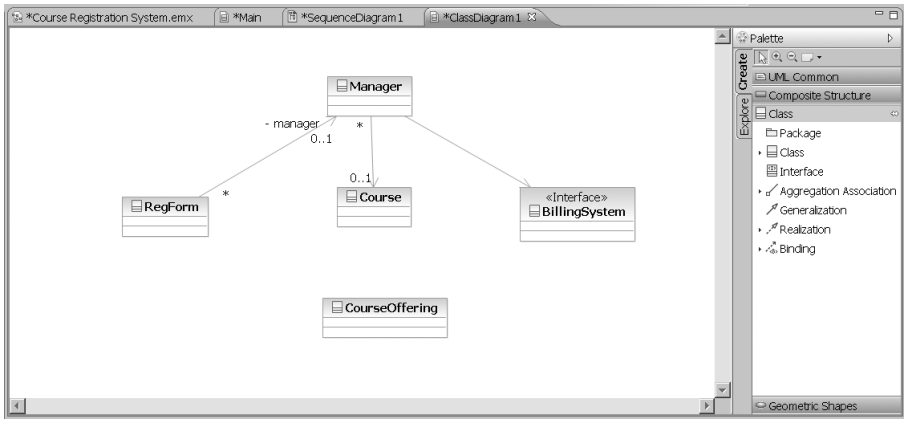


**Рис. 21.36.** Ассоциации между классами

1. На палитре инструментов располагается значок ненаправленной ассоциации. Поскольку этот тип ассоциации нас не устраивает, щелкнем по треугольнику слева. В результате откроется меню вариантов (рис. 21.37), в котором выбираем нужную направленную ассоциацию (Directed Association).

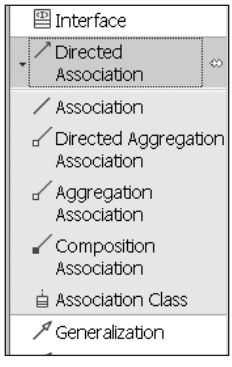


**Рис. 21.37.** Варианты ассоциаций между классами

2. Щелкнем по классу RegForm и перетащим линию ассоциации на класс Manager.
3. Повторим предыдущие шаги для ввода следующих отношений:

- от Manager к Course;
- от Manager к BillingSystem.

Ассоциации задают пути между объектами-партнерами одинакового уровня.

Агрегация фиксирует неравноправные связи. Она показывает отношение между целым и его частями. Создадим отношение агрегации между классом Course и классом CourseOffering (рис. 21.38), так как предложение курса CourseOfferings является частью агрегата — класса Course.

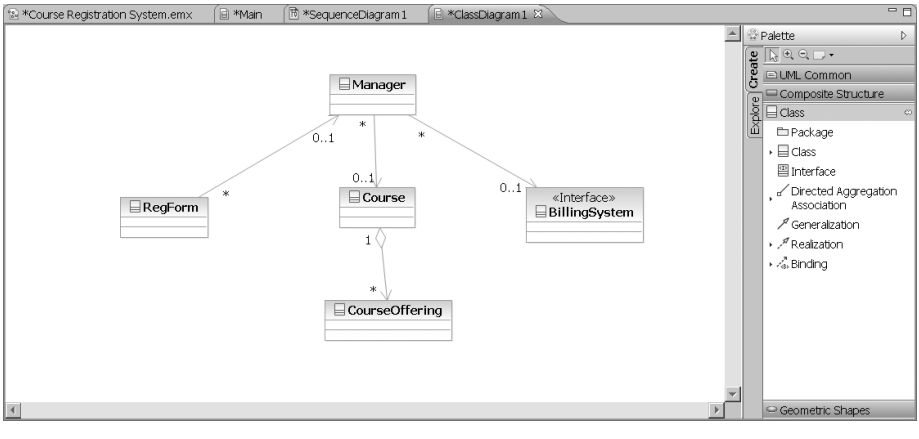


**Рис. 21.38.** Отношение агрегации

1. На палитре инструментов выберем значок направленной агрегации (Directed Agregation Association).
2. Щелкнем по классу, представляющему целое — Course.
3. Перетащим линию агрегации на класс CourseOffering, представляющий часть.

Объем информации, характеризующей стрелки отношений, может быть различным, как это видно на рис. 21.36, 21.38. Желательная степень подробности задается пунктом Filters контекстного меню, открывающегося при щелчке правой кнопкой по линии отношения (рис. 21.39).

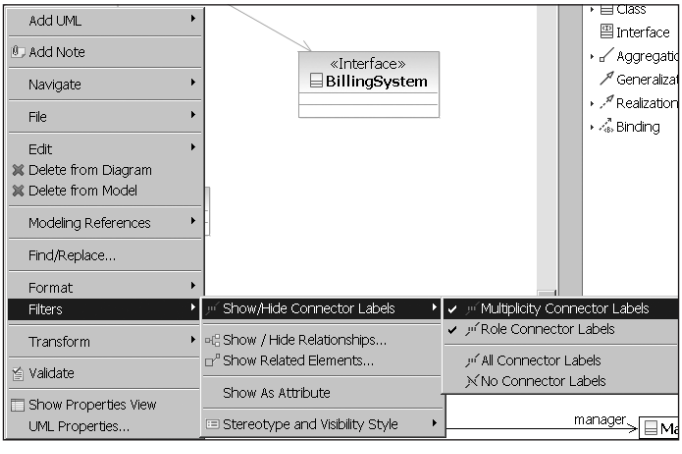


**Рис. 21.39.** Выбор степени подробности информации об отношении

Для отображения того, «как много» объектов участвует в отношении, к ассоциациям и агрегациям диаграммы могут добавляться индикаторы мощности. Порядок добавления индикаторов мощности:

1. Щелкнуть левой кнопкой по линии агрегации между классами Course и CourseOffering.
2. На странице Properties под окном диаграммы (рис. 21.40) выбрать закладку General.

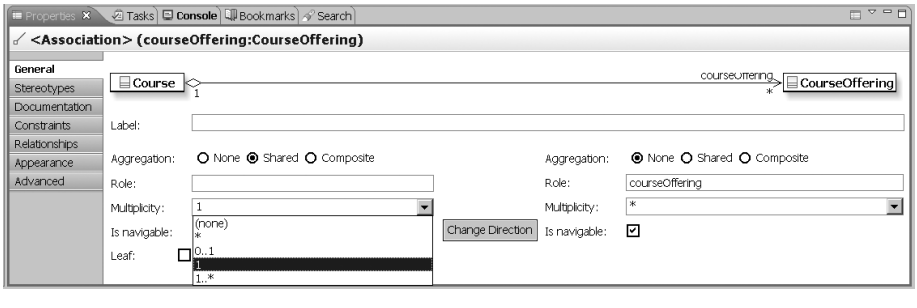


**Рис. 21.40.** Индикаторы мощности

3. В левом раскрывающемся списке Multiplicity задать мощность на левом полюсе агрегации.
4. В правом раскрывающемся списке Multiplicity задать мощность на правом полюсе агрегации.

5. Заметим, что в этой закладке можно задавать и другие параметры. Например, роли для каждого полюса.

Вспомним, что задание имени — это первый из трех шагов определения класса. Любой класс должен инкапсулировать в себе структуру данных и поведение, которое определяет возможности обработки этой структуры. Примем, что на фиксацию структуры ориентируется второй шаг, а на фиксацию поведения — третий шаг.

Структура класса представляется набором его атрибутов. Структура находится путем исследования предметных требований и соглашений между разработчиками и заказчиками. В нашей модели каждое предложение курса (CourseOffering) является атрибутом (attribute) класса-агрегата Course.

Конечно, класс CourseOffering тоже имеет атрибуты. Определим один из них — количество студентов (numberStudents):

1. В диаграмме классов щелкнем по классу CourseOffering.
2. В появляющемся графическом контекстном меню (рис. 21.41) выберем левый значок (Add Attribute). Это приведет к добавлению в класс атрибута.

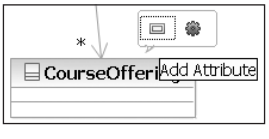


**Рис. 21.41.** Добавление атрибута

3. Пока новый атрибут остается выделенным, введем его имя — numberStudents. Введенный атрибут появится в изображении класса на диаграмме (рис. 21.42).

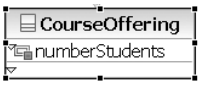


**Рис. 21.42.** Класс CourseOffering с добавленным атрибутом numberStudents

Итак, два шага формирования класса сделаны. Перейдем к третьему шагу — заданию поведения класса.

Поведение класса представляется набором его операций. Исходная информация об операциях класса находится в диаграммах последовательности. В операции отображаются сообщения из диаграмм последовательности. При этом обычно сообщения переименовываются — производится согласование имени сообщения и имени операции Причины переименования просты и понятны. Во-первых, имя операции должно отражать ее принадлежность к классу (а не к источнику соответствующего сообщения). Во-вторых, имя операции должно указывать на ее обязанность, а не на способ ее реализации. В-третьих, имя должно быть допустимым с точки зрения синтаксиса языка программирования, который будет использоваться для кодирования класса.

Для примера наполним операциями класс CourseOffering. Как видно из диаграммы последовательности, классу присущи две операции, представленные сообщениями add John и accepting students?. Сначала создадим операцию, соответствующую

первому из этих сообщений, присвоив ей имя add. Процесс похож на наполнение класса атрибутами.

1. В диаграмме классов щелкнем по классу CourseOffering.
2. В появляющемся графическом контекстном меню (рис. 21.43) выберем правый значок (Add Operation). Это приведет к добавлению в класс операции.

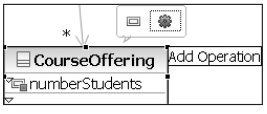


**Рис. 21.43.** Добавление операции

3. Пока новая операция остается выделенной, вводим ее имя — add.
4. Действуя аналогично, дополним класс второй операцией, представляющей сообщение accepting students?, присвоив этой операции имя offeringOpen.

   Теперь изображение класса CourseOffering примет следующий вид (рис. 21.44).

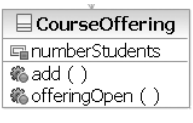


**Рис. 21.44.** Класс CourseOffering с добавленными атрибутом и операциями

Теперь мы можем привязать обобщенные объекты из диаграммы последовательности к конкретным классам.

Откроем диаграмму последовательности. Будем поочередно перетаскивать значки классов (из браузера проектов) на прямоугольники соответствующих обобщенных объектов. Результат перетаскивания демонстрирует рис. 21.45.

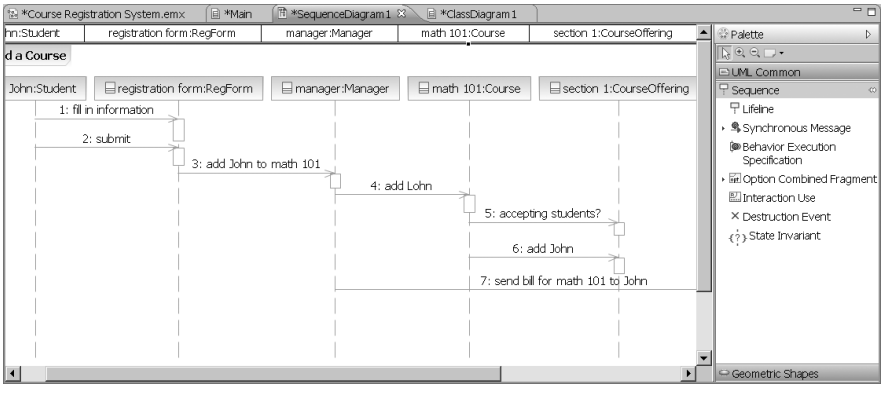


**Рис. 21.45.** Привязка обобщенных объектов к классам

Видим, что имя каждого обобщенного объекта удлинилось, в нем появились две части, разделенные двоеточием. Слева от двоеточия записывается собственное имя объекта, а справа — имя класса.

В заключение согласуем имена сообщений и операций классов. Продемонстрируем согласование на примере сообщения accepting students?.

1. Выполним двойной щелчок по имени сообщения accepting students?. Появится диалоговое окошко, предлагающее два варианта: создать новую операцию (Create New Operation), выбрать существующую операцию (Select Existing Operation). Для варианта «существующей операции» указаны две операции класса CourseOffering.

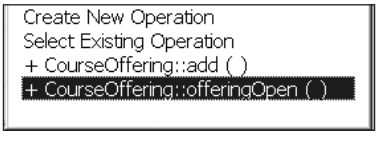


**Рис. 21.46.** Привязка к операции

2. Как было определено ранее, этому сообщению соответствует операция offeringOpen() класса CourseOffering (последняя строка в диалоговом окошке на рис. 21.46). Выполним щелчок по этой строке. Имя сообщения на диаграмме последовательности заменяется именем операции (рис. 21.47).

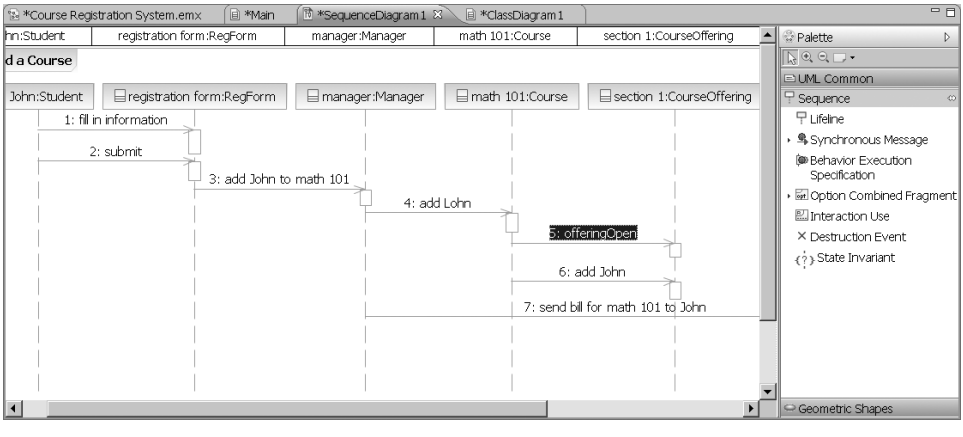


**Рис. 21.47.** Диаграмма последовательности после привязки к операции класса

Аналогично поступаем с остальными сообщениями в диаграмме последовательности.

## Генерация программного кода

Прежде чем станет возможным преобразование UML-модели в программный код, нужно создать так называемую *конфигурацию трансформации* (transformation configuration). Она включает информацию, используемую в процессе трансформации: уникальное имя, источник и целевой объект трансформации, а также ряд свойств, специфичных для данного вида трансформации.

В нашем примере мы рассмотрим трансформацию разработанной модели UML в программный код на языке Java, в результате чего элементы модели отображаются в текст Java-классов. Генерацию программного кода начнем с создания конфигурации трансформации.

1. Выбираем в браузере проектов модель Course Registration System, щелкнув по ней правой кнопкой мыши.
2. В открывшемся меню выбираем пункт New, а в последующем подменю — пункт Other (рис. 21.48).

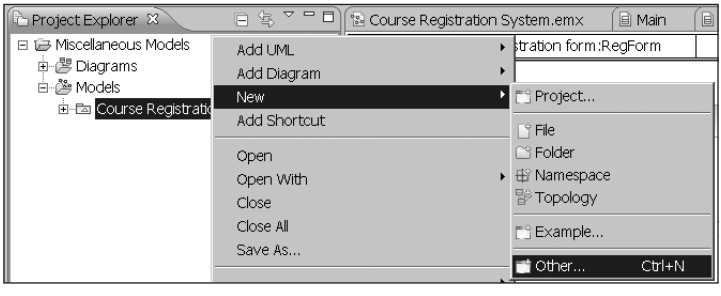


**Рис. 21.48.** Переход к созданию конфигурации трансформации

3. В появляющемся на экране окне Select a Wizard выбираем пункт Transformations > Transformation Configuration (рис. 21.49) и делаем щелчок по Next.

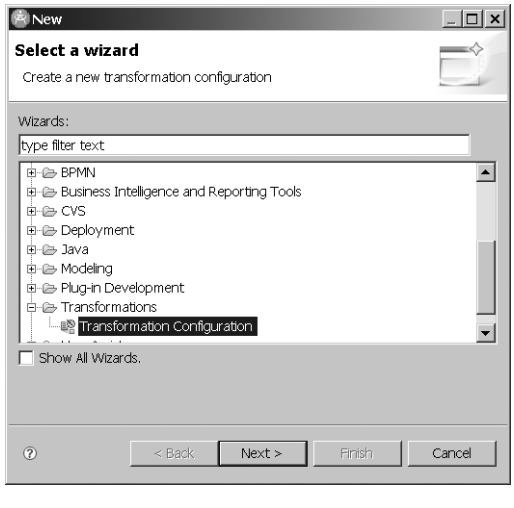


**Рис. 21.49.** Первый шаг создания конфигурации трансформации

4. В появившемся окне Specify a Configuration Name and Transformation (рис. 21.50) в списке Transformation выбираем Java Transformations > UML to Java, а в поле Name заносим имя конфигурации трансформации CourseRegistrationSystemUMLtoJava. Делаем щелчок по Next.

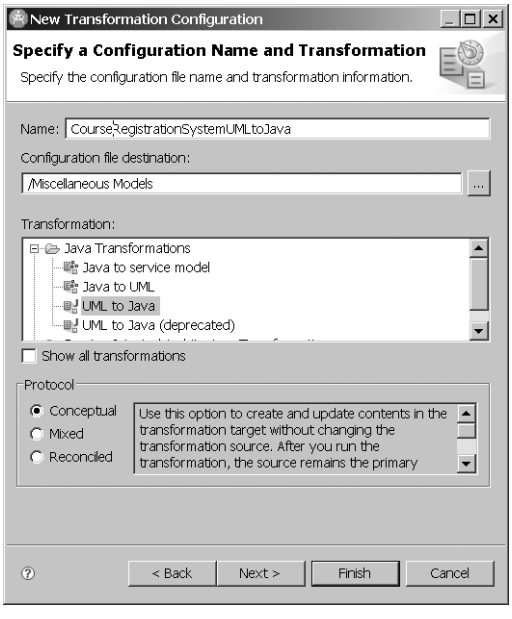


**Рис. 21.50.** Определение имени и типа конфигурации трансформации

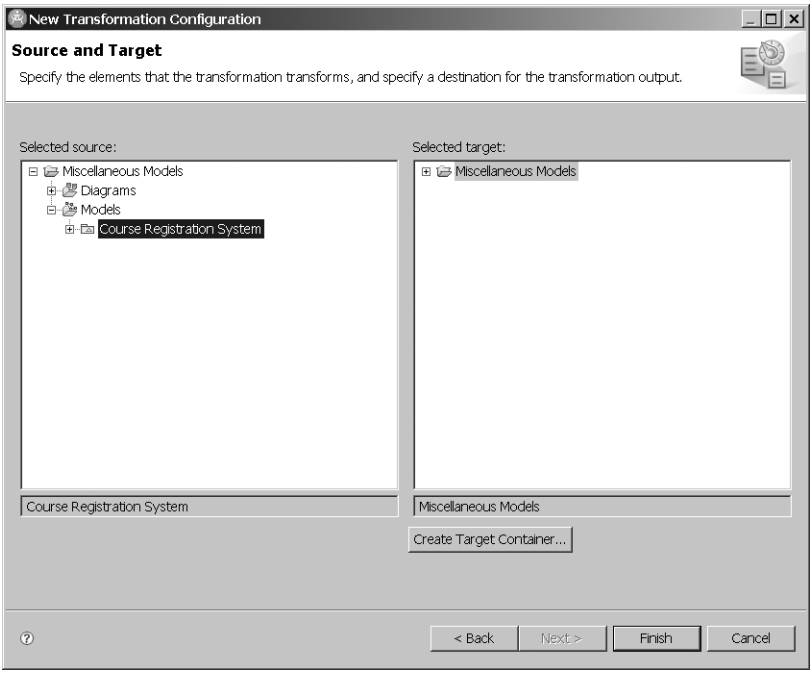


**Рис. 21.51.** Определение преобразуемой модели UML и местоположения конфигурации трансформации

5. В появляющемся окне Source and Target (рис. 21.51) в качестве источника выбираем нашу модель UML Course Registration Model. Для хранения конфигурации трансформации создадим контейнер, для чего щелкнем по кнопке Create Target Container... .
6. В очередном окне (Create Java Project) указываем имя создаваемого Java-проекта (назовем его CourseRegisterSystemJavaProject) и щелкаем по Finish (рис. 21.52). После возврата к окну Source and Target также щелкаем по Finish.

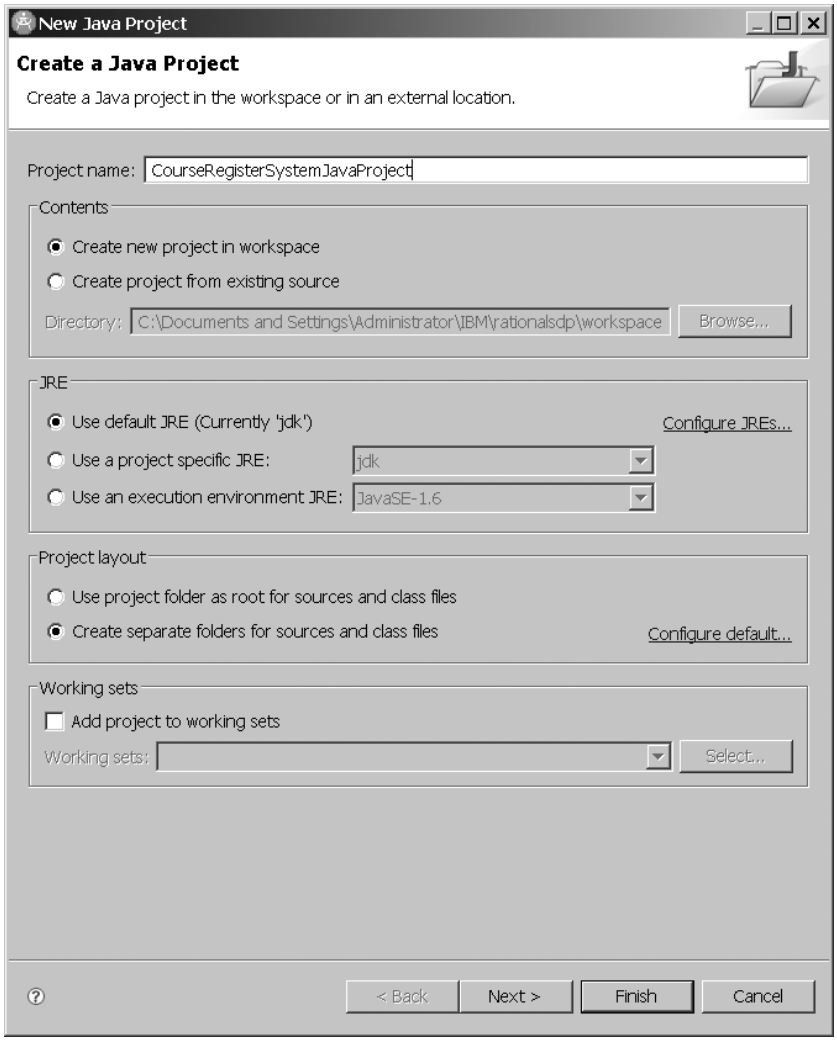


**Рис. 21.52.** Завершение формирования конфигурации трансформации

Мы создали конфигурацию трансформации, которая имеет название, а также CourseRegisterSystemJavaProject, но файлы Java еще не сгенерированы. Поэтому сле-

дующий шаг — это преобразование модели UML в Java-код. После выполненных шагов изображение на экране соответствует рис. 21.53.

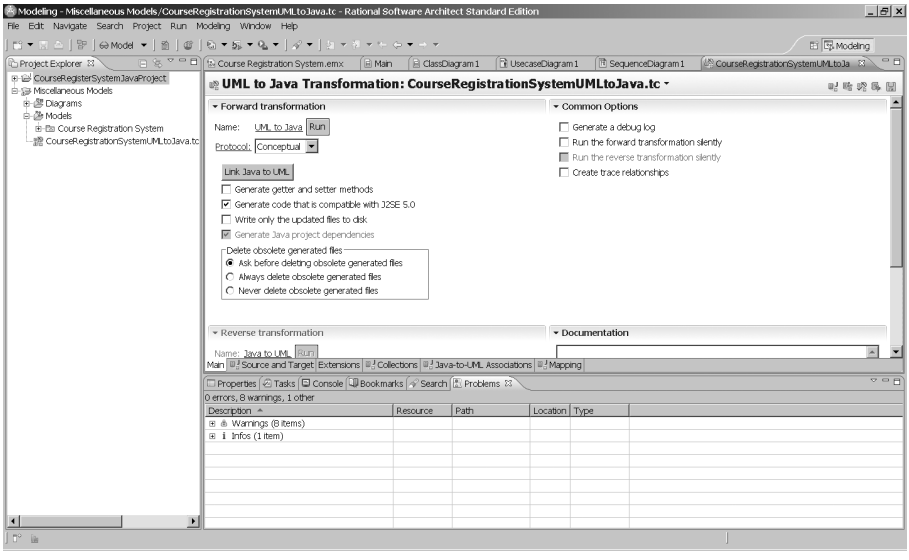


**Рис. 21.53.** Экран RSA после завершения создания конфигурации трансформации

Для выполнения трансформации (рис. 21.54):

1) сделаем правый щелчок по названию файла конфигурации CourseRegistrationSystemUMLtoJava.tc в браузере проектов;
2) в появившемся меню выберем пункт Transforn > UML to Java.

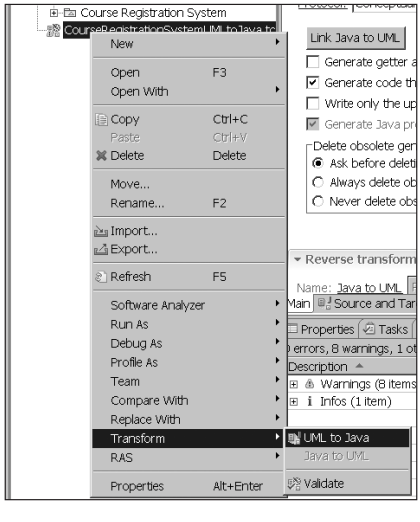


**Рис. 21.54.** Переход к операции трансформации

В результате будет сгенерирован код Java, который можно посмотреть в проекте Java. На рис. 21.55 показан код Java, соответствующий классу CourseOffering из модели UML.

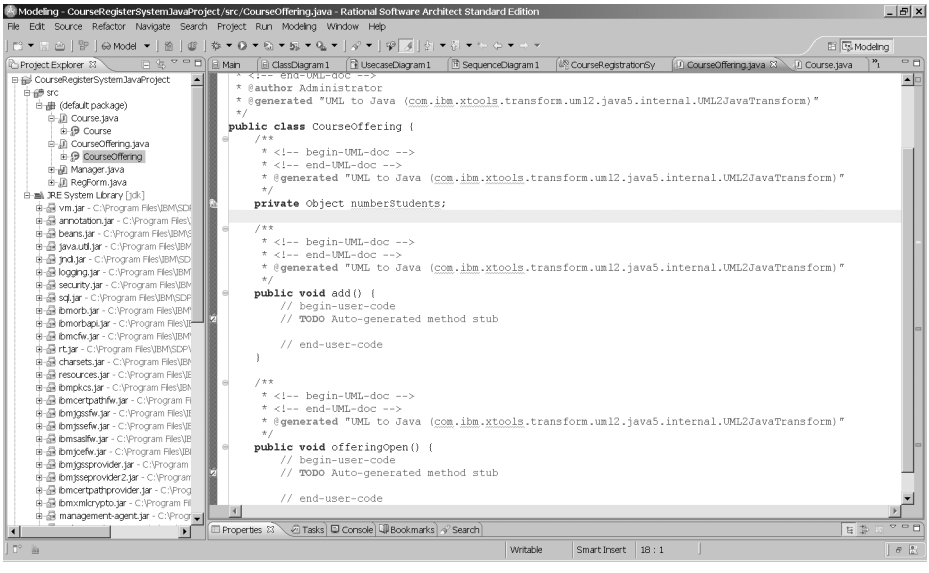


**Рис. 21.55.** Сгенерированный код Java для класса CourseOffering

В процессе генерации RSA отображает логическое описание класса в каркас программного кода — в коде появляются языковые описания имени класса, атрибутов класса и заголовки методов. Кроме того, для описания тела каждого метода формируется программная заготовка. Появляются и программные связи классов. Подразумевается, что программист будет дополнять этот код, работая в конкретной среде программирования. В качестве примера в листинге 21.1 приведен сгенерированный каркас кода для класса CourseOffering.

**Листинг 21.1.** Программный код для класса CourseOffering

```
public class CourseOffering {
    private Object numberStudents;
     * <!-- begin-UML-doc -->
     * <!-- end-UML-doc -->
    public void add() {
        // begin-user-code
        // TODO Auto-generated method stub
        // end-user-code
    }
    public void offeringOpen() {
        // begin-user-code
        // TODO Auto-generated method stub
        // end-user-code
    }
}
```

Видим, что в коде класса CourseOffering предусмотрен один приватный атрибут и два публичных метода.

Для поддержки однонаправленных ассоциаций RSA создает атрибут в классе-источнике сообщения. Например, в рассмотренной модели предусмотрена направленная ассоциация из класса RegForm в класс Manager, поэтому в коде класса RegForm обнаруживаем приватный атрибут manager:

```
public class RegForm {

    private Manager manager;
}
```

От класса Manager поддержка этой ассоциации не требуется.

## Трансформация программного кода в модель UML

Теперь выполним обратное преобразование Java-кода в модель UML, внеся небольшое изменение в текст для класса CourseOffering.

Например, добавим параметр и определим тип возвращаемого значения для метода add() (новый заголовок примет вид public int add(int a)), а также введем новый метод public void sub(int b).

Сформируем новую конфигурацию трансформации.

1. Выбираем в браузере проектов пункт CourseRegisterJavaProject, щелкнув по нему правой кнопкой мыши.
2. В открывшемся меню выбираем пункт New, а в последующем подменю — пункт Other.
3. В появляющемся на экране окне Select a Wizard выбираем пункт Transformations > Transformation Configuration и делаем щелчок по Next.
4. В появившемся окне Specify a Configuration Name and Transformation в списке Transformation выбираем Java Transformations > Java to UML, а в поле Name заносим имя конфигурации трансформации CourseRegistrationSystemJavatoUML. Делаем щелчок по Next. Следствием наших действий становится появление окна Source and Target.
5. В окне Source and Target в качестве источника выбираем CourseRegisterJavaProject. Для хранения формируемой конфигурации создадим контейнер, для чего щелкнем на кнопке Create Target Container... .
6. В появляющемся окне Create Model выбираем Standard Template (стандартный шаблон) и щелкаем по Next.
7. Выводится следующее диалоговое окно с тем же названием Create Model. Оно позволяет выбрать категорию создаваемой модели (раздел Categories) и один из шаблонов, предлагаемых для выбранной категории (раздел Templates). Для нашего примера выберем категорию Analysis and Design и шаблон Blank Analysis Package (пустой пакет анализа). В поле File name наберем название файла с нашей моделью Course Registration System UML Reverse Model.
8. Щелкнем по кнопке Finish. В итоге вернемся к окну Source and Target, которое теперь будет иметь вид, представленный на рис. 21.56.

9. В разделе Select Target выберем пункт Course Registration System UML Reverse Model и щелкнем по Finish. В результате создается файл CourseRegistrationSystemJavatoUML.tc, который можно увидеть в окне браузера проектов.

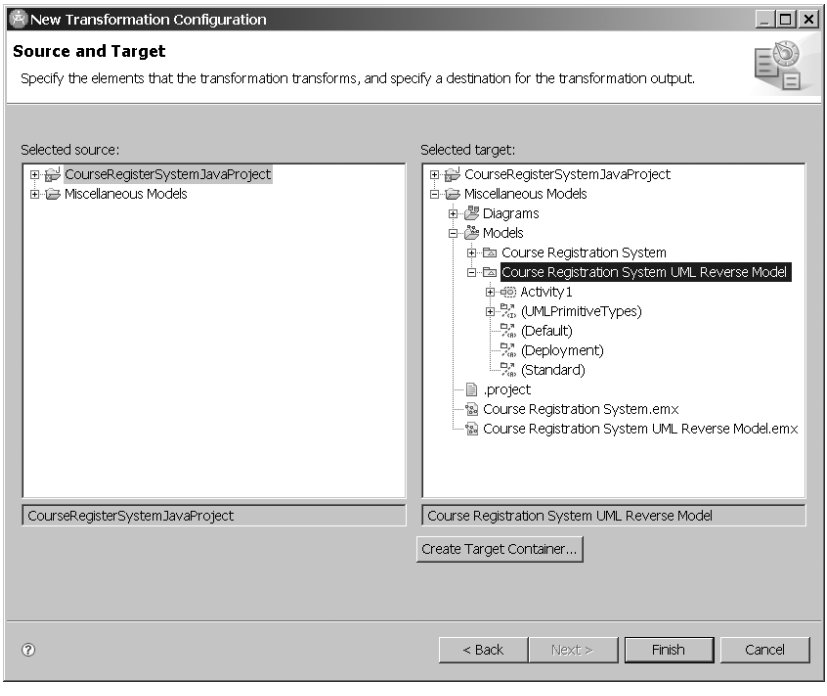


**Рис. 21.56.** Вид окна Source and Target после описанных действий

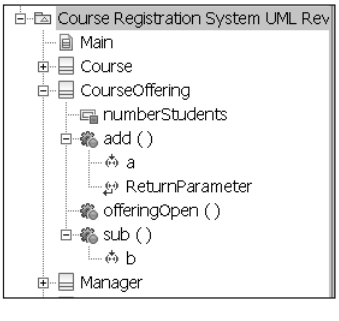


**Рис. 21.57.** Модель UML, созданная в результате трансформации

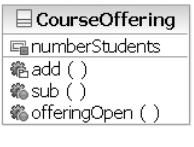


**Рис. 21.58.** Модифицированный класс CourseOffering

После создания конфигурации приступим к самой трансформации:

1. Сделаем правый щелчок по названию файла конфигурации CourseRegistrationSystemJavatoUML.tc в браузере проектов.

2. В появившемся меню выберем пункт Transforn > Java to UML.
3. На экране появляется окно Merge trasnformed model. К этому моменту сформирована промежуточная модель. В левом разделе окна показаны изменения, которые предлагается перенести в целевую модель UML. Убирая галочки в соответствующих позициях левого раздела, можно отказаться от переноса отдельных изменений.
4. Соглашаемся на полный перенос (все галочки установлены) и щелкаем по OK.

В результате наших действий создана модель Course Registration System UML Reverse Model, что подтверждает строка в браузере проектов (рис. 21.57).

Как видно, изменения Java-кода отражены в модели UML. Об этом свидетельствует и модифицированное представление класса (рис. 21.58).

# Приложение Б

# Внутренние и внешние метрики качества

Это приложение описывает внутренние и внешние метрики для оценки качества систем и программного обеспечения.

**Таблица Б.1.** Детализация характеристики «функциональная пригодность»

| Атрибут | Метрика |
|---|---|
| Функциональная полнота (functional coverage) | X = 1 – A / B<br>A = количество отсутствующих функций<br>B = количество заданных функций |
| Функциональная правильность (functional correctness) | X = 1 – A / B<br>A = количество неправильных функций<br>B = количество рассмотренных функций |
| Функциональная целесообразность (functional appropriateness) | X = 1 – A / B<br>A = количество отсутствующих или неправильных функций, требуемых для достижения цели пользователя<br>B = количество функций, требуемых для достижения целей конкретного пользователя |

**Таблица Б.2.** Детализация характеристики «эффективность исполнения»

| Атрибут | Метрика |
|---|---|
| Среднее время отклика (mean response time) | $X = \sum_{i=1}^{n} (A_i) / n$<br>$A_i$ = время отклика системы на действие или событие $i$<br>$n$ = количество измеренных событий отклика |
| Согласованность времени отклика (response time conformance) | X = A / B<br>A = среднее время отклика<br>B = заданное время отклика |

| Атрибут | Метрика |
|---|---|
| Среднее время оборота (mean turn-around time) | $X = \sum_{i=1}^{n} (B_i - A_i) / n$<br>$A_i$ = время начала работы $i$<br>$B_i$ = время завершения работы $i$<br>$n$ = количество измеренных работ |
| Согласованность времени оборота (turnaround time conformance) | X = A / B<br>A = среднее время оборота<br>B = заданное время оборота |
| Согласованность пропускной способности (throughput conformance) | $X = (\sum_{i=1}^{n} (A_i / B_i) / n) / C$<br>$A$ = количество работ, завершенных за время наблюдения<br>$B$ = период наблюдения<br>$C$ = заданная пропускная способность<br>$n$ = количество наблюдений |
| Среднее время использования процессора (mean processor utilization) | $X = \sum_{i=1}^{n} (A_i / B_i) / n$<br>$A_i$ = время процессора, фактически используемое для выполнения данного набора задач<br>$B_i$ = время для выполнения задач в период наблюдения $i$<br>$n$ = количество наблюдений |
| Средняя емкость используемой памяти (mean memory utilization) | $X = \sum_{i=1}^{n} (A_i / B_i) / n$<br>$A_i$ = размер памяти, фактически используемой для выполнения данного набора задач<br>$B_i$ = размер памяти для выполнения задач в период наблюдения $i$<br>$n$ = количество измеренных наборов задач |
| Среднее время использования устройств ввода-вывода (mean I/O devices utilization) | $X = \sum_{i=1}^{n} (A_i / B_i) / n$<br>$A_i$ = время занятости устройства ввода-вывода для выполнения данного набора задач<br>$B_i$ = время выполнения операций ввода-вывода для выполнения задач<br>$n$ = количество измеренных событий |

| Атрибут | Метрика |
|---|---|
| Использование памяти (storage utilization) | X = A / B<br>A = емкость вторичной памяти, действительно требуемой для выполнения данного набора задач<br>B = емкость вторичной памяти, доступная для задач |
| Использование полосы пропускания (bandwidth utilization) | X = A / B<br>A = реально измеренное во времени среднее значение полосы пропускания<br>B = доступное значение полосы пропускания |
| Согласованность возможности обработки транзакций (transaction processing capacity conformance) | $X = (\sum_{i=1}^{n} (A_i / B) / C$<br>$A_i$ = количество активных транзакций на момент $i$<br>B = общая длительность операции<br>C = требуемая возможность обработки транзакций для заданного временного интервала |
| Согласованность возможности пользовательского доступа (user access capacity conformance) | X = A / B<br>A = количество пользователей, одновременно обращающихся к системе в определенное время<br>B = заданная возможность требуемого пользовательского доступа |
| Согласованность возрастания пользовательского доступа (user access increase conformance) | X = A / B<br>A = реальное количество пользователей, успешно добавленных за единицу времени<br>B = количество пользователей в единицу времени, ожидаемое для увеличения |

**Таблица Б.3.** Детализация характеристики «совместимость»

| Атрибут | Метрика |
|---|---|
| Согласованность сосуществования (co-existence conformance) | X = A / B<br>A= количество других программных продуктов, с которыми может сосуществовать данный продукт<br>B= количество программных продуктов, которые определены для сосуществования с продуктом в операционной среде |
| Согласованность формата обмена данными (data exchange format conformance) | X = A / B<br>A = количество форматов данных, которыми обмениваются с другим ПО<br>B = количество форматов данных, которое определено для обмена |
| Согласованность протоколов для обмена данными (data exchange protocol conformance) | X = A / B<br>A = количество поддерживаемых протоколов обмена данными<br>B = количество протоколов обмена данными, которое определено для поддержки |

| Атрибут | Метрика |
|---|---|
| Согласованность внешнего интерфейса (external interface conformance) | X = A / B<br>A = количество реализованных внешних интерфейсов<br>B = общее количество определенных внешних интерфейсов |

**Таблица Б.4.** Детализация характеристики «практичность»

| Атрибут | Метрика |
|---|---|
| Полнота описания (description completeness) | X = A / B<br>A= количество сценариев использования, описанных в описании продукта или пользовательских документах<br>B= количество сценариев использования продукта |
| Возможность демонстраций (demonstration capability) | X = A / B<br>A = количество демонстрируемых функций<br>B = количество функций, способных продемонстрировать свою привлекательность |
| Информативность точек входа (entry point selfdescriptiveness) | X = A / B<br>A = количество стартовых страниц, которые поясняют цель веб-сайта<br>B = количество входных страниц на веб-сайте |
| Полнота руководства для пользователя (user guidance completeness) | X = A / B<br>A= количество функций, описанных в пользовательской документации и/или требуемые средства помощи<br>B= количество реализованных функций, которые нужно документировать |
| Входные поля по умолчанию (entry fields defaults) | X = A / B<br>A = количество входных полей, чьи значения по умолчанию автоматически заполняются в ходе операции<br>B = количество входных полей, которые должны иметь значения по умолчанию |
| Понятность сообщений об ошибках (error messages understandability) | X = A / B<br>A= количество сообщений об ошибках, которые объясняют причины возникновения и, возможно, предлагают способы решения<br>B= количество реализованных сообщений об ошибках |
| Поясняющий пользовательский интерфейс (self-explanatory user interface) | X = A / B<br>A = реальное количество информационных элементов и шагов, которые объясняют то, что пользователь должен понимать<br>B = количество необходимых информационных элементов и шагов, нужных для завершения общих задач пользователем-новичком |

| Атрибут | Метрика |
|---|---|
| Естественная последовательность операций (operational consistency) | X = 1 – A / B<br>A = количество конкретных диалоговых задач, которые выполняются не в естественном порядке<br>B= количество конкретных диалоговых задач, которые должны выполняться в естественной последовательности |
| Ясность сообщений (message clarity) | X = A / B<br>A = количество сообщений, которые подсказывают правильный результат или дают советы пользователю<br>B = количество реализованных сообщений |
| Функциональная настраиваемость (functional customizability) | X = A / B<br>A = количество функций и операционных процедур, которые могут быть настроены для повышения удобства пользователя<br>B = количество функций и операционных процедур, для которых пользователи могут получить преимущества при настройке |
| Настраиваемость пользовательского интерфейса (user interface customizability) | X = A / B<br>A= количество элементов пользовательского интерфейса, которые могут быть настроены<br>B= количество элементов пользовательского интерфейса, которые могут получить преимущества при настройке |
| Возможность отслеживания (monitoring capability) | X = A / B<br>A = количество функций, имеющих возможность отслеживания состояния<br>B = количество функций, которые могут получить преимущества при возможности отслеживания |
| Возможность отмены (undo capability) | X = A / B<br>A = количество задач, которые обеспечивают возможности отмены, или предложения по реконфигурации<br>B = количество задач, для которых пользователи могут получить преимущества за счет реконфигурации или возможности отмены |
| Понятность терминологии (terminology understandability) | X = A / B<br>A = количество терминов, которые должны быть понятны предполагаемым пользователям<br>B = количество терминов, которые используются в пользовательском интерфейсе |
| Согласованность появления (appearance consistency) | X = 1 – A / B<br>A = количество пользовательских интерфейсов с подобными элементами, но по-разному появляющимися в интерфейсах<br>B= количество пользовательских интерфейсов с подобными элементами |

| Атрибут | Метрика |
|---|---|
| Недопущение ошибки пользовательской операции (avoidance of user operation error) | X = A / B<br>A = реализованное количество пользовательских действий и вводов для защиты системы от причинения каких-то неприятностей<br>B = количество пользовательских действий и вводов, которые могут защитить систему от причинения каких-то неприятностей |
| Коррекция ошибки пользовательской записи (user entry error correction) | X = A / B<br>A = количество ошибок в записях, для которых автоматически предлагаются правильные значения<br>B = количество обнаруженных ошибок в записях |
| Устраняемость ошибки пользователя (user error recoverability) | X = A / B<br>A = количество пользовательских ошибок, устраняемых системой<br>B = количество пользовательских ошибок, которые могут произойти в течение операции |
| Эстетика внешнего вида пользовательских интерфейсов (appearance aesthetics of user interfaces) | X = A / B<br>A= количество отображаемых интерфейсов, эстетически приятных при появлении<br>B= количество отображаемых интерфейсов |
| Доступность для пользователей с когнитивной инвалидностью (accessibility for users with cognitive disability) | X = A / B<br>A = количество функций, доступных для пользователей с когнитивной инвалидностью<br>B = количество реализованных функций |
| Доступность для пользователей с физической инвалидностью (accessibility for users with physical disability) | X = A / B<br>A = количество функций, доступных для пользователей с физической инвалидностью<br>B = количество реализованных функций |
| Доступность для пользователей со слуховой инвалидностью (accessibility for users with hearing disability) | X = A / B<br>A = количество функций, доступных для пользователей со слуховой инвалидностью<br>B = количество реализованных функций |
| Доступность для пользователей с визуальной инвалидностью (accessibility for users with visual disability) | X = A / B<br>A = количество функций, доступных для пользователей с визуальной инвалидностью<br>B = количество реализованных функций |
| Поддерживаемые языки (supported languages) | X = A / B<br>A = количество реально поддерживаемых языков<br>B = количество языков, нуждающихся в поддержке |

**Таблица Б.5.** Детализация характеристики «надежность»

| Атрибут | Метрика |
|---|---|
| Устранение неисправностей (fault correction) | X = A / B<br>A = количество неисправностей, устраняемых при проектировании, программировании и тестировании<br>B = количество неисправностей, обнаруженных при проектировании, программировании и тестировании |
| Соответствие среднего времени наработки на отказ (mean time between failure (MTBF) conformance) | X = ( A / B ) / C<br>A= время операции<br>B= общее количество реально произошедших системных/программных отказов<br>C = требуемый порог для MTBF |
| Соответствие интенсивности отказов (failure rate conformance) | X = 1 – A / B<br>A = количество отказов, зафиксированных за единицу времени<br>B = требуемый порог для интенсивности отказов |
| Тестовое покрытие (test coverage) | X = A / B<br>A = количество реальных тестов, выполняемых в ходе тестирования<br>B = количество тестов, которые должны выполняться для покрытия требований |
| Доступность системной операции (availability of system operation) | X = A / B<br>A = реальное время системной операции<br>B = время системной операции в соответствии с ее расписанием |
| Соответствие среднего времени простоя (mean down time conformance) | X = ( A / B ) / C<br>A= общее время простоя<br>B= количество наблюдаемых поломок<br>C = требуемый порог для среднего времени простоя |
| Доступность системы в определенные дни (system availability in special days) | X = A / B<br>A = продолжительность работы системы в определенные дни<br>B = общая продолжительность определенных дней |
| Предотвращение отказов (failure avoidance) | X = A / B<br>A = количество предотвращенных критических и серьезных отказов (на основе тестов)<br>B = количество тестов из паттерна отказов, выполненных в ходе тестирования (большей частью вызывающих отказ) |
| Избыточность компонентов (redundancy of components) | X = A / B<br>A = количество установленных избыточных системных компонентов<br>B = общее количество системных компонентов |

| Атрибут | Метрика |
|---|---|
| Соответствие среднего времени уведомления о неисправности (mean fault notification time conformance) | $X = (\sum_{i=1}^{n} (A_i - B_i) / n) / C$<br>$A_i$ = время на уведомление о неисправности системы<br>$B_i$ = время на обнаружение неисправности<br>$C$ = требуемый порог для среднего времени уведомления о неисправности<br>$n$ = количество обнаруженных неисправностей |
| Соответствие среднего времени восстановления (mean recovery time conformance) | $X = (\sum_{i=1}^{n} (A_i / n) / B$<br>$A_i$ = общее время на восстановление отказавшей программы/системы и повторную инициализацию для каждого отказа $i$<br>$B$ = требуемый порог для среднего времени восстановления<br>$n$ = общее количество отказов |
| Полнота резервного копирования данных (backup data completeness) | X = A / B<br>A = количество элементов данных, регулярно подвергаемых резервному копированию<br>B = количество элементов данных, требующих резервного копирования для устранения ошибки |

**Таблица Б.6.** Детализация характеристики «защищенность»

| Атрибут | Метрика |
|---|---|
| Контролируемость доступа (access controllability | X = 1 – A / B<br>A= количество элементов данных, которые могут быть доступны без авторизации<br>B= количество элементов данных, которым требуется контроль доступа |
| Корректность шифрования данных (data encryption correctness) | X = A / B<br>A = количество корректно зашифрованных/расшифрованных элементов данных<br>B = количество элементов данных, которым требуется шифрование/расшифровка |
| Прочность криптографических алгоритмов (strength of cryptographic algorithms) | X = 1 – A / B<br>A = количество сломанных (подвергаемых риску слома) криптографических алгоритмов<br>B = количество используемых криптографических алгоритмов |

| Атрибут | Метрика |
|---|---|
| Соответствие целостности данных (data integrity conformance) | X = 1 – A / B<br>A = количество доступов, в которых реально разрушены данные<br>B = количество доступов, где разрушение или поломка данных могла быть предотвращена |
| Предотвращение разрушения внутренних данных (internal data corruption prevention) | X = A / B<br>A = количество действительно реализованных методов предотвращения разрушения данных<br>B = доступное и рекомендованное количество методов предотвращения разрушения данных |
| Валидация доступов к массиву (validity of array accesses) | X = A / B<br>A = количество доступов к массиву, в которых пользовательский ввод был проверен по диапазону<br>B = количество доступов к массиву, где пользовательский ввод ограничен программными модулями |
| Использование цифровой сигнатуры (utilization of digital signature) | X = A / B<br>A = количество событий, обработанных с использованием цифровой сигнатуры<br>B = количество событий, нуждающихся в обеспечении безотказности |
| Проверяемость доступа (access auditability) | X = A / B<br>A = количество доступов, записанных в логе системы<br>B = количество реально выполненных доступов к системе или данным |
| Соответствие сохранения системного лога (system log retention conformance) | X = A / B<br>A = промежуток времени, в течение которого системный лог сохраняется в стабильной памяти<br>B = требуемый период сохранения системного лога в стабильной памяти |
| Соответствие протокола аутентификации (authentication protocol conformance) | X = A / B<br>A = количество обеспеченных протоколов аутентификации (например, ID пользователя /пароль или карта доступа)<br>B = количество протоколов аутентификации, требуемое спецификацией |
| Соответствие правил аутентификации (authentication rules conformance) | X = A / B<br>A = реализованное количество правил аутентификации для защищенной системы<br>B = количество правил аутентификации, требуемое для защищенной системы |

**Таблица Б.7.** Детализация характеристики «сопровождаемость»

| Атрибут | Метрика |
|---|---|
| Соответствие сцепления компонентов (coupling of components conformance) | X = A / B<br>A = количество компонентов, которые были реализованы с минимальным влиянием друг на друга<br>B = количество компонентов, которые нуждаются в независимости |
| Цикломатическая сложность (cyclomatic complexity) | X = 1 – A / B<br>A = количество программных модулей, для которых цикломатическая сложность задана в виде порога<br>B = количество программных модулей в продукте |
| Повторная используемость активов (reusability of assets) | X = A / B<br>A = количество активов, которые были спроектированы и реализованы для повторного использования<br>B = количество активов в системе |
| Соответствие правилам кодирования (conformance to coding rules) | X = A / B<br>A = количество модулей, соответствующих правилам кодирования для определенной системы<br>B = количество модулей в разработанной системе |
| Соответствие полноты системного лога (system log completeness conformance) | X = A / B<br>A= количество логов, реально записанных в системе<br>B= количество логов, для которых в ходе операции нужен журнал аудита |
| Эффективность функции диагностики (diagnosis function effectiveness) | X = A / B<br>A = количество функций диагностики, полезных для анализа причин<br>B = количество реализованных функций диагностики |
| Достаточность функции диагностики (diagnosis function sufficiency conformance) | X = A / B<br>A = количество реализованных функций диагностики<br>B = количество функций диагностики, требуемых спецификацией |
| Эффективность модификаций (modification efficiency) | $X = \sum_{i=1}^{n} (A_i / B_i) / n$<br>$A_i$ = общее время, расходуемое на конкретный тип модификации $i$<br>$B_i$ = ожидаемое время на выполнение конкретного типа модификации $i$<br>$n$ = измеренное количество модификаций |
| Корректность модификаций (modification correctness) | X = 1 – (A / B)<br>A = количество модификаций, вызываемых инцидентами или отказами в течение определенного периода после реализации<br>B = количество модификаций, реализованных в течение определенного периода |

| Атрибут | Метрика |
|---|---|
| Возможность модификаций (modification capability) | X = A / B<br>A = количество реально модифицированных элементов (за определенный период)<br>B = количество элементов, которые нужно модифицировать |
| Полнота функций тестирования (test function completeness conformance) | X = A / B<br>A =количество функций тестирования, реализованных по спецификации<br>B = количество требуемых функций тестирования |
| Автономность тестирования (autonomous testability) | X = A / B<br>A = количество тестов, которые могут быть смоделированы с помощью заглушки, устраняющей зависимость от тестирования других систем<br>B = количество тестов, которые зависят от других систем |
| Перезапуски тестов (test restartability) | X = A / B<br>A = количество тестов, в которых тестировщик может сделать паузу, а затем рестартовать с указанной точки (для проверки в пошаговом режиме)<br>B = количество тестов, в которых выполняемый тест может быть приостановлен |

**Таблица Б.8.** Детализация характеристики «переносимость»

| Атрибут | Метрика |
|---|---|
| Адаптируемость аппаратной среды (hardware environmental adaptability) | X = 1 – A / B<br>A = количество функций, чьи задачи не завершены или чьи результаты недостаточно отражают требования тестирования<br>B = количество функций, тестируемых в различных аппаратных средах |
| Адаптируемость системной программной среды (system software environmental adaptability) | X = 1– A / B<br>A = количество функций, чьи задачи не завершены или чьи результаты недостаточно отражают требования тестирования<br>B = количество функций, тестируемых в различных системных программных средах |
| Адаптируемость операционной среды (operational environment adaptability) | X = 1 – A / B<br>A = количество функций, чьи задачи не завершены или чьи результаты недостаточно отражают требования тестирования в пользовательской среде<br>B = количество функций, тестируемых в различных операционных средах |
| Эффективность времени инсталляции (installation time efficiency) | $X = \sum_{i=1}^{n} (A_i / B_i) / n$<br>$A_i$ = общее время, потраченное на инсталляцию $i$<br>$B_i$ = ожидаемое время для выполнение инсталляции $i$<br>$n$ = измеренное количество инсталляций |

| Атрибут | Метрика |
|---|---|
| Легкость инсталляции (ease of installation) | X = A / B<br>A = количество вариантов, где пользователь успешно изменяет операцию инсталляции (для своего удобства)<br>B = количество вариантов, где пользователь пытается изменять операцию инсталляции (для своего удобства) |
| Сходство использования (usage similarity) | X = A / B<br>A = количество пользовательских функций, которые могут выполняться без дополнительного обучения или каких-то «обходных маневров»<br>B = количество пользовательских функций в измененном программном продукте |
| Эквивалентность качества продукта (product quality equivalence) | X = A / B<br>A = количество характеристик качества нового продукта, лучших или эквивалентных характеристикам замещенного продукта<br>B = количество характеристик качества замещенного продукта, имеющих отношение к новому продукту |
| Функциональный охват (инклюзивность) (functional inclusiveness) | X = A / B<br>A = количество функций, вырабатывающих результаты, подобные прежним<br>B = количество функций, протестированных перед заменой одного ПО на другое |
| Возможность повторного использования/импорта данных (data reusability/import capability) | X = A / B<br>A = количество данных, которые можно продолжать использовать (так же, как и прежде)<br>B = количество данных, которые продолжительно использовались в замещенном ПО |

**Таблица Б.9.** Группы метрик для функциональной пригодности

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Функциональная полнота | обе |
| 2 | Функциональная правильность | обе |
| 3 | Функциональная целесообразность | обе |

**ПРИМЕЧАНИЕ**

Пометка «обе» означает, что метрика является как внешней, так и внутренней.

**Таблица Б.10.** Группы метрик для эффективности исполнения

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Среднее время отклика | обе |
| 2 | Согласованность времени отклика | обе |

| Номер | Метрика | Принадлежность |
|---|---|---|
| 3 | Среднее время оборота | обе |
| 4 | Согласованность времени оборота | обе |
| 5 | Согласованность пропускной способности | обе |
| 6 | Среднее время использования процессора | внешняя |
| 7 | Средняя емкость используемой памяти | внешняя |
| 8 | Среднее время использования устройств ввода-вывода | внешняя |
| 9 | Использование памяти | внешняя |
| 10 | Использование полосы пропускания | внешняя |
| 11 | Согласованность возможности обработки транзакций | обе |
| 12 | Согласованность возможности пользовательского доступа | обе |
| 13 | Согласованность возрастания пользовательского доступа | внешняя |

**Таблица Б.11.** Группы метрик для совместимости

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Согласованность сосуществования | внешняя |
| 2 | Согласованность формата обмена данными | обе |
| 3 | Согласованность протоколов для обмена данными | обе |
| 4 | Согласованность внешнего интерфейса | обе |

**Таблица Б.12.** Группы метрик для практичности

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Полнота описания | обе |
| 2 | Возможность демонстраций | обе |
| 3 | Информативность точек входа | обе |
| 4 | Полнота руководства для | обе |
| 5 | Входные поля по умолчанию | обе |
| 6 | Понятность сообщений об ошибках | обе |
| 7 | Поясняющий пользовательский интерфейс | обе |
| 8 | Естественная последовательность операций | обе |
| 9 | Ясность сообщений | обе |
| 10 | Функциональная настраиваемость | обе |
| 11 | Настраиваемость пользовательского интерфейса | обе |
| 12 | Возможность отслеживания | обе |
| 13 | Возможность отмены | обе |
| 14 | Понятность терминологии | обе |
| 15 | Согласованность появления | обе |
| 16 | Недопущение ошибки пользовательской операции | обе |

| Номер | Метрика | Принадлежность |
|---|---|---|
| 17 | Коррекция ошибки пользовательской записи | обе |
| 18 | Устраняемость ошибки | обе |
| 19 | Эстетика внешнего вида пользовательских интерфейсов | обе |
| 20 | Доступность для пользователей с когнитивной инвалидностью | обе |
| 21 | Доступность для пользователей с физической инвалидностью | обе |
| 22 | Доступность для пользователей со слуховой инвалидностью | обе |
| 23 | Доступность для пользователей с визуальной инвалидностью | обе |
| 24 | Поддерживаемые языки | обе |

**Таблица Б.13.** Группы метрик для надежности

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Устранение неисправностей | обе |
| 2 | Соответствие среднего времени наработки на отказ | внешняя |
| 3 | Соответствие интенсивности отказов | внешняя |
| 4 | Тестовое покрытие | внешняя |
| 5 | Доступность системной операции | внешняя |
| 6 | Соответствие среднего времени простоя | внешняя |
| 7 | Доступность системы в определенные дни | внешняя |
| 8 | Предотвращение отказов | внешняя |
| 9 | Избыточность компонентов | обе |
| 10 | Соответствие среднего времени уведомления о неисправности | внешняя |
| 11 | Соответствие среднего времени восстановления | внешняя |
| 12 | Полнота резервного копирования данных | обе |

**Таблица Б.14.** Группы метрик для защищенности

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Контролируемость доступа | обе |
| 2 | Корректность шифрования данных | обе |
| 3 | Прочность криптографических алгоритмов | обе |
| 4 | Соответствие целостности данных | обе |
| 5 | Предотвращение разрушения внутренних данных | обе |
| 6 | Валидация доступов к массиву | внутренняя |
| 7 | Использование цифровой сигнатуры | обе |
| 8 | Проверяемость доступа | обе |
| 9 | Соответствие сохранения системного лога | обе |

| Номер | Метрика | Принадлежность |
|---|---|---|
| 10 | Соответствие протокола аутентификации | обе |
| 11 | Соответствие правил аутентификации | обе |

**Таблица Б.15.** Группы метрик для сопровождаемости

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Соответствие сцепления компонентов | обе |
| 2 | Цикломатическая сложность | внутренняя |
| 3 | Повторная используемость активов | обе |
| 4 | Соответствие правилам кодирования | внутренняя |
| 5 | Соответствие полноты системного лога | обе |
| 6 | Эффективность функции диагностики | обе |
| 7 | Достаточность функции диагностики | обе |
| 8 | Эффективность модификаций | обе |
| 9 | Корректность модификаций | обе |
| 10 | Возможность модификаций | обе |
| 11 | Полнота функций тестирования | обе |
| 12 | Автономность тестирования | обе |
| 13 | Перезапуски тестов | обе |

**Таблица Б.16.** Группы метрик для переносимости

| Номер | Метрика | Принадлежность |
|---|---|---|
| 1 | Адаптируемость аппаратной среды | внешняя |
| 2 | Адаптируемость системной программной среды | внешняя |
| 3 | Адаптируемость операционной среды | внешняя |
| 4 | Эффективность времени инсталляции | внешняя |
| 5 | Легкость инсталляции | внешняя |
| 6 | Сходство использования | обе |
| 7 | Эквивалентность качества продукта | обе |
| 8 | Функциональный охват (инклюзивность) | внешняя |
| 9 | Возможность повторного использования/импорта данных | внешняя |

# Приложение В

# Терминология языка UML и унифицированного процесса

В данном приложении приведен словарь основных терминов языка UML и унифицированного процесса разработки, описываемого в учебнике.

## Словарь терминов

| | |
|---|---|
| Абстрактный класс (abstract class) | Класс, объект которого не может быть создан непосредственно |
| Агрегат (aggregate) | Класс, описывающий «целое» в отношении агрегации |
| Агрегация (aggregation) | Специальная форма ассоциации, определяющая отношение «часть-целое» между агрегатом (целым) и частями |
| Актер (actor) | Связанный набор ролей, исполняемый пользователями при взаимодействии с элементами Use case |
| Активация (activation) | Выполнение соответствующего действия |
| Активный класс (active class) | Класс, экземпляры которого являются активными объектами. Активный класс изображается в виде прямоугольника, левая и правая стороны которого рисуются двойными линиями. См. *процесс, задача, поток* |
| Активный объект (active object) | Объект, являющийся владельцем процесса или потока, которые инициируют управляющую деятельность |
| Артефакт (artifact) | Документ, отчет или выполняемый элемент. Артефакт может вырабатываться, обрабатываться или потребляться |
| Асинхронное действие (asynchronous action) | Запрос, отправляемый объекту без паузы для ожидания результата |
| Ассоциация (association) | Семантическое отношение между классификаторами, задающее набор связей между их экземплярами |
| Атрибут (attribute) | Именованная характеристика классификатора, задающая набор возможных значений, которые определяют состояния экземпляров классификатора (например, объектов) |

| | |
|---|---|
| Бизнес-модель (business model) | Определяет абстракцию организации, для которой создается система |
| Бинарная ассоциация (binary association) | Ассоциация между двумя классами |
| Взаимодействие (interaction) | Поведение, заключающееся в обмене набором сообщений между набором объектов (в определенном контексте и для достижения определенной цели) |
| Видимость (visibility) | Показывает, как может быть увидено и использовано другими данное имя |
| Временный объект (transient object) | Объект, существующий только во время выполнения задачи или процесса, которые его создали |
| Действие (action) | Исполняемое атомарное вычисление. Действие инициируется при получении объектом сообщения или изменении значения его атрибута. В результате действия изменяется состояние объекта |
| Делегирование (delegation) | Способность объекта посылать сообщение другому объекту в ответ на прием чужого сообщения |
| Деятельность (activity) | Задает состояния вычислений, в которых проявляется некоторое поведение |
| Диаграмма (diagram) | Графическое представление набора элементов, обычно в виде связного графа, в вершинах которого находятся предметы, а дуги представляют собой их отношения |
| Диаграмма взаимодействия (interaction diagram) | Диаграмма, показывающая взаимодействие, включающее в себя набор обобщенных объектов и их отношений, а также пересылаемые между ними сообщения. Диаграммы взаимодействия относятся к динамическому представлению системы. Это общий термин, применяемый к различным видам диаграмм, на которых изображено взаимодействие участников, включая диаграммы коммуникации и диаграммы последовательности |
| Диаграмма деятельности (activity diagram) | Диаграмма, показывающая разложение некоторой деятельности на ее составные части. Диаграммы деятельности относятся к динамическому представлению системы. Диаграмма деятельности отображает поток управления и данных от одной деятельности к другой |
| Диаграмма классов (class diagram) | Диаграмма, показывающая набор классов, интерфейсов, кооперации, а также их отношения. Диаграмма классов относится к статическому проектному представлению системы. Эта диаграмма показывает набор декларативных (статических) элементов |
| Диаграмма объектов (object diagram) | Диаграмма, показывающая набор объектов и их отношений в некоторый момент времени. Диаграмма объектов относится к статическому проектному представлению или статическому представлению процессов системы |
| Диаграмма последовательности (sequence diagram) | Диаграмма взаимодействия, выделяющая временную последовательность передачи сообщений |
| Диаграмма коммуникации (communication diagram) | Диаграмма взаимодействия, которая выделяет структурную организацию обобщенных объектов (ролей), посылающих и принимающих сообщения; диаграмма, которая демонстрирует организацию взаимодействия между экземплярами и их связи друг с другом |

| | |
|---|---|
| Диаграмма конечного автомата (state machine diagram) | Диаграмма, показывающая конечный автомат. Диаграммы конечных автоматов относятся к динамическому представлению системы |
| Диаграмма развертывания (deployment diagram) | Диаграмма, показывающая набор узлов и их отношения. Диаграмма развертывания относится к статическому представлению размещения системы |
| Диаграмма Use Case (use case diagram) | Диаграмма, показывающая набор элементов Use Case, актеров и их отношений. Диаграмма Use Case относится к статическому представлению Use Case, создаваемому для системы |
| Единица дистрибуции (distribution unit) | Набор объектов или компонентов, которые предназначены для выполнения одной задачи или работы на одном процессоре |
| Зависимость (dependency) | Семантическое отношение между двумя предметами, при котором изменение одного предмета (независимого предмета) влияет на семантику другого предмета (зависимого предмета) |
| Задача (task ) | Единичный путь выполнения программы, динамической модели или другого представления потока управления; нить или процесс |
| Запустить (fire) | Выполнить переход из состояния в состояние |
| Иерархия вложенности (containment hierarchy) | Иерархия пространств имен, содержащих элементы и отношения вложенности между ними |
| Импорт (import) | В контексте пакетов — зависимость, показывающая, на классы какого пакета могут ссылаться классы данного пакета (включая пакеты, рекурсивно вложенные в данный) |
| Имя (name) | То, как вы называете сущность, отношение или диаграмму; строка, используемая для идентификации элемента |
| Интерфейс (interface) | Набор операций, используемых для описания услуг класса или компонента |
| Исполняемый модуль (executable) | Программа, которая может выполняться в узле |
| Использование (usage) | Зависимость, при которой один элемент (клиент) для корректного функционирования нуждается в присутствии другого элемента (поставщика) |
| Кардинальное число (cardinality) | Число элементов в наборе |
| Каркас (framework) | Архитектурный паттерн, предоставляющий расширяемый шаблон приложения в какой-либо предметной области |
| Класс (class) | Описание набора объектов, имеющих одинаковые атрибуты, операции, отношения и семантику |
| Класс-ассоциация (association class) | Элемент моделирования, имеющий одновременно характеристики класса и ассоциации. Класс-ассоциация может рассматриваться как ассоциация, имеющая также характеристики класса, или как класс, обладающий характеристиками ассоциации |
| Классификатор (classifier) | Механизм описания структурных и поведенческих характеристик. Классификаторами являются интерфейсы, классы, типы данных, элементы Use Case, компоненты и узлы |
| Клиент (client) | Классификатор, запрашивающий услуги у другого классификатора |

| | |
|---|---|
| Композит (composite) | Класс, связанный с одним или более классами отношением композиции |
| Композиция (composition) | Сильная форма агрегации, при которой время жизни частей и целого совпадают. Части не существуют отдельно и при удалении композита должны быть уничтожены |
| Компонент (component) | Модульная часть системы, которая соответствует набору интерфейсов и обеспечивает реализацию набора интерфейсов |
| Компонентная диаграмма (component diagram) | Диаграмма, показывающая набор компонентов и их отношений. Компонентные диаграммы относятся к статическому компонентному представлению системы |
| Конечный автомат (state machine) | Поведение, которое определяется последовательностью состояний, через которые проходит объект в течение своей жизни в ответ на поступление сообщений, вместе с его реакцией на эти сообщения |
| Конкретный класс (concrete class) | Класс, для которого возможно создание экземпляров |
| Контейнер (container) | Объект, создаваемый для хранения других объектов и предоставляющий операции для доступа к своему содержимому в определенном порядке |
| Контекст (context) | Набор связанных элементов, ориентированных на достижение определенной цели, например определение операции |
| Кооперация (collaboration) | Сообщество классов, интерфейсов и других элементов, работающих вместе с целью реализации некоторого кооперативного поведения. Кооперация больше, чем простая сумма элементов. Описание того, как элементы, такие как элементы Use Case или операции, реализуются набором классификаторов и ассоциаций, играющих определенные роли определенным образом |
| «Линия жизни» (lifeline) | См. *линия жизни объекта* |
| Линия жизни объекта (object lifeline) | Роль участника взаимодействия на диаграмме последовательности, которая отражает существование обобщенного объекта в течение некоторого периода времени. На диаграмме последовательности изображается вертикальной линией. Символ в верхней части линии изображает имя и тип участника |
| Местоположение (location) | Место размещения компонента в узле |
| Метакласс (metaclass) | Класс, экземпляры которого являются классами |
| Метод (method) | Реализация операции. Определяет алгоритм или процедуру, обеспечивающую операцию. |
| Механизм расширения (extensibility mechanism) | Один из трех механизмов (стереотипы, теговые величины и ограничения), который может использоваться для контролируемого расширения UML |
| Множественная классификация (multiple classification) | Семантическая вариация обобщения, в которой объект может принадлежать более чем одному классу |
| Множественное наследование (multiple inheritance) | Семантическая вариация обобщения, в которой тип может иметь более одного супертипа |
| Множественность (multiplicity) | Спецификация диапазона возможных кардинальных чисел набора |
| Модель (Model) | Семантически ограниченное абстрактное представление системы |
| Модель Use Case (Use case model) | Определяет функциональные требования к системе |

| | |
|---|---|
| Модель анализа (analysis model) | Интерпретирует требования к системе в терминах проектной модели |
| Модель области определения (domain model) | Фиксирует контекстное окружение системы |
| Модель процессов (process model) | Определяет параллелизм в системе и механизмы синхронизации |
| Модель развертывания (deployment model) | Определяет аппаратную топологию, в которой исполняется система |
| Модель реализации (implementation model) | Определяет части, которые используются для сборки и реализации физической системы |
| Наследование (inheritance) | Механизм, при помощи которого более специализированные элементы включают в себя структуру и поведение более общих элементов |
| Наследование интерфейса (interface inheritance) | Наследование интерфейса более специализированным элементом, не включает наследования реализации |
| Нить (thread) | Облегченный поток управления, который может выполняться параллельно с другими нитями того же процесса |
| Область действия (scope) | Контекст, который придает имени определенный смысл |
| Обобщение (generalization) | Отношение обобщения/специализации, когда объекты специализированного элемента (подтипа) могут замещать объекты обобщенного элемента (супертипа) |
| Объект (object ) | См. *экземпляр* |
| Объект длительного хранения (persistent object) | Объект, сохраняющийся после завершения процесса или задачи, в ходе которой он был создан |
| Объектный язык ограничений (object constraint language (OCL)) | Формальный язык, используемый для создания ограничений, не имеющих побочных эффектов |
| Обязанность (responsibility) | Контракт или обязательство типа или класса |
| Ограничение (constraint) | Расширение семантики элемента UML, позволяющее добавлять к нему новые правила или изменять существующие |
| Одиночное наследование (single inheritance) | Семантический вариант обобщения, при котором каждый тип может иметь только один супертип |
| Операция (operation) | Обслуживание, которое может запрашиваться у объекта. Операция имеет сигнатуру, которая задает допустимые фактические параметры. |
| Отношение (relationship) | Семантическая связь между элементами |
| Отношение трассировки (trace) | Зависимость, указывающая на историческую связь или связь обработки между двумя элементами, представляющими одну и ту же концепцию, без определения правил вывода одного элемента из другого |
| Отправитель (сообщения) (sender) | Объект, посылающий экземпляр сообщения объекту-получателю |
| Отправление (send) | Посылка экземпляра сообщения от отправителя получателю |
| Пакет (package) | Механизм общего назначения для группировки элементов |
| Параллелизм (concurrency) | Осуществление двух или более видов деятельности в один и тот же временной интервал. Параллелизм может быть осуществлен путем квантования процессорного времени или одновременного выполнения двух или более потоков |

| | |
|---|---|
| Параметр (parameter) | Определение переменной, которая может изменяться, передаваться или возвращаться |
| Паттерн (pattern) | Паттерн является решением типичной проблемы в определенном контексте |
| Переход (transition) | Отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, в случае некоторого события и выполнения определенных условий совершит некоторые действия и перейдет во второе состояние |
| Плавательная дорожка (swim lane) | Область на диаграмме деятельности для назначения ответственного за действие |
| Побуждение (stimulus) | Операция или сигнал |
| Подсистема (subsystem) | Группировка элементов, в которой каждый элемент содержит описание поведения, предоставляемого другим элементам подсистемы |
| Подтип (subtype) | В отношении обобщения — специализация другого типа, супертипа |
| Получатель (receiver) | Объект, обрабатывающий экземпляр сообщения, поступивший от объекта — отправителя |
| Полюс (конец) ассоциации (association end) | Конечная точка ассоциации, которая связывает ассоциацию с классификатором |
| Полюс (конец) связи (link end) | Экземпляр полюса (конца) ассоциации |
| Поставщик (supplier) | Тип, класс или компонент, предоставляющие услуги, используемые другими |
| Постусловие (postcondition) | Условие, которое должно выполняться после завершения операции |
| Представление (view) | Проекция модели, рассматриваемая с определенной точки зрения, в которой показаны существенные и опущены несущественные детали |
| Предусловие (precondition) | Условие, которое должно выполняться при вызове операции |
| Прием (receive) | Обработка экземпляра сообщения, поступившего от объекта — отправителя |
| Примечание (comment) | Примечание, добавляемое к элементу или группе элементов |
| Примечание (note) | Комментарий, добавляемый к элементу или набору элементов |
| Примитивный тип (primitive type) | Предопределенный базовый тип, например целое число или строка |
| Проектная модель (design model) | Определяет словарь проблемы и ее решение |
| Пространство имен (namespace) | Часть модели, в которой могут определяться и использоваться имена. Внутри пространства имен каждое имя имеет единственный смысл |
| Процесс (process) | Тяжеловесный поток управления, который может выполняться параллельно с другими процессами |
| Рабочий поток процесса (process workflow) | Логическая группировка действий |
| Реализация (realization) | Семантическое отношение между классификаторами, когда один классификатор определяет контракт, который другие классификаторы должны гарантированно выполнять |
| Роль (role) | Определенное поведение сущности в определенном контексте |

| | |
|---|---|
| Связывание (binding) | Создание конкретного элемента на основе шаблона (путем сопоставления параметрам шаблона конкретных аргументов) |
| Связь (link) | Семантическая связь между объектами, экземпляр ассоциации |
| Сигнал (signal) | Спецификация асинхронного стимула, передаваемого от экземпляра к экземпляру |
| Сигнатура (signature) | Имя и параметры характеристики поведения |
| Синхронное действие (synchronous action) | Запрос, при котором отправивший его объект прерывает работу, ожидая результата |
| Система (system) | Набор подсистем, организованный для достижения определенной цели и описываемый набором моделей, с разных точек зрения |
| Событие (event) | Определение значимого происшествия, ограниченного во времени и пространстве, в контексте конечных автоматов. Событие может запустить переход из одного состояния в другое состояние |
| Сообщение (message) | Спецификация передачи информации между объектами в ожидании того, что будет обеспечена требуемая деятельность. Получение экземпляра сообщения обычно рассматривается как экземпляр события |
| Состояние (state) | Условия или ситуация в течение жизни объекта, когда он удовлетворяет некоторому условию, выполняет некоторую деятельность или ждет некоторого события |
| Состояние действия (action state) | Состояние, которое представляет собой исполнение единичного действия, обычно вызов операции |
| Спецификация (specification) | Текстовая запись синтаксиса и семантики определенного строительного блока, описание того, что он из себя представляет или что он делает |
| Спецификация выполнения (execution specification) | Спецификация выполнения (активация) — период выполнения операции участником взаимодействия. На диаграмме последовательности изображается прямоугольником на линии жизни участника. Символ на диаграмме последовательности, указывающий период времени, в течение которого участник взаимодействия (обобщенный объект) выполняет действие |
| Стереотип (stereotype) | Расширение словаря UML, позволяющее нам создавать новые типы строительных блоков, порождая их от существующих. Новые блоки специализированы для решения определенных проблем |
| Сторожевое условие (guard condition) | Условие, которое должно быть выполнено для запуска ассоциированного с ним перехода |
| Супертип (supertype) | В отношении обобщения — обобщение другого типа, подтипа |
| Сценарий (scenario) | Определенная последовательность действий, иллюстрирующая поведение |
| Теговая величина (tagged value) | Расширение характеристик элемента UML, позволяющее помещать в спецификацию элемента новую информацию |
| Тестовая модель (test model) | Определяет тестовые варианты для проверки системы |
| Тип (type) | Стереотип класса, используемый для определения предметной области объекта и операций (но не методов), применимых к этому объекту |

| | |
|---|---|
| Тип данных (datatype) | Тип, задающий набор неидентифицированных значений и операций для их обработки. Типы данных включают в себя как простые встроенные типы (такие, как числа и строки), так и перечислимые типы (например, логический тип) |
| Узел (node) | Физический элемент, существующий во время работы системы и предоставляющий вычислительный ресурс, обычно имеющий память, а часто — и возможность выполнения операций |
| Украшение (adornment) | Детализация спецификации элемента, добавляемая к его основной графической нотации |
| Фасад (facade) | Фасад — это стереотипный пакет, не содержащий ничего, кроме ссылок на элементы модели, находящиеся в другом пакете. Он используется для обеспечения «публичного» представления некоторой части содержимого пакета |
| Фокус управления (focus of control) | Старое название термина «спецификация выполнения». Символ на диаграмме последовательности, указывающий период времени, в течение которого участник взаимодействия (обобщенный объект) выполняет действие |
| Характеристика (property) | Именованная величина, обозначающая характеристику элемента |
| Шаблон (template) | Параметризованный элемент |
| Экземпляр (instance) | Конкретная реализация абстракции, сущность, к которой может быть применен набор операций, она имеет состояние для сохранения результатов применения операций. Синоним объекта |
| Экспорт (export) | В контексте пакетов — действие, делающее элемент видимым вне его собственного пространства имен |
| Элемент (element) | Единичная составная часть модели |
| Этап Конструирование (Construction phase) | Этап построения программного продукта в виде серии инкрементных итераций |
| Этап Начало (Inception phase) | Этап спецификации представления продукта |
| Этап Переход (Transition phase) | Этап внедрения программного продукта в среду пользователя (промышленное производство, доставка и применение) |
| Этап Развитие (Elaboration phase) | Этап планирования необходимых действий и требуемых ресурсов |
| n-арная ассоциация (n–ary association) | Ассоциация между n классами. Если n равно двум, ассоциация бинарная. См. *бинарная ассоциация* |
| Элемент Use Case (use case) | Описание набора, состоящего из нескольких последовательностей действий системы, которые производят для отдельного актера видимый результат |