ФГБОУ ВО «Воронежский государственный
технический университет»

Е.Н. Королев

# МЕТОДЫ ПРЕДСТАВЛЕНИЯ ДАННЫХ

Утверждднно Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2010

УДК 681.38+681.3

Королев Е.Н. Модели представления данных: учеб. пособие/ Е.Н. Королев. Воронеж: ГОУВПО “Воронежский государственный технический университет”, 2010. 124 с.

В учебном пособии рассматриваются основы типы моделей представления данных и работа с ними с помощью языка программирования Java.

Издание соответствует требованиям Государственного образовательного стандарта высшего профессионального образования по направлению подготовки магистров по направлению «Информатика и вычислительная техника», очной формы обучения.

Пособие предназначено студентам специальностей естественно-технического профиля, аспирантам и специалистам, занимающимся вопросами практической работы с различными моделями представления данных.

Предназначено для студентов очной формы обучения.

Ил. 44. Библиогр.: 4 назв.

Научный редактор д-р техн. наук, проф. Я.Е. Львович

Рецензенты: кафедра информационного обеспечения и моделирования агроэкономических систем Воронежского государственного аграрного университета им. К.Д. Глинки (зав. кафедрой д-р экон. наук, доц. А.В. Улезько);
д-р техн. наук, проф. О.Ю. Макаров

# ВВЕДЕНИЕ

Объектно-ориентированный анализ, проектирование и разработка сложных информационных систем - задача актуальная и востребованная. Современные информационные системы оперируют огромными потоками данных, которые представляются в виде множества различных форм. Современным специалистам, подготовленным в области новых информационных технологий, необходимо ориентироваться в технологиях и методах работы с большим разнообразием сложных типов данных, представленных в виде обобщенных структур, называемых моделями, т.к. они отражают представление пользователя о данных реального мира.

Работа с моделями представления данных ведется на всех этапах проектирования и разработки информационных систем. Наиболее важным этапом проектирования является этап описания, визуализации и документирования объектно-ориентированных систем и бизнес-процессов с ориентацией на их последующую реализацию в виде программного обеспечения. На этапе проектирования ведется разработка интегрированной модели системы, которая включает в себя разработку как концептуальных, так и физических моделей. В качестве основного языка для выполнения этих функций предлагается UML (Unified Modeling Language). На этапе разработки информационных систем необходима разработка типов данных в виде моделей, для чего используется механизм классов, разработка моделей для представления визуальных компонент, разработка моделей для представления данных в СУБД в виде инфологических и даталогических моделей.

Важно не только иметь представление о принципах построения таких моделей, но и иметь практические навыки формирования таких моделей и работы с ними на современных языках программировнаия, таких как Java.

# 1. ПРЕДСТАВЛЕНИЕ ДАННЫХ

## 1.1. Основные типы данных

Данные, хранящиеся в памяти ЭВМ, представляют собой совокупность нулей и едениц (битов). Биты объединяются в последовательности: байты, слова и т.д. Каждому участку оперативной памяти, который может вместить один байт или слово, присваивается порядковый номер (адрес).

Какой смысл заключен в данных, какими символами они выражены - буквенными или цифровыми, что означает то или иное число - все это определяется программой обработки. Все данные необходимые для решения практических задач подразделяются на несколько типов, причем понятие тип связывается не только с представлением данных в адресном пространстве, но и со способом их обработки.

Любые данные могут быть отнесены к одному из двух типов: основному (простому), форма представления которого определяется архитектурой ЭВМ, или сложному, конструируемому пользователем для решения конкретных задач.

Данные простого типа это - символы, числа и т.п. элементы, дальнейшее дробление которых не имеет смысла. Из элементарных данных формируются структуры (сложные типы) данных.

Некоторые структуры:

- **Массив** (функция с конечной областью определения) - простая совокупность элементов данных одного типа, средство оперирования группой данных одного типа. Отдельный элемент массива задается индексом. Массив может быть одномерным, двумерным и т.д. Разновидностями одномерных массивов переменной длины являются структуры типа кольцо, стек, очередь и двухсторонняя очередь.
- **Запись** (декартово произведение) - совокупность элементов данных разного типа. В простейшем случае запись содержит постоянное количество элементов, которые называют полями. Совокупность записей одинаковой структуры называется файлом. (Файлом называют также набор данных во внешней памяти, например, на магнитном диске). Для того, чтобы иметь возможность извлекать из файла отдельные записи, каждой записи присваивают уникальное имя или номер, которое служит ее

идентификатором и располагается в отдельном поле. Этот идентификатор называют ключом.

Такие структуры данных как массив или запись занимают в памяти ЭВМ постоянный объем, поэтому их называют статическими структурами. К статическим структурам относится также множество.

Имеется ряд структур, которые могут изменять свою длину - так называемые динамические структуры. К ним относятся дерево, список, ссылка.

Важной структурой, для размещения элементов которой требуется нелинейное адресное пространство, является дерево. Существует большое количество структур данных, которые могут быть представлены как деревья. Это, например, классификационные, иерархические, рекурсивные и др. структуры.


Рис. 1. Классификация типов данных

## 1.2. Обобщенные структуры или модели данных

Выше мы рассмотрели несколько типов структур, являющихся совокупностями элементов данных: массив, дерево, запись. Более сложный тип данных может включать эти структуры в качестве элементов. Например, элементами записи может быть массив, стек, дерево и т.д.

Существует большое разнообразие сложных типов данных, но исследования, проведенные на большом практическом материале, показали, что среди них можно выделить несколько наиболее общих. Обобщенные структуры называют также **моделями**

**данных**, т.к. они отражают представление пользователя о данных реального мира.

Любая модель данных должна содержать **три компоненты**:

1. структура данных - описывает точку зрения пользователя на представление данных.

2. набор допустимых операций, выполняемых на структуре данных. Модель данных предполагает, как минимум, наличие языка определения данных (ЯОД), описывающего структуру их хранения, и языка манипулирования данными (ЯМД), включающего операции извлечения и модификации данных.

3. ограничения целостности - механизм поддержания соответствия данных предметной области на основе формально описанных правил.

В процессе исторического развития в СУБД использовалось следующие модели данных:

- иерархическая,
- сетевая,
- реляционная.

**В последнее время все большее значение приобретает объектно-ориентированный подход к представлению данных.**

Объектно-ориентированная парадигма.

Сразу же необходимо заметить, что общепринятого определения "объектно-ориентированной модели данных" не существует. Сейчас можно говорить лишь о неком "объектном" подходе к логическому представлению данных и о различных объектно-ориентированных способах его реализации.

**Класс — способ описания типа**

Для описания типов в Java используется механизм классов. За исключением базовых (иначе — элементарных) типов (int, char, float и др.) и интерфейсов (что это такое, мы рассмотрим позже), все остальные типы — это классы.

В простейшем случае описание класса выглядит так

```
class MyClass {

. . . // тело класса

}
```

Здесь **class** — ключевое слово, MyClass — имя класса. Внутри фигурных скобок находится тело класса.

| Тип | Описатель | Размер | Комментарий |
|---|---|---|---|
| Логический | boolean | ?* | - |
| Символьный | char | 2 байта | Unicode |
| Байтовый | byte* | 1 байт | (-128 - 127) |
| Короткий целый | short | 2 байта | (-215 — 215-1) |
| Целый | int | 4 байта | (-231 — 231-1) |
| Длинный целый | long | 8 байт | (-263 — 263-1) |
| Вещественный | float | 4 байта | - |
| Вещественный двойной точности | double | 8 байт | - |
| Пустой | void* | - | - |

**Любая модель данных должна включать три аспекта: структурный, целостный и манипуляционный**. Посмотрим, как они реализуются на основе объектно-ориентированная парадигмы программирования:

**Структура.**

Структура объектной модели описываются с помощью трех ключевых понятий:

- **инкапсуляция** - каждый объект обладает некоторым внутренним состоянием (хранит внутри себя запись данных), а также набором методов - процедур, с помощью которых (и только таким образом) можно получить доступ к данным, определяющим внутреннее состояние объекта, или изменить их. Таким образом, объекты можно рассматривать как самостоятельные сущности, отделенные от внешнего мира.

Пример:

```
  Class Point {  // вводим новый тип данных - объект "точка"
    X,Y : int;     // данные объекта - координаты точки
    .....
 Point(X : int, Y : int);
 //конструктор объекта - процедура, вызываемая
 // при определении переменной на базе объекта и
 // присваивающая значения его данным
    .....
```

```
Draw();
  //метод "нарисовать точку"

Erase();          // метод "стереть точку"
Move(newX,newY);  //метод "переместить точку"(изменяет данные
объекта)
int GetX();       // метод "получить значение поля X"
int GetY();       // метод "получить значение поля Y"
     .....
// все методы должны быть описаны, например
// реализация метода Move:

     Move(newX : int, newY : int) {
       X=newX;              // запись новых данных в объект
       Y=newY;              //
     }
   }                         // конец описания объекта

   // основная процедура программы

     Point A(0,0);   // создать новый объект и присвоить ему данные

     for(int i=0;i<100;i++)
{        // создать цикл
         A.Draw();         // нарисовать точку
         A.Hide();         // стереть точку
         A.Move(i,i*10);  // присвоить экземпляру объекта новые
данные
     }                //

     print(A.GetX(),A.GetY());
// получить и напечатать данные объекта
```

Из этого примера видно, что мы не можем напрямую обратиться к данным объекта, а должны вызывать метод Move для изменения его данных и GetX, GetY для считывания значений этих данных. Т.е. объект скрывает свою внутренню структуру, именно

это свойство и называется "инкапсуляцией".

- **наследование** - подразумевает возможность создавать из классов объектов новые классы объекты, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков).

Пример:

```
   Class Circle extend Point {
 // создаем новый объект "окружность",
 //наследующий свойства объекта "точка"
  Radius : int;
 // добавляем новое поле "радиус", поля X и Y наследуются
 // от родительского объекта
   .............
   Circle(X:int,Y:int,Radius:int); // конструктор нового объекта
   .............
   Draw();          // переопределяем некоторые методы
   Hide();          // родительского объекта, метод Move наследуется
   .............
   ChangeRadius(Radius);  // вводим новый метод "изменить радиус"
   .............
   GetRadius(); // вводим новый метод "получить значение радиуса"
                // методы GetX и GetY наследуются от родительского
                // объекта
 }
```

- **полиморфизм** - различные объекты могут по разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы.

Пример:

```
 Begin
  Point A(100,100);
  Circle B(200,200,50);

  A.Draw();       // рисует точку
  B.Draw();       // рисует окружность
End.
```

**Целостность данных:**

Для поддержания целостности объектно-ориентированный подход предлагает использовать следующие средства:

- автоматическое поддержание отношений наследования;
- возможность объявить некоторые поля данных и методы объекта как "скрытые", не видимые для других объектов; такие поля и методы используются только методами самого объекта;
- создание процедур контроля целостности внутри объекта.

**Средства манипулирования данными:**

Работа с данными в объектно-ориентированном программировании ведется с помощью одного из объектно-ориентированных языков программирования общего назначения, например, C++ или Java.

## 1.3. Классификация моделей данных

Одними из основополагающих в концепции баз данных являются обобщенные категории "данные" и "модель данных".

Понятие "данные" в концепции баз данных — это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы. Примеры данных: Петров Николай Степанович, $30 и т. д. Данные не обладают определенной структурой, данные становятся информацией тогда, когда пользователь задает им определенную структуру, то есть осознает их смысловое содержание. Поэтому центральным понятием в области баз данных является понятие модели. Не существует однозначного определения этого термина, у разных авторов эта абстракция определяется с некоторыми различиями, но тем не менее можно выделить нечто общее в этих определениях.

Модель данных - это некоторая абстракция, которая, будучи приложена к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

На рис. 2. представлена классификация моделей данных.

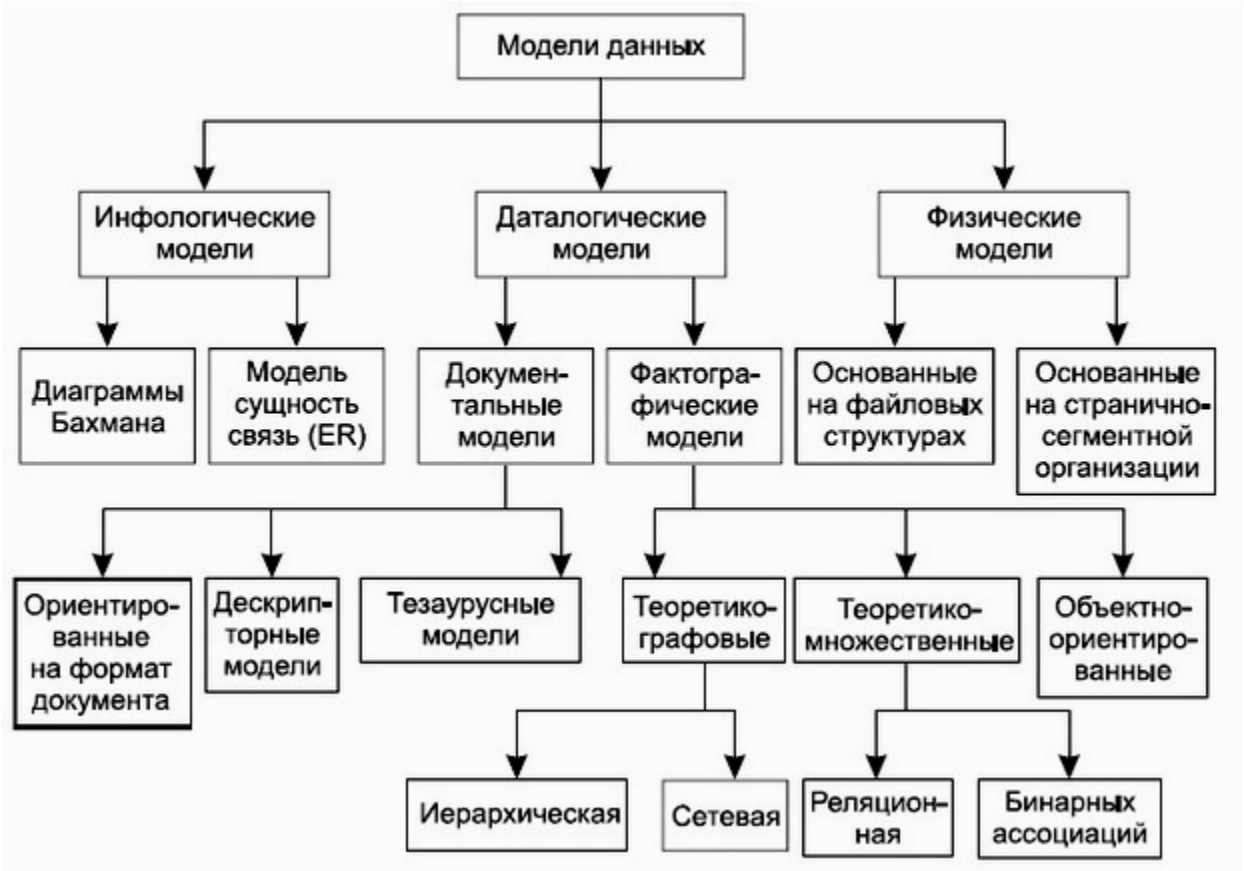



Рис. 2. Классификация моделей данных

В соответствии с рассмотренной ранее трехуровневой архитектурой мы сталкиваемся с понятием модели данных по отношению к каждому уровню. И действительно, физическая модель данных оперирует категориями, касающимися организации внешней памяти и структур хранения, используемых в данной операционной среде. В настоящий момент в качестве **физических моделей** используются различные методы размещения данных, основанные на файловых структурах: это организация файлов прямого и последовательного доступа, индексных файлов и инвертированных файлов, файлов, использующих различные методы хэширования, взаимосвязанных файлов. Кроме того, современные СУБД широко используют страничную организацию данных. Физические модели данных, основанные на страничной организации, являются наиболее перспективными.

Наибольший интерес вызывают модели данных, используемые на концептуальном уровне. По отношению к ним внешние модели называются подсхемами и используют те же абстрактные категории, что и концептуальные модели данных.

Кроме трех рассмотренных уровней абстракции при проектировании БД существует еще один уровень, предшествующий им. Модель этого уровня должна выражать информацию о предметной области в виде, независимом от используемой СУБД. Эти модели называются инфологическими, или семантическими, и отражают в естественной и удобной для разработчиков и других пользователей форме информационно-логический уровень абстрагирования, связанный с фиксацией и описанием объектов предметной области, их свойств и их взаимосвязей.

**Инфологические модели данных** используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а даталогические модели уже поддерживаются конкретной СУБД.

**Даталогическая модель данных**

Под даталогической понимается модель, отражающая логические взаимосвязи между элементами данных безотносительно их содержания и физической организации. При этом даталогическая модель разрабатывается с учетом конкретной реализации СУБД, также с учетом специфики конкретной предметной области на основе ее инфологической модели.

**Документальные модели** данных соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.

**Модели, ориентирванные на формат документа – HTML, XML.**

Модели, основанные на языках разметки документов, связаны прежде всего со стандартным общим языком разметки — SGML (Standart Generalised Markup Language), который был утвержден ISO в качестве стандарта еще в 80-х годах.

Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тегов (ссылок), их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования тегов осуществляется при помощи специального набора правил, называемых DTD-описаниями, которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил,

описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду некоторой своей сложности SGML использовался в основном для описания синтаксиса других языков (наиболее известным из которых является HTML), и немногие приложения работали с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор инструкций — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране программы-клиента и определяют этим самым способ представления документа, но не его структуру. В качестве элемента гипертекстовой базы данных, описываемой HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеет возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его сегодня главным механизмом представления информации в Интернете.

Однако HTML сегодня уже не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. И ему на смену был предложен новый язык гипертекстовой разметки, мощный, гибкий и, одновременно с этим, удобный язык XML. В чем же заключаются его достоинства?

XML (Extensible Markup Language) — это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Он используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. То есть сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания.

**Тезаурусные модели** основаны на принципе организации словарей, содержат определенные языковые конструкции и

принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

**Дескрипторные модели** — самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствовал дескриптор — описатель. Этот дескриптор имел жесткую структуру и описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой документальной БД. Например, для БД, содержащей описание патентов, дескриптор содержал название области, к которой относился патент, номер патента, дату выдачи патента и еще ряд ключевых параметров, которые заполнялись для каждого патента. Обработка информации в таких базах данных велась исключительно по дескрипторам, то есть по тем параметрам, которые характеризовали патент, а не по самому тексту патента.

На уровне **физической модели** электронная БД представляет собой файл или их набор в формате TXT, CSV, Excel, DBF, XML либо в специализированном формате конкретной СУБД. Также в СУБД в понятие физической модели включают специализированные виртуальные понятия, существующие в её рамках — таблица, табличное пространство, сегмент, куб, кластер и т. д.

**Иерархическая модель данных** — логическая модель данных в виде древовидной структуры.

**Сетевая модель данных** - это представление данных сетевыми структурами типов записей и связанных отношениями мощности один-к-одному или один-ко-многим.

**Реляционная модель** – представление данных в виде таблиц, между которыми опрелены определенные отношения.

## 1.4. Последовательность создания информационной модели данных

Процесс создания информационной модели данных начинается с определения концептуальных требований ряда пользователей. Концептуальные требования могут определяться и для некоторых задач (приложений), которые в ближайшее время реализовывать не планируется. Это может несколько повысить трудоемкость работы, однако поможет наиболее полно учесть все нюансы функциональности, требуемой для разрабатываемой системы, и снизит вероятность ее переделки в дальнейшем. Требования отдельных пользователей интегрируются в едином “обобщенном представлении”. Последнее называют концептуальной моделью.

Концептуальная модель представляет объекты и их взаимосвязи без указания способов их физического хранения.

Таким образом, концептуальная модель является, по существу, моделью предметной области. При проектировании концептуальной модели все усилия разработчика должны быть направлены в основном на структуризацию данных и выявление взаимосвязей между ними без рассмотрения особенностей реализации и вопросов эффективности обработки. Проектирование концептуальной модели основано на анализе решаемых на этом предприятии задач по обработке данных. Концептуальная модель включает описания объектов и их взаимосвязей, представляющих интерес в рассматриваемой предметной области и выявляемых в результате анализа данных. Здесь имеются в виду данные, используемые как в уже разработанных прикладных программах, так и в тех, которые только будут реализованы.

Концептуальная модель транслируется затем в модель данных, совместимую с выбранной СУБД. Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемыми средствами выбранной СУБД. Это потребует изменения концептуальной модели. Версия концептуальной модели, которая может быть обеспечена конкретной СУБД, называется логической моделью.

Логическая модель отражает логические связи между

элементами данных вне зависимости от их содержания и среде хранения.

Логическая модель данных может быть реляционной, иерархической или сетевой. Пользователям выделяются подмножества этой логической модели, называемые внешними моделями, отражающие их представления о предметной области. Внешняя модель соответствует представлениям, которые пользователи получают на основе логической модели, в то время как концептуальные требования отражают представления, которые пользователи первоначально желали иметь и которые легли в основу разработки концептуальной модели. Логическая модель отображается в физическую память

Физическая модель, определяющая размещение данных, методы доступа и технику индексирования, называется внутренней моделью системы.

Внешние модели никак не связаны с типом физической памяти, в которой будут храниться данные, и с методами доступа к этим данным. С другой стороны, если концептуальная модель способна учитывать расширение требований к системе в будущем, то вносимые в нее изменения не должны оказывать влияния на существующие внешние модели. Основное различие между указанными выше тремя типами моделей данных (концептуальной, логической и физической) состоит в способах представлении взаимосвязей между объектами. При проектировании БД нам потребуется различать взаимосвязи между объектами, между атрибутами одного объекта и между атрибутами различных объектов.

Моделирование данных проводится как поуровневый спуск от концептуальной модели к логической, а затем к физической модели.

Различие уровней представления данных на каждом этапе проектирования представлено в следующей таблице:

| | |
|---|---|
| **КОНЦЕПТУАЛЬНЫЙ УРОВЕНЬ**<br>• сущности<br>• атрибуты<br>• связи | Представление аналитика |

| | |
|---|---|
| **ЛОГИЧЕСКИЙ УРОВЕНЬ**<br>• записи<br>• элементы данных<br>• связи между записями | Представление программиста |
| **ФИЗИЧЕСКИЙ УРОВЕНЬ**<br>• группирование данных<br>• индексы<br>• методы доступа | Представление администратора |

## 2. МОДЕЛИ, ОСНОВАННЫЕ НА ЯЗЫКАХ РАЗМЕТКИ ДОКУМЕНТОВ

### 2.1. Язык XML

**XML** (Extensible Markup Language) — расширяемый язык разметки, фактически представляющий собой свод общих синтаксических правил. XML — текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML). XML является упрощённым подмножеством языка SGML.

Целью создания XML было обеспечение совместимости при передаче структурированных данных между разными системами обработки информации, особенно при передаче таких данных через Интернет. Словари, основанные на XML (например, RDF, RSS, MathML, XHTML, SVG), сами по себе формально описаны, что позволяет программно изменять и проверять документы на основе этих словарей, не зная их семантики, то есть не зная смыслового значения элементов. Важной особенностью XML также является применение так называемых пространств имён.

XML — это иерархическая структура, предназначенная для хранения любых данных, визуально структура может быть

представлена как дерево. Важнейшее обязательное синтаксическое требование заключается в том, что документ имеет только один корневой элемент. Это означает, что текст или другие данные всего документа должны быть расположены между единственным начальным корневым тегом и соответствующим ему конечным тегом.

Следующий простейший пример — правильно построенный документ XML:

<book>Это книга: "Книжечка"</book>

Первая строка XML-документа называется объявлением XML - это необязательная строка, указывающая версию стандарта XML, также здесь может быть указана кодировка символов и внешние зависимости.

<?xml version="1.0" encoding="UTF-8"?>

Спецификация требует, чтобы процессоры XML обязательно поддерживали Юникод-кодировки UTF-8 и UTF-16. Признаются допустимыми, поддерживаются и широко используются (но не обязательны) другие кодировки, основанные на стандарте ISO/IEC 8859, также допустимы другие кодировки, например, русские Windows-1251, KOI-8.

Комментарий может быть размещен в любом месте дерева. XML комментарии размещаются внутри пары тегов <!-- и -->. Два знака дефис (--) не могут быть применены ни в какой части внутри комментария.

<!-- Это комментарий. -->

Ниже приведён пример простого кулинарного рецепта, размеченного с помощью XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="хлеб" preptime="5" cooktime="180">
<title>Простой хлеб</title>
<ingredient amount="3" unit="стакан">Мука</ingredient>
<ingredient amount="0.25" unit="грамм">Дрожжи</ingredient>
<ingredient amount="1.5" unit="стакан">Тёплая вода</ingredient>
<ingredient amount="1" unit="чайная ложка">Соль</ingredient>
<instructions>
<step>Смешать все ингредиенты и тщательно замесить.</step>
<step>Закрыть тканью и оставить на один час в тёплом
```

помещении.</step>
<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>
</instructions>
</recipe>

Структура.

Остальная часть этого XML-документа состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержимое. Элемент обычно состоит из открывающего и закрывающего тегов, обрамляющих текст и другие элементы. Открывающий тег состоит из имени элемента в угловых скобках, например, <step>; закрывающий тег состоит из того же имени в угловых скобках, но перед именем ещё добавляется косая черта, например, </step>. Содержимым элемента (англ. content) называется всё, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы. Ниже приведён пример XML-элемента, который содержит открывающий тег, закрывающий тег и содержимое элемента:

<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>

Кроме содержания у элемента могут быть атрибуты — пары имя-значение, добавляемые в открывающий тег после названия элемента. Значения атрибутов всегда заключаются в кавычки (одинарные или двойные), одно и то же имя атрибута не может встречаться дважды в одном элементе. Не рекомендуется использовать разные типы кавычек для значений атрибутов одного тега.

<ingredient amount="3" unit="стакан">Мука</ingredient>

В приведённом примере у элемента «ingredient» есть два атрибута: «amount», имеющий значение «3», и «unit», имеющий значение «стакан». С точки зрения XML-разметки, приведённые атрибуты не несут никакого смысла, а являются просто набором символов.

**Документальные модели** данных соответствуют представлению о слабоструктурированной информации,

ориентированной в основном на свободные форматы документов, текстов на естественном языке. XML будем рассматривать как язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Он используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов.

То есть сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания. Очень часто разработчику программного обеспечения необходимо вынести настройку программы за пределы исходного текста, чтобы конфигурирование можно было проводит без перекомпиляции всей системы. К подобным настройкам можно отнести сетевые порты, адреса серверов, пути на жестком диске и даже внешний вид приложения. Многие разработчики придумывают свою систему: текстовые файлы, xml файлы, механизм properties.

Для самого широкого применения больше всего подходит xml. Как правило, большинство администраторов и программистов с xml уже знакомы, и редактирование настроек будет весьма тривиальным процессом. Да и иерархическая модель xml позволяет лучше скомпоновать настройки в группы. Примером подобного xml файла с настройками может служить фрагмент:

```
<config>
<listen>
<ip>192.168.5.2</ip>
<port>5677</port>
</listen>
<mail>
<smtp>mail.acme.com</smtp>
<login>username</login>
<password>12345</password>
</mail>
<performance>
<threads>10</threads>
<openfiles>50</openfiles>
<sessions>100</sessions>
<sessionTimeout>1h</sessionTimeout>
</performance>
```

</config>

Далее приведен пример приложения, которое прочитает подобный файл, и позволит читать все параметры. Для исполнения примеров на языке Java Вам понадобится JDK 1.5 или выше (хотя примеры сработают и в среде 1.4). Основная библиотека доступна для скачивания с сайта http://configloader.sourceforge.net/ где Вы можете скачать и исходные текста, и уже скопилированную библиотеку в виде JAR. ConfigLoader поддерживает загрузку конфигураций из файлов и текстовых переменных. Загрузка из текстовых переменных удобна при получении настроек с сервера, по сети.

Сами примеры были написаны в IDE Eclispe, но аналогичные действия можно произвести в любой другой среде разработке, или при работе с командной строки.

**Скачаем все необходимые библиотеки**

ConfigLoader - доступен на сайте по адресу: http://configloader.sourceforge.net/index.php/Downloads файл configloader_*.jar.

Дополнительно нам понадобится библиотека, реализующая SAX – simple api for XML. В мире java представлено множество подобных разработок, но для определенности воспользуемся xerces – решением от сообщества apache.

Актуальную версию можно скачать здесь:

http://xerces.apache.org/xerces2-j/download.cgi.

Здесь Вам понадобится файл - Xerces-J-bin.*.zip

На самом деле в ходе выполнении лабораторной работы Вам не потребуется скачивать все эти библиотеки, для их получения необходимо обратиться к преподавателю.

Для определенности мы хотим, чтобы наш первый класс читал секцию mail из файла конфигурации: smtp, login и password; и выдавал эти данные на консоль. Создадим новый класс “Example1” в пакете “configartice”. Для начала нам необходимо импортировать нужные классы, точнее класс:

```
import om.romanenco.configloader.ConfigLoader;
```

В самом теле класса создадим метод main

```
public static void main(String[] args) {
```

```
...
...
}
```

Наши дальнейшие строки кода мы будем добавлять в тело этого метода. Первая наша строка – создание объекта, который и позволит нам удобно работать с конфигурациями.

```
ConfigLoader config = new ConfigLoader();
```

Следующим шагом будет загрузка самой конфигурации, сделать это тоже весьма просто:

```
config.LoadFromFile("xml/example1.xml");
```

Теперь самое интересное, нам надо прочитать значения трех тегов smtp, login и password, которые вложены в тэги mail и config.

Код при этом прост:

```
System.out.println("Host = " +
config.getTagValue("config.mail.smtp"));

System.out.println("Username = " +
config.getTagValue("config.mail.login"));

System.out.println("Password = " +
config.getTagValue("config.mail.password"));
```

Не трудно заметить, как нам обратиться нужному тегу: надо просто задать цепочку имен тегов, начиная от корневого и так до него нужного нам.

Полный текст нашего примера будет таким:

```
package configarticle;
import com.romanenco.configloader.ConfigLoader;

public class Example1 {

public static void main(String[] args) {

ConfigLoader config = new ConfigLoader();
config.LoadFromFile("xml/example1.xml");
```

```
 System.out.println("Host = " +
config.getTagValue("config.mail.smtp"));
 System.out.println("Username = " +
config.getTagValue("config.mail.login"));
 System.out.println("Password = " +
config.getTagValue("config.mail.password"));
 }

}
```

Как мы видим, подобный подход к оформлению конфигурационных файлов создает удобства и для разработчика, и для конечного пользователя.

Разработчик может использовать простой и надежный подход во всех проектах, а пользователь (даже не высококвалифицированный) будет иметь возможность ясно настраивать приложение.

## 2.2. INI файл

Ini файлы – это простые текстовые файлами с расширением ini (в принципе можно и с любым другим). Ini файлы предназначены для хранения настроек программы.

Ini файлы разделены на разделы, содержащие идентификаторы, которым, в свою очередь, можно присвоить значения. В общем виде структура ini файла такова:

```
[Раздел1]

Идентификатор1=Значение1
Идентификатор2=Значение2

[Раздел2]

Идентификатор1=Значение1

И т.д.
```

Выражение *Идентификатор = Значение* очень часто называют ключом.

Пример ini файла:

```
[LOGS Database Driver]
DriverName=oracle.jdbc.driver.OracleDriver
DataBaseURL=jdbc:oracle:thin:@192.168.0.245:1521:xionet
UserName=xiouser
Password=user
[LOGS Directories]
TempDirectory=c:\\Sqlarch\\
ArchiveDirectory=c:\\Sqlarch\\
ReserveDirectory=c:\\sqlarch\\
[LOGS System Info]
SystemInfoId=1
[Timeouts]
RetryTimeout=10
```

Для чтения данных из ini файла можно использовать класс java.util.Properties и его метод load. После создания экземпляра класса Properties (например ini) и запуска его метода load можно получить имена полей (идентификаторов) и их значения следующим образом:

```
Enumeration e = ini.propertyNames() ;
        while (e.hasMoreElements()) {
        Object element=e.nextElement();
        System.out.print(element+"=");
        System.out.println(ini.getProperty((String)element));
    }
```

## 3. ПРЕДСТАВЛЕНИЕ ДАННЫХ В СУБД

### 3.1. Иерархическая модель представления данных

**Иерархическая модель данных** — логическая модель данных в виде древовидной структуры. Иерархическая модель данных

представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих перевернутое дерево (граф). Данная модель характеризуется такими параметрами, как уровни, узлы, связи. Принцип работы модели таков, что несколько узлов более низкого уровня соединяется при помощи связи с одним узлом более высокого уровня.

Часто организацию данных в СУБД иерархического типа определяется в терминах: элемент, агрегат, запись (группа), групповое отношение, база данных.

- ***Атрибут (элемент данных)*** - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.
- ***Запись*** - именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи - конкретная запись с конкретным значением элементов
- ***Групповое отношение*** - иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.

При графическом изображении групповые отношения изображают дугами ориентированного графа, а типы записей - вершинами (диаграмма Бахмана).

Для групповых отношений в иерархической модели обеспечивается автоматический режим включения и фиксированное

членство. Это означает, что для запоминания любой некорневой записи в БД должна существовать ее родительская запись (подробнее о режимах включения и исключения записей сказано в параграфе о сетевой модели). При удалении родительской записи автоматически удаляются все подчиненные.

Пример:

Рассмотрим следующую модель данных предприятия (рис. 3). Предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

Поэтому, для информационной системы управления персоналом необходимо создать групповое отношение, состоящее из родительской записи ОТДЕЛ (НАИМЕНОВАНИЕ_ОТДЕЛА, ЧИСЛО_РАБОТНИКОВ) и дочерней записи СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД). Это отношение показано на рис. 3(a) (Для простоты полагается, что имеются только две дочерние записи).

Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры : заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи ЗАКАЗЧИК(НАИМЕНОВАНИЕ_ЗАКАЗЧИКА, АДРЕС), КОНТРАКТ(НОМЕР, ДАТА,СУММА), ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ_ОТДЕЛА) (рис. 3(b)).

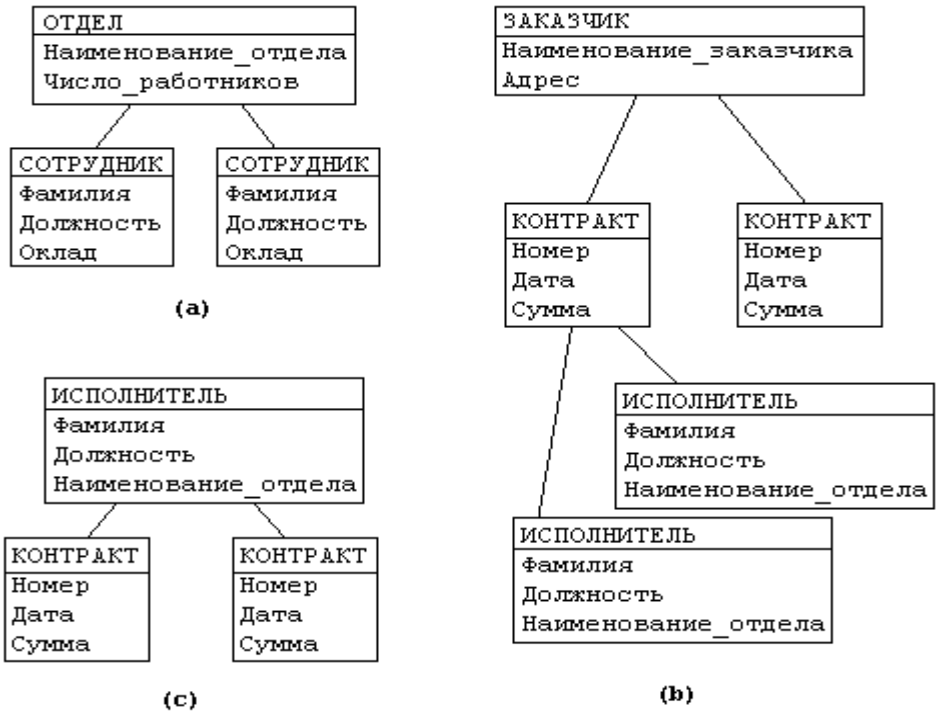



Рис. 3. Иерархическая модель данных предприятия

Из этого примера видны недостатки иерархических БД:

- Частично дублируется информация между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ (такие записи называют парными), причем в иерархической модели данных не предусмотрена поддержка соответствия между парными записями.
- Иерархическая модель реализует отношение между исходной и дочерней записью по схеме 1:N, то есть одной родительской записи может соответствовать любое число дочерних. Допустим теперь, что исполнитель может принимать участие более чем в одном контракте (т.е. возникает связь типа M:N). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором ИСПОЛНИТЕЛЬ будет являться исходной записью, а КОНТРАКТ - дочерней (рис. (c)). Таким образом, мы опять вынуждены дублировать информацию.

*Операции над данными, определенные в иерархической модели:*

- ДОБАВИТЬ в базу данных новую запись. Для корневой записи обязательно формирование значения ключа.
- ИЗМЕНИТЬ значение данных предварительно

извлеченной записи. Ключевые данные не должны подвергаться изменениям.

- УДАЛИТЬ некоторую запись и все подчиненные ей записи.
- ИЗВЛЕЧЬ:
  - извлечь корневую запись по ключевому значению, допускается также последовательный просмотр корневых записей
  - извлечь следующую запись (следующая запись извлекается в порядке левостороннего обхода дерева)

В операции ИЗВЛЕЧЬ допускается задание условий выборки (например, извлечь сотрудников с окладом более 1 тысячи руб.)

Как видим, все операции изменения применяются только к одной "текущей" записи (которая предварительно извлечена из базы данных). Такой подход к манипулированию данных получил название "навигационного".

## 3.2. Сетевая модель представления данных

*Сети* - естественный способ представления отношений между объектами. Они широко применяются в математике, исследованиях операций, химии, физике, социологии и других областях знаний. Сети обычно могут быть представлены математической структурой, которая называется *направленным графом.* Направленный граф имеет простую структуру. Он состоит из точек или *узлов,* соединенных стрелками или ребрами. В контексте моделей данных узлы можно представлять как типы записей данных, а ребра представляют отношения один-к-одному или один-ко-многим. Структура графа делает возможными простые представления иерархических отношений (таких, как генеалогические данные) .

*Сетевая модель данных* - это представление данных сетевыми структурами типов записей и связанных отношениями мощности один-к-одному или один-ко-многим. В конце 60-х конференция по языкам систем данных (Conference on Data Systems Languages, CODASYL) поручила подгруппе, названной Database Task Group (DTBG), разработать стандарты систем управления базами данных. На DTBG оказывала сильное влияние архитектура, использованная в одной из самых первых СУБД, Iategrated Data Store (IDS),

созданной ранее компанией General Electric.Это привело к тому, что была рекомендована сетевая модель.

Документы Database Task Group (DTBG) (группа для разработки стандартов систем управления базами данных) от 1971 года остается основной формулировкой сетевой модели, на него ссылаются как на модель CODASYL DTBG. Она послужила основой для разработки сетевых систем управления базами данных нескольких производителей. IDS (Honeywell) и IDMS (Computer Associates) - две наиболее известных коммерческих реализации. В сетевой модели существует две основные структуры данных: типы записей и наборы:

- *Тип записей.* Совокупность логически связанных элементов данных.
- *Набор.* В модели DTBG отношение один-ко-многим между двумя типами записей.
- *Простая сеть.* Структура данных, в которой все бинарные отношения имеют мощность один-ко-многим.
- *Сложная сеть.* Структура данных, в которой одно или несколько бинарных отношений имеют мощность многие-ко-многим.
- *Тип записи связи*. Формальная запись, созданная для того, чтобы преобразовать сложную сеть в эквивалентную ей простую сеть.

В модели DBTG возможны только простые сети, в которых все отношения имеют мощность один-к-одному или один-ко-многим. Сложные сети, включающие одно или несколько отношений многие-ко-многим, не могут быть напрямую реализованы в модели DBTG. Следствием возможности создания искусственных формальных записей является необходимость дополнительного объема памяти и обработки, однако при этом модель данных имеет простую сетевую форму и удовлетворяет требованиям DBTG.

Иерархическая структура преобразовывается в сетевую следующим образом (рис. 4):

- древья (a) и (b), показанные на рис. 3, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два

групповых отношения;

- для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, см. рис. 4.(Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту.)

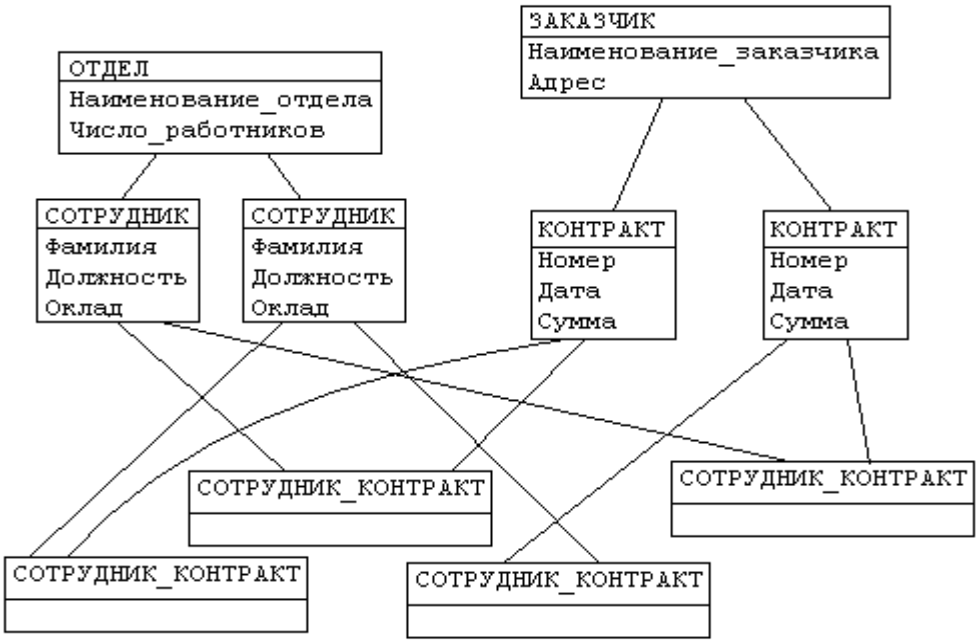



Рис. 4. Сетевая модель данных предприятия

Реализация групповых отношений в сетевой модели осуществляется с использованием специально вводимых дополнительных полей - указателей (адресов связи или ссылок), которые устанавливают связь между владельцем и членом группового отношения. Запись может состоять в отношениях разных типов (1:1, 1:N, M:N). Заметим, что если один из вариантов установления связи 1:1 очевиден (в запись – владелец отношения, поля которой соответствуют атрибутам сущности, включается дополнительное поле – указатель на запись – член отношения), то возможность представления связей 1:N и M:N таким же

образом весьма проблематична. Поэтому наиболее распространенным способом организации связей в сетевых СУБД является введение дополнительного типа записей (и соответственно, дополнительного файла), полями которых являются указатели.

Наиболее существенным недостатком сетевой модели является "жесткость" получаемой концептуальной схемы. Связи закреплены в записях в виде указателей. При появлении новых аспектов использования этих же данных может возникнуть необходимость установления новых связей между ними. Это требует введения в записи новых указателей, т.е. изменения структуры БД, и, соответственно, переформирования всей базы данных.

СУБД, поддерживающие сетевую модель, широко использовались на вычислительных системах серии IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать IDMS, UNIBAD (БАНК), и их аналоги СЕДАН, СЕТОР. На персональных компьютерах сетевые СУБД не получили широкого распространения. Примером сетевой СУБД для персонального компьютера является db_VISTA III. Отметим, что система db_VISTA реализована на языке С и поэтому является переносимой. Система может эксплуатироваться на ПЭВМ типа IBM PC, SUN, Macintosh.

## 3.3. Реляционная модель представления данных

Учитывая отмеченные в предыдущих разделах недостатки сетевых и иерархических моделей, можно сформулировать желательные требования к модели данных:

- модель должна быть понятна пользователю, не имеющему особых навыков в программировании;
- появление новых аспектов использования данных и необходимость введения новых связей не должны приводить к реструктуризации всей модели данных и базы данных в целом.

Моделью данных, удовлетворяющей вышеуказанным требованиям, является реляционная модель, часто называемая также табличной.

Основным используемым понятиям здесь является понятие отношения, представляемого в виде таблицы, столбцы которой соответствуют атрибутам сущности (структура строки таблицы аналогична структуре записи). Каждый атрибут может принимать определенное множество значений, называемое доменом. Строка таблицы с конкретными значениями полей здесь называется кортежем (соответствует понятию "экземпляр записи"). Поля таблицы предполагаются элементарными (неделимыми). Таким образом, понятие "таблица" здесь соответствует понятию "файл" модели данных. Первичный ключ здесь – минимальный набор атрибутов, однозначно идентифицирующий кортеж в отношении.

В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей. Рассмотренный ранее пример базы данных, содержащей сведения о подразделениях предприятия и работающих в них сотрудниках, применительно к реляционной модели будет иметь вид:


Рис. 5. База данных о подразделениях и сотрудниках предприятия.

Например, связь между отношениями ОТДЕЛ и СОТРУДНИК создается путем копирования первичного ключа *"Номер_отдела"* из первого отношения во второе. Таким образом:

- для того, чтобы получить список работников данного подразделения, необходимо

1. из таблицы ОТДЕЛ установить значение атрибута

*"Номер_отдела"*, соответствующее данному "Наименованию_отдела"

2. выбрать из таблицы СОТРУДНИК все записи, значение атрибута *"Номер_отдела"* которых равно полученному на предыдущем шаге.

- для того, чтобы узнать в каком отделе работает сотрудник, нужно выполнить обратную операцию:

1. определяем *"Номер_отдела"* из таблицы СОТРУДНИК
2. по полученному значению находим запись в таблице ОТДЕЛ.

Атрибуты, представляющие собой копии ключей других отношений, называются внешними ключами.

Таким образом, реляционная модель представления данных представляет собой модель, в которой данные хранятся в виде таблиц, между которыми установлены определенные связи. Для работы с такими моделями данных с использованием языка Java предлагается следующая схема:

*Пошаговая инструкция.*

Связь с базой данных с помощью JDBC.

1. Установка связи между Java-программой и диспетчером базы данных.
2. Передача SQL-команды в базу данных с помощью объекта Statement.
3. Чтение полученных результатов из базы данных и использование их в программе.

**Первый этап**

Первый этап включает в себя подключение драйвера базы данных и непосредственно подключение к ней. Реализация этого этапа представляется следующим образом:

Class.forName(“org.gjt.mm.mysql.Driver”);

Conection connection=DriverManager.getConnection (“jdbc:mysql://localhost:3306/NewsDB”, “User”, “Password”);

После регистрации драйвера с помощью диспетчера драйверов его можно применять для подключения к базе данных. Для этого диспетчеру драйверов следует сообщить о создании нового подключения. В ответ на это диспетчер драйверов вызовет соответствующий драйвер и возвратит ссылку на установленное подключение. Для создания подключения необходимо указать

место расположения базы данных, а также (как правило, для большинства баз данных) учетное имя и пароль.

**Второй этап**

Объект Statement предназначен для хранения SQL-команд. При пересылке объекта Statement базе данных с помощью установленного подключения СУБД запустит заданную SQL-команду и возвратит результат ее выполнения в виде объекта ResultSet.

Реализация второго этапа выглядит так:

```
Statement statement=connection.createStatement();
ResultSet=statement.executeQuery(“select * from vus”);
```

**Третий этап**

Работа с результатами выполнения запроса. Объект ResultSet функционирует как курсор, в котором доступ к отдельной строке можно осуществить с помощью команды SELECT. Для перехода к следующей строке необходимо вызвать метод next(), который возвращает логическое значение, например значение false возвращается после перебора всех строк, т.е. когда следующей строки уже нет.

Внутри строки содержимое столбцов считывается в порядке слева направо. (Напомним. что команду SELECT можно использовать для выборки необходимых для работы столбцов в заданном порядке.) Специалистами фирмы Sun разработаны методы qetTun() для всех основных типов данных, которые совместимы с SQL.

Рассмотрим пример.

```
ResultSet theResults =
theStatement.executeQuery
("SELECT title, author, retailPrice FROM books");
while (theResults.next())
{
String theTitle = theResults.getString("title");
String theAuthor = theResults.getString("author");
float thePrice = theResults.getFloat("retailPrice");
System.out.println(theTitle + " " + theAuthor + " " + thePrice.
toString());
}
```

Код будет более понятным, если извлекать столбцы по их названию, но для небольшого повышения производительности их можно извлекать и по порядковому номеру, как показано в следующем примере:

float thePrice = theResults.getFloat(3);

По окончании работы с объектами Statement и Connection для них обоих рекомендуется вызвать метод close (). Это позволяет высвободить ресурсы, как правило, задолго до того. как сборщик мусора приступит к своей работе.

Пример подключения к базе данных MySql

Приведем пример подключения и работы с базой данных MySql:

```
import java.sql.*;
……
try      {
Class.forName(“org.gjt.mm.mysql.Driver”);
Conection connection=DriverManager.getConnection
(“jdbc:mysql://localhost:3306/NewsDB”, “User”, “Password”);
Statement statement=connection.createStatement();
}
catch (ClassNotFoundExeception e)
        {
System.out.println(“Cannot connection to database”);
        }
        catch (SQLException e)
        {
        System.out.println(“Cannot connection to database”);
        }
        ResultSet=statement.executeQuery(“select * from vus”);
        while (rs.next())
                {
                String idS=rs.getString(1);
                System.out.println(“строка=”+idS);
                }
        rs.close();
        statement.close();
```

```
connection.close();
```

## 3.4 Теория нормальных форм

Реляционная база данных содержит как структурную, так и семантическую информацию. Структура базы данных определяется числом и видом включенных в нее отношений, и связями типа "один ко многим", существующими между кортежами этих отношений. Семантическая часть описывает множество функциональных зависимостей, существующих между атрибутами этих отношений. Дадим определение функциональной зависимости.

Определение:

Если даны два атрибута X и Y некоторого отношения, то говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X соответствует ровно одно значение Y.

Функциональная зависимость обозначается X -> Y. Отметим, что X и Y могут представлять собой не только единичные атрибуты, но и группы, составленные из нескольких атрибутов одного отношения.

Можно сказать, что функциональные зависимости представляют собой связи типа "один ко многим", существующие внутри отношения.

Некоторые функциональные зависимости могут быть нежелательны.

Определение:

Избыточная функциональная зависимость - зависимость, заключающая в себе такую информацию, которая может быть получена на основе других зависимостей, имеющихся в базе данных.

Корректной считается такая схема базы данных, в которой отсутствуют избыточные функциональные зависимости. В противном случае приходится прибегать к процедуре декомпозиции (разложения) имеющегося множества отношений. При этом порождаемое множество содержит большее число отношений, которые являются проекциями отношений исходного множества. (Операция проекции описана в разделе, посвященном реляционной

алгебре). Обратимый пошаговый процесс замены данной совокупности отношений другой схемой с устранением избыточных функциональных зависимостей называется нормализацией.

Условие обратимости требует, чтобы декомпозиция сохраняла эквивалентность схем при замене одной схемы на другую, т.е. в результирующих отношениях:

- не должны появляться ранее отсутствовавшие кортежи;
- на отношениях новой схемы должно выполняться исходное множество функциональных зависимостей.

**1NF - первая нормальная форма.**

Для обсуждения первой нормальной формы необходимо дать два определения:

Простой атрибут - атрибут, значения которого атомарны (неделимы).

Сложный атрибут - получается соединением нескольких атомарных атрибутов, которые могут быть определены на одном или разных доменах. (его также называют вектор или агрегат данных).

Теперь можно дать определение первой нормальной формы: отношение находится в 1NF если значения всех его атрибутов атомарны.

Рассмотрим пример:

В базе данных отдела кадров предприятия необходимо хранить сведения о служащих, которые можно попытаться представить в отношении СЛУЖАЩИЙ(НОМЕР_СЛУЖАЩЕГО, ИМЯ, ДАТА_РОЖДЕНИЯ, ИСТОРИЯ_РАБОТЫ, ДЕТИ).

Из внимательного рассмотрения этого отношения следует, что атрибуты "история_работы" и "дети" являются сложными, более того, атрибут "история_работы" включает еще один сложный атрибут "история_зарплаты".

Данные агрегаты выглядят следующим образом:

- ИСТОРИЯ_РАБОТЫ (ДАТА_ПРИЕМА, НАЗВАНИЕ, ИСТОРИЯ_ЗАРПЛАТЫ),
- ИСТОРИЯ_ЗАРПЛАТЫ (ДАТА_НАЗНАЧЕНИЯ, ЗАРПЛАТА),
- ДЕТИ (ИМЯ_РЕБЕНКА, ГОД_РОЖДЕНИЯ).

Их связь представлена на рис. 6.

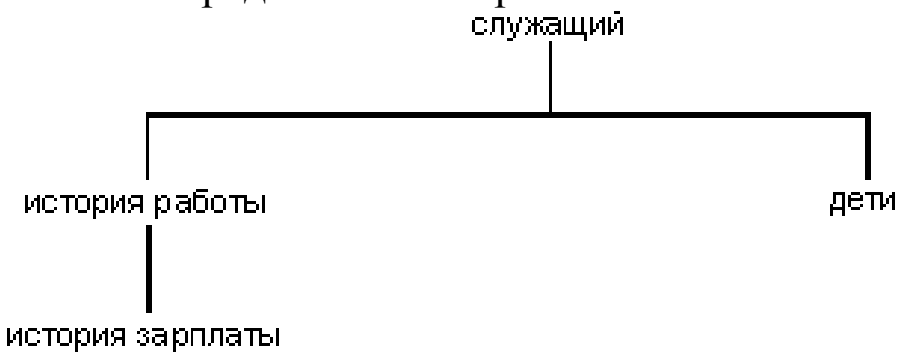



Рис. 6. Исходное отношение.

Для приведения исходного отношения СЛУЖАЩИЙ к первой нормальной форме необходимо декомпозировать его на четыре отношения, так как это показано на следующем рисунке:

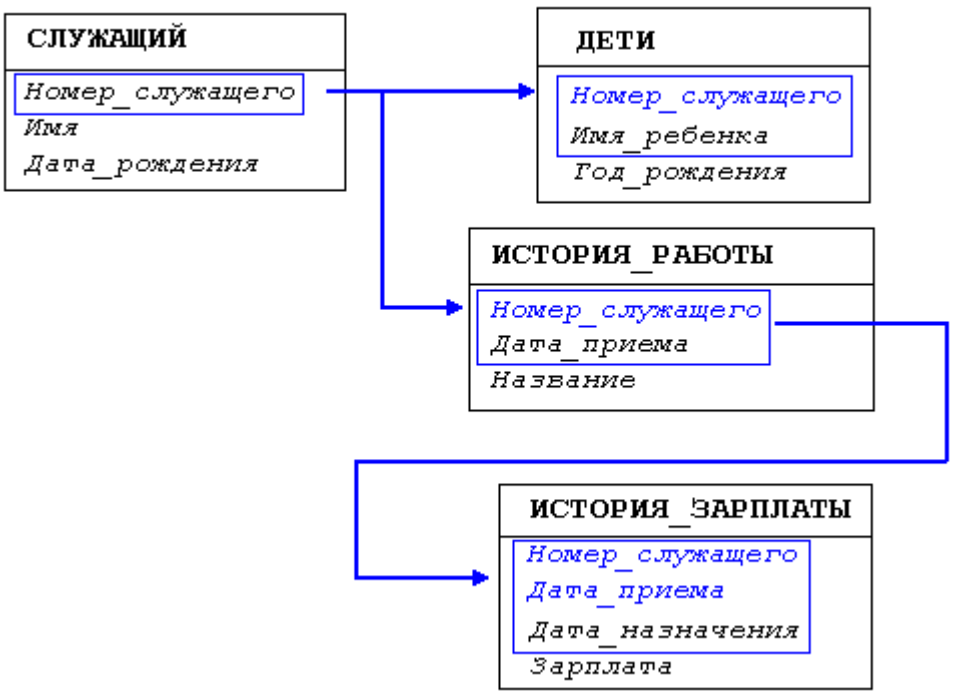



Рис. 7. Нормализованное множество отношений.

Здесь первичный ключ каждого отношения выделен синей рамкой, названия внешних ключей набраны шрифтом синего цвета. Напомним, что именно внешние ключи служат для представления функциональных зависимостей, существующих в исходном отношении. Эти функциональные зависимости обозначены линиями со стрелками.

Алгоритм нормализации описан Е.Ф.Коддом следующим образом:

- Начиная с отношения, находящегося на верху дерева (рис. 6), берется его первичный ключ, и каждое непосредственно подчиненное отношение расширяется путем вставки домена или комбинации доменов этого первичного ключа.
- Первичный ключ каждого расширенного таким образом отношения состоит из первичного ключа, который был у этого отношения до расширения и добавленного первичного ключа родительского отношения.
- После этого из родительского отношения вычеркиваются все непростые домены, удаляется верхний узел дерева, и эта же процедура повторяется для каждого из оставшихся поддеревьев.

**2NF - вторая нормальная форма.**

Очень часто первичный ключ отношения включает несколько атрибутов (в таком случае его называют составным) - см., например, отношение ДЕТИ, показанное на рис. 7. При этом вводится понятие полной функциональной зависимости.

Определение: неключевой атрибут функционально полно зависит от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.

Пример:

Пусть имеется отношение ПОСТАВКИ (N_ПОСТАВЩИКА, ТОВАР, ЦЕНА).

Поставщик может поставлять различные товары, а один и тот же товар может поставляться разными поставщиками. Тогда ключ отношения - "N_поставщика + товар". Пусть все поставщики поставляют товар по одной и той же цене. Тогда имеем следующие функциональные зависимости:

- N_поставщика, товар -> цена
- товар -> цена

Неполная функциональная зависимость атрибута "цена" от ключа приводит к следующей аномалии: при изменении цены товара необходим полный просмотр отношения для того, чтобы

изменить все записи о его поставщиках. Данная аномалия является следствием того факта, что в одной структуре данных объединены два семантических факта. Следующее разложение дает отношения во 2НФ:

- ПОСТАВКИ (N_ПОСТАВЩИКА, ТОВАР)
- ЦЕНА_ТОВАРА (ТОВАР, ЦЕНА)

Таким образом, можно дать

**Определение второй нормальной формы:**

Отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

**3NF - третья нормальная форма.**

Перед обсуждением третьей нормальной формы необходимо ввести понятие **транзитивной функциональной зависимости**.

**Определение:**

Пусть X, Y, Z - три атрибута некоторого отношения. При этом X-->Y и Y-->Z, но обратное соответствие отсутствует, т.е. Z -/-> Y и Y-/->X. Тогда Z транзитивно зависит от X.

Пусть имеется отношение ХРАНЕНИЕ (ФИРМА, СКЛАД, ОБЪЕМ), которое содержит информацию о фирмах, получающих товары со складов, и объемах этих складов. Ключевой атрибут - "фирма". Если каждая фирма может получать товар только с одного склада, то в данном отношении имеются следующие функциональные зависимости:

- фирма -> склад
- склад -> объем

При этом возникают аномалии:

- если в данный момент ни одна фирма не получает товар со склада, то в базу данных нельзя ввести данные о его объеме (т.к. не определен ключевой атрибут)
- если объем склада изменяется, необходим просмотр всего отношения и изменение кортежей для всех фирм, связанных с данным складом.

Для устранения этих аномалий необходимо декомпозировать исходное отношение на два:

- ХРАНЕНИЕ (ФИРМА, СКЛАД)

- ОБЪЕМ_СКЛАДА (СКЛАД, ОБЪЕМ)

**Определение третьей нормальной формы:**

Отношение находится в 3НФ, если оно находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

# 4. МОДЕЛИРОВАНИЕ ДАННЫХ

## 4.1. Подходы к моделированию данных

Существует два основных способа проектирования программных систем и данных - структурное проектирование, основанное на алгоритмической декомпозиции, и объектно-ориентированное проектирование, основанное на объектно-ориентированной декомпозиции. Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение объектам действия.

Алгоритмическую декомпозицию можно представить как обычное разделение алгоритмов, где каждый модуль системы выполняет один из этапов общего процесса. При объектно-ориентированной декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира.

Объектная декомпозиция имеет несколько преимуществ перед алгоритмической.

• Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов.

• Объектно-ориентированные системы более гибки и проще эволюционируют со временем.

В объектно-ориентированном анализе существует четыре основных типа моделей: **динамическая, статическая, логическая и физическая**.

Через них можно выразить результаты анализа и проектирования, выполненные в рамках любого проекта. Эти модели в совокупности семантически достаточно

универсальны, чтобы разработчик мог выразить все заслуживающие внимания стратегические и тактические решения, которые он должен принять при анализе системы и формировании ее архитектуры.

Кроме того, эти модели достаточно полны, чтобы служить техническим проектом реализации практически на любом объектно-ориентированном языке программирования.

Фактически все сложные системы можно представить одной и той же канонической формой - в виде двух ортогональных иерархий одной системы: классов и объектов. Каждая иерархия является многоуровневой, причем в ней классы и объекты более высокого уровня построены из более простых. Какой класс или объект выбран в качестве элементарного, зависит от рассматриваемой задачи. Объекты одного уровня имеют четко выраженные связи, особенно это касается компонентов структуры объектов.

Внутри любого рассматриваемого уровня находится следующий уровень сложности. Структуры классов и объектов не являются независимыми:

каждый элемент структуры объектов представляет специфический экземпляр определенного класса. Объектов в сложной системе обычно гораздо больше, чем классов. С введением структуры классов в ней размещаются общие свойства экземпляров классов.

**Сущность структурного подхода** к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. Общая модель системы строится в виде некоторой иерархической структуры, которая отражает различные уровни абстракции с ограниченным числом компонентов на каждом из уровней. Одним из главных принципов структурного системного анализа является выделение на каждом из уровней абстракции только наиболее существенных компонентов или элементов системы.

Все наиболее распространенные методологии структурного

подхода базируются на ряде общих принципов. В качестве двух базовых принципов используются следующие:

- принцип "разделяй и властвуй" - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;
- принцип иерархического упорядочивания - принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта). Основными из этих принципов являются следующие:

- принцип абстрагирования - заключается в выделении существенных аспектов системы и отвлечения от несущественных;
- принцип формализации - заключается в необходимости строгого методического подхода к решению проблемы;
- принцип непротиворечивости - заключается в обоснованности и согласованности элементов;
- принцип структурирования данных - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используется ряд средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм). Наиболее распространенной диаграммой, предназначенной для моделирования данных, является диаграмма ERD (Entity-Relationship Diagrams) "сущность-связь".

## 4.2. Диаграммы потоков данных (DFD)

Диаграммы DFD *(Data Flow Diagramming)* могут дополнить то, что уже отражено в модели IDEF0, поскольку они описывают потоки данных, позволяя проследить, каким образом происходит обмен информацией как внутри системы между бизнес-функциями,

так и системы в целом с внешней информационной средой.

Для усиления функциональности в данной нотации диаграмм предусмотрены специфические элементы, предназначенные для описания информационных и документопотоков, такие как внешние сущности и хранилища данных.

Без объекта "внешняя сущность" аналитику бывает иногда сложно определить, откуда пришли в компанию данные документы. Или какие документы еще приходят от, такой внешней сущности как, например, "клиент". Объект "хранилище данных" является уникальным обозначением длительного хранения, очередности обработки, резерва документов.

Это представление потоков совместно с хранилищами данных и внешними сущностями делает модели DFD более похожими на физические характеристики системы — движение объектов, хранение объектов, поставка и распространение объектов.

Основой данной методологии графического моделирования информационных систем является специальная технология построения диаграмм потоков данных DFD. В разработке методологии DFD приняли участие многие аналитики, среди которых следует отметить Э. Йордона (E. Yourdon). Он является автором одной из первых графических нотаций DFD. В настоящее время наиболее распространенной является так называемая нотация Гейна-Сарсона (Gene-Sarson), основные элементы которой будут рассмотрены в этом разделе.

Диаграммы потоков данных (DFD – Data Flow Diagram) являются основным средством моделирования функциональных требований проектируемой системы. С их помощью эти требования разбиваются на функциональные компоненты (процессы) и представляются в виде сети, связанной потоками данных. Главная цель таких средств – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Система в контексте DFD представляется в виде некоторой информационной модели, основными компонентами которой являются различные потоки данных, которые переносят информацию от одной подсистемы к другой. Каждая из подсистем выполняет определенные преобразования входного потока данных и передает результаты обработки информации в виде потоков

данных для других подсистем.

Для изображения DFD традиционно используются две различные нотации: Йодана (Yourdon) и Гейна-Сарсона (Gane-Sarson). Далее при построении примеров будет использоваться нотация Йодана, все исключения будут предварительно оговариваться.

На диаграммах функциональные требования представляются с помощью процессов и хранилищ, связанных потоками данных. Основные компоненты диаграмм потоков данных изображены на рис. 8. Опишем их назначение.

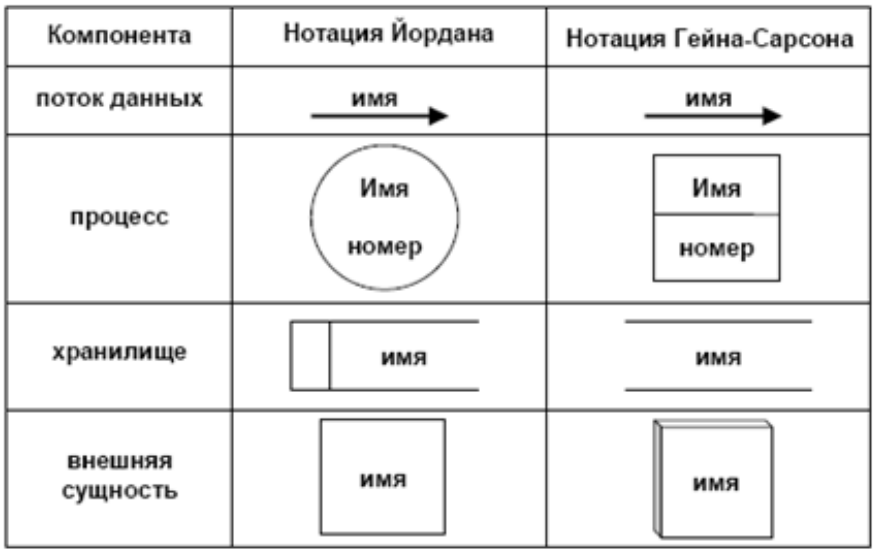



Рис. 8. Основные символы диаграммы потоков данных

**Потоки данных** являются механизмами, использующимися для моделирования передачи информации (или даже физических компонент) из одной части системы в другую. Важность этого объекта очевидна: он дает название целому инструменту. Потоки на диаграммах обычно изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение процесса состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Это имя должно содержать глагол в неопределенной форме с последующим дополнением (например, «вычислить максимальную высоту»). Кроме того, каждый процесс должен иметь уникальный номер для ссылок на него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Накопитель данных или хранилище представляет собой абстрактное устройство или способ хранения информации, перемещаемой между процессами. Хранилище (накопитель) данных позволяет на определенных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее определения, при этом данные могут выбираться в любом порядке. Имя хранилища должно идентифицировать его содержимое и быть существительным. В случае, когда поток данных входит или выходит в/из хранилища, и его структура соответствует структуре хранилища, он должен иметь то же самое имя, которое нет необходимости отражать на диаграмме.

Накопитель данных может быть физически реализован различными способами, но наиболее часто предполагается его реализация в электронном виде на магнитных носителях

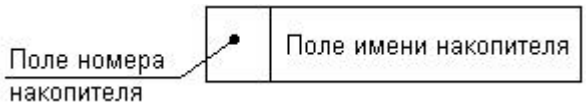



Рис. 9. Изображение накопителя на диаграмме потоков данных

**Внешняя сущность (терминатор)** представляет сущность вне контекста системы, являющуюся источником или приемником системных данных. Ее имя должно содержать существительное, например, *склад товаров*. Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке. Примерами внешних сущностей могут служить: клиенты организации, заказчики, персонал, поставщики.

**Контекстная диаграмма и детализация процессов**

Декомпозиция DFD осуществляется на основе процессов: каждый процесс может раскрываться с помощью DFD нижнего уровня.

Важную специфическую роль в модели играет специальный вид DFD - контекстная диаграмма, моделирующая систему наиболее общим образом.

Контекстная диаграмма отражает интерфейс системы с

внешним миром, а именно, информационные потоки между системой и внешними сущностями, с которыми она должна быть связана. Она идентифицирует эти внешние сущности, а также, как правило, единственный процесс, отражающий главную цель или природу системы насколько это возможно. И хотя контекстная диаграмма выглядит тривиальной, несомненная ее полезность заключается в том, что она устанавливает границы анализируемой системы. Каждый проект должен иметь ровно одну контекстную диаграмму, при этом нет необходимости в нумерации единственного ее процесса.

DFD первого уровня строится как декомпозиция процесса, который присутствует на контекстной диаграмме.

Построенная диаграмма первого уровня также имеет множество процессов, которые в свою очередь могут быть декомпозированы в DFD нижнего уровня. Таким образом, строится иерархия DFD с контекстной диаграммой в корне дерева. Этот процесс декомпозиции продолжается до тех пор, пока процессы могут быть эффективно описаны с помощью коротких (до одной страницы) миниспецификаций обработки (спецификаций процессов). При таком построении иерархии DFD каждый процесс более низкого уровня необходимо соотнести с процессом верхнего уровня. Обычно для этой цели используются структурированные номера процессов. Так, например, если мы детализируем процесс номер 2 на диаграмме первого уровня, раскрывая его с помощью DFD, содержащей три процесса, то их номера будут иметь следующий вид: 2.1, 2.2 и 2.3. При необходимости можно перейти на следующий уровень, т.е. для процесса 2.2 получим 2.2.1, 2.2.2. и т.д.

## Декомпозиция данных и соответствующие расширения диаграмм потоков данных

Индивидуальные данные в системе часто являются независимыми. Однако иногда необходимо иметь дело с несколькими независимыми данными одновременно. Например, в

системе имеются потоки *яблоки*, *апельсины* и *груши*. Эти потоки могут быть сгруппированы с помощью введения нового потока *фрукты*. Для этого необходимо определить формально поток *фрукты* как состоящий из нескольких элементов-потомков. Такое определение задается в словаре данных. В свою очередь поток *фрукты* сам может содержаться в потоке-предке *еда* вместе с потоками *овощи, мясо* и др. Такие потоки, объединяющие несколько потоков, получили название групповых.

Обратная операция, расщепление потоков на подпотоки, осуществляется с использованием группового узла (рис.38), позволяющего расщепить поток на любое число подпотоков. При расщеплении также необходимо формально определить подпотоки в словаре данных.

Аналогичным образом осуществляется и декомпозиция потоков через границы диаграмм, позволяющая упростить детализирующую DFD. Пусть имеется поток *фрукты*, входящий в детализируемый процесс. На детализирующей этот процесс диаграмме потока *фрукты* может не быть вовсе, но вместо него могут быть потоки *яблоки* и *апельсины* (как будто бы они переданы из детализируемого процесса). В этом случае должно существовать определение потока *фрукты*, состоящего из подпотоков *яблоки* и *апельсины*, для целей балансирования.

Применение этих операций над данными позволяет обеспечить структуризацию данных, увеличивает наглядность и читабельность диаграмм.

Для обеспечения декомпозиции данных и некоторых других сервисных возможностей к DFD добавляются следующие типы объектов.

1. **Групповой узел.** Предназначен для расщепления и объединения потоков.

В некоторых случаях может отсутствовать (т.е. фактически вырождаться в точку слияния/расщепления потоков на диаграмме).

2. **Узел-предок.** Позволяет увязывать входящие и выходящиепотоки между детализируемым процессом и детализирующей DFD.

3. **Неиспользуемый узел.** Применяется в ситуации, когда декомпозиция данных производится в групповом узле, при этом требуются не все элементы входящего в узел потока.

4. **Узел изменения имени.** Позволяет неоднозначно именовать потоки, при этом их содержимое эквивалентно. Например, если при проектировании разных частей системы один и тот же фрагмент данных получил различные имена, то эквивалентность соответствующих потоков данных обеспечивается узлом изменения имени. При этом один из потоков данных является входным для данного узла, а другой - выходным.

5. **Текст** в свободном формате в любом месте диаграммы.

Возможный способ изображения этих узлов приведен на рис. 10.

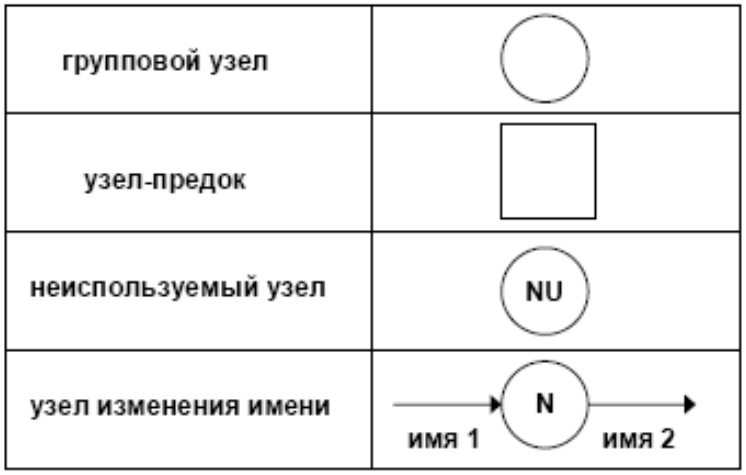



Рис. 10. Расширения диаграммы потоков данных

**Построение модели**

Главная цель построения иерархического множества DFD заключается в том, чтобы сделать требования ясными и понятными на каждом уровне детализации, а также разбить эти требования на части с точно определенными отношениями между ними. Для достижения этого целесообразно пользоваться следующими рекомендациями:

1. Размещать на каждой диаграмме от 3 до 6-7 процессов. Верхняя граница соответствует человеческим возможностям одновременного восприятия и понимания структуры сложной системы с множеством внутренних связей, нижняя граница выбрана по соображениям здравого смысла: нет необходимости детализировать процесс диаграммой, содержащей всего один или два процесса.

2. Не загромождать диаграммы несущественными на данном уровне деталями.

3. Декомпозицию потоков данных осуществлять параллельно с декомпозицией процессов; эти две работы должны выполняться одновременно, а не одна после завершения другой.

4. Выбирать ясные, отражающие суть дела, имена процессов и потоков для улучшения понимаемости диаграмм, при этом стараться не использовать аббревиатуры.

5. Однократно определять функционально идентичные процессы на самом верхнем уровне, где такой процесс необходим, и ссылаться на него на нижних уровнях.

6. Пользоваться простейшими диаграммными техниками: если что-либо возможно описать с помощью DFD, то это и необходимо делать, а не использовать для описания более сложные объекты.

7. Отделять управляющие структуры от обрабатывающих структур (т.е. процессов), локализовать управляющие структуры.

В соответствии с этими рекомендациями процесс построения модели разбивается на следующие этапы:

1. Расчленение множества требований и организация их в основные функциональные группы.

2. Идентификация внешних объектов, с которыми система должна быть связана.

3. Идентификация основных видов информации, циркулирующей между системой и внешними объектами.

4. Предварительная разработка контекстнойдиаграммы, на которой основные функциональные группы представляются процессами, внешние объекты – внешними сущностями, основные виды информации – потоками данных между процессами и внешними сущностями.

5. Изучение предварительной контекстной диаграммы и внесение в нее изменений по результатам ответов на возникающие при этом изучении вопросы по всем ее частям.

6. Построение контекстной диаграммы путем объединения всех процессов предварительной диаграммы в один процесс, а также группирования потоков.

7. Формирование DFD первого уровня на базе процессов предварительной контекстной диаграммы.

8. Проверка основных требований по DFD первого уровня.

9. Декомпозиция каждого процесса текущей DFD с помощью детализирующей диаграммы или спецификации процесса.

10. Проверка основных требований по DFD соответствующего уровня.

11. Добавление определений новых потоков в словарь данных при каждом их появлении на диаграммах.

12. Параллельное (с процессом декомпозиции) изучение требований (в том числе и вновь поступающих), разбиение их на элементарные и идентификация процессов или спецификаций процессов, соответствующих этим требованиям.

13. Проведение ревизии после построения двух-трех уровней с целью проверки корректности и улучшения понимаемости модели. Построение спецификации процесса (а не простейшей диаграммы) в случае, если некоторую функцию сложно или невозможно выразить комбинацией процессов.

Пример построения DFD диаграммы

В качестве примера создания модели рассмотрим фрагмент проекта системы обработки заказов клиентов и их обслуживания. Организация принимает от клиентов заказы на поставку товаров. Клиенту выписывается счет, после оплаты которого происходит отгрузка товара. Основными функциями системы являются выписка счета клиенту, прием оплаты от клиента, отгрузка товаров.

Построим контекстную диаграмму системы. Для этого изобразим основную функцию рассматриваемой системы и внешние по отношению к ней сущности, а также взаимосвязи между внешними сущностями и функцией системы.

Здесь сущность Склад является внешней по отношению к системе, т.к. рассматриваемая система не выполняет складских операций. Таким образом она лишь получает необходимую для работы информацию от Склада. Клиенты делают заказы на покупку товаров и получают товары, что изображено в виде стрелок с соответствующими наименованиями.

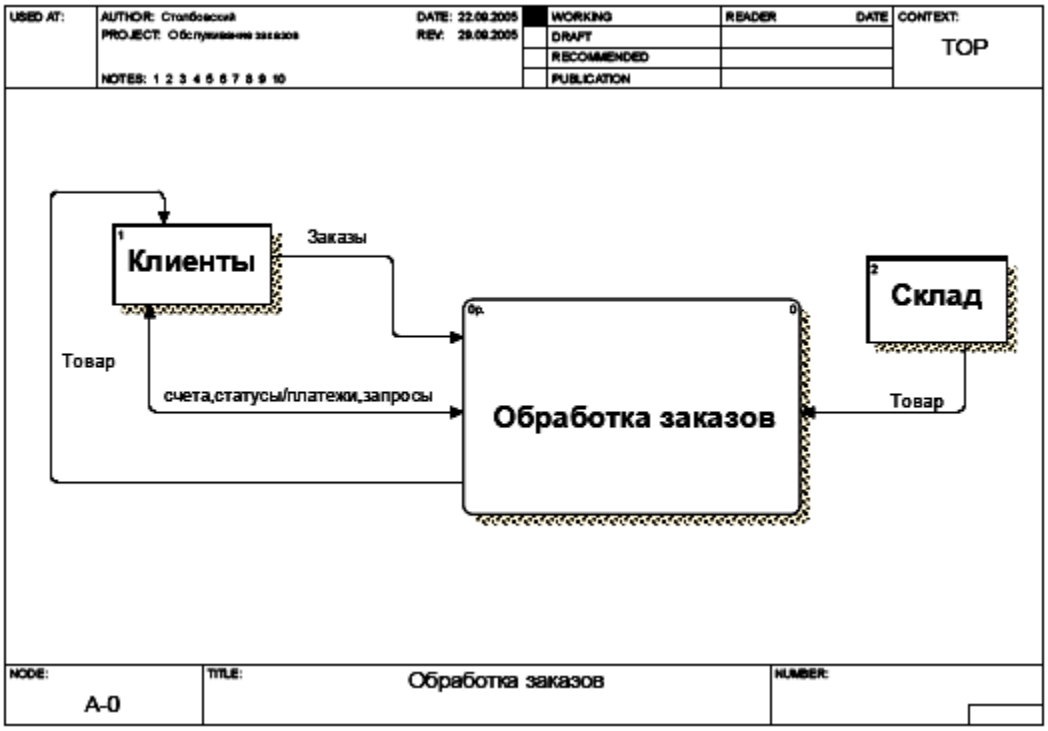



Рис. 11. Контекстная диаграмма системы «Обработка заказов клиентов»

После создания контекстной диаграммы, постараемся рассмотреть функции системы более подробно и построить в результате диаграмму детализации первого уровня. В начале рассмотрения примера было указано, что система должна выполнять три основных функции: выписка счета клиенту, прием оплаты от клиента, отгрузка товаров. Изобразим эти фукнции на диаграмме детализации, а также определим и изобразим хранилища данных и потоки данных между функциями, хранилищами и контекстной диаграммой.

Результат построения диаграммы детализации первого уровня представлен на рис.12.

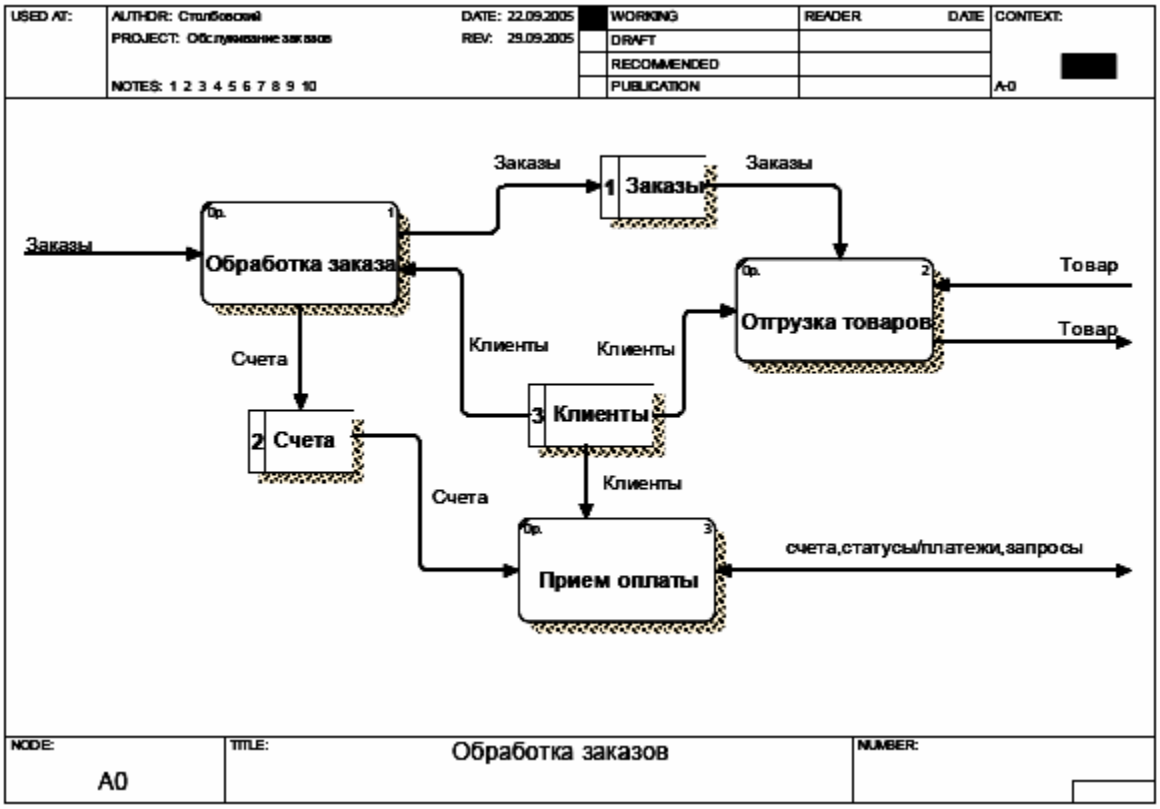



Рис. 12. Диаграмма детализации первого уровня системы «Обработка заказов клиентов»

Здесь, помимо трех функций системы представлены три хранилища, предназначенные для хранения данных о заказах, счетах и клиентах.

Рассмотренный пример является упрощенным и предназначен для знакомства с техникой построения как отдельных DFD диаграмм, так и их иерархии. Все рассмотренные выше принципы и техники построения подобных диаграмм применимы на практике.

Пример 2. Рассмотрим процесс *СДАТЬ ЭКЗАМЕН*. У нас есть две сущности *СТУДЕНТ* и *ПРЕПОДАВАТЕЛЬ*. Опишем потоки данных, которыми обменивается наша проектируемая система с внешними объектами.

Со стороны сущности *СТУДЕНТ* опишем информационные потоки. Для сдачи экзамена необходимо, чтобы у *СТУДЕНТА* была *ЗАЧЕТКА*, а также чтобы он имел *ДОПУСК К ЭКЗАМЕНУ*. Результатом сдачи экзамена, т.е. выходными потоками будут *ОЦЕНКА ЗА ЭКЗАМЕН* и *ЗАЧЕТКА*, в которую будет проставлена *ОЦЕНКА*.

Со стороны сущности *ПРЕПОДАВАТЕЛЬ* информационные потоки следующие: *ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ*, согласно которой будет известно, что *СТУДЕНТ* допущен до экзамена, а также официальная бумага, куда будет занесен результат экзамена, т.е. *ОЦЕНКА ЗА ЭКЗАМЕН*, проставленная в *ЭКЗАМЕНАЦИОННУЮ ВЕДОМОСТЬ*.

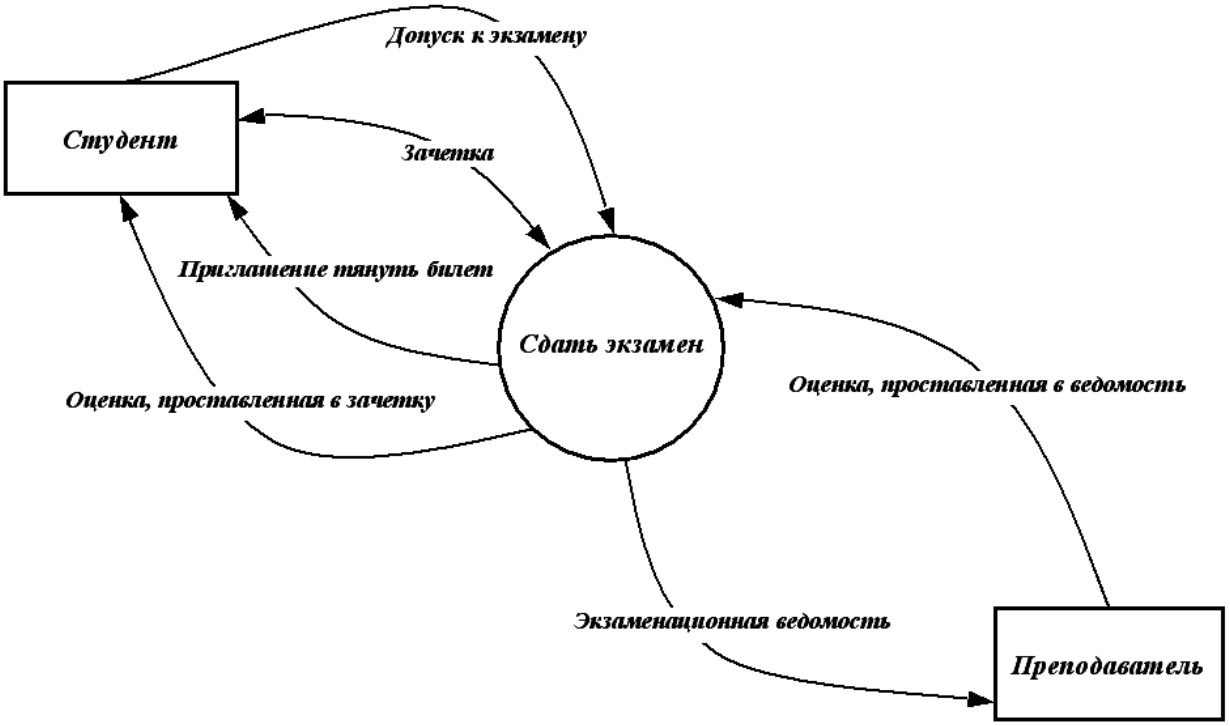



Рис. 13. DFD-диаграмма процесса «Сдать экзамен»

Теперь детализируем процесс СДАЧА ЭКЗАМЕНА. Этот процесс будет содержать следующие процессы (рис. 14): ВЫТЯНУТЬ БИЛЕТ {1.1}, ПОДГОТОВИТЬСЯ К ОТВЕТУ {1.2}, ОТВЕТИТЬ НА БИЛЕТ {1.3}, ПРОСТАВЛЕНИЕ ОЦЕНКИ {1.4}.

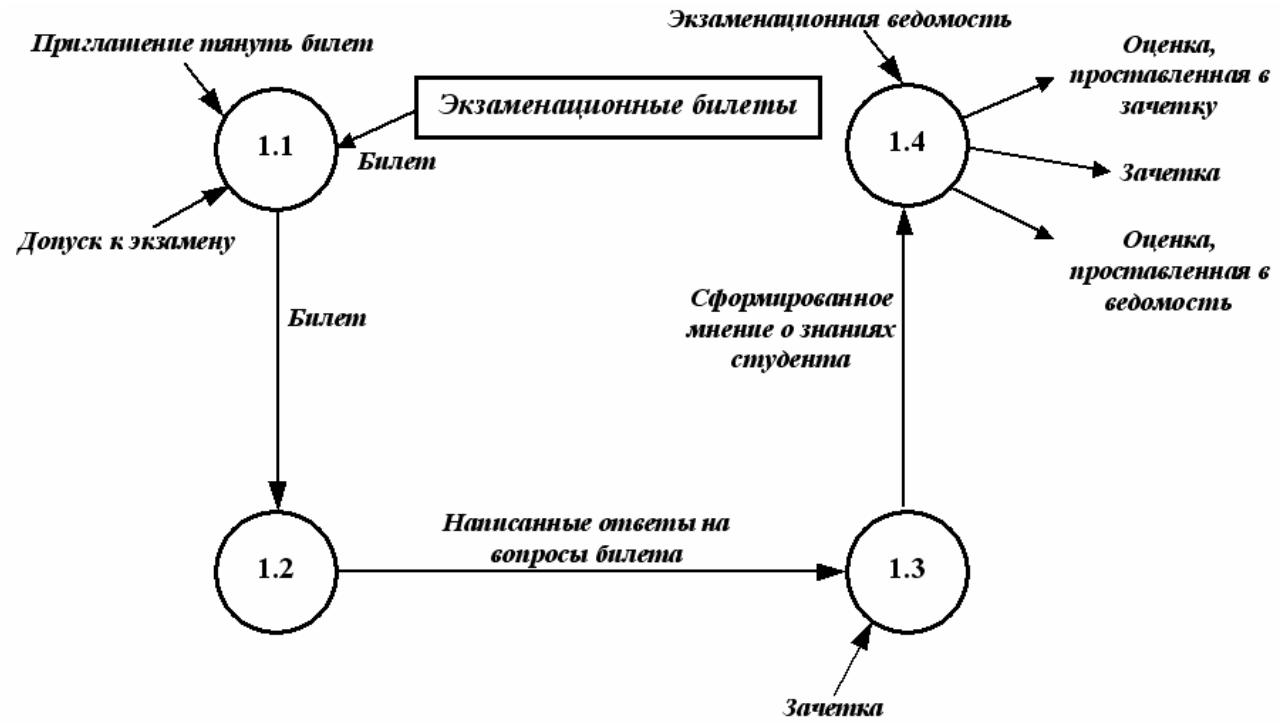



Рис. 14 Декомпозиция 1-го уровня DFD-диаграммы процесса «Сдать экзамен»

Пример 3. Построим DFD-диаграмму для предприятия, строящего свою деятельность по принципу "изготовление на заказ". На основании полученных заказов формируется план выпуска продукции на определенный период. В соответствии с этим планом определяются потребность в комплектующих изделиях и материалах, а также график загрузки производственного оборудования. После изготовления продукции и проведения платежей, готовая продукция отправляется заказчику.

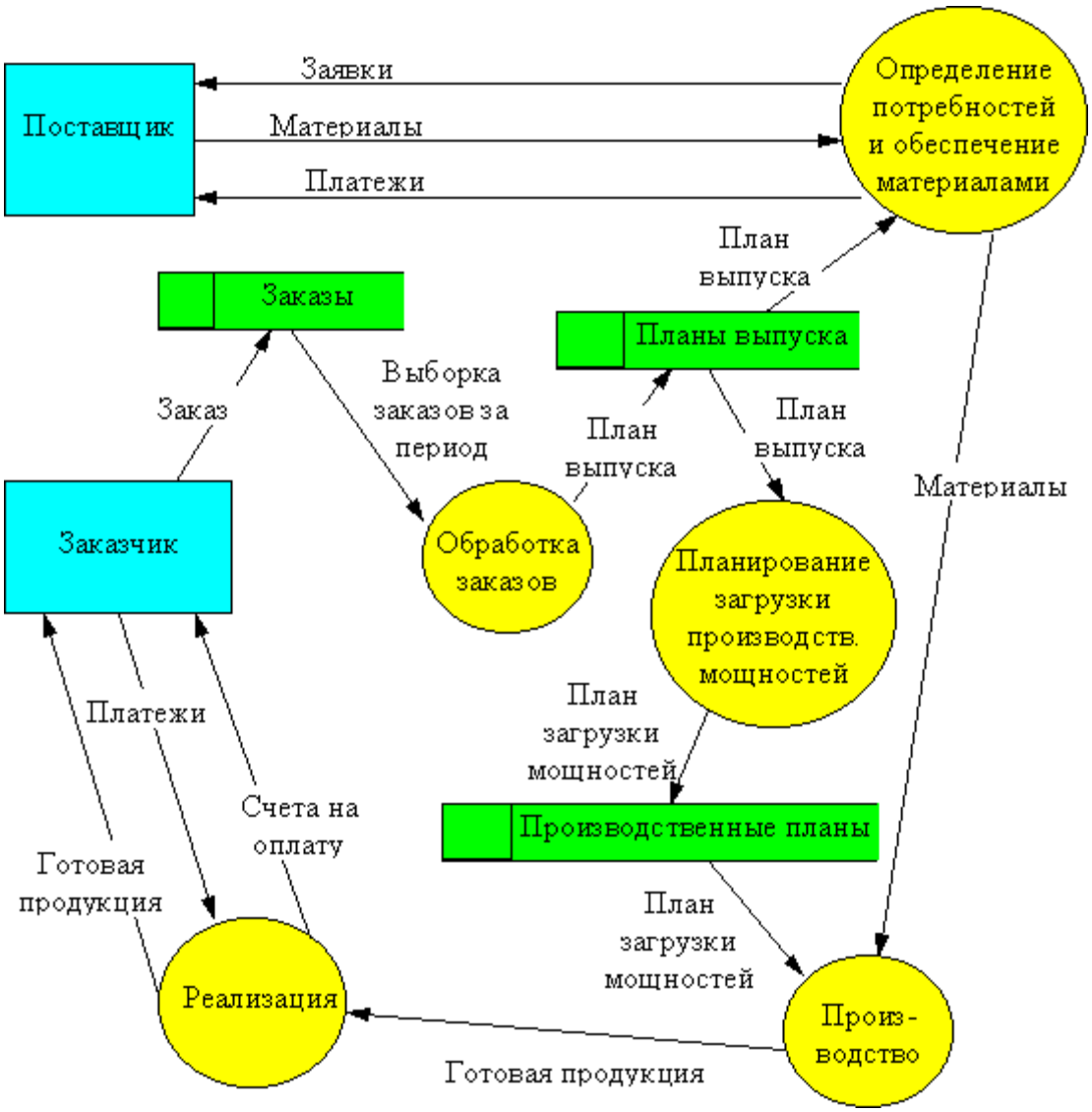



Рис. 15. DFD диаграмма процесса «Изготовление на заказ»

Эта диаграмма представляет самый верхний уровень функциональной модели. Естественно, это весьма грубое описание предметной области. Уточнение модели производится путем детализации необходимых функций на DFD-диаграмме следующего уровня. Так мы можем разбить функцию "Определение потребностей и обеспечение материалами" на подфункции "Определение потребностей", "Поиск поставщиков", "Заключение и анализ договоров на поставку", "Контроль платежей", "Контроль поставок", связанные собственными потоками данных, которые будут представлены на отдельной диаграмме. Детализация модели должна производится до тех пор, пока она не будет содержать всю информацию, необходимую для построения ифнормационной системы.

## 4.3. Диаграммы сущность-связь (ERD)

Цель моделирования данных состоит в обеспечении разработчика ИС концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются диаграммы "сущность-связь" (ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Наибольшее распространение получили следующие нотации, используемые при построении *ER*-диаграмм: нотация *Чена*, нотация *Мартина*, нотация *DEF1X*, нотация *Баркера.*

Данная нотация была предложена П. Ченом (P. Chen) в его известной работе 1976 года и получила дальнейшее развитие в работах Р. Баркера (R. Barker). Диаграммы "сущность-связь" (ERD) предназначены для графического представления моделей данных разрабатываемой программной системы и предлагают некоторый набор стандартных обозначений для определения данных и отношений между ними. С помощью этого вида диаграмм можно описать отдельные компоненты концептуальной модели данных и совокупность взаимосвязей между ними, имеющих важное значение для разрабатываемой системы.

Основными понятиями являются понятия сущности и связи. При этом под сущностью (entity) понимается произвольное множество реальных или абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. В этом случае каждый рассматриваемый объект может являться экземпляром одной и только одной сущности, должен иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности.

Примерами сущностей могут быть: банк, клиент банка, счет клиента, аэропорт, пассажир, рейс, компьютер, терминал, автомобиль, водитель. Каждая из сущностей может

рассматриваться с различной степенью детализации и на различном уровне абстракции, что определяется конкретной постановкой задачи. Для графического представления сущностей используются специальные обозначения (рис. 16).


Рис. 16. Графические изображения для обозначения сущностей

Связь (relationship) определяется как отношение или некоторая ассоциация между отдельными сущностями. Примерами связей могут являться родственные отношения типа "отец-сын" или производственные отношения типа "начальник-подчиненный". Другой тип связей задается отношениями "иметь в собственности" или "обладать свойством". Различные типы связей графически изображаются в форме ромба с соответствующим именем данной связи (рис. 17).

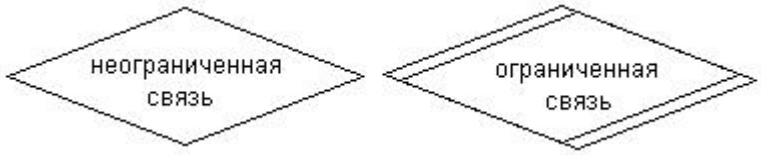
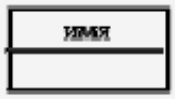



Рис. 17. Графические изображения для обозначения связей

**Конструктивные элементы нотации Чена**

| № | Элемент диаграммы | Обозначает | № | Элемент диаграммы | Обозначает |
|---|---|---|---|---|---|
| 1 |  | независимая сущность | 6 |  | атрибут |
| 2 |  | зависимая сущность | 7 |  | первичный ключ |
| 3 |  | родительская сущность в иерархической связи | 8 |  | внешний ключ (понятие внешнего ключа вводится в реляционной модели данных) |
| 4 |  | Связь | 9 |  | многозначный атрибут |
| 5 |  | идентифицирующая связь | 10 |  | получаемый (наследуемый) атрибут в иерархических связях |

Графическая модель данных строится таким образом, чтобы связи между отдельными сущностями отражали не только семантический характер соответствующего отношения, а также кратность участвующих в данных отношениях экземпляров сущностей.

Рассмотрим в качестве простого примера ситуацию, которая описывается двумя сущностями: "Сотрудник" и "Компания". При этом в качестве связи естественно. использовать отношение принадлежности сотрудника данной компании. Если учесть соображения о том, что в компании работают несколько сотрудников, и эти сотрудники не могут быть работниками других компаний, то данная информация может быть представлена графически в виде следующей диаграммы "сущность-связь" (рис. 18). На данном рисунке буква "N" около связи означает тот факт, что в компании могут работать более одного сотрудника, при этом значение N заранее не фиксируется. Цифра "1" на другом конце связи означает, что сотрудник может работать только в одной конкретной компании, т. е. не допускается прием на работу

сотрудников по совместительству из других компаний или учреждений.


Рис. 18. Диаграмма "сущность-связь" для примера сотрудников некоторой компании

Несколько иная ситуация складывается в случае рассмотрения сущностей "сотрудник" и "проект", и связи "участвует в работе над проектом" (рис. 19). Поскольку в общем случае один сотрудник может участвовать в разработке нескольких проектов, а в разработке одного проекта могут принимать участие несколько сотрудников, то данная связь является многозначной. Данный факт специально отражается на диаграмме указанием букв "N" и "M" около соответствующих сущностей, при этом выбор конкретных букв не является принципиальным.


Рис. 19. Диаграмма "сущность-связь" для примера сотрудников, участвующих в работе над проектами

Рассмотренные две диаграммы могут быть объединены в одну, на которой будет представлена информация о сотрудниках компании, участвующих в разработке проектов данной компании (рис. 20). При этом может быть введена дополнительная связь, характеризующая проекты данной компании.

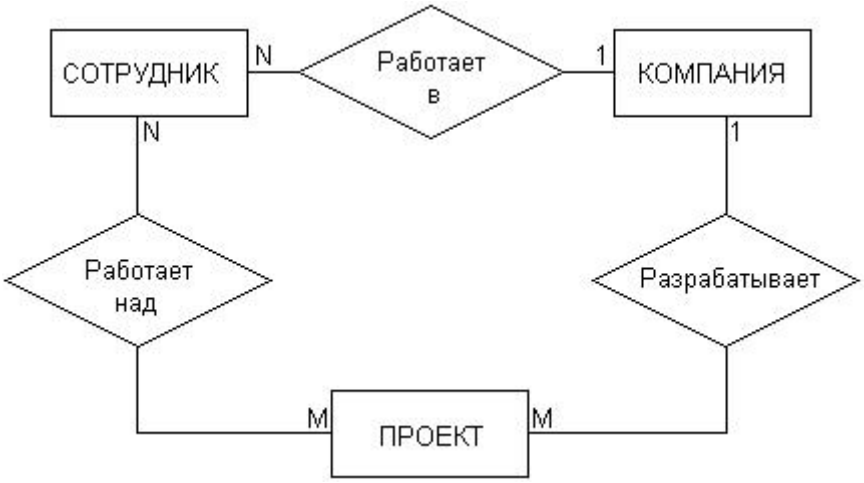



Рис. 20. Диаграмма "сущность-связь" для общего примера компании

На указанных диаграммах могут быть отражены более сложные зависимости между отдельными сущностями, которые отражают обязательность выполнения некоторых дополнительных условий, определяемых спецификой решаемой задачи и моделируемой предметной области. В частности, могут быть отражены связи подчинения одной сущности другой или введения ограничений на действие отдельных связей. В подобных случаях используются дополнительные графические обозначения, отражающие особенности соответствующей семантики.

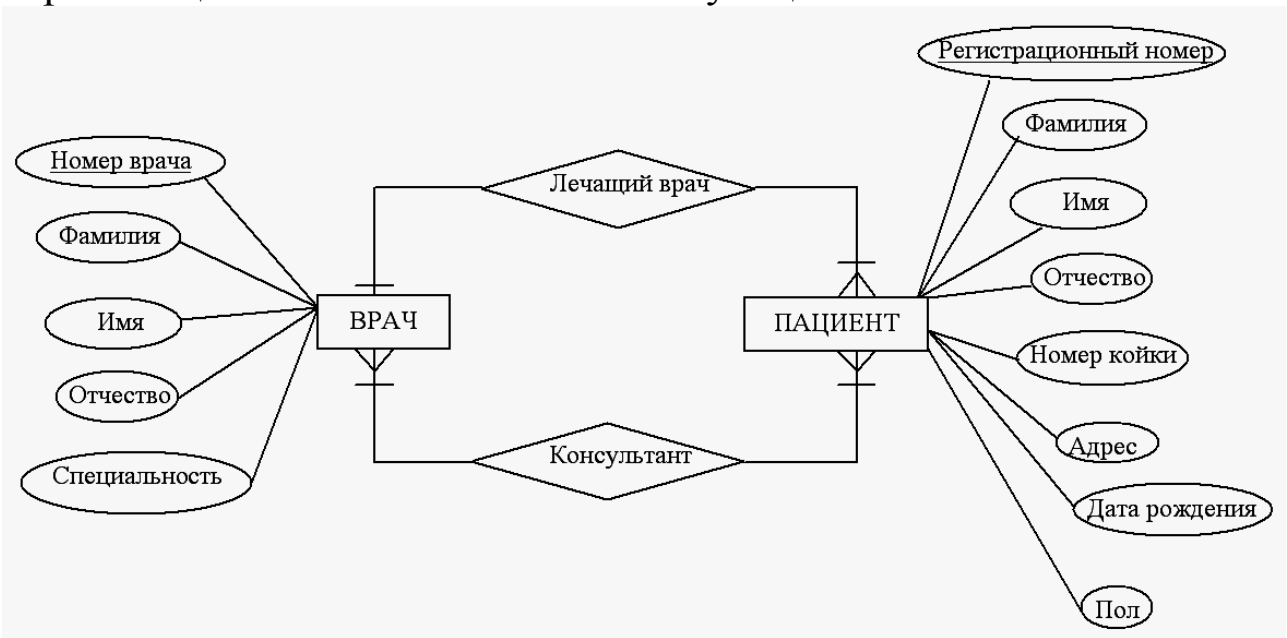



Рис. 21. Фрагмент концептуальной схемы в нотации Чена

## 4.4. Проектирование структур данных с использованием нотации IDEF1X/IE

На сегодняшний день наиболее проработанным и часто используемым подходом при проектировании структур данных информационных хранилищ является структурный подход. В рамках структурного подхода используются несколько нотаций, предназначенных для моделирования структур данных информационных систем. Наиболее распространенными из них сегодня являются IDEF1X и IE. Рассмотрим методологию и используемые в ее рамках нотации более подробно.

Уровни моделирования данных делятся на общие модели, изображающие сущности в общем не детализированном виде, содержащем сущности важные для рассматриваемой предметной области и модели, детально изображающие взаимосвязи между сущностями предметной области в терминах конкретной СУБД, выбранной для реализации модели данных. Таким образом, на самом нижнем уровне представления модели в значительной степени зависят от выбранной реализации (СУБД) и, следовательно, модель, созданная для использования в СУБД Access будет значительно отличаться от SQL Server и тем более Oracle. Модели верхнего уровня технологически независимы и могут содержать информацию об элементах, физически не сохраняемых в БД.

Таким образом, уровни создания информационной модели можно представить следующим образом (рис. 22).

Модели верхнего уровня делятся на две категории. ERD (Entity Relationship Diagram) диаграммы Сущность-связь содержат наиболее общие для рассматриваемой предметной области сущности и связи между ними. KB (Key Based) Models – Ключевые модели (модели, основанные на ключах) устанавливают границы требований к информации предметной области и содержат все ее сущности. В KB моделях начинают проявляться детали и особенности строения данных.

Модели нижнего уровня также делятся на две категории. FA (Fully Attributed) модель с полным набором атрибутов, все отношения которой приведены к третьей нормальной форме, содержащая все необходимые элементы для полной

реализации базы данных. TM (Transformation Model) модель, представляющая собой трансформационную модель из реляционной в модель, предназначенную для реализации в конкретной СУБД с учетом ее особенностей. TM модель в большинстве случаев не удовлетворяет условиям третьей нормальной формы, т.к. она оптимизирована для использования в рамках конкретной СУБД с учетом ее особенностей и возможностей. В зависимости от уровня интеграции ИС физическая модель данных может быть как уровня системы в целом (глобальная модель), отображая взаимодействие информационных объектов масштаба ИС, так и уровня подсистем (локальная модель), отображая, соответственно, элементы, присущие только подсистемам ИС.

Заметим, что в идеальной ситуации должно существовать несколько локальных физических моделей, каждая из которых предназначена для своей подсистемы с возможностью взаимного использования элементов моделей.


Рис. 22. Уровни создания моделей данных

**Логические модели**

Существует три уровня логических моделей, предназначенных для моделирования бизнес информации: диаграммы сущность-связь (ER), Ключевые модели (KB) и модели с полным набором атрибутов (FA). ER и KB модели также называют «моделями предметной области», т.к. они содержат элементы и их взаимосвязи

внутри всей предметной области, являясь таким образом, более «широкими» по сравнению с моделями, используемыми для автоматизации в рамках конкретного проекта, охватывающего обычно часть предметной области. В свою очередь FA модели называют «моделями проекта», т.к. они обычно содержат описание части предметной области, необходимой в рамках решения конкретной задачи автоматизации.

**ER диаграммы**

ER диаграммы представляют собой диаграммы верхнего уровня, содержащие общие сущности и связи между ними отображающие общее представление о всей предметной области.

Основная цель ER диаграмм, является представление информации о бизнес требованиях, способных обеспечить возможность общего планирования разработки информационной системы. Эти модели не детализированы (содержат только общие сущности и атрибуты). В них возможно использовать связи между сущностями типа «многие-ко-многим», при этом ключевые атрибуты обычно не указываются. Это прежде всего модель для ее дальнейшего представления и обсуждения.

**KB модели**

KB модели описывают общие структуры данных, содержащие элементы в широких пределах предметной области. Сюда включаются все сущности и ключевые атрибуты, а также основные неключевые атрибуты.

Основная цель KB моделей дать широкое представление о сущностях и атрибутах, используемых в данной части предметной области. Эти модели создают возможность для построения детализированных моделей реализации. KB модели охватывают ту же область что и ER модели предметной области, но содержат больше деталей.

**FA модели**

FA модели представляют собой модели данных, приведенные к третьей нормальной форме, содержащие все сущности, атрибуты и связи необходимые для реализации в рамках одного проекта автоматизации. Данная модель включает размер экземпляра сущности, пути и скорость доступа, а также образцы возможностей доступа к данным.

**Физические модели**

Существует также два типа физических моделей: трансформационные модели и модели, привязанные к СУБД. Физические модели содержат всю необходимую для реализации разработчиками базы данных на основе логической модели. Трансформационная модель является также «проектной моделью», описывающей часть общей структуры данных, необходимой в рамках создания одного проекта (части создания интегрированной ИС).

**Трансформационная модель**

Целью трансформационной модели является обеспечение администратора базы данных необходимой информацией для физического создания эффективной базы данных, а также создание условий для определения и записи элементов данных и записей в словарь данных. Кроме того важной задачей трансформационной модели является помощь программистам в выборе физической структуры хранения данных при создании прикладного программного обеспечения.

На основе этой модели может проводиться также анализ и сравнение физической реализации (структуры БД) с бизнес требованиями, определенными предметной областью для определения адекватности физической реализации бизнес требованиям.

**Модель данных СУБД**

Трансформационная модель напрямую преобразуется в модель данных СУБД путем преобразования ее элементов в элементы реляционной модели данной конкретной СУБД с учетом ее особенностей. При этом первичные ключи обычно становятся уникальными индексами. Альтернативные ключи также могут лечь в основу создания индексов. Мощность связей может быть реализована либо за счет возможностей контроля ссылочной целостности СУБД, либо за счет логики приложения.

В основе всех типов перечисленных моделей лежит единая система графических обозначений, основанная на нотациях IDEF1X и IE.

**Графические обозначения нотаций IDEF1X и IE**

Существует несколько методологий визуального моделирования данных. Наиболее распространенными из них

являются две.

1. **IDEF1X** (Integration Definition for Information Modeling – интегрированное описание информационных моделей).

2. **IE** (Information Engineering – информационная инженерия).

**Интегрированное описание информационных моделей**

**IDEF1X** – высоко структурированная методология моделирования данных, расширяющая методологию **IDEF1,** принятую в качестве стандарта

**FIPS** (Federal Information Processing Standards – федеральный орган стандартов обработки информации). IDEF1X использует строго структурированный набор типов конструкций моделирования и приводит к модели данных, которая требует понимания физической природы данных до того, как такая информация может стать доступной.

Жесткая структура IDEF1X принуждает разработчика модели назначать сущностям характеристики, которые могут не отвечать реалиям окружающего мира. Например, IDEF1X требует, чтобы все подтипы сущностей были эксклюзивными. Это приводит к тому, что персона не может быть одновременно клиентом и сотрудником. В то время как реальная практика говорит нам другое.

**Информационный инжиниринг**

Информационный инжиниринг (Information Engineering – IE) использует для управления информацией подход, направляемый бизнесом, и применяет другую нотацию для представления бизнес-правил. IE служит расширением и развитием нотации и базовых концепций методологии ER, предложенной П. Ченом.

IE обеспечивает инфраструктуру поддержки требований к информации путем интеграции корпоративного стратегического планирования с разрабатываемыми информационными системами. Подобная интеграция позволяет более тесно увязать управление информационными ресурсами с долговременными стратегическими перспективами корпорации. Этот подход, направляемый требованиями бизнеса, приводит многих разработчиков моделей к выбору IE вместо других методологий, которые, в основном, концентрируют внимание на решении сиюминутных задач разработки. IE предлагает последовательность действий, приводящую корпорацию к определению всехсвоих информационных потребностей по сбору и управлению данными и

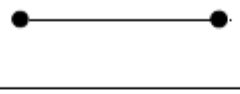
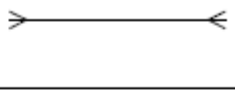
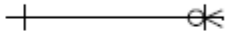
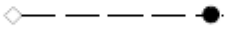
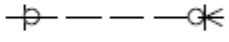
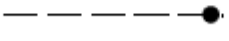
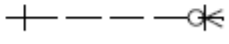
выявлению взаимосвязей между информационными объектами. В результате, требования к информации ясно формулируются на основе директив управления и могут быть непосредственно переведены в информационную систему управления, которая будет поддерживать стратегические потребности в информации.

Основными элементами методологий IDEF1X и IE являются сущности и связи между ними. Отличие в рассматриваемых методологиях заключается в используемых нотациях, т.е. правилах графического обозначения в основном связей между сущностями. Рассмотрим типы существующих связей более подробно.

Одним из наиболее распространенных и доступных CASE средств, предназначенных для проектирования структур данных, является ERWin фирмы Computer Associates. Несмотря на свой немолодой (по меркам программной индустрии) возраст, этот программный продукт отвечает современным требованиям по ведению процесса проектирования и реализации БД, поддерживает возможности прямого и обратного инжиниринга.

При описании примеров БД в данном курсе, автор ориентировался на возможность использования именно этого CASE средства, а все фрагменты, связанные с описанием особенностей выполнения тех или иных операций в ERWin помечены знаком .

Ниже приведено сопоставление обозначений связей, принятых в нотациях IDEF1X и IE.

| Тип связи | IDEF1X | IE |
|---|---|---|
| «многие-ко-многим» |  |  |
| «один-ко-многим»<br>идентифицирующая |  |  |
| неидентифицирующая (пустые значения допустимы) |  |  |
| неидентифицирующая (пустые значения не допустимы) |  |  |
| «категориальная» |  |  |

**Процесс создания модели данных**

Процесс создания модели данных можно рассматривать как последовательный переход от модели бизнес процессов к обобщенной диаграмме «сущность-связь» уровня предметной области, а затем к другим моделям более низкого уровня. Конечной целью данного процесса является получение физической модели данных уровня СУБД, готовой для ее реализации в рамках конкретной СУБД. Такая модель может порой значительно отличаться от физической модели той же базы данных, предназначенной для реализации в рамках другой СУБД. Несмотря на то, что значимость всего процесса создания модели данных определяется получаемым конечным результатом в виде физической модели данных, все промежуточные результаты представляют собой также большую практическую ценность, т.к. позволяют проводить анализ данных на более высоком логическом уровне и могут использоваться для адаптации модели базы данных к изменившимся требованиям предметной области. Кроме того, они представляют собой важные элементы документации созданной модели данных.

Таким образом, процесс создания модели данных можно представить в виде последовательности следующих этапов.

1. Выделение сущностей рассматриваемой предметной области и связей между ними на основе модели бизнес процессов. Построение диаграммы «сущность-связь» уровня предметной области. Фактически, этот этап является начальным этапом создания инфологической модели данных.

2. Выделение ключевых атрибутов-признаков сущностей, полученных в первом пункте и создание ключевой модели данных уровня предметной области. Фактически, создание полной инфологической модели данных.

3. Выделение из созданной в п.2 инфологической модели предметной области элементов (сущностей), относящихся к области данного проекта реализации, выделение полного набора атрибутов этих сущностей и создание модели с полным набором атрибутов уровня проекта. Фактически, это означает создание инфологической модели данных проекта, являющейся проекцией инфологической модели уровня предметной области.

4. Преобразование модели п.3 с учетом особенностей выбранной модели данных. В большинстве случаев, с

использованием особенностей реляционной модели данных. Фактически, это означает создание даталогической модели данных.

5. Модификация, расширение, преобразование модели п.4 с учетом особенностей выбранной СУБД и создание модели данных СУБД. Фактически это означает создание физической модели данных. Модификация и расширение означает, что даталогическая модель может подвергаться достаточно серьезным преобразованиям, которые допускают добавление новых, изменение или удаление старых сущностей, размещенных на даталогической модели.

Результирующая модель может очень сильно отличаться от исходной инфологической. Степень отличия зависит от того, как предполагается реализовывать приложение, осуществляющее обработку данных БД и возможностей СУБД. Однако основные элементы инфологической модели данных (в основном независимые сущности), скорее всего, останутся неизменными.

Начальная инфологическая модель данных строится для всей моделируемой предметной области и должна обеспечить целостное ее представление. По мере приближения к физическому уровню, необходимо сужать рассматриваемую область, ограничиваясь рамками конкретного проекта. В результате может быть получено несколько моделей данных, соответствующих различным проектам.

**Пример создания сложной IDEF1X модели**

В качестве примера создания модели данных ИС рассмотрим процесс создания базы данных на основе модели потребительского кредитования.

В результате проведенного анализа, были сформированы следующие сущности:

- Заемщик
- Банк
- ПлатежиЗаемщика
- Документы
- ВыдачаКредита
- ПервоначальныйВзнос
- ЧерныйСписокЗаемщиков
- Договор
- ДоговорПоручительства

- НачисленныеПроценты
- СчетСсудный

Определим связи между этими сущностями и построим начальную ER диаграмму. В результате анализа взаимоотношений сущностей была построена следующая диаграмма


Рис. 23. Диаграмма «сущность-связь» модели Потребительское кредитование

Практически все элементы этой диаграммы не нуждаются в комментариях за исключением следующих. Сущности Банк и ЧерныйСписокЗаемщиков не связаны ни с какими другими сущностями модели. Рассмотрим какими соображениями пользовался автор при создании этой модели. Сущность Банк представляет собой информацию о Банке, выдающем кредит. Хотя и планируется использование системы в рамках одного банка, но все равно необходимы его реквизиты, которые удобно хранить в отдельной таблице. Эта таблица никак логически не связана ни с одной другой сущностью рассматриваемой модели. Если же планируется использование системы в разных банках (например филиалах), необходимо связать сущность Банк с сущностью Договор связью «один-ко-многим». Сущность ЧерныйСписокЗаемщиков предназначена для хранения информации о заемщиках, которым в будущем никогда не

будет выдаваться кредит по разным причинам. Предполагается, что заемщик будет попадать в черный список путем копирования данных о нем из таблицы Заемщик. Таким образом, атрибуты сущности аналогичны атрибутам сущности Заемщик, но она не связана ни с одной другой сущностью т.к. информация о неблагонадежных заемщиках нигде (в других сущностях) не используется. Одним из вариантов организации хранения черного списка заемщиков является создание отдельного атрибута в сущности Заемщик, по значению которого было бы возможно определить входит ли он в черный список.

Построим ключевую модель рассматриваемой системы. Для этого определим список атрибутов сущностей рассматриваемой предметной области и выделим из их состава ключевые атрибуты.

Построив ключевую модель необходимо еще раз взглянуть на нее, и, просматривая ее сущности и связи, а также совокупность атрибутов каждой сущности, попытаться оценить ее правильность и законченность уже с точки зрения организации хранения данных. В результате этого анализа можно прийти к выводу о необходимости преобразования некоторых идентифицирующих связей в неидентифицирующие для исключения миграции атрибутов в состав первичных ключей сущностей. В результате проведенных преобразований получим ключевую модель данных, изображенную на рис.24.

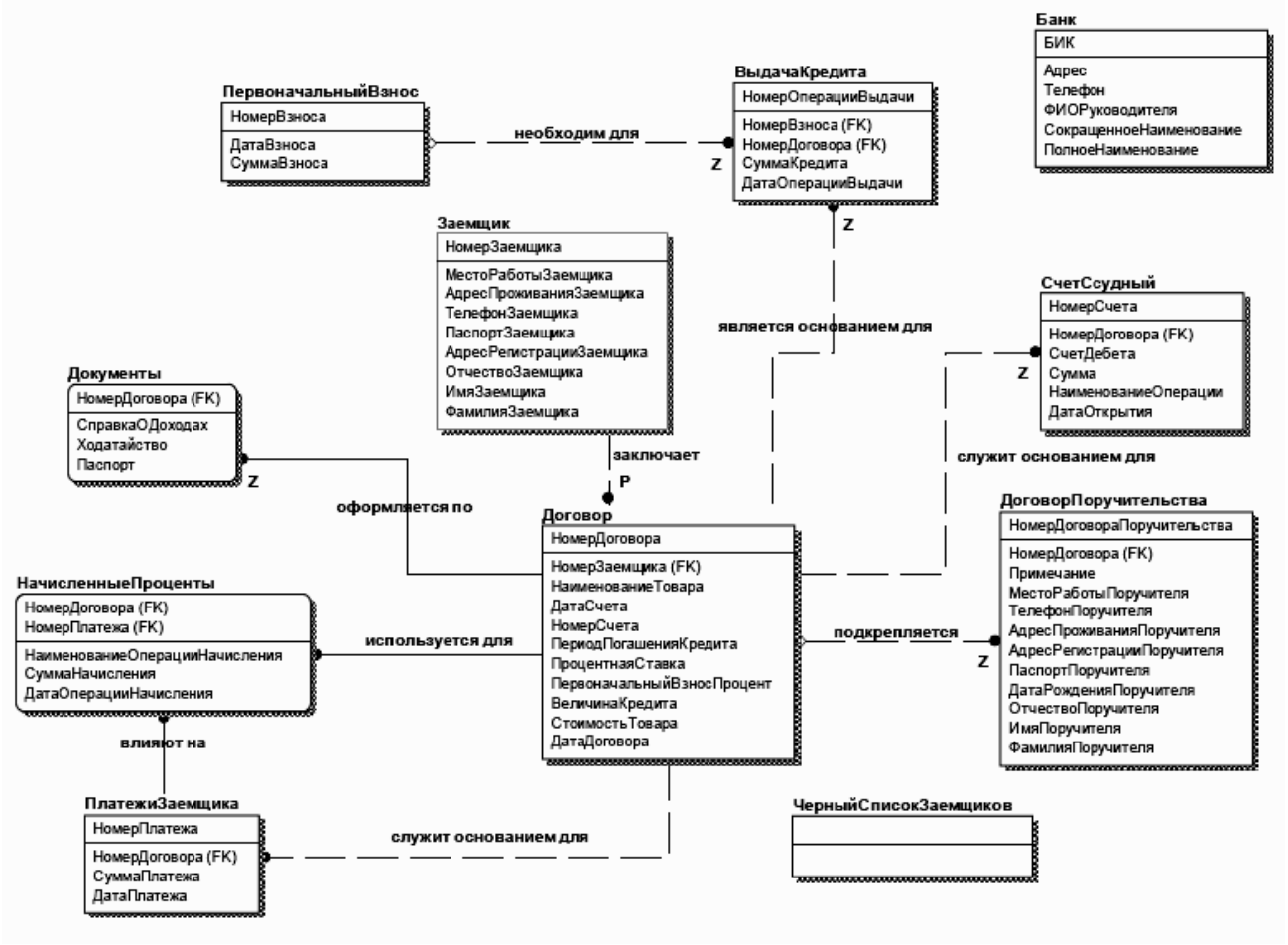



Рис. 24. Предварительная ключевая модель «Потребительское кредитование»

В процессе определения связей, атрибутов и ключей сущностей были замечено следующее. Сущность ДоговорПоручительства по существу содержит информацию о человеке, выступающем в роли поручителя. Эта информация может быть использована для создания договора поручительства, но саму сущность правильнее было бы назвать Поручитель. Сущность Документы должна содержать информацию о документах, которые требуются для оформления кредита, однако только паспортные данные заемщика заносятся в базу данных при оформлениикредита. Все остальные документы предоставляются на бумажных носителях, а информация, содержащаяся в них в базу не вносится. Следовательно, сущность Документы можно вообще исключить, тем более что мощность ее связи с сущностью Договор «один-к-одному», а паспортные данные перенести в сущность Заемщик. Точнее, в сущности Заемщик уже есть атрибут ПаспортЗаемщика, но т.к. он состоит из нескольких частей, вместо атрибута

ПаспортЗаемщика, создадим атрибуты ПаспортЗаемщикаСерия, ПаспортЗаемщикаНомер, ПаспортЗаемщикаДатаВыдачи, ПаспортЗаемщикаКемВыдан. Аналогичные действия можно проделать и с атрибутом ПаспортПоручителя сущности Поручитель. Эти действия, кстати, соответствуют операциям, производимым в рамках проведения нормализации отношений, которые необходимо проводить после создания даталогической модели.

Сущности ВыдачаКредита и СчетССудный фактически являются одинаковыми т.к. в каждой из них содержится информация о денежных средствах, выдаваемых заемщику в качестве кредита, а т.к. в данном случае не рассматриваются вопросы, связанные с формированием банковских проводок, осуществляющих перечисление денежных средств с одного счета на другой, смысл сущности СчетССудный теряется. Следовательно, ее можно удалить.

Т.е. система предназначена прежде всего для регистрации выдачи кредитов, начисления процентов и контроля погашения кредитов. Все остальные вопросы, связанные с бухгалтерией банка остаются вне рамок системы.

Сущность ПервоначальныйВзнос соединена отношением «один-к-одному» с сущностью ВыдачаКредита. С логической точки зрения это не совсем правильно, т.к. первоначальный взнос производится при заключении договора и в некоторых случаях может отсутствовать. Следовательно правильнее связать сущность ПервоначальныйВзнос с сущностью Договор так, как показано на рис.25, на котором приведен окончательный вариант ключевой модели системы Потребительское кредитование.

Следующим этапом является создание модели с полным набором атрибутов. Для этого необходимо проверить каждую сущность на полноту входящих в нее атрибутов и присвоить каждому из них соответствующий тип данных.

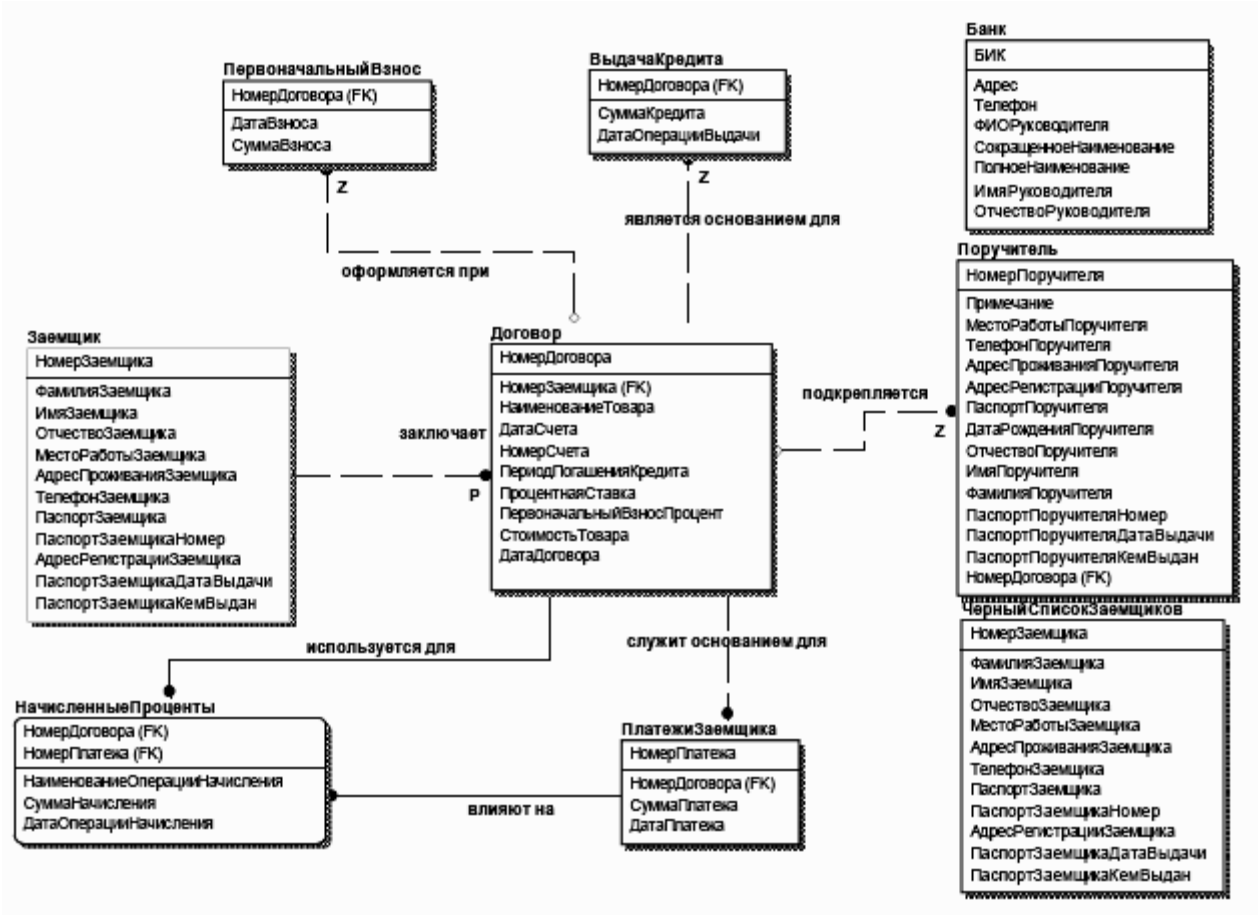



Рис. 25. Окончательный вариант ключевой модели системы «Потребительское кредитование»

# 5. ПОСТРОЕНИЕ ИНТЕГРИРОВАННОЙ МОДЕЛИ СЛОЖНОЙ СИСТЕМЫ

## 5.1. Язык моделирования UML

Язык UML предоставляет в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи,

которые могут быть решены с использованием данных моделей. При этом одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Другим принципом построения моделей сложных систем является принцип многомодельности. Этот принцип представляет собой утверждение о том, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Применительно к методологии ООАП это означает, что достаточно полная модель сложной системы допускает некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы. При этом наиболее общими представлениями сложной системы принято считать статическое и динамическое представления, которые в свою очередь могут подразделяться на другие более частные представления.)

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. При этом исходная или первоначальная модель сложной системы имеет наиболее общее представление (метапредставление). Такая модель строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы.


Рис. 26. Общая схема взаимосвязей моделей и представлений сложной системы в процессе объектно-ориентированного анализа и проектирования

Формальное описание самого языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

- Мета-метамодель
- Метамодель
- Модель
- Объекты пользователя

Уровень мета-метамодели образует исходную основу для всех метамодельных представлений. Главное предназначение этого уровня состоит в том, чтобы определить язык для спецификации метамодели. Мета-метамодель определяет модель языка UML на самом высоком уровне абстракции и является наиболее компактным ее описанием. С другой стороны, мета-метамодель может специфицировать несколько метамоделей, чем достигается потенциальная гибкость включения дополнительных понятий.

Метамодель является экземпляром или конкретизацией мета-метамодели. Главная задача этого уровня — определить язык для спецификации моделей. Данный уровень является более конструктивным, чем предыдущий, поскольку обладает более развитой семантикой базовых понятий. Все основные понятия языка UML — это понятия уровня метамодели. Примеры таких

понятий — класс, атрибут, операция, компонент, ассоциация и многие другие. Именно рассмотрению семантики и графической нотации понятий уровня метамодели посвящена данная книга.

Модель в контексте языка UML является экземпляром метамодели в том смысле, что любая конкретная модель системы должна использовать только понятия метамодели, конкретизировав их применительно к данной ситуации. Это уровень для описания информации о конкретной предметной области. Однако если для построения модели используются понятия языка UML, то необходима полная согласованность понятий уровня модели с базовыми понятиями языка UML уровня метамодели. Примерами понятий уровня модели могут служить, например, имена полей проектируемой базы данных, такие как имя и фамилия сотрудника, возраст, должность, адрес, телефон. При этом данные понятия используются лишь как имена соответствующих информационных атрибутов.

Конкретизация понятий модели происходит на уровне объектов. В настоящем контексте объект является экземпляром модели, поскольку содержит конкретную информацию относительно того, чему в действительности соответствуют те или иные понятия модели. Примером объекта может служить следующая запись в проектируемой базе данных: "Илья Петров, 30 лет, иллюзионист, ул. Невидимая, 10-20, 100-0000".

Описание семантики языка UML предполагает рассмотрение базовых понятий только уровня метамодели, который представляет собой лишь пример или частный случай уровня мета-метамодели. Метамодель UML является по своей сути, скорее логической моделью, чем физической или моделью реализации. Особенность логической модели заключается в том, что она концентрирует внимание на декларативной или концептуальной семантике, опуская детали конкретной физической реализации моделей. При этом отдельные реализации, использующие данную логическую метамодель, должны быть согласованы с ее семантикой, а также поддерживать возможности импорта и экспорта отдельных логических моделей.

В то же время, логическая метамодель может быть реализована различными способами для обеспечения требуемого уровня производительности и надежности соответствующих

инструментальных средств. В этом заключается недостаток логической модели, которая не содержит на уровне семантики требований, обязательных для ее эффективной последующей реализации. Однако согласованность метамодели с конкретными "моделями реализации является обязательной для всех разработчиков программных средств, обеспечивающих поддержку языка UML.

Метамодель языка UML имеет довольно сложную структуру, которая включает в себя порядка 90 метаклассов, более 100 метаассоциаций и почти 50 стереотипов, число которых возрастает с появлением новых версий языка. Чтобы справиться с этой сложностью языка UML, все его элементы организованы в логические пакеты. Поэтому рассмотрение языка UML на метамодельном уровне заключается в описании трех его наиболее общих логических блоков или пакетов: основные элементы, элементы поведения и общие механизмы.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. В терминах языка UML определены следующие виды диаграмм:

- Диаграмма вариантов использования (use case diagram)
- Диаграмма классов (class diagram)
- Диаграммы поведения (behavior diagrams)
    - Диаграмма состояний (statechart diagram)
    - Диаграмма деятельности (activity diagram)
    - Диаграммы взаимодействия (interaction diagrams)
        - Диаграмма последовательности (sequence diagram)
        - Диаграмма кооперации (collaboration diagram)
- Диаграммы реализации (implementation diagrams)
    - Диаграмма компонентов (component diagram)
    - Диаграмма развертывания (deployment diagram)

Из перечисленных выше диаграмм некоторые служат для обозначения двух и более других подвидов диаграмм. При этом в качестве самостоятельных представлений в языке UML используются следующие диаграммы:

1. Диаграмма вариантов использования
2. Диаграмма классов.
3. Диаграмма состояний.

4. Диаграмма деятельности.
5. Диаграмма последовательности.
6. Диаграмма кооперации.
7. Диаграмма компонентов.
8. Диаграмма развертывания.

Перечень этих диаграмм и их названия являются каноническими в том смысле, что представляют собой неотъемлемую часть графической нотации языка UML. Более того, процесс ООАП неразрывно связан с процессом построения этих диаграмм. При этом совокупность построенных таким образом диаграмм является самодостаточной в том смысле, что в них содержится вся информация, которая необходима для реализации проекта сложной системы.

Каждая из этих диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML. При этом диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. Диаграмма классов является, по своей сути, логической моделью, отражающей статические аспекты структурного построения сложной системы.

Диаграммы поведения также являются разновидностями логической модели, которые отражают динамические аспекты функционирования сложной системы. И, наконец, диаграммы реализации служат для представления физических компонентов сложной системы и поэтому относятся к ее физической модели. Таким образом, интегрированная модель сложной системы в нотации UML представляется в виде совокупности указанных выше диаграмм ( рис. 27).

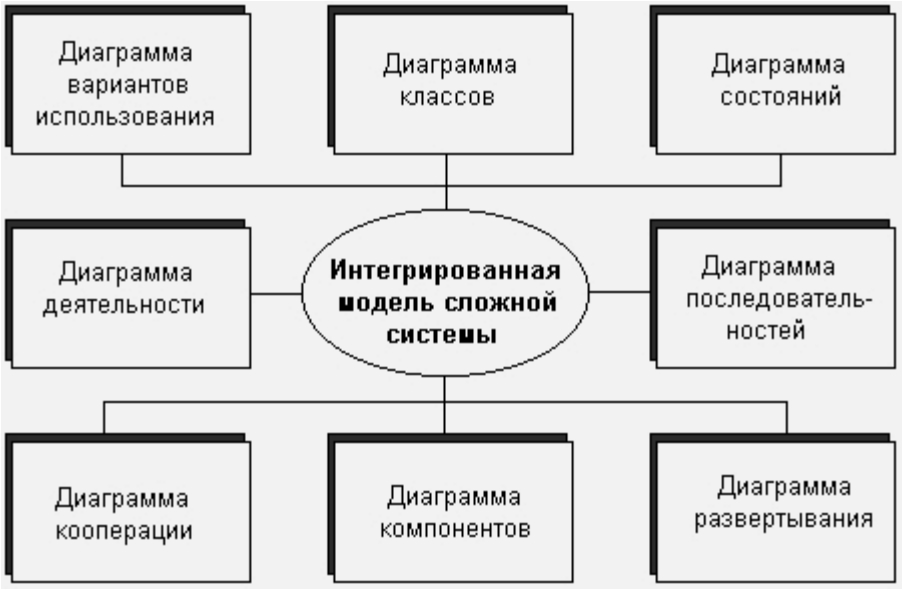



Рис. 27. Интегрированная модель сложной системы в нотации UML

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы

Концептуальная модель выражается в виде диаграмм прецедентов (use case diagram). Этот тип диаграмм служит для проведения итерационного цикла общей постановки задачи вместе с заказчиком.

Логическая модель позволяет определять два различных взгляда на системы: статический и динамический. Статический подход выражается диаграммами классов (class diagram).

Динамический подход описывается двумя типами диаграмм: диаграммами взаимодействия объектов; диаграммами последовательности взаимодействий.

Физическая модель задается компонентной диаграммой (component diagram), которая описывает распределение реализации классов по модулям, и диаграммой развертывания (deployment diagram).

## 5.2. Построение концептуальной модели.

### 5.2.1. Разработка диаграмм вариантов использования

Визуальное моделирование в UML можно представить, как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме, так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами,

каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис. Следует заметить, что два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности. Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

- Отношение ассоциации (association relationship)
- Отношение расширения (extend relationship)
- Отношение обобщения (generalization relationship)
- Отношение включения (include relationship)

При этом общие свойства вариантов использования могут быть представлены тремя различными способами, а именно с помощью отношений расширения, обобщения и включения.

**Отношение ассоциации**

Отношение ассоциации является одним из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования. Другими словами, ассоциация

специфицирует семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования.

**Отношение расширения**

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров. В метамодели отношение расширения является направленным и указывает, что применительно к отдельным примерам некоторого варианта использования должны быть выполнены конкретные условия, определенные для расширения данного варианта использования. Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования. Данное отношение включает в себя некоторое условие и ссылки на точки расширения в базовом варианте использования. Чтобы расширение имело место, должно быть выполнено определенное условие данного отношения. Ссылки на точки расширения определяют те места в базовом варианте использования, в которые должно быть помещено соответствующее расширение при выполнении условия.

Один из вариантов использования может быть расширением для нескольких базовых вариантов, а также иметь в качестве собственных расширений несколько других вариантов. Базовый вариант использования может дополнительно никак не зависеть от своих расширений.

**Отношение включения**

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного

компонента в последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения. При этом предполагается, что даже если экземпляр первого варианта использования может иметь несколько включаемых в себя экземпляров других вариантов, выполняемые ими действия должны закончиться к некоторому моменту, после чего должно быть продолжено выполнение прерванных действий экземпляра первого варианта использования в соответствии с заданным для него поведением.

Один вариант использования может быть включен в несколько других вариантов, а также включать в себя другие варианты. Включаемый вариант использования может быть независимым от базового варианта в том смысле, что он предоставляет последнему некоторое инкапсулированное поведение, детали реализации которого скрыты от последнего и могут быть легко перераспределены между несколькими включаемыми вариантами использования. Более того, базовый вариант может зависеть только от результатов выполнения включаемого в него поведения, но не от структуры включаемых в него вариантов.

Отношение включения, направленное от варианта использования A к варианту использования B, указывает, что каждый экземпляр варианта A включает в себя функциональные свойства, заданные для варианта B. Эти свойства специализируют поведение соответствующего варианта A на данной диаграмме.

### 5.2.2. Пример построения диаграммы вариантов использования

В качестве примера рассмотрим процесс моделирования системы продажи товаров по каталогу, которая может быть использована при создании соответствующих информационных систем.

В качестве актеров данной системы могут выступать два субъекта, один из которых является продавцом, а другой — покупателем. Каждый из этих актеров взаимодействует с рассматриваемой системой продажи товаров по каталогу и является ее пользователем, т. е. они оба обращаются к соответствующему сервису "Оформить заказ на покупку товара".

Полученная диаграмма вариантов использования будет выглядеть следующим образом:


Рис. 28. Диаграмма вариантов использования

Приведенная выше диаграмма вариантов использования, в свою очередь, может быть детализирована далее с целью более глубокого уточнения предъявляемых к системе требований и конкретизации деталей ее последующей реализации. В рамках общей парадигмы ООАП подобная детализация может выполняться в двух основных направлениях.

С одной стороны, детализация может быть выполнена на основе установления дополнительных отношений типа отношения "обобщение-специализация" для уже имеющихся компонентов

диаграммы вариантов использования. Так, в рамках рассматриваемой системы продажи товаров может иметь самостоятельное значение и специфические особенности отдельная категория товаров — компьютеры. В этом случае диаграмма может быть дополнена вариантом использования "Оформить заказ на покупку компьютера" и актерами "Покупатель компьютера" и "Продавец компьютеров", которые связаны с соответствующими компонентами диаграммы отношением обобщения (рис. 29.).


Рис. 29. Уточненная диаграмма вариантов использования

Построение диаграммы вариантов использования является самым первым этапом процесса объектно-ориентированного анализа и проектирования, цель которого — представить совокупность требований к поведению проектируемой системы. Спецификация требований к проектируемой системе в форме диаграммы вариантов использования представляет собой самостоятельную модель, которая в языке UML получила название модели вариантов использования и имеет свое специальное стандартное имя или стереотип "useCaseModel".

### 5.2.3. Рекомендации по разработке диаграмм вариантов использования

Главное назначение диаграммы вариантов использования заключается в формализации функциональных требований к

системе с помощью понятий соответствующего пакета и возможности согласования полученной модели с заказчиком на ранней стадии проектирования. Любой из вариантов использования может быть подвергнут дальнейшей декомпозиции на множество подвариантов использования отдельных элементов, которые образуют исходную сущность. Рекомендуемое общее количество актеров в модели — не более 20, а вариантов использования — не более 50. В противном случае модель теряет свою наглядность и, возможно, заменяет собой одну из некоторых других диаграмм.

Семантика построения диаграммы вариантов использования должна определяться следующими особенностями рассмотренных выше элементов модели. Отдельный экземпляр варианта использования по своему содержанию является выполнением последовательности действий, которая инициализируется посредством экземпляра сообщения от экземпляра актера. В качестве отклика или ответной реакции на сообщение актера экземпляр варианта использования выполняет последовательность действий, установленную для данного варианта использования. Экземпляры актеров могут генерировать новые экземпляры сообщений для экземпляров вариантов использования.

Подобное взаимодействие будет продолжаться до тех пор, пока не закончится выполнение требуемой последовательности действий экземпляром варианта использования, и соответствующий экземпляр актера (и никакой другой) не получит требуемый экземпляр сервиса. Окончание взаимодействия означает отсутствие инициализации экземпляров сообщений от экземпляров актеров для соответствующих экземпляров вариантов использования.

Варианты использования могут быть специфицированы в виде текста, а в последующем — с помощью операций и методов вместе с атрибутами, в виде графа деятельности, посредством автомата или любого другого механизма описания поведения, включающего предусловия и постусловия. Взаимодействие между вариантами использования и актерами может уточняться на диаграмме кооперации, когда описываются взаимосвязи между сущностью, содержащей эти варианты использования, и окружением или внешней средой этой сущности.

В случае, когда для представления иерархической структуры проектируемой системы используются подсистемы, система может

быть определена в виде вариантов использования на всех уровнях. Отдельные подсистемы или классы могут выступать в роли таких вариантов использования. При этом вариант, соответствующий некоторому из этих элементов, в последующем может уточняться множеством более мелких вариантов использования, каждый из которых будет определять сервис элемента модели, содержащийся в сервисе исходной системы. Вариант использования в целом может рассматриваться как суперсервис для уточняющих его подвариантов, которые, в свою очередь, могут рассматриваться как подсервисы исходного варианта использования.

Функциональность, определенная для более общего варианта использования, полностью наследуется всеми вариантами нижних уровней. Однако следует заметить, что структура элемента-контейнера не может быть представлена вариантами использования, поскольку они могут представлять только функциональность отдельных элементов модели. Подчиненные варианты использования кооперируются для совместного выполнения суперсервиса варианта использования верхнего уровня. Эта кооперация также может быть представлена на диаграмме кооперации в виде совместных действий отдельных элементов модели.

Отдельные варианты использования нижнего уровня могут участвовать в нескольких кооперациях, т. е. играть определенную роль при выполнении сервисов нескольких вариантов верхнего уровня. Для отдельных таких кооперaций могут быть определены соответствующие роли актеров, взаимодействующих с конкретными вариантами использования нижнего уровня. Эти роли будут играть актеры нижнего уровня модели системы. Хотя некоторые из таких актеров могут быть актерами верхнего уровня, это не противоречит принятым в языке UML семантическим правилам построения диаграмм вариантов использования. Более того, интерфейсы вариантов использования верхнего уровня могут полностью совпадать по своей структуре с соответствующими интерфейсами вариантов нижнего уровня.

Окружение вариантов использования нижнего уровня является самостоятельным элементом модели, который в свою очередь содержит другие элементы модели, определенные для этих вариантов использования. Таким образом, с точки зрения общего

представления верхнего уровня взаимодействие между вариантами использования нижнего уровня определяет результат выполнения сервиса варианта верхнего уровня. Отсюда следует, что в языке UML вариант использования является элементом-контейнером.

Варианты использования классов соответствуют операциям этого класса, поскольку сервис класса является по существу выполнением операций данного класса. Некоторые варианты использования могут соответствовать применению только одной операции, в то время как другие — конечного множества операций, определенных в виде последовательности операций. В то же время одна операция может быть необходима для выполнения нескольких сервисов класса и поэтому будет появляться в нескольких вариантах использования этого класса.

Реализация варианта использования зависит от типа элемента модели, в котором он определен. Например, поскольку варианты использования класса определяются посредством операций этого класса, они реализуются соответствующими методами. С другой стороны, варианты использования подсистемы реализуются элементами, из которых состоит данная подсистема. Поскольку подсистема не имеет своего собственного поведения, все предлагаемые подсистемой сервисы должны представлять собой композицию сервисов, предлагаемых отдельными элементами этой подсистемы, т. е., в конечном итоге, классами. Эти элементы могут взаимодействовать друг с другом для совместного обеспечения требуемого поведения отдельного варианта использования. Такое совместное обеспечение требуемого поведения описывается специальным элементом языка UML — кооперация или сотрудничество. Здесь лишь отметим, что кооперации используются как для уточнения спецификаций в виде вариантов использования нижних уровней диаграммы, так и для описания особенностей их последующей реализации.

Если в качестве моделируемой сущности выступает система или подсистема самого верхнего уровня, то отдельные пользователи вариантов использования этой системы моделируются актерами. Такие актеры, являясь внутренними по отношению к моделируемым подсистемам нижних уровней, часто в явном виде не указываются, хотя и присутствуют неявно в модели подсистемы. Вместо этого варианты использования непосредственно

обращаются к тем модельным элементам, которые содержат в себе подобные неявные актеры, т. е. экземпляры которых играют роли таких актеров при взаимодействии с вариантами использования. Эти модельные элементы могут содержаться в других пакетах или подсистемах. В последнем случае роли определяются в том пакете, к которому относится соответствующая подсистема.

С системно-аналитической точки зрения построение диаграммы вариантов использования специфицирует не только функциональные требования к проектируемой системе, но и выполняет исходную структуризацию предметной области. Последняя задача сочетает в себе не только следование техническим рекомендациям, но и является в некотором роде искусством, умением выделять главное в модели системы. Хотя рациональный унифицированный процесс не исключает итеративный возврат в последующем к диаграмме вариантов использования для ее модификации, не вызывает сомнений тот факт, что любая подобная модификация потребует, как по цепочке, изменений во всех других представлениях системы. Поэтому всегда необходимо стремиться к возможно более точному представлению модели именно в форме диаграммы вариантов использования.

Если же варианты использования применяются для спецификации части системы, то они будут эквивалентны соответствующим вариантам использования в модели подсистемы для части соответствующего пакета. Важно понимать, что все сервисы системы должны быть явно определены на диаграмме вариантов использования, и никаких других сервисов, которые отсутствуют на данной диаграмме, проектируемая система не может выполнять по определению. Более того, если для моделирования реализации системы используются сразу несколько моделей (например, модель анализа и модель проектирования), то множество вариантов использования всех пакетов системы должно быть эквивалентно множеству вариантов использования модели в целом.

## 5.3. Построение логической модели, отражающей статические аспекты работы сложной системы

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов. Декомпозиция некоторого представления на отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области. При этом

компоненты диаграммы соответствуют элементам статической семантической модели. Модель системы, в свою очередь, должна быть согласована с внутренней структурой классов, которая описывается на языке UML.

Класс

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 30). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

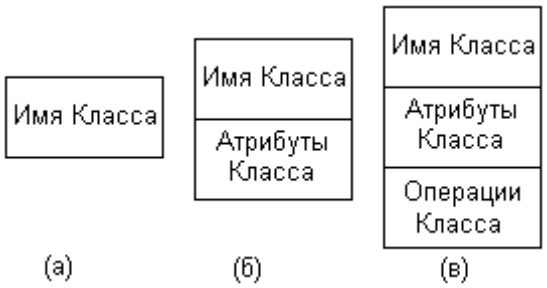



Рис. 30. Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 30, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 30, б) и операциями (рис. 30, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры

графического изображения классов на диаграмме классов приведены на рис. 31. В первом случае для класса "Прямоугольник" (рис. 31, а) указаны только его атрибуты — точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 31, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 31, в) дополнительно изображена четвертая секция, в которой указано исключение — отказ от обработки просроченной кредитной карточки.

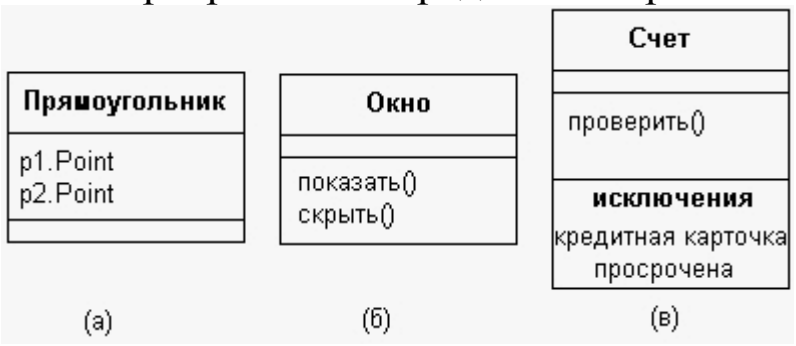



Рис. 31. Примеры графического изображения классов на диаграмме

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент",

"Продавец", "Менеджер", "Офис", "Студент" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель — двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий <Имя_пакета>::<Имя_класса>. Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

Атрибуты класса

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения.

Методы класса

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций

характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения

Отношения между классами

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями или связями в языке UML являются:

- Отношение зависимости (dependency relationship)
- Отношение ассоциации (association relationship)
- Отношение обобщения (generalization relationship)
- Отношение реализации (realization relationship)

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

Рассмотрим сходства и различия между следующими классами: цветы, маргаритки, красные розы, желтые розы, лепестки и божьи коровки. Мы можем заметить следующее:

- Маргаритка - цветок.
- Роза - (другой) цветок.
- Красная и желтая розы - розы.
- Лепесток является частью обоих видов цветов.
- Божьи коровки питаются вредителями, поражающими некоторые цветы.

Из этого простого примера следует, что классы, как и объекты, не существуют изолированно. В каждой проблемной области ключевые абстракции взаимодействуют многими интересными способами, что мы и должны отразить в проекте.

Отношения между классами могут означать одно из двух. Во-первых, у них может быть что-то общее. Например, и маргаритки, и розы - это разновидности цветов: и те, и другие имеют ярко

окрашенные лепестки, испускают аромат и так далее. Во-вторых, может быть какая-то семантическая связь. Например, красные розы больше похожи на желтые розы, чем на маргаритки. Но между розами и маргаритками больше общего, чем между цветами и лепестками. Также существует симбиотическая связь между цветами и божьими коровками: божьи коровки защищают цветы от вредителей, которые, в свою очередь, служат пищей божьим коровкам. UML диаграмма отшений между этими классами представлена на рис. 32.

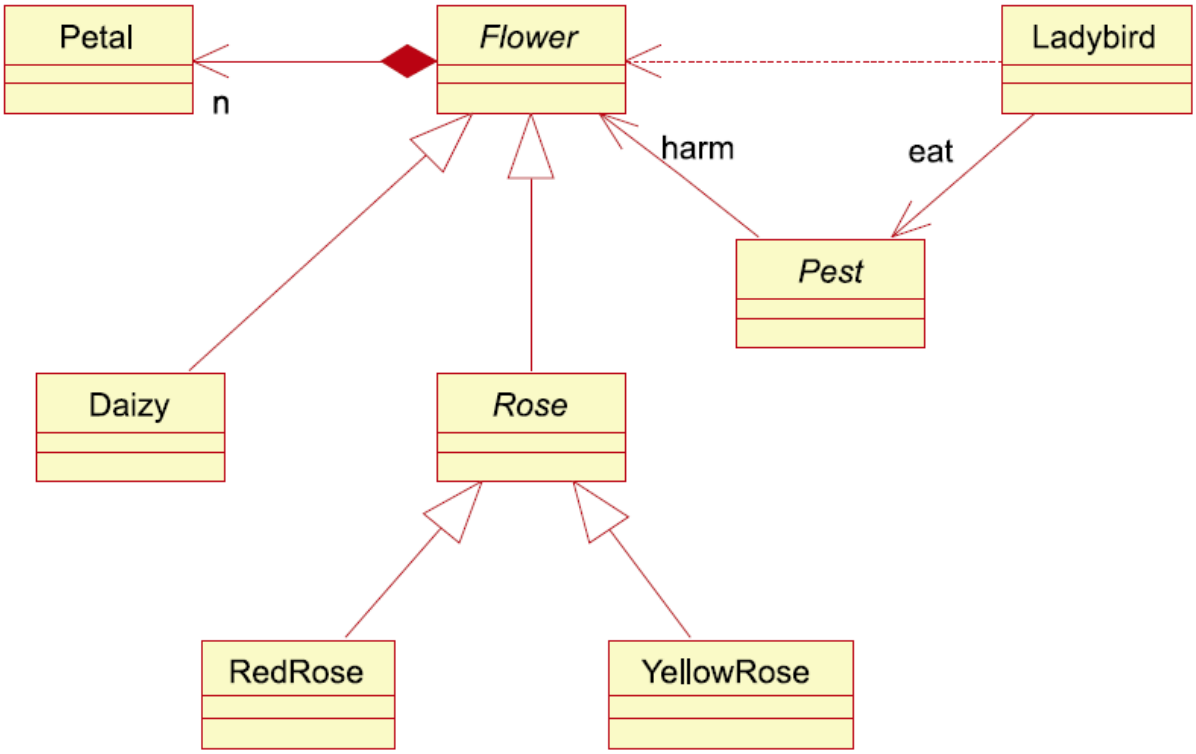



Рис. 32. Отношения между классами

Рассмотрим основные типы отношений между классами на данном примере. Во-первых, это отношение "обобщение/специализация" (общее и частное), известное как "is-a". Розы суть цветы, что значит: розы являются специализированным частным случаем, подклассом более общего класса "цветы". Во вторых, это отношение "целое/ часть", известное как "part of". Так, лепестки являются частью цветов. В-третьих, это семантические, смысловые отношения, ассоциации. Например, божьи коровки ассоциируются с цветами - хотя, казалось бы, что у них общего. Или вот: розы и свечи - и то, и другое можно использовать для украшения стола.

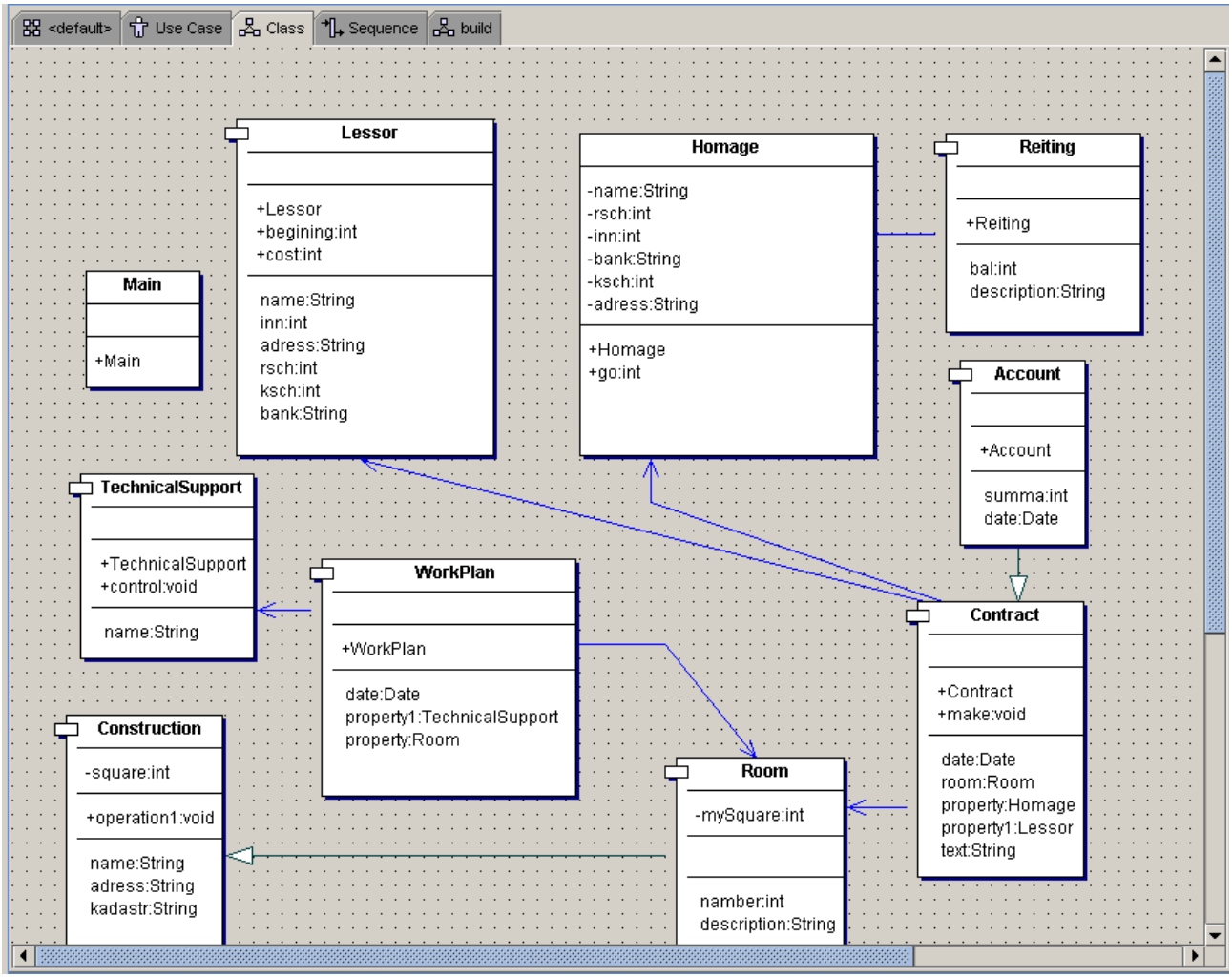



Рис. 33. Пример графического изображения диаграммы классов с различными типами отношений между классами, разработанной в среде Together

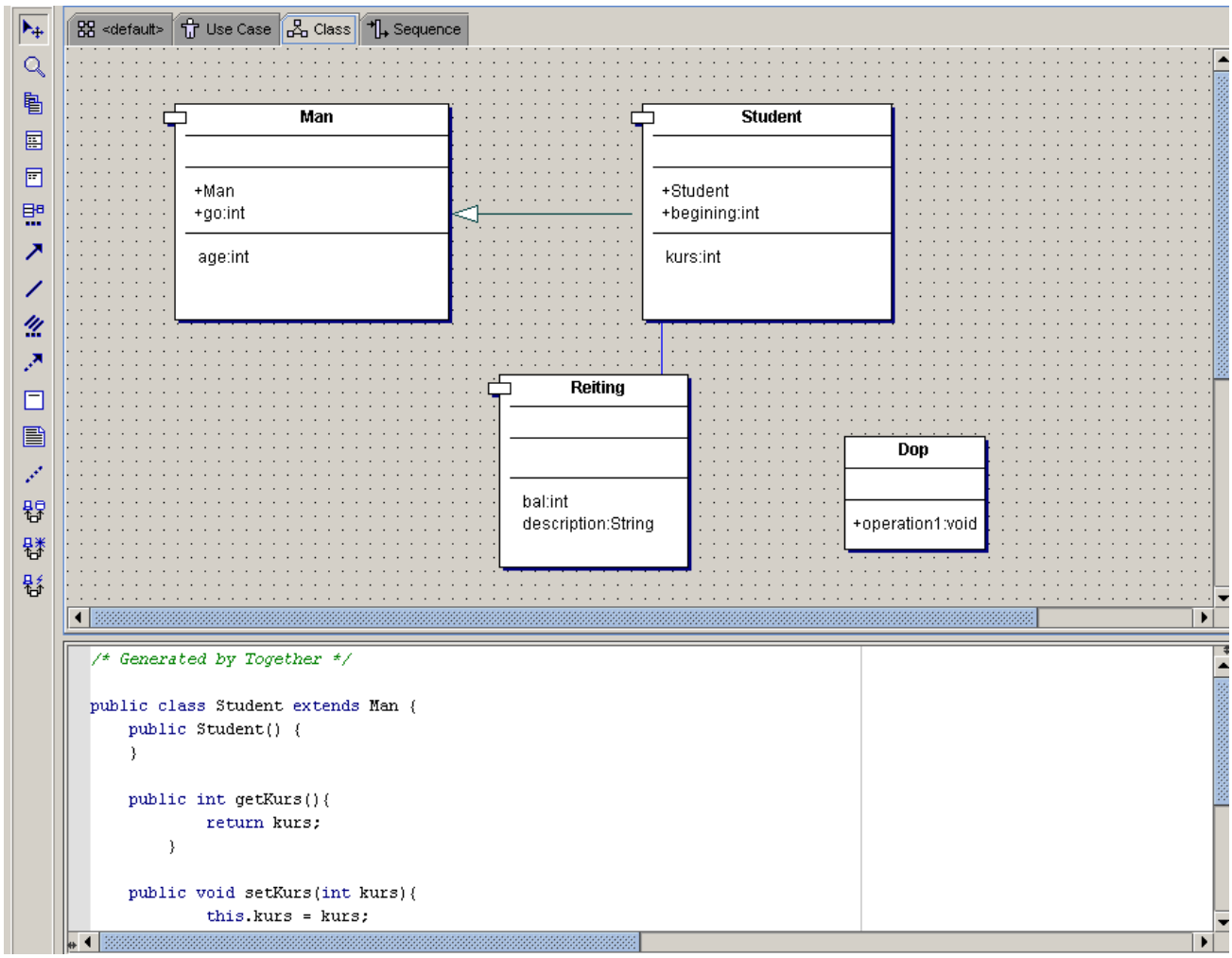



Рис. 34. Пример графического изображения диаграммы классов с элементами сгенерированного кода, разработанной в среде Together

Шаблоны классов, автоматически сгенерированные средой Together представлены ниже:

```
/* Generated by Together */

public class Reiting {
    public int getBal(){
        return bal;
    }

    public void setBal(int bal){
        this.bal = bal;
```

```
        }

    public String getDescription(){
            return description;
        }

    public void setDescription(String description){
            this.description = description;
        }

    private int bal;
    private String description;
}


/* Generated by Together */

public class Student extends Man {
    public Student() {
    }

    public int getKurs(){
            return kurs;
        }

    public void setKurs(int kurs){
            this.kurs = kurs;
        }

    private int kurs;
    private Reiting lnkClass1;
}

/* Generated by Together */

public class Man {
       private int age;
    public Man(int age) {
```

```
        this.age=age;
    }

    public int getAge()
    {
        return age;
    }
    public void setAge(int age)
    {
    }
    public int go()
    {
    }
}
```

Рекомендации по построению диаграмм классов

Процесс разработки диаграммы классов занимает центральное место в ООАП сложных систем. От умения правильно выбрать классы и установить между ними взаимосвязи часто зависит не только успех процесса проектирования, но и производительность выполнения программы. Как показывает практика ООП, каждый программист в своей работе стремится в той или иной степени использовать уже накопленный личный опыт при разработке новых проектов. Это обусловлено желанием свести новую задачу к уже решенным, чтобы иметь возможность использовать не только проверенные фрагменты программного кода, но и отдельные компоненты в целом (библиотеки компонентов).

Такой стереотипный подход позволяет существенно сократить сроки реализации проекта, однако приемлем лишь в том случае, когда новый проект концептуально и технологически не слишком отличается от предыдущих. В противном случае платой за сокращение сроков проекта может стать его реализация на устаревшей технологической базе. Что касается собственно объектной структуризации предметной области, то здесь уместно придерживаться тех рекомендаций, которые накоплены в ООП.

При определении классов, атрибутов и операций и задании их имен и типов перед отечественными разработчиками всегда встает

невольный вопрос: какой из языков использовать в качестве естественного, русский или английский? С одной стороны, использование родного языка для описания модели является наиболее естественным способом ее представления и в наибольшей степени отражает коммуникативную функцию модели системы. С другой стороны, разработка модели является лишь одним из этапов разработки соответствующей системы, а применение инструментальных средств для ее реализации в абсолютном большинстве случаев требует использования англоязычных терминов.

Отвечая на поставленный выше вопрос, следует отметить, что наиболее целесообразно придерживаться следующих рекомендаций. При построении диаграммы вариантов использования, являющейся наиболее общей концептуальной моделью проектируемой системы, применение русскоязычных терминов является не только оправданным с точки зрения описания структуры предметной области, но и эффективным с точки зрения коммуникативного взаимодействия с заказчиком и пользователями. При построении остальных типов диаграмм следует придерживаться разумного компромисса.

В частности, на начальных этапах разработки диаграмм целесообразность использования русскоязычных терминов вполне очевидна и оправдана. Однако, по мере готовности графической модели для реализации в виде программной системы и передачи ее для дальнейшей работы программистам, акцент может смещаться в сторону использования англоязычных терминов, которые в той или иной степени отражают особенности языка программирования, на котором предполагается реализация данной модели.

Более того, использование CASE-инструментариев для автоматизации ООАП, чаще всего, накладывает свои собственные требования на язык спецификации моделей. Именно по этой причине большинство примеров в литературе даются в англоязычном представлении, а при их переводе на русский может быть утрачена не только точность формулировок, но и семантика соответствующих понятий.

После разработки диаграммы классов процесс ООАП может быть продолжен в двух направлениях. С одной стороны, если поведение системы тривиально, то можно приступить к разработке

диаграмм кооперации и компонентов. Однако для сложных динамических систем поведение представляет важнейший аспект их функционирования. Детализация поведения осуществляется последовательно при разработке диаграмм состояний, последовательности и деятельности.

## 5.4. Построение логической модели, отражающей динамические аспекты работы сложной системы

### 5.4.1. Разработка диаграмм последовательности (sequence diagram)

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. Во-первых, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Этот вид канонических диаграмм является предметом изучения настоящей главы.

Ранее, при изучении диаграмм состояния и деятельности, было отмечено одно немаловажное обстоятельство. Хотя рассмотренные диаграммы и используются для спецификации динамики поведения систем, время в явном виде в них не присутствует. Однако временной аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели в языке UML используются диаграммы последовательности.

Во-вторых, можно рассматривать структурные особенности взаимодействия объектов. Для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации.

Объекты

На диаграмме последовательности изображаются

исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно — слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни (рис. 35). Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который, как известно, представляет собой экземпляр класса.

Не исключается ситуация, когда имя объекта может отсутствовать на диаграмме последовательности. В этом случае указывается только имя класса, а сам объект считается анонимным.

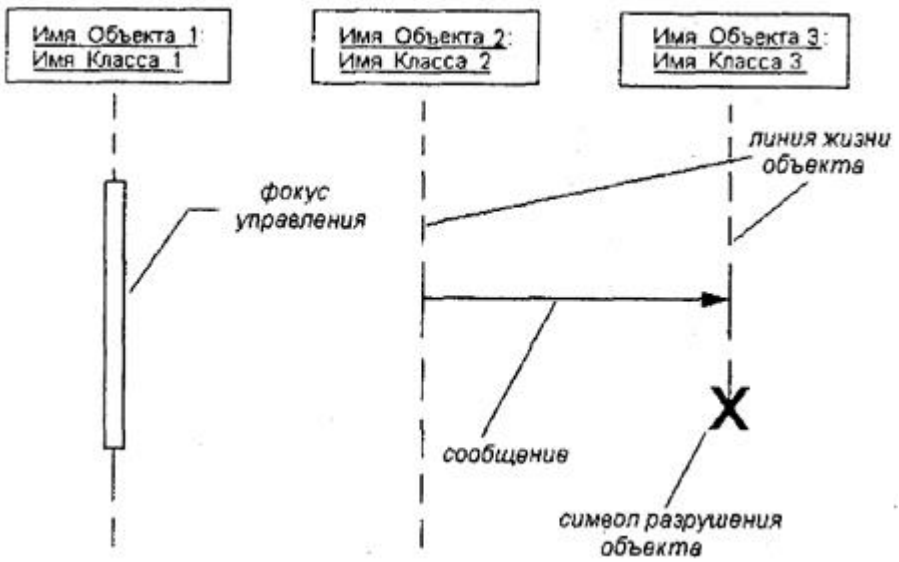



Рис. 35. Различные графические примитивы диаграммы последовательности

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (объект 1 на рис. 35). Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при

взаимодействии друг с другом.

Второе измерение диаграммы последовательности — вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

Линия жизни объекта

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней (объекты 1 и 2 на рис. 35).

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы "X" (объект 3 на рис. 35). Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Вовсе не обязательно создавать все объекты в начальный момент времени. Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и

повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той ее части, которая соответствует моменту создания объекта (объект 6 на рис. 36). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Очевидно, объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.

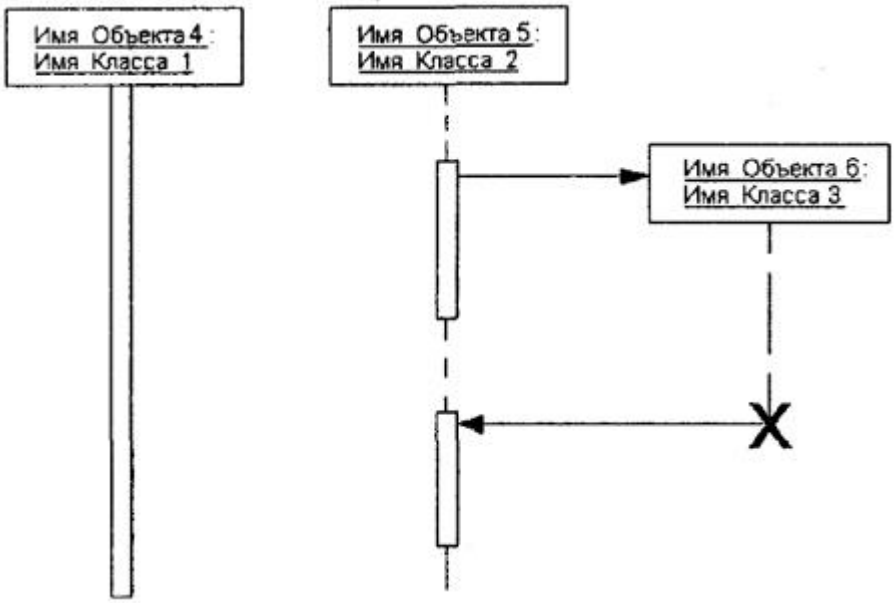



Рис. 36. Графическое изображение различных вариантов линий жизни и фокусов управления объектов

Фокус управления

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника (см. объект 1 на рис. 35), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона — окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни (объект 4 на рис. 36),

если на всем ее протяжении он является активным.

С другой стороны, периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (объект 5 на рис. 36). Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь. В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления (рис. 37). Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой. Речь идет о том, что наличие во многих языках программирования специальных средств построения рекурсивных процедур требует визуализации соответствующих понятий в форме графических примитивов. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается это рекурсивное взаимодействие (объект 7 на рис. 37).

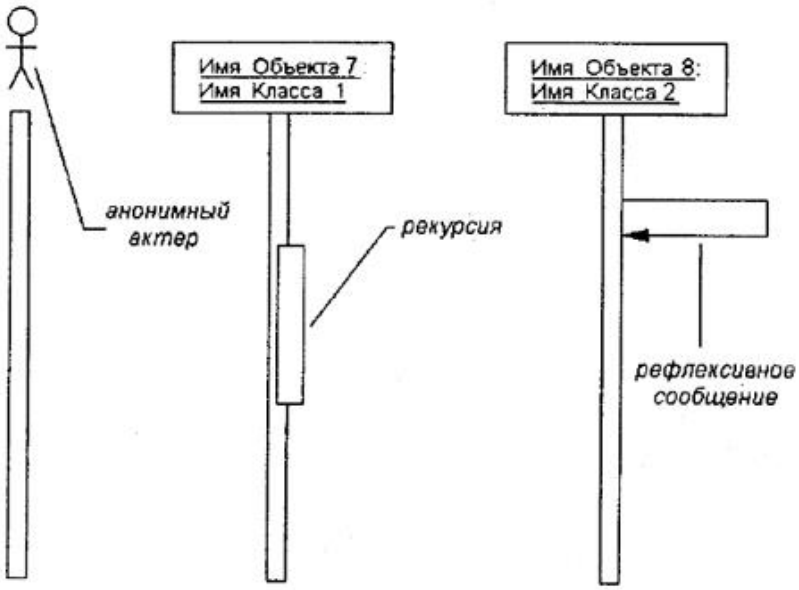



Рис. 37. Графическое изображение актера, рекурсии и рефлексивного сообщения на диаграмме последовательности

Сообщения

Как было отмечено выше, цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий. Сообщения могут инициировать выполнение операций объектом соответствующего класса, а параметры этих операций передаются вместе с сообщением. На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе.

В таком контексте каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту,

который его получает. Иногда отправителя сообщения называют клиентом, а получателя — сервером. При этом сообщение от клиента имеет форму запроса некоторого сервиса, а реакция сервера на запрос после получения сообщения может быть связана с выполнением определенных действий или передачи клиенту необходимой информации тоже в форме сообщения.

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение.

- Первая разновидность сообщения является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки всегда соприкасается с фокусом управления или линией жизни того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект зачастую получает и фокус управления, становясь активным.
- Вторая разновидность сообщения используется для обозначения простого (не вложенного) потока управления. Каждая такая стрелка указывает на прогресс одного шага потока. При этом соответствующие сообщения обычно являются асинхронными, т. е. могут возникать в произвольные моменты времени. Передача такого сообщения обычно сопровождается получением фокуса управления объектом, его принявшим.
- Третья разновидность явно обозначает асинхронное сообщение между двумя объектами в некоторой процедурной последовательности. Примером такого сообщения может служить прерывание операции при возникновении исключительной ситуации. В этом случае информация о такой ситуации передается вызывающему объекту для продолжения процесса дальнейшего взаимодействия.
- Наконец, последняя разновидность сообщения используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации

объекта. В то же время считается, что каждый вызов процедуры имеет свою пару — возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные сообщения, стрелка возврата должна указываться явным образом.

Обычно сообщения изображаются горизонтальными стрелками, соединяющими линии жизни или фокусы управления двух объектов на диаграмме последовательности. При этом неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами. Считается также, что за время передачи сообщения с соответствующими объектами не может произойти никаких событий. Другими словами, состояния объектов остаются без изменения. Если же это предположение не может быть признано справедливым, то стрелка сообщения изображается под некоторым наклоном, так чтобы конец стрелки располагался ниже ее начала.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые рефлексивные сообщения (объект 8 на рис. 37). Такие сообщения изображаются прямоугольником со стрелкой, начало и конец которой совпадают. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые аргументы или параметры, в зависимости от конкретных значений которых может быть получен различный результат. Соответствующие параметры будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Построение диаграммы последовательности сводится к добавлению или удалению отдельных объектов и сообщений, а также к их спецификации. Доступ к спецификации этих элементов организован либо через контекстное меню, либо через пункт меню Browse - Specification (Браузер - Спецификация). При добавлении сообщений на диаграмму последовательности они получают по

умолчанию свой номер в последовательности.

Пример построения диаграммы последовательности

В качестве примера рассмотрим построение диаграммы последовательности для моделирования процесса телефонного разговора с использованием обычной телефонной сети. Объектами в этом примере являются: два абонента а и Ь, два телефонных аппарата end, коммутатор и сам разговор как объект моделирования. При этом как коммутатор, так и разговор являются анонимными объектами.

На первом этапе располагаем выбранные объекты на предполагаемой диаграмме (рис. 38). Заметим, что абонентов мы будем рассматривать как актеров, причем первый из них — а — играет активную роль, а второй — b — пассивную роль. Поэтому первый получает фокус управления сразу после своего появления в системе, а второй имеет только линию жизни. Коммутатор также имеет постоянную активность, что изображается его фокусом управления. Разговор как объект появляется только после установки соединения и уничтожается с его прекращением. Поэтому он будет изображен позже на этой же диаграмме последовательности.

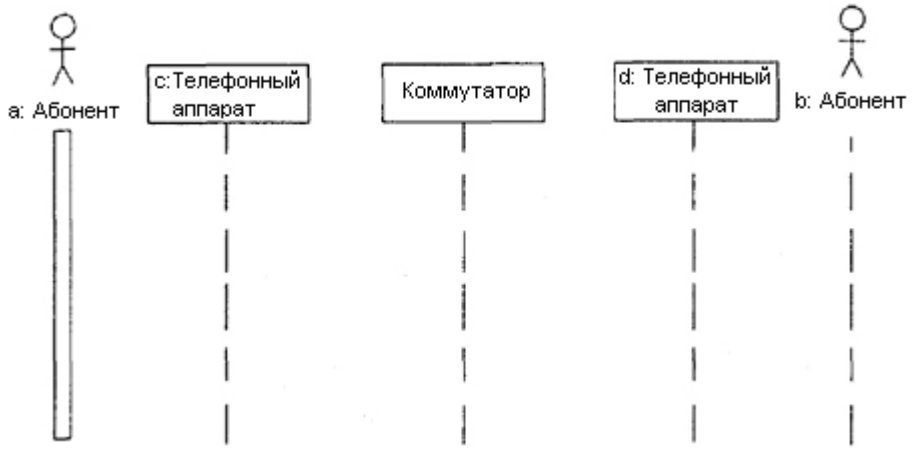



Рис. 38. Начальный фрагмент диаграммы последовательности для моделирования телефонного разговора

Процесс взаимодействия в этой системе начинается с поднятия трубки телефонного аппарата первым абонентом. Тем самым он посылает сообщение телефонному аппарату с, которое переводит этот аппарат в активное состояние и вызывает действие — подачу

тонового сигнала в телефонную трубку для первого абонента. Следующее действие также инициируется первым абонентом — набор цифр телефонного номера. Это представлено в форме итеративного сообщения со знаком "*" слева от его имени.

Заметим, что поднятие телефонной трубки и набор цифр номера являются физическими действиями и поэтому изображаются в форме простых асинхронных сообщений. После набора цифр'номера телефона аппарат с рекурсивно вызывает процедуру посылки коммутационных импульсов на коммутатор. Последний инициирует создание нового объекта в моделируемой системе — телефонного разговора. Дополненный фрагмент диаграммы последовательности изображен на рис. 39.

После создания анонимный объект "разговор" сразу получает фокус активности и посылает сообщение телефонному аппарату d на выполнение действия — звонка вызова. При этом второй абонент снимает трубку (асинхронное сообщение), тем самым устанавливается прямое соединение между абонентами а и Ь. После того как абоненты опустят трубки, разговор заканчивается. Тем самым объект "разговор" уничтожается. Окончательный вариант диаграммы последовательности может содержать некоторые временные ограничения и комментарии (рис. 40). Назначение отдельных сообщений соответствуют рассмотренным действиям.

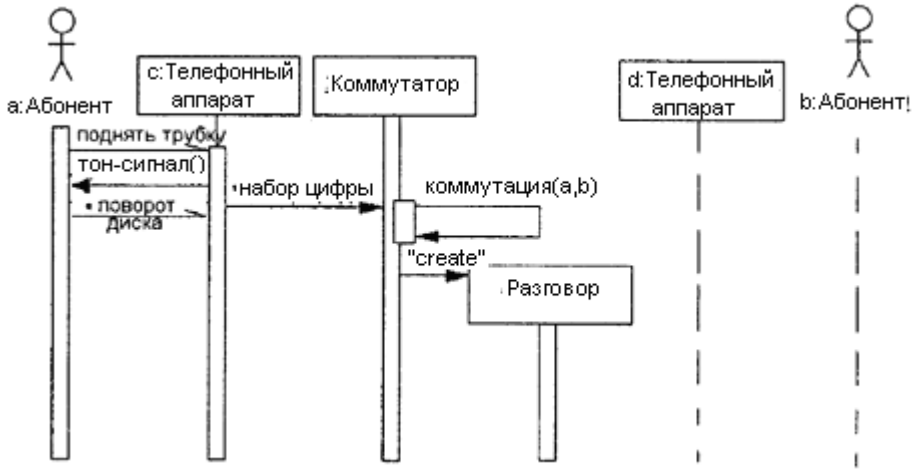



Рис. 39. Дополненный фрагмент диаграммы последовательности для моделирования телефонного разговора

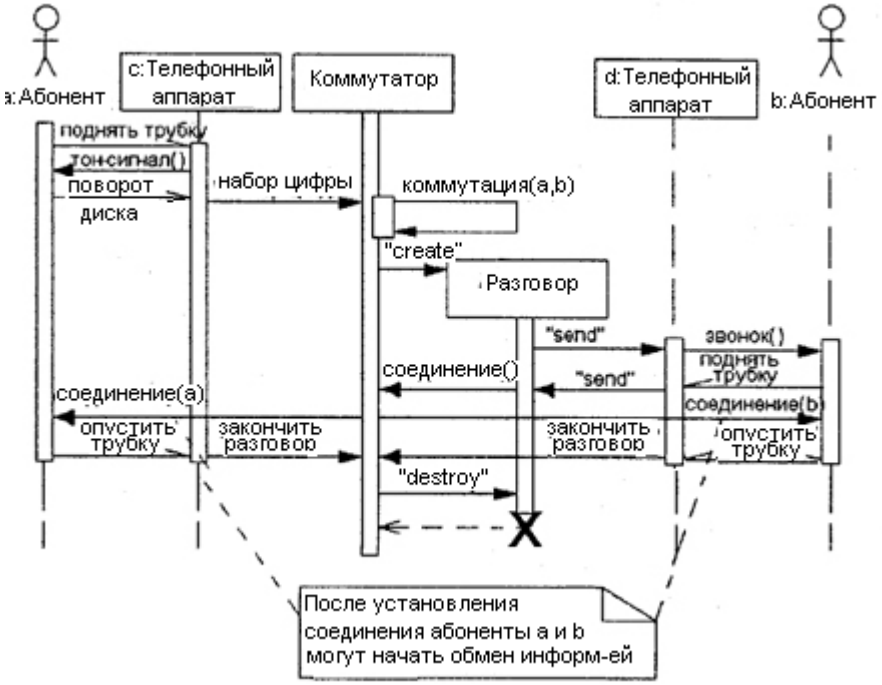



Рис. 40. Окончательный вариант диаграммы последовательности для моделирования телефонного разговора

Ниже приводится пример построенной диаграммы последовательности в среде Together (рис. 41).

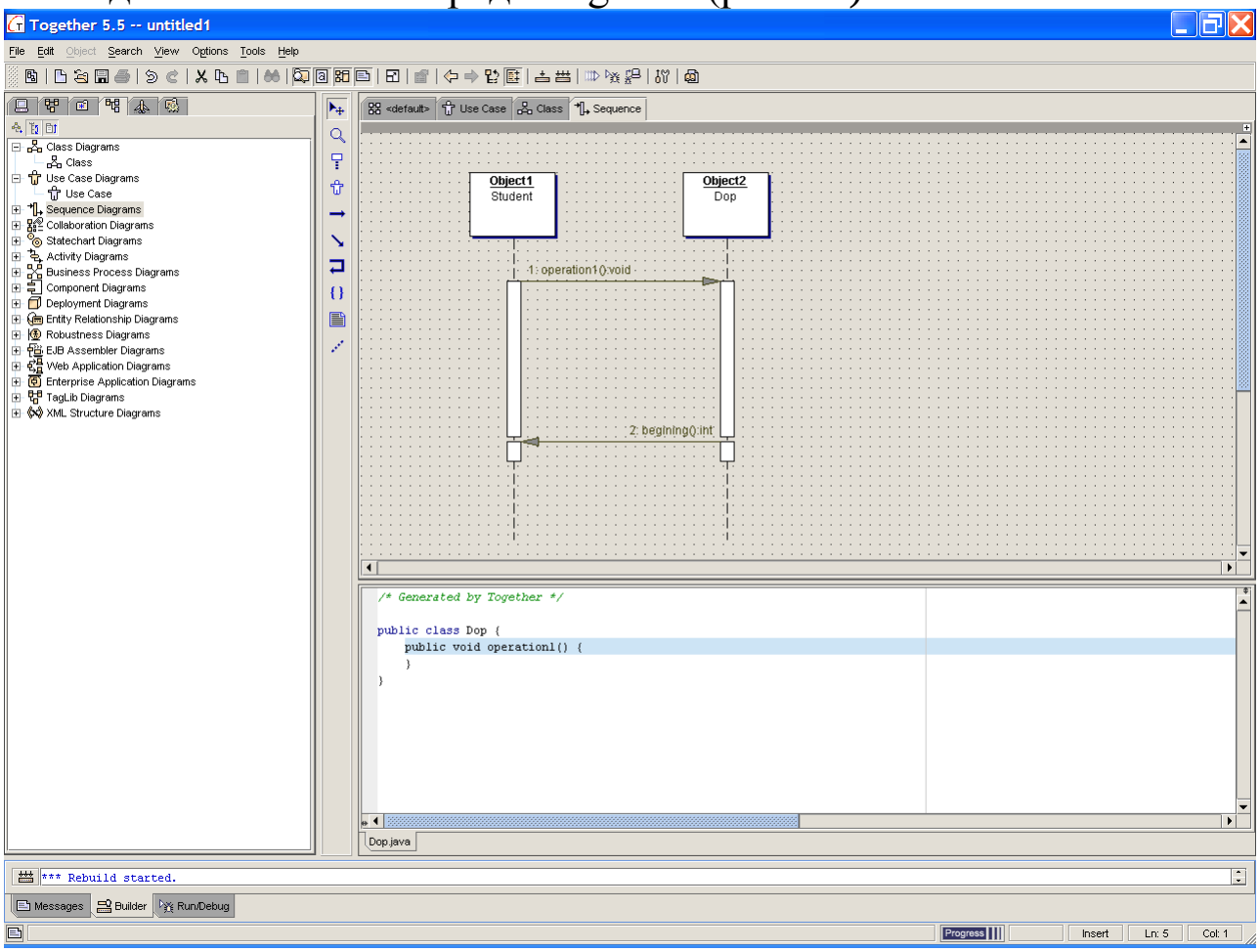



Рис. 41. Пример диаграммы последовательности

Заключительные рекомендации по построению диаграмм последовательности

Как уже отмечалось, построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Здесь необходимо установить, какие объекты будут существовать постоянно, а какие временно — только на период выполнения ими требуемых действий.

Когда объекты визуализированы, можно приступать к спецификации сообщений. При этом следует учитывать те роли, которые играют сообщения в системе. При необходимости уточнения этих ролей надо использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.

Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. Однако следует помнить, что каждый альтернативный поток управления может существенно затруднить понимание построенной модели. Поэтому общим правилом является визуализация каждого потока управления на отдельной диаграмме последовательности. В этой ситуации такие отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия.

Дальнейшая детализация диаграммы последовательности связана с введением временных ограничений на выполнение отдельных действий в системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Однако необходимость синхронизировать сложные потоки управления, как правило, требуют введение в модель таких ограничений. Общая их запись должна следовать семантике языка объектных ограничений, который рассмотрен в приложении.

### 5.4.2. Разработка диаграмм деятельности (activity diagram)

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Конечно, мы не всегда задумываемся о том, что подобные операции относятся к столь научным категориям. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

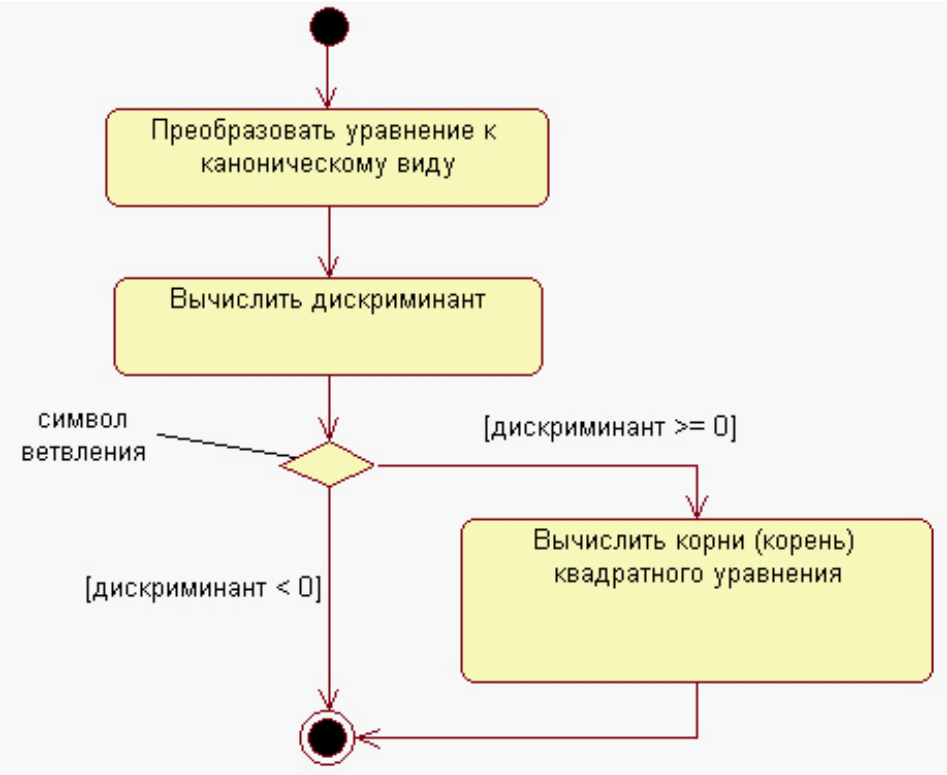



Рис. 42. Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения


Рис. 43. Различные варианты ветвлений на диаграмме деятельности

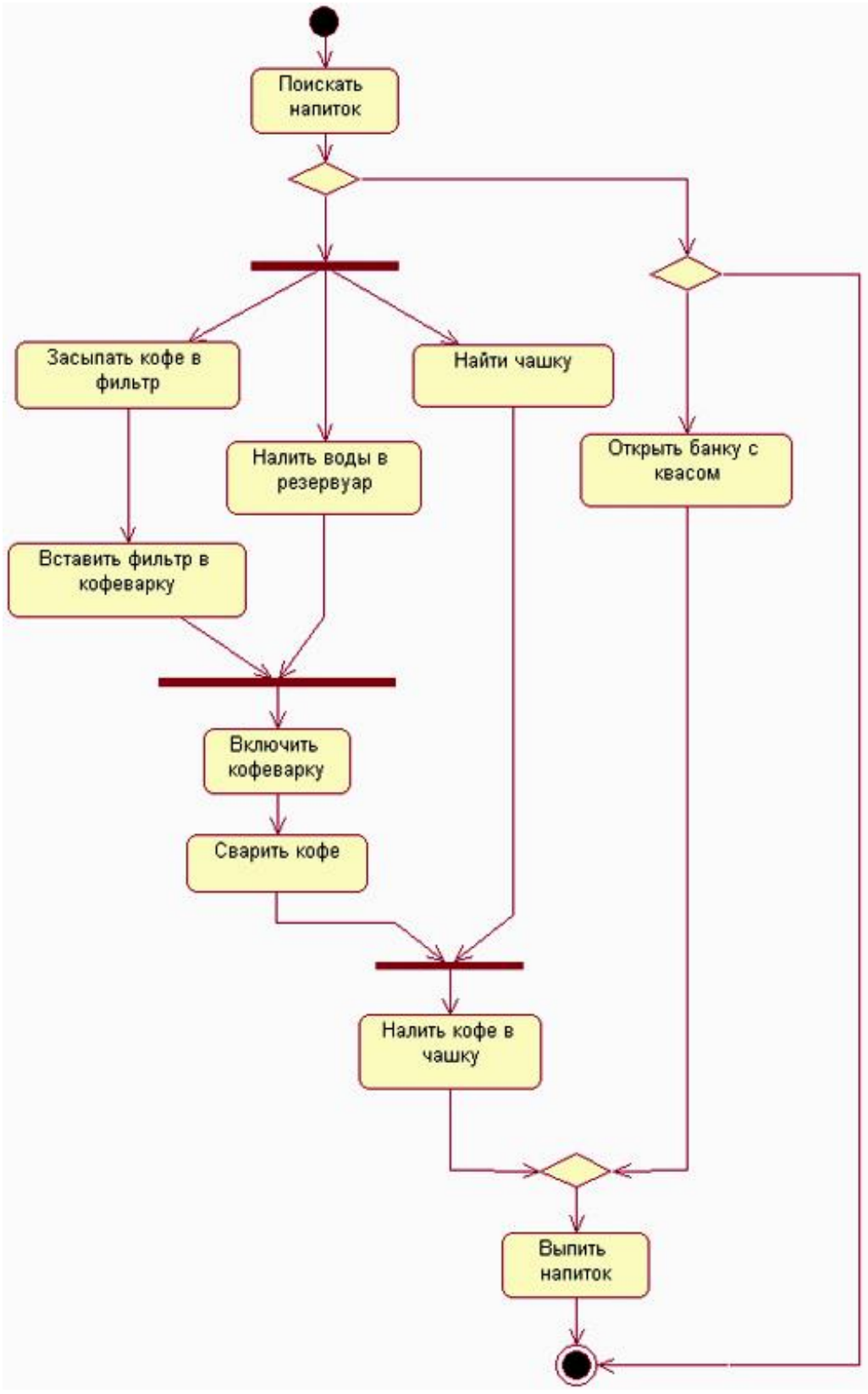



Рис. 44. Диаграмма деятельности для примера с приготовлением напитка

## Рекомендации по построению диаграмм деятельности

Диаграммы деятельности играют важную роль в понимании процессов реализации алгоритмов выполнения операций классов и потоков управления в моделируемой системе. Используемые для этой цели традиционные блок-схемы алгоритмов обладают серьёзными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений компаний и фирм.

Содержание диаграммы деятельности во многом напоминает диаграмму состояний, хотя и не тождественно ей. Поэтому многие рекомендации по построению последней оказываются справедливыми применительно к диаграмме деятельности. В частности, эта диаграмма строится для отдельного класса, варианта использования, отдельной операции класса или целой подсистемы.

С одной стороны, на начальных этапах проектирования, когда детали реализации деятельностей в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения под-деятельностей, которые в совокупности образуют деятельность подсистем. В последующем, по мере разработки диаграмм классов и состояний, эти под-деятельности уточняются в виде отдельных вложенных диаграмм деятельности компонентов подсистем, какими выступают классы и объекты.

С другой стороны, отдельные участки рабочего процесса в существующей системе могут быть хорошо отлаженными, и у разработчиков может возникнуть желание сохранить этот механизм выполнения действий в проектируемой системе. Тогда строится диаграмма деятельности для этих участков, отражающая конкретные особенности выполнения действий с использованием дорожек и объектов. В последующем такая диаграмма вкладывается в более общие диаграммы деятельности для подсистемы и системы в целом, сохраняя свой уровень детализации.

Таким образом, процесс объектно-ориентированного анализа и проектирования сложных систем представляется как последовательность итераций нисходящей и восходящей

разработки отдельных диаграмм, включая и диаграмму деятельности. Доминирование того или иного из направлений разработки определяется особенностями конкретного проекта и его новизной.

В случае типового проекта большинство деталей реализации действий могут быть известны заранее на основе анализа существующих систем или предшествующего опыта разработки систем-прототипов. Для этой ситуации доминирующим будет восходящий процесс разработки (Зачем изобретать велосипед заново?). Использование типовых решений может существенно сократить время разработки и избежать возможных ошибок при реализации проекта.

При разработке проекта новой системы, процесс функционирования которой основан на новых технологических решениях, ситуация представляется более сложной. А именно, до начала работы над проектом могут быть неизвестны не только детали реализации отдельных деятельностей, но и само содержание этих деятельностей становится предметом разработки. В данном случае доминирующим будет нисходящий процесс разработки от более общих схем к уточняющим их диаграммам. При этом достижение такого уровня детализации всех диаграмм, который достаточен для понимания особенностей реализации всех действий и деятельностей, может служить признаком завершения отдельных этапов работы над проектом.

В заключение следует заметить, что диаграмма деятельности, так же как и другие виды канонических диаграмм, не содержит средств выбора оптимальных решений. При разработке сложных проектов проблема выбора оптимальных решений становится весьма актуальной. Рациональное расходование средств, затраченных на разработку и эксплуатацию системы, повышение ее производительности и надежности зачастую определяют конечный результат всего проекта. В такой ситуации можно рекомендовать использование дополнительных средств и методов, ориентированных на аналитико-имитационное исследование моделей системы на этапе разработки ее проекта.

В частности, при построении диаграмм деятельности сложных систем могут быть успешно использованы различные классы сетей Петри (классические, логико-алгебраические, стохастические,

нечеткие и др.) и нейронных сетей. Применение этих формализмов позволяет не только получить оптимальную структуру поведения системы на ее модели, но и специфицировать целый ряд дополнительных характеристик системы, которые не могут быть представлены на диаграмме деятельности и других диаграммах UML.

# 6. ЗАДАНИЯ НА ЛАБОРАТОРНЫЕ РАБОТЫ

ЛАБОРАТОРНАЯ РАБОТА № 1

Создание и работа с простыми моделями представления данных с помощью языка Java.

Цель работы

Изучить модели представления данных в виде классов, работу с моделями данных для представления визуальных компонент JTable (класс TableModel), JList (класс ListModel), работу с файловыми данными, модели, основанные на файловых структурах ini, xml.

Задание на лабораторную работу № 1

1. Создать класс, представляющий собой модель данных «Студент», «Автомобиль». В каждом таком классе должно быть не менее 5 полей и соответственно по два метода доступа к ним (set и get методы).

2. Создать модель данных в виде ini файла и в виде xml файла. Написать программу на языке java, выполняющую чтение данных из этих файлов. Для чтения данных из ini файла использовать класс java.util.Properties и его метод load. После создания экземпляра класса Properties (например ini) и запуска его метода load можно получить имена полей (идентификаторов) и их значения следующим образом:

```
Enumeration e = ini.propertyNames() ;
    while (e.hasMoreElements()) {
    Object element=e.nextElement();
    System.out.print(element+"=");
    System.out.println(ini.getProperty((String)element));
  }
```

Для работы с xml файлом использовать класс ConfigLoader, пакеты xerces.jar, configloader_1_5.jar.

3. Создать модели данных для визуальных компонент JTable и JList. Написать программу на языке java, отображающие компоненты JTable и JList.

ЛАБОРАТОРНАЯ РАБОТА № 2

Работа с реляционными моделями данных.

Цель работы

Создание моделей данных для представления информации из базы данных. Визуальное отображение ее в компоненте JTable с использованием модели javax.swing.table.TableModel.

Задание на лабораторную работу № 2

Разработать модель (класс Kafedra) для представления информации из базы данных. Информация задается в таблице (create table kafedra (id int, name varchar(30), fone varchar(10), address varchar(40), age int ). Имя базы данных в СУБД MySql: example.

```
mysql> describe kafedra;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | YES  |     | NULL    |       |
| name   | varchar(30) | YES  |     | NULL    |       |
| fone   | varchar(10) | YES  |     | NULL    |       |
| adress | varchar(40) | YES  |     | NULL    |       |
| age    | int(11)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Написать программу на языке Java, загружающую информацию из таблицы kafedra ("select * from kafedra") в написанную модель с последующим отображением ее в визуальном компоненте JTable с использованием модели javax.swing.table.TableModel;. Дописать программу, отображающую таблицу, не зная количества, названий и типов данных столбцов. Программе в качестве параметра должно передаваться только имя таблицы. Для получения информации о таблице использовать метод getMetaData() класса ResultSet.

Для получения количества и названий столбцов использовать методы getColumnCount() и getColumnLabel класса ResultSetMetaData. Для получения значения из таблицы, не зная его типа использовать метод getObject класса ResultSet.

ЛАБОРАТОРНАЯ РАБОТА № 3

Описание интегрированной модели сложной системы средствами языка UML (концептуальные модели).

Цель работы

Создание концептуальных моделей, логических моделей, отражающих статические аспекты функционирования сложной системы

Задание на лабораторную работу № 3

Разработать концептуальную модель (диаграмма вариантов использования, диаграмма классов) проектируемой информационной системы на выбранную тему. Название проектируемой системы согласовать с преподавателем.

## ЛАБОРАТОРНАЯ РАБОТА № 4

Описание интегрированной модели сложной системы средствами языка UML (логические модели).

Цель работы

Создание логических моделей, отражающих динамические аспекты функционирования сложной системы.

Задание на лабораторную работу № 4

Разработать логические модели проектируемой информационной системы, отражающие динамические аспекты работы (диаграмма последовательности, диаграмма деятельности) на тему, выбранную в лабораторной работе №3.

# ЗАКЛЮЧЕНИЕ

Возможности современных информационных технологий с каждым годом становятся все более широкими и затрагивают все больше сфер жизни человека. Глобальными информационными системами пользуются все больше и в самых различных сферах жизни. Работа с моделями представления данных ведется на всех этапах проектирования и разработки информационных систем. На этапе проектирования ведется разработка интегрированной модели системы, которая включает в себя разработку как концептуальных, так и физических моделей. Проектирование таких сложных моделей становится все более трудной и важной задачей.

В учебном пособии описываются типы моделей данных, иерархия моделей данных, уровни представления (концептуальный, логический, физический), реляционная модель данных, ER – модель, функциональная модель данных. Приводится сравнение различных моделей данных концептуального уровня.

Для проектирования, разработки и поддержки моделей данных, используемых в современных информационных системах, требуются специалисты со знаниями методологий объектно-ориентированного анализа и проектирования с использованием языка UML, объектно-ориентированного программирования на языке Java, знаниями методологии построения функциональных моделей SADT, нотаций IDEF, методологии проектирования данных ERD, DFD. Данное учебное пособие предназначено для освоения технологий построения моделей данных с использованием средств программирования Java и методологий SADT, DFD, ERD, UML.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Буч Г. Язык UML: руководство пользователя/ Г. Буч, Д. Рамбо, А. Джекобсон М.: ДМК, 2000.

2. Леоненков А.В. Самоучитель UML / А.В. Леоненков.- СПб.: БХВ – Петербург, 2001. – 304 с.: ил.

3. Королев Е.Н. Проектирование информационных систем с помощью языка UML: учеб. пособие/ Е.Н. Королев. Воронеж: ВГТУ, 2009. – 95 с.

4. Королев Е.Н. Проектирование и разработка приложений на языке Java: учеб. пособие/ Е.Н. Королев. Воронеж: ВГТУ, 2008. – 137 с.

# ОГЛАВЛЕНИЕ

Учебное издание

Королев Евгений Николаевич

МОДЕЛИ ПРЕДСТАВЛЕНИЯ ДАННЫХ

В авторской редакции

Компьютерный набор Е.Н. Королева

Подписано в печать 22.03.2010.
Формат 60x84/16. Бумага для множительных аппаратов.
Усл. печ. л. 7,8. Уч.-изд. л. 6,3. Тираж 250 экз.
Зак. №

ГОУВПО “Воронежский государственный технический университет”
394026 Воронеж, Московский просп., 14