

А.М. Вендров

ПРАКТИКУМ

**ПО ПРОЕКТИРОВАНИЮ
ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ
ЭКОНОМИЧЕСКИХ
ИНФОРМАЦИОННЫХ
СИСТЕМ**



ПРАКТИКУМ

ПО ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЭКОНОМИЧЕСКИХ ИНФОРМАЦИОННЫХ СИСТЕМ

Допущено

Министерством образования Российской Федерации
в качестве учебного пособия

для студентов высших учебных заведений,
обучающихся по специальностям

351400 "Прикладная информатика в экономике",

351500 "Математическое обеспечение

и администрирование информационных систем"



Москва
"Финансы и статистика"
2002

РЕЦЕНЗЕНТЫ:

кафедра проектирования экономических информационных систем

Московского государственного университета
экономики, статистики и информатики (МЭСИ);

Г.Н. Калянов,

доктор технических наук, профессор

Вендров А.М.

В29 Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие. - М.: Финансы и статистика, 2002. - 192 с: ил.
ISBN 5-279-02440-6

Практикум является дополнением к одноименному учебнику. Он содержит практические задания и упражнения на построение моделей программных систем с применением как структурного, так и объектно-ориентированного подхода (с использованием стандартного языка моделирования UML). Построение моделей и языка UML выполняется с помощью CASE-средства Rational Rose.

Для студентов высших учебных заведений, обучающихся по специальностям 351400 «Прикладная информатика в экономике» и 351500 «Математическое обеспечение и администрирование информационных систем». Может быть полезен разработчикам и пользователям систем программного обеспечения.

В 2404000000-091
010(01) - 2002

233-2001

УДК 004.415.2:33(076.5)
ББК 65ф.я73

ISBN 5-279-02440-6

А.М. Вендров, 2002

В современной практике проектирования программного обеспечения (ПО) широко применяются визуальные модели, они представляют собой средства для описания, проектирования и документирования архитектуры системы. По мнению одного из авторитетнейших специалистов в области объектно-ориентированного подхода Гради Буча, моделирование является центральным звеном всей деятельности по созданию качественного ПО. Модели строятся для того, чтобы понять и осмыслить структуру и поведение будущей системы, облегчить управление процессом ее создания и уменьшить возможный риск, а также документировать принимаемые проектные решения. Хорошие модели служат основой взаимодействия участников проекта и гарантируют корректность архитектуры.

Одним из факторов, от которых зависит успех проекта, является наличие строгого стандарта языка моделирования, включающего элементы модели - фундаментальные концепции моделирования и их семантику, нотацию - визуальное представление элементов моделирования, и руководство по использованию языка - правила применения его элементов в рамках построения тех или иных типов моделей ПО.

Проблемы автоматизации проектирования ПО (в частности, визуального моделирования) породили потребность в программно-технологических средствах специального класса - CASE-средствах. Термин CASE (Computer Aided Software Engineering) имеет весьма широкое толкование. Первоначальное значение термина CASE ограничивалось вопросами автоматизации разработки только лишь ПО, а в настоящее время оно приобрело новый смысл и охватывает процесс разработки сложных систем в целом.

CASE-технология представляет собой совокупность методов проектирования ПО, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки и сопровождения ПО и разрабатывать приложения в соответствии с информационными потребностями пользователей.

Цель настоящего практикума - формирование навыков самостоятельного практического применения современных методов и средств проектирования ПО информационных систем, основанных на использовании визуального моделирования и CASE-средств.

Практикум является дополнением к учебнику «Проектирование программного обеспечения экономических информационных систем», выпущенному издательством «Финансы и статистика» в 2000 г. Он содержит практические задания и упражнения на построение моделей программных систем с использованием как структурного, так и объектно-ориентированного подхода (с использованием стандартного языка моделирования UML). Построение моделей и диаграмм UML выполняется с помощью CASE-средства Rational Rose.

Практикум состоит из трех глав. В главе 1 рассматривается применение структурного подхода к анализу и проектированию ПО (моделирование потоков данных, моделирование данных и комплексное использование моделей структурного подхода при проектировании систем). Глава 2 посвящена применению объектно-ориентированного подхода к проектированию ПО (на основе языка UML). Рассматриваются моделирование требований к системе (варианты использования), анализ и проектирование системы, комплексное использование моделей объектно-ориентированного подхода, работа с CASE-средством Rational Rose. В главе 3 дается оценка трудоемкости разработки ПО (метод функциональных точек).

Практикум подготовлен в соответствии с Государственным образовательным стандартом по специальности 351400 «Прикладная информатика в экономике», но может быть использован также студентами и преподавателями других специальностей, связанных с проектированием информационных систем и программного обеспечения, в частности 351500 «Математическое обеспечение и администрирование информационных систем» и 010200 «Прикладная математика и информатика». Он может быть полезен также системным аналитикам и разработчикам программного обеспечения.

Материал практикума используется в рамках курса «Объектно-ориентированный анализ и проектирование» на факультете вычислительной математики и кибернетики МГУ.

ПРИМЕНЕНИЕ СТРУКТУРНОГО ПОДХОДА К АНАЛИЗУ И ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. ОСНОВНЫЕ СВЕДЕНИЯ О СТРУКТУРНЫХ МЕТОДАХ АНАЛИЗА И ПРОЕКТИРОВАНИЯ

В структурном подходе к анализу и проектированию используются в основном две группы средств, описывающих функциональную структуру системы и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых являются:

- *DFD (Data Flow Diagrams)* - диаграммы потоков данных;
- *SADT (Structured Analysis and Design Technique - MeTOji.cpyK-турного анализа и проектирования)* - модели и соответствующие функциональные диаграммы;
- *ERD (Entity-Relationship Diagrams)* - диаграммы «сущность-связь».

Диаграммы потоков данных и диаграммы «сущность-связь» - наиболее часто используемые в CASE-средствах виды моделей.

Конкретный вид перечисленных диаграмм и интерпретация их конструкций зависят от стадии жизненного цикла ПО, на которой они применяются.

На стадии формирования требований к ПО SADT-модели и DFD используются, как правило, для моделирования бизнес-процессов (использование SADT-моделей ограничивается только данной стадией, поскольку они не предназначены для проектирования ПО). С помощью ERD выполняется описание данных на концептуальном уровне, не зависящем от средств реализации базы данных (СУБД).

На стадии анализа и проектирования ПО DFD используются для описания структуры проектируемой системы, при этом они могут уточняться, расширяться и дополняться новыми конструкциями. Аналогично ERD уточняются и дополняются новыми конструкциями, описывающими представление данных на логическом уровне, пригодном для последующей генерации схемы базы данных. Вышеперечисленные модели могут дополняться диаграммами, отражающими системную архитектуру ПО, структурные схемы программ, иерархию экранных форм и меню и др. Состав диаграмм в каждом конкретном случае зависит от сложности системы и необходимой полноты ее описания.

1.1.1. ОБЩИЕ СВЕДЕНИЯ О МОДЕЛИРОВАНИИ ПОТОКОВ ДАННЫХ (ПРОЦЕССОВ)

Диаграммы потоков данных являются основным средством моделирования функциональных требований к проектируемой системе. С их помощью эти требования представляются в виде иерархии функциональных компонентов (процессов), связанных потоками данных. Главная цель такого представления - продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить связи между этими процессами.

Для построения DFD традиционно используются две различные нотации, соответствующие методам Йордана и Гейна - Сэрсона. Эти нотации незначительно отличаются друг от друга графическим изображением символов. Далее при построении примеров будет использоваться нотация Гейна - Сэрсона.

В соответствии с данными методами модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы с внешними входами и выходами. Они детализируются с помощью диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию

диаграмм до тех пор, пока не будет достигнут уровень декомпозиции, на котором процессы становятся элементарными и детализировать их далее не имеет смысла.

Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те, в свою очередь, преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям - потребителям информации.

Основными компонентами диаграмм потоков данных являются:

- внешние сущности;
- системы и подсистемы;
- процессы;
- накопители данных;
- потоки данных.

Внешняя сущность - это материальный объект или физическое лицо, представляющее собой источник или приемник информации, например заказчики, персонал, поставщики, клиенты, склад. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что они находятся за границами анализируемой системы. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой системы, если это необходимо, или, наоборот, часть процессов может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

Внешняя сущность обозначается квадратом (рис. 1.1), расположенным как бы над диаграммой и бросающим на нее тень для того, чтобы можно было выделить этот символ среди других обозначений.



Рис. 1.1. Графическое изображение внешней сущности

Модель сложной системы может быть представлена на так называемой *контекстной диаграмме* в виде одной *системы* как единого целого либо может быть декомпозирована на ряд *подсистем* (рис. 1.2).

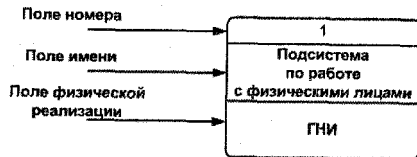


Рис. 1.2. Подсистема по работе с физическими лицами:
ГНИ - Государственная налоговая инспекция

Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: например, с помощью подразделения организации (отдела), выполняющего обработку входных документов и выпуск отчетов, программы, аппаратного устройства и т.д. Процесс на диаграмме потоков данных изображен на рис. 1.3.



Рис. 1.3. Графическое изображение процесса

Номер процесса служит для его идентификации в иерархии диаграмм. В поле имени вводится наименование процесса в виде предложения с активным однозначным глаголом в неопределенной форме (*вычислить, рассчитать, проверить, определить, создать, получить*), за которым следуют существительные в винительном падеже, например: «Ввести сведения о налогоплательщиках», «Выдать информацию о текущих расходах», «Проверить поступление денег».

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных. Это абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и спустя некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных (рис. 1.4) идентифицируется буквой «D» и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика.

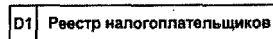


Рис. 1.4. Графическое изображение накопителя данных

Накопитель данных в общем случае является прообразом будущей базы данных, и описание хранящихся в нем данных должно соответствовать информационной модели (ERD).

Поток данных определяет информацию, передаваемую от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по коммуникационному каналу между двумя устройствами, пересылаемыми по почте письмами, магнитными носителями, переносимыми с одного компьютера на другой, и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рис. 1.5). Каждый поток данных имеет имя, отражающее его содержание.

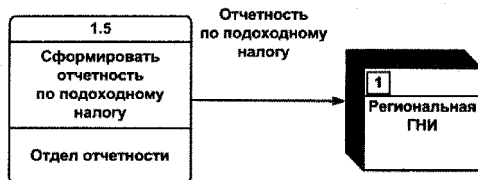


Рис. 1.5. Поток данных

Главная цель построения иерархии DFD заключается в том, чтобы сделать требования к системе 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:

- 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 процесс или два;
- не загромождать диаграммы не существенными на данном уровне деталями;
- декомпозицию потоков данных осуществлять параллельно с декомпозицией процессов. Эти две работы должны выполняться одновременно;
- выбирать ясные имена процессов и потоков, при этом не использовать аббревиатуры.

Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы. Перед построением контекстной DFD необходимо проана-

лизировать внешние события (внешние сущности), оказывающие влияние на функционирование системы. Количество потоков на контекстной диаграмме должно быть по возможности небольшим, поскольку каждый из них может быть в дальнейшем разбит на несколько потоков на следующих уровнях диаграммы.

Для проверки контекстной диаграммы можно составить список событий. Он должен состоять из описаний действий внешних сущностей (событий) и соответствующих реакций системы на события. Каждое событие должно соответствовать одному (или более) потоку данных: входные потоки интерпретируются как воздействия, а выходные потоки - как реакция системы на входные потоки.

Если для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги стандартного формата, и, кроме того, единственный главный процесс не раскрывает структуры такой системы. Признаками сложности (имеется в виду контекст) могут быть: наличие большого количества внешних сущностей (десять и более), распределенная природа системы, многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

Иерархия контекстных диаграмм определяет взаимодействие основных функциональных подсистем как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует система.

Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры системы на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в создании которых участвуют разные организации и коллективы разработчиков.

После построения контекстных диаграмм полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстных диаграммах, выполняется ее детализация с помощью DFD. Это можно сделать путем построения диаграммы для каждого события. Каждое событие представляется в виде процесса с соответствующими входными и выходными потоками, накопителями данных, внешними сущностями и ссылками на другие процессы для описания связей между этим процессом и его окружением.

Каждый процесс на DFD, в свою очередь, может быть детализирован с помощью DFD или спецификации (если процесс элементарный). При детализации должны выполняться следующие правила:

- правило балансировки - при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников или приемников данных может иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеют информационную связь детализируемые подсистема или процесс на родительской диаграмме;
- правило нумерации - при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т.д.

Спецификация процесса должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

Спецификация является терминальной вершиной иерархии DFD. Решение о завершении детализации процесса и использовании спецификации принимается аналитиком, который учитывает следующие критерии:

- наличие у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- возможность описания преобразования данных процессом в виде последовательного алгоритма;
- выполнение процессом единственной логической функции преобразования входной информации в выходную;
- возможность описания логики процесса с помощью спецификации небольшого объема (не более 20 - 30 строк).

Спецификации должны удовлетворять следующим требованиям:

- для каждого процесса нижнего уровня должна существовать одна и только одна спецификация;
- спецификация должна определять способ преобразования входных потоков в выходные;
- на стадии формирования требований нет необходимости определять метод реализации этого преобразования;
- спецификация должна стремиться к ограничению избыточности - не следует переопределять то, что уже было определено на диаграмме;
- набор конструкций для построения спецификации должен быть простым и понятным.

Фактически спецификации представляют собой описания алгоритмов задач, выполняемых процессами. Спецификации содержат номер и/или имя процесса, списки входных и выходных данных и тело (описание) процесса, являющееся спецификацией алгоритма или операции, трансформирующей входные потоки данных в выходные. Известно большое количество разнообразных методов, позволяющих описать тело процесса. Соответствующие этим методам языки могут варьироваться от структурированного естественного языка или псевдокода до визуальных языков проектирования.

Структурированный естественный язык применяется для читабельного, достаточно строгого описания спецификаций процессов. Он представляет собой разумное сочетание строгости языка программирования и читабельности естественного языка и состоит из подмножества слов, организованных в определенные логические структуры, арифметических выражений и диаграмм.

В состав языка входят следующие основные символы:

- глаголы, ориентированные на действие и применяемые к объектам;
- термины, определенные на любой стадии проекта ПО (например, задачи, процедуры, символы данных и т.п.);
- предлоги и союзы, используемые в логических отношениях;
- общеупотребительные математические, физические и технические термины;
- арифметические уравнения;
- таблицы, диаграммы, графы и т.п.;
- комментарии.

К управляющим структурам языка относятся последовательная конструкция, конструкция выбора и итерация (цикл).

При использовании структурированного естественного языка приняты следующие соглашения:

- логика процесса выражается в виде комбинации последовательных конструкций, конструкций выбора и итераций;
- глаголы должны быть активными, однозначными и ориентированными на целевое действие (*заполнить, вычислить, извлечь, а не модернизировать, обработать*);
- логика процесса должна быть выражена четко и однозначно.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается с помощью структур данных. Для каждого потока данных формируется список всех его элементов данных, затем элементы данных объединяются в структуры данных, соответствующие более крупным объектам данных (например, строкам документов или объектам предметной области). Каждый объект должен включать описания элементов, являющихся его атрибутами. Структуры данных могут содержать альтернативы, условные вхождения и итерации. *Условное вхождение* показывает, что данный компонент может отсутствовать в структуре (например, структура «данные о страховании» для объекта «служаший»). *Альтернатива* означает, что в структуру может входить один из перечисленных элементов. *Итерация* предусматривает вхождение любого числа элементов в указанном диапазоне (например, элемент «имя ребенка» для объекта «служаший»). Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для *непрерывных данных* могут указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для *дискретных данных* может указываться таблица допустимых значений.

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные недетализированные объекты следует детализировать, вернувшись к предыдущим шагам разработки. В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть записаны.

1.1.2. ОБЩИЕ СВЕДЕНИЯ О МОДЕЛИРОВАНИИ ДАННЫХ

Цель моделирования данных состоит в обеспечении разработчика системы 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), нотация которых была впервые введена Питером Ченом в 1976 г. Базовыми понятиями ERD являются:

Сущность (Entity) - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области.

Каждая сущность должна обладать *уникальным идентификатором*. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

- иметь уникальное имя; к одному и тому же имени должна всегда применяться одна и та же интерпретация; одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- обладать одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
- обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности.

Каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь (Relationship) - поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области, при которой каждый экземпляр одной сущности ассоциирован с произвольным (в том числе нулевым) количеством экземпляров другой сущности, и наоборот.

Атрибут (Attribute) - любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количествен-

ной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, предметов и т.д.). Экземпляр атрибута - это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. На диаграмме «сущность-связь» атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

1.1.3. ПОДХОД, ИСПОЛЗУЕМЫЙ В CASE-СРЕДСТВЕ SILVERRUN

В CASE-средстве Silverrun для концептуального моделирования данных (на стадии формирования требований) используется нотация, несколько отличающаяся от нотации Чена. На ERD-диаграмме сущность обозначается прямоугольником, содержащим, имя сущности (рис. 1.6), а связь в отличие от нотации Чена - не ромбом, а овалом, связанным линией с каждой из взаимодействующих сущностей. Числа над линиями означают степень и обязательность связи.



Рис. 1.6. Обозначение сущностей и связей

В данном примере пара (0,N) означает:

- физическое лицо может не иметь банковского счета (необязательная связь) либо иметь много счетов (степень связи - N);
- каждый банковский счет может принадлежать одному (обязательная связь) и только одному физическому лицу (степень связи - 1).

При описании атрибутов в верхней части прямоугольника располагается имя сущности, а в нижней части - список атрибутов, описывающих сущность. Обычно идентификаторы появляются в начале списка атрибутов. Пример графического представления сущности Юридическое лицо приведен на рис. 1.7.

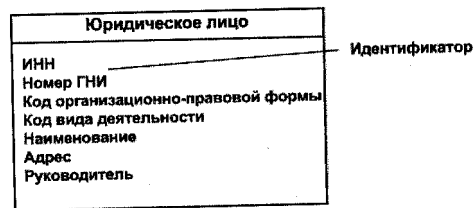


Рис. 1.7. Графическое представление сущности

Существуют следующие виды идентификаторов:

- *первичный/альтернативный*: сущность может иметь несколько идентификаторов. Один должен являться основным (первичным), а другие - альтернативными. Первичный идентификатор на диаграмме подчеркивается. Альтернативные идентификаторы предваряются символами <1> для первого альтернативного идентификатора, <2> для второго и т.д. В концептуальном моделировании данных различие первичных и альтернативных идентификаторов обычно не используется. В реляционной модели, полученной из концептуальной модели данных, первичные ключи используются в качестве внешних ключей. Альтернативные идентификаторы не копируются в качестве внешних ключей в другие таблицы;
- *простой/составной* (рис. 1.8): идентификатор, состоящий из одного атрибута, является простым, из нескольких атрибутов - составным;
- *абсолютный/относительный*: если все атрибуты, составляющие идентификатор, принадлежат сущности, то идентификатор является абсолютным. Если один или более атрибутов идентификатора принадлежат другой сущности, то идентификатор яв-



Рис. 1.8. Составной идентификатор

ляется относительным. Когда первичный идентификатор является относительным, сущность определяется как зависимая сущность, поскольку ее идентификатор зависит от другой сущности. На рис. 1.9 идентификатор сущности Строка_заказа является относительным. Он включает идентификатор сущности Заказ, что показано на рисунке подчеркиванием 1.

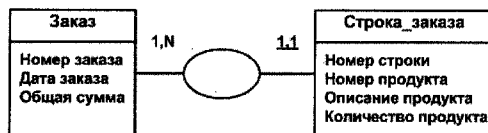


Рис. 1.9. Относительный идентификатор

Как и сущности, связи могут иметь атрибуты. На рис. 1.10 показан пример атрибутов связи. В этом примере Для того, чтобы найти оценку студента, нужно знать не только идентификатор студента, но и номер курса. Оценка не является атрибутом студента или атрибутом курса; она является атрибутом обеих этих сущностей. Это атрибут связи между студентом и курсом, которая в примере называется «регистрация».

Связь между сущностями в концептуальной модели данных является типом, который представляет множество экземпляров связи между экземплярами сущностей. Для того чтобы идентифицировать

Определенный экземпляр сущности, используется идентификатор сущности. Точно так же для определения экземпляров связи между сущностями требуется идентификатор связи. Так, в примере на рис. 1.10 идентификатором отношения «регистрация» являются идентификатор студента и номер курса, поскольку вместе они определяют конкретный экземпляр связи студентов и курсов.

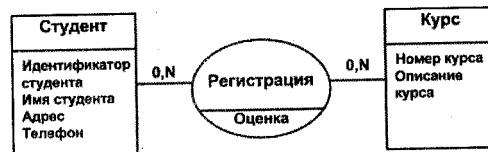


Рис. 1.10. Связь с атрибутами

В связи «супертип-подтип» (рис. 1.11) общие атрибуты типа определяются в сущности-супертипе, сущность-подтип наследует все атрибуты супертипа. Экземпляр подтипа существует только при условии существования определенного экземпляра супертипа. Подтип не может иметь идентификатора (он импортирует его из супертипа).

В дальнейшем в процессе проектирования базы данных концептуальная модель данных преобразуется в реляционную модель, для описания которой используется отдельная графическая нотация. Каждая конструкция концептуальной модели преобразуется в таблицы или колонки таблиц, являющиеся двумя основными конструкциями реляционных баз данных.

Основным различием между реляционной и концептуальной моделями является представление связи: в концептуальной модели связь может соединять любое количество сущностей, а в реляционной модели связь является либо унарной, либо бинарной (она не может связывать больше двух различных таблиц).

Покажем на примере, как строится концептуальная модель данных системы учета персонала предприятия. Допустим, что предприятие состоит из отделов, в которых работают сотрудни-

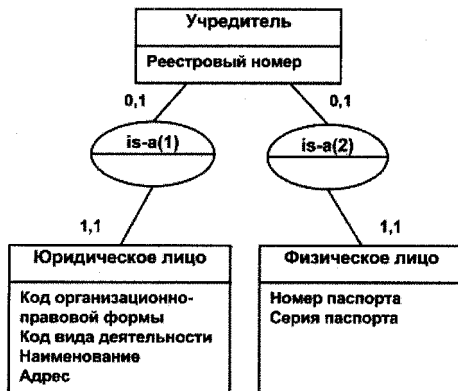


Рис. 1.11. Связь «супертип-подтип»

ки. Оклад каждого сотрудника зависит от занимаемой им должности. Далее предположим, что на предприятии разрешено совмещение должностей, т.е. каждый сотрудник может иметь более чем одну должность (и работать более чем в одном отделе), причем может занимать неполную ставку. В то же время одну и ту же должность могут занимать одновременно несколько сотрудников.

На основе данной информации можно выделить следующие сущности с соответствующими атрибутами:

- ОТДЕЛ (Номер_отдела);
- СОТРУДНИК (Табельный_номер, ФИО);
- ДОЛЖНОСТЬ (Наименование_должности, Оклад)

и связь РАБОТАЕТ_В между ними с атрибутом Ставка, определяющим, какую часть должностного оклада получает данный сотрудник. Таким образом получаем следующую диаграмму (рис. 1.12).

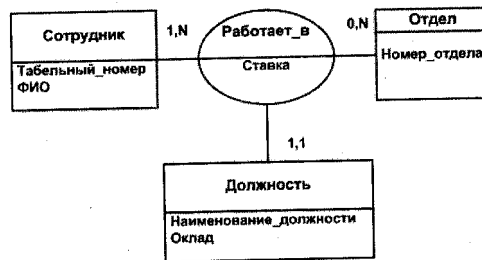


Рис. 1.12. Начальная диаграмма

Тернарная связь, изображенная на диаграмме, наиболее адекватно отражает информацию о предметной области. Действительно, она однозначно отображает тот факт, что оклад сотрудника зависит от его должности, отдела, где он работает, и ставки. Однако в этом случае возникают определенные проблемы с определением степени связи. Хотя, как было сказано, каждый работник может занимать несколько должностей, а в штате каждого отдела существуют вакансии с различными должностями, тем не менее класс принадлежности сущности ДОЛЖНОСТЬ на приведенном рисунке установлен в (1,1). Это объясняется тем, что ДОЛЖНОСТЬ ассоциируется фактически не с сущностями СОТРУДНИК и ОТДЕЛ, а со связью между ними.

Попытаемся отобразить ассоциации сотрудников, отделов и должностей с помощью бинарных связей.

В этом случае для адекватного описания семантики предметной области необходимо ввести еще одну сущность ШТАТНАЯ_ЕДИНИЦА, которая фактически заменяет собой связь РАБОТАЕТ_В и поэтому имеет атрибут *штатная ставка* (рис. 1.13).

Дополним модель некоторыми сущностями, которые будут полезны при моделировании данных рассматриваемого предприятия. Эти сущности должны быть связаны с заказчиками, контрактами и рабочими группами. С одним заказчиком может быть заключено более одного контракта, каждый контракт заключен с конкретным заказчиком, а каждый заказчик имеет хотя бы один контракт. Для

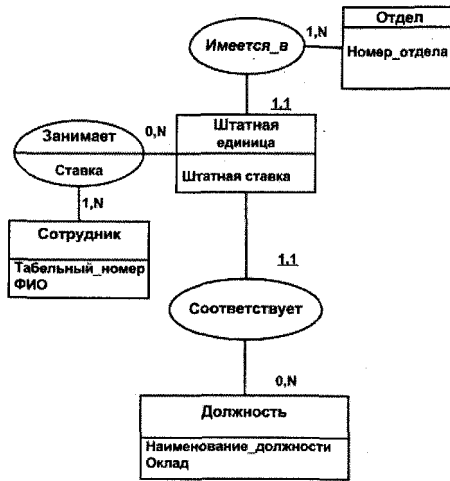
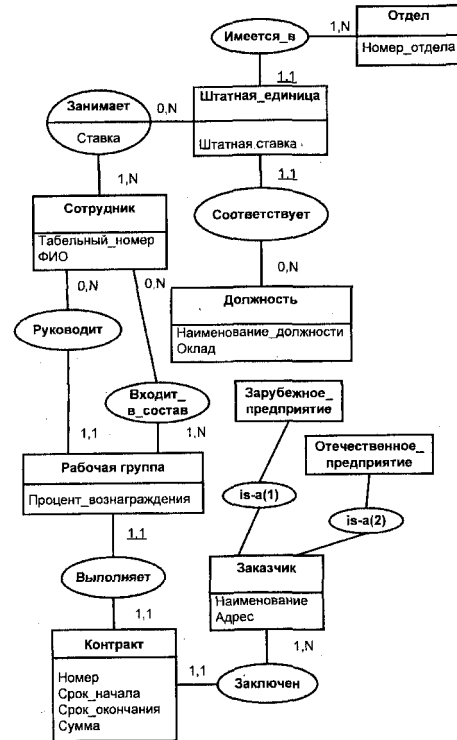


Рис. 1.13. Диаграмма с бинарными связями

Обобщая все приведенные выше рассуждения, получим новую диаграмму «сущность-связь» (рис. 1.14).



выполнения каждого контракта создается рабочая группа, в которую входят сотрудники разных отделов. Каждый сотрудник может входить в несколько (в том числе и ни в одну) рабочих групп, а каждая группа должна включать не менее одного сотрудника. Рабочая группа характеризуется процентом вознаграждения, который отражает долю стоимости контракта, предназначенную для оплаты труда членов соответствующей рабочей группы. Рабочая группа создается только после того, как будет подписан контракт с заказчиком, и прекращает свое существование по выполнению контракта. Как правило, один из членов рабочей группы является руководителем по отношению к другим сотрудникам, входящим в ее состав. Что касается заказчиков, следует различать национальную принадлежность юридических лиц, с которыми предприятие вступает в договорные отношения (отечественные компании и зарубежные фирмы).

В заключение ответим на следующие вопросы:

1. Как изменится диаграмма «сущность-связь» в том случае, если процент вознаграждения по всем контрактам будет одинаков? *Отпадает необходимость в сущности РАБОЧАЯ_ГРУППА. Все ее связи перейдут к сущности КОНТРАКТ.*

2. Что изменится в диаграмме, если будет запрещено совместительство должностей, т.е. каждый сотрудник будет иметь право занимать только одну должность со ставкой 1? *Связь «работает_в» не будет иметь атрибутов. При декомпозиции ее на бинарные связи получим сущность ШТАТНАЯ_ЕДИНИЦА, также не имеющую атрибутов.*

1.1.4. ОБЩИЕ СВЕДЕНИЯ О ФУНКЦИОНАЛЬНЫХ МОДЕЛЯХ, ИСПОЛЗУЕМЫХ НА СТАДИИ ПРОЕКТИРОВАНИЯ

Функциональные модели, используемые на *стадии проектирования ПО*, предназначены для описания функциональной структуры проектируемой системы. Построенные ранее DFD при этом уточняются, расширяются и дополняются новыми конструкциями. Помимо DFD могут использоваться и другие диаграммы, отражающие системную архитектуру ПО, иерархию экранных форм и меню, структурные схемы программ (структурные карты) и т.д. Состав диаграмм и степень их детализации определяются необходимой полнотой описания системы для непосредственного перехода к ее последующей реализации (программированию).

Так, переход от модели бизнес-процессов, представленной в виде DFD, к *модели системных процессов* может происходить следующим образом:

- внешние сущности на контекстной диаграмме заменяются или дополняются техническими устройствами (например, рабочими станциями, принтерами и т.д.);

- для каждого потока данных определяется, посредством каких технических устройств информация передается или производится;
- процессы на диаграмме нулевого уровня заменяются соответствующими процессорами - обрабатывающими устройствами (процессорами могут быть как технические устройства - ПК конечных пользователей, рабочие станции, серверы баз данных, так и служащие-исполнители);
- определяется и изображается на диаграмме тип связи между процессорами (например, локальная сеть - LAN - Local Area Network);
- определяются задачи для каждого процессора (приложения, необходимые для работы системы), для них строятся соответствующие диаграммы. Определяется тип связи между задачами;
- устанавливаются ссылки между задачами и процессами диаграмм потоков данных следующих уровней.

Иерархия экранных форм моделируется с помощью *диаграмм последовательностей экранных форм*. Совокупность таких диаграмм представляет собой абстрактную модель пользовательского интерфейса системы, отражающую последовательность появления экранных форм в приложении. Построение диаграмм последовательностей экранных форм выполняется следующим образом:

- на DFD выбираются интерактивные процессы нижнего уровня. Интерактивные процессы нуждаются в пользовательском интерфейсе, поэтому можно определить экранную форму для каждого такого процесса;
- диаграмма изображается в виде прямоугольника для каждого интерактивного процесса на нижнем уровне диаграммы;
- определяется структура меню. Для этого интерактивные процессы группируются в меню (либо так же, как в модели процессов, либо другим способом -- по функциональным признакам или в зависимости от принадлежности к определенным объектам);
- формы с меню изображаются над формами, соответствующими интерактивным процессам, и соединяются с ними переходами в виде стрелок, направленных от меню к формам;
- определяется верхняя форма (главная форма приложения), связывающая все формы с меню.

1.2.
**ВЫПОЛНЕНИЕ УЧЕБНОГО
ПРОЕКТА**

1.2.1.
ПОСТАНОВКА ЗАДАЧИ

Администрация больницы заказала разработку информационной системы для отдела приема пациентов и медицинского секретариата. Новая система предназначена для обработки данных о врачах, пациентах, приеме пациентов и лечении. Система должна выдавать отчеты по запросу врачей или администрации.

Во время пред проектного обследования составлено следующее описание деятельности рассматриваемых подразделений.

Перед приемом в больницу проводится встреча пациента и врача. Врач сообщает в отдел приема пациентов об ожидаемом приеме больного и передает данные о нем. Пациент может быть принят в больницу более чем один раз, но если пациент ранее не лечился в больнице, то ему присваивается регистрационный номер и записываются его данные (фамилия, имя и отчество, адрес и дата рождения). Пациент должен быть зарегистрирован в системе до приема в больницу.

Спустя некоторое время врач оформляет в отделе приема пациентов прием больного. При этом определяется порядковый номер приема и запоминаются данные приема пациента. После этого отдел приема посылает сообщение врачу для подтверждения приема больного. В это сообщение включаются регистрационный номер пациента и его фамилия, порядковый номер приема, дата начала лечения и номер палаты.

В день приема пациент сообщает в отдел приема о своем прибытии и передает данные о себе (или изменения в данных). Отдел приема проверяет и при необходимости корректирует данные о пациенте. Если пациент не помнит свой регистрационный номер, то, выполняется соответствующий запрос. После регистрации пациент получает регистрационную карту, содержащую ФИО пациента, адрес, дату рождения, номер телефона, группу крови, название страховой компании и номер страховки.

Во время пребывания в больнице пациент может лечиться у нескольких врачей; каждый врач назначает один или более курсов лечения, но каждый курс лечения назначается только одним врачом. Данные о курсах лечения передаются в медицинский секретариат, который занимается координацией лечения пациентов, регистрируются и хранятся там. Данные включают номер врача, номер пациента, порядковый номер приема, название курса лечения, дату назначения, время и примечания.

При необходимости врач запрашивает в медицинском секретариате историю болезни пациента, содержащую данные о курсах лечения, полученных пациентом. История болезни представляется в виде табл. 1.1.

Таблица 1.1

История болезни пациента			
История болезни		Дата: 01-02-1998	
Данные с 01-01-1992 по 01-02-1998			
Номер пациента: 732	ФИО: Иванов С.П.	Дата рождения: 15-07-55	
Номер приема: 649	Дата приема: 06-07-1992	Палата: 4	
Заболевание: перелом ноги		Врач: Петров П.С., хирург	
	Дата	Время	Примечание
	06-07-1992	12.00	Наложен гипс
	20-07-1992	14.00	Состояние хорошее
	22-07-1992	10.00	Лечение закончено
Заболевание: ОРЗ		Врач: Иванов А.С., терапевт	
	Дата	Время	Примечание
	10-07-1992	10.00	Назначена терапия
	15-07-1992	10.00	Состояние удовлетворительное
	20-07-1992	10.00	Состояние хорошее
	25-07-1992	10.00	Выписан

Продолжение

История болезни		Дата: 01-02-1998	
Номер приема: 711	Дата приема: 12-04-1993'	Палата: 8	
Заболевание: сердечный приступ		Врач: Сидоров И.И., кардиолог	
	Дата	Время	Примечание

После окончания курсов лечения лечащий врач принимает решение о выписке пациента. Когда пациент выписывается, он сообщает об этом в отдел приема. Отдел приема регистрирует данные о выписке, включая дату выписки, и дает пациенту справку об окончании пребывания в больнице.

Врач передает данные о себе и изменения в данных в отдел приема. Данные о враче включают номер врача, ФИО, адрес, дату рождения, номер домашнего телефона, специализацию, номер кабинета, номер рабочего телефона.

Администрация больницы может запросить отчет о пациентах. В запросе указывается интервал времени (начальная и конечная даты). Отчет должен быть представлен в следующей форме (табл. 1.2).

Таблица 1.2

Отчет о пациентах						
Отчет о пациентах						
Данные с 01-12-1997 по 01-02-1998						
Номер	ФИО	Адрес	Телефон	Дата рождения	Страховая компания	страховки

Администрация больницы запрашивает также обзор заболеваний за последнюю неделю каждый понедельник в 9.00.

1.2.2. ОПИСАНИЕ КОНТЕКСТА СИСТЕМЫ ПРИЕМА ПАЦИЕНТОВ В БОЛЬНИЦЕ И ПОСТРОЕНИЕ НАЧАЛЬНОЙ КОНТЕКСТНОЙ ДИАГРАММЫ

Построим начальную контекстную диаграмму потоков данных в нотации Гейна - Сэрсона (рис. 1.15). Нарисуем нулевой процесс и присвоим ему имя системы (Система приема пациентов). Поскольку моделируется деятельность отдела приема пациентов и медицинского секретариата, внешними сущностями являются Врач, Пациент и Администрация больницы. Нарисуем внешние сущности и соединим их с нулевым процессом посредством потоков данных. Потоки данных соответствуют документам, запросам или сообщениям, которыми внешние сущности обмениваются с системой.

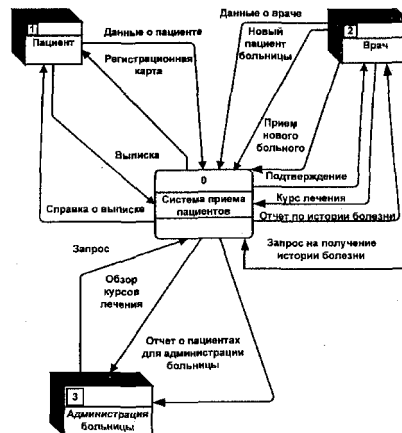


Рис. 1.15. Начальная контекстная диаграмма

1.2.3. СПЕЦИФИКАЦИЯ СТРУКТУР ДАННЫХ

Определим состав потоков данных и подготовим исходную информацию для конструирования концептуальной модели данных,

1. Пометим символом «*» все структуры и элементы данных типа «итерация», которые могут иметь повторяющиеся значения.

2. Пометим символом «^» все структуры и элементы данных типа «условное вхождение», присутствие которых является необязательным.

3. Пометим символом «|» все структуры и элементы данных типа «альтернатива», если в соответствующую структуру может входить только один из перечисленных элементов.

4. Повторим действия 2-5, объединяя структуры и элементы данных в более крупные структуры. В результате для каждого потока данных должна быть сформирована иерархическая (древовидная) структура, конечные элементы (листья) которой являются элементами данных, узлы дерева являются структурами данных, а верхний узел дерева соответствует потоку данных в целом. Результат можно представить в виде текстового описания, подобного описанию структур данных в языках программирования.

Примеры спецификации структур данных:

ОТЧЕТ_ПО_ИСТОРИИ_БОЛЕЗНИ

Текущая_дата
Начальная_дата
Конечная_дата

СТРОКА_ПАЦИЕНТА

Номер_пациента
Фино

Дата_рождения

СТРОКА_ПРИЕМА*

Номер_приема /* порядковый номер приема нового больного */

Дата_приема

Номер_палаты

СТРОКА_КУРСА_ЛЕЧЕНИЯ*

Наименование заболевания

Имя_врача
Специализация
СТРОКА_РЕЗУЛЬТАТОВ_ЛЕЧЕНИЯ*
Дата
Время
Примечание

ИНФОРМАЦИЯ_ОТ_ВРАЧА

Прием_нового_больного
Данные_о_враче
Новый_пациент_больницы
Запрос_на_получение_истории_болезни
Курс_лечения

ОТЧЕТ_О_ПАЦИЕНТАХ

Текущая_дата
Начальная_дата
Конечная_дата

СТРОКА_ПАЦИЕНТА*

Номер_пациента
Фино

Адрес

Телефон

Дата_рождения

СТРАХОВАНИЕ*

Страховая_компания
Номер_страховки

1.2.4. ПОСТРОЕНИЕ НАЧАЛЬНОГО ВАРИАНТА КОНЦЕПТУАЛЬНОЙ МОДЕЛИ ДАННЫХ

Используя нотацию CASE-средства Silverrun, выделим и нарисует сущности для каждого объекта данных в системе приема пациентов. Рассмотрим каждую возможную пару сущностей и установим существование связи между ними. Связь долж-

на отражать наличие взаимодействия между сущностями, причем в системе должна сохраняться информация об этом взаимодействии.

Нарисуем диаграмму «сущность-связь». Присвоим наименование каждой связи и зададим ее характеристики (степень связи и обязательность) (рис. 1.16).

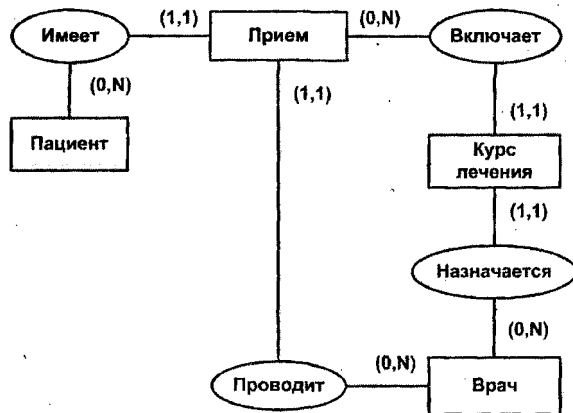


Рис. 1.16. Начальный вариант концептуальной модели данных

Предлагается ответить самостоятельно на следующие вопросы:

1. Почему степень связи между Пациентом и Приемом, Приемом и Курсом лечения, Врачом и Курсом лечения равна 0, а не 1?
2. Почему отсутствует связь между Врачом и Пациентом?

1.2.5. ПОСТРОЕНИЕ ДИАГРАММ ПОТОКОВ ДАННЫХ НУЛЕВОГО И ПОСЛЕДУЮЩИХ УРОВНЕЙ

Детализируем начальную контекстную диаграмму для завершения анализа функционального аспекта поведения системы.

1. Декомпозируем начальную контекстную диаграмму. Для этого можно построить диаграмму для каждого события. Поставим каждому событию в соответствие процесс, нарисует входные и выходные потоки, накопители данных, внешние сущности и ссылки на другие процессы для описания связей между этим процессом и его окружением.

2. Сведем все построенные диаграммы в одну диаграмму нулевого уровня.

3. Проверим соответствие между контекстной диаграммой и диаграммой нулевого уровня (каждый поток данных между системой и внешней сущностью на диаграмме нулевого уровня должен быть представлен и на контекстной диаграмме).

4. Разделим процессы на группы, которые имеют много общего (работают с одинаковыми данными и/или имеют сходные функции). Нарисуем их вместе на диаграмме более низкого (первого) уровня, а на диаграмме нулевого уровня объединим в один процесс. Накопители данных, используемые процессами из одной группы, перенесем из нулевого уровня на первый, а из нулевого уровня удалим и заменим одной базой данных.

5. Декомпозируем сложные процессы и проверим соответствие различных уровней модели процессов.

6. Опишем накопители данных посредством структур данных. 7. Опишем процессы нижнего уровня посредством спецификаций.

Результаты представлены на рис. 1.17-1.20.

Описание накопителей данных приведено ниже.

Накопитель данных: пациенты

Номер_пациента
 Фамилия
 Адрес
 Телефон
 Дата_рождения
 Группа_крови
 Страхование

Накопитель данных: приемы

Номер_приема
 Номер_пациента
 Дата_приема
 Дата_выписки
 Номер_палаты

Накопитель данных: курсы лечения

Номер_приема
 Номер_специалиста
 Наименование_заболевания

ПРИМЕЧАНИЕ

Дата
 Время
 Примечание

Накопитель данных: специалисты

Номер_специалиста
 Имя_специалиста
 Адрес
 Телефон
 Дата_рождения
 Специализация
 Номер_кабинета
 Рабочий_телефон

Мини-спецификация процесса: Зарегистрировать пациента

BEGIN

GET Фино и Дата_рождения

FIND пациент

IF пациент найден THEN

DISPLAY данные пациента

ELSE

GET Адрес

DETERMINE Номер_пациента

INSERT пациент

ENDIF

PRINT подтверждение

END

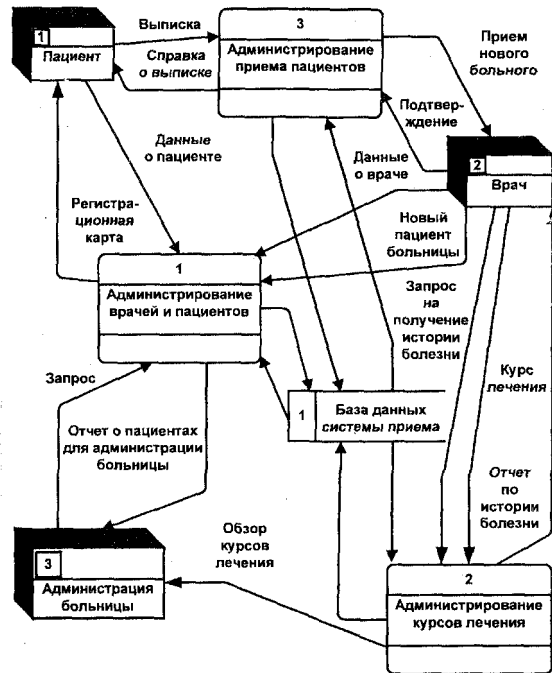


Рис. 1.17. Диаграмма потоков данных
нулевого уровня

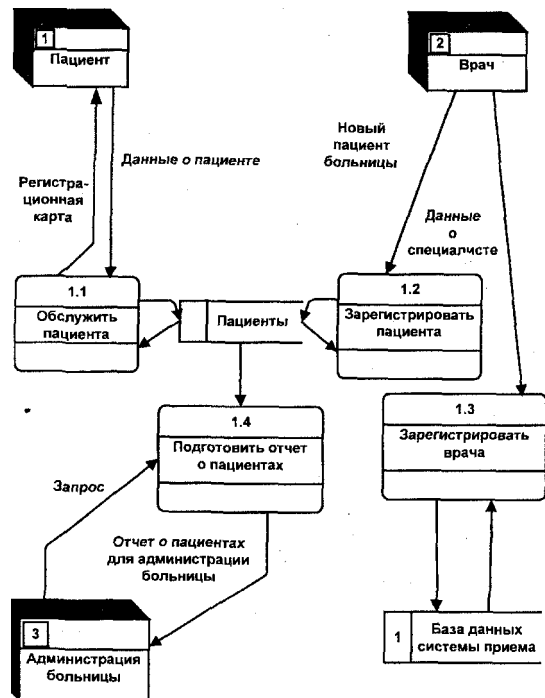


Рис. 1.18. Диаграмма потоков данных первого уровня для процесса 1

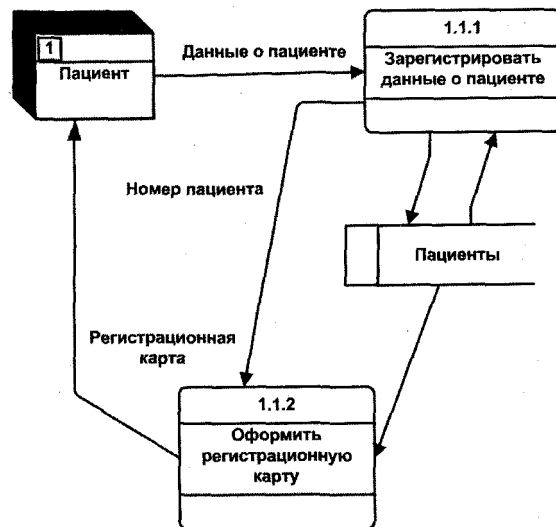


Рис. 1.19. Диаграмма потоков данных второго уровня для процесса 1.1

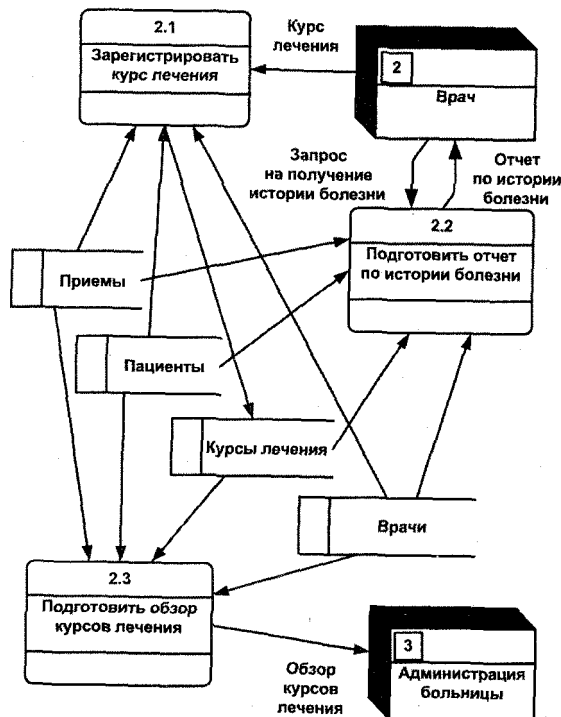


Рис. 1.20. Диаграмма потоков данных первого уровня для процесса 2

1.2.6. УТОЧНЕНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ ДАННЫХ

Определим атрибуты сущностей и уточним построенную модель данных.

1. Используя построенные ранее структуры данных, уточним атрибуты каждой сущности и нарисуем их на ER-диаграмме. Внешние ключи можно не показывать, поскольку они определяются связями между сущностями.

2. Выделим атрибуты-идентификаторы и подчеркнем их.

3. Проверим связи, выделим (при необходимости) зависимые от идентификатора сущности и связи «супертип-подтип».

4. Проверим соответствие между описанием структур данных и концептуальной моделью (все элементы данных должны присутствовать на диаграмме в качестве атрибутов).

Результат представлен на рис. 1.21.

1.2.7. ПОСТРОЕНИЕ ДИАГРАММ СИСТЕМНЫХ ПРОЦЕССОВ И ДИАГРАММ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ЭКРАННЫХ ФОРМ

Результаты перехода от модели бизнес-процессов к модели системных процессов представлены на рис. 1.22 - 1.23. На диаграмме системных процессов первого уровня вместо отдельных Процессов введены процессоры - компьютеры, на которых выполняются соответствующие процессы.

Результаты построения абстрактной модели пользовательского интерфейса системы, отражающей последовательность появления экранных форм в приложении, представлены на рис. 1.24.

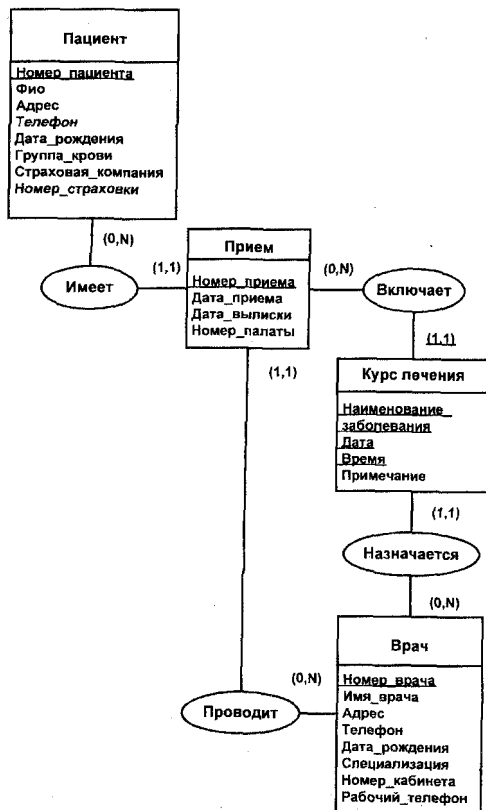


Рис. 1.21. Уточненный вариант концептуальной модели данных

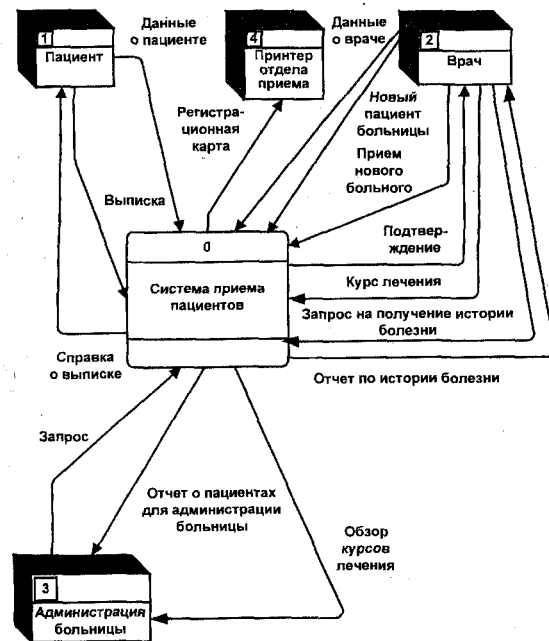


Рис. 1.22. Диаграмма системных процессов нулевого уровня

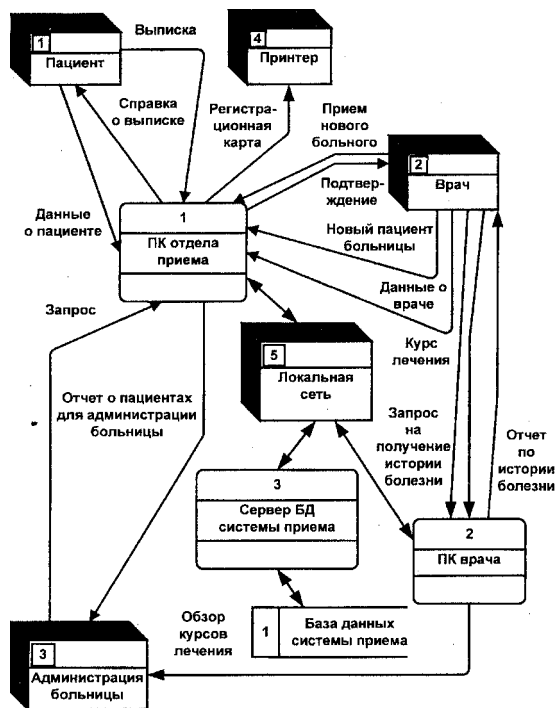


Рис. 1.23. Диаграмма системных процессов первого уровня

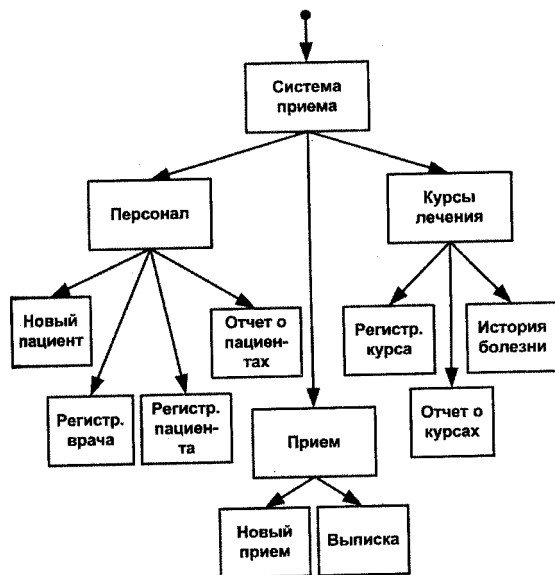


Рис. 1.24. Диаграмма последовательности экранных форм

1.3. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА

Задание 1. Построить концептуальную модель данных для информационной системы колледжа

В колледже работают N преподавателей. О каждом преподавателе известна следующая информация: фамилия, имя, отчество, год рождения, пол, образование, учебное заведение, которое он окончил (предполагается, что каждый из них окончил не более одного специального учебного заведения; если он окончил среднее и высшее учебное заведение, то фиксируется информация только о последнем из них), специальность, иностранные языки, которыми владеет преподаватель, и степень владения ими, адрес, информация о детях (ФИО, год рождения).

Каждый преподаватель может вести один или несколько предметов. За каждой группой студентов закреплен один руководитель.

Имеется расписание занятий, в котором зафиксировано, в какое время, в какой день недели, в какой аудитории, какая группа занимается, каким предметом и какой преподаватель его ведет.

Некоторые преподаватели ведут специальные семинары. Каждый преподаватель может вести несколько семинаров. Имеется расписание работы семинаров. Известно, кто из студентов посещает каждый семинар.

В колледже работают спортивные секции. Занятия в каждой из секций ведут несколько тренеров. Занятия в секциях ведутся по группам. Каждый студент может заниматься в одной или нескольких секциях. В каждой из этих секций он записан в определенную группу. Каждая группа закреплена за одним определенным тренером.

Имеется расписание работы секций, в котором указано, какая группа, в какое время, с каким тренером занимается, а также место проведения занятий. Каждый тренер для каждой группы ведет журнал посещения занятий ее членами.

Задание 2. Построить диаграммы потоков данных и концептуальную модель данных для системы начисления зарплаты

В компании поставлена задача создания новой системы начисления зарплаты взамен морально устаревшей существующей системы. Новая система должна предоставлять служащим возможность записывать электронным способом информацию из карточки учета рабочего времени и автоматически формировать чеки на оплату, учитывающие количество отработанных часов и общий Объем продаж (для служащих, получающих комиссионное вознаграждение). В системе должна храниться информация обо всех служащих компании.

Часть служащих получает почасовую оплату. Она начисляется на основе карточек учета рабочего времени, каждая из которых содержит дату и количество часов, отработанных в соответствии с конкретным тарифом. Если какой-либо служащий отработал в день более 8 часов, сверхурочное время оплачивается с Коэффициентом 1,5. Служащие-почасовики получают зарплату Каждую пятницу.

Часть служащих получает фиксированный оклад, однако они тоже представляют свои карточки учета рабочего времени. Благодаря этому система может вести учет количества часов, отработанных в соответствии с конкретными тарифами. Такие служащие получают зарплату в последний рабочий день месяца.

Некоторые из служащих с фиксированным окладом также получают комиссионное вознаграждение, учитывающее объем продаж. Они представляют заказы на поставку, отражающие дату и объем продаж. Процент комиссионного вознаграждения определяется индивидуально для каждого служащего и может составлять 10, 15, 25 или 35%.

Одной из наиболее часто используемых возможностей новой системы является формирование различных отчетов: запросить количество отработанных часов, суммарную зарплату, оставшееся время отпуска и т.д.

Служащие вправе выбирать способ оплаты за работу. Они могут получать свои чеки на оплату по почте, на счет в банке или на руки в офисе.

Администратор системы курирует информацию о служащих. В его обязанности входят ввод данных о новых служащих, удаление данных и изменение любой информации о служащем, такой, как имя, адрес и способ оплаты, а также формирование различных отчетов для руководства.

Приложение Начисление зарплаты запускается автоматически каждую пятницу и в последний рабочий день месяца, рассчитывая в эти дни зарплату соответствующих служащих. Начисление зарплаты должно производиться автоматически, без ручной обработки.

Система должна взаимодействовать с существующей банковской системой посредством интерфейса электронных транзакций.

ПРИМЕНЕНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА К АНАЛИЗУ И ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1. ОСНОВНЫЕ СВЕДЕНИЯ О ЯЗЫКЕ UML 2.1.1. СРЕДСТВА UML

Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов. Стандарт UML версии 1.1, принятый OMG в 1997 г., предлагает следующий набор диаграмм для моделирования:

- *диаграммы вариантов использования (use case diagrams)* - для моделирования бизнес-процессов организации и требований к создаваемой системе;
- *диаграммы классов (class diagrams)* - для моделирования статической структуры классов системы и связей между ними;
- *диаграммы поведения системы (behavior diagrams)*;
- *диаграммы взаимодействия (interaction diagrams)*;
- *диаграммы последовательности (sequence diagrams)* и *кооперативные диаграммы (collaboration diagrams)* - для моделирования процесса обмена сообщениями между объектами;
- *диаграммы состояний (statechart diagrams)* - для моделирования поведения объектов системы при переходе из одного состояния в другое;

- *диаграммы деятельности (activity diagrams)* - для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельности;
- *диаграммы реализации (implementation diagrams)*;
- *диаграммы компонентов (component diagrams)* - для моделирования иерархии компонентов (подсистем) системы;
- *диаграммы размещения (deployment diagrams)* - для моделирования физической архитектуры системы.

2.1.2. ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Действующее лицо (actor) - это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования.

Действующие лица делятся на три основных типа - пользователи системы, другие системы, взаимодействующие с данной, и время. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Для наглядного представления вариантов использования в качестве основных элементов процесса разработки ПО применяются диаграммы вариантов использования. На рис. 2.1 показан пример такой диаграммы для банковской системы с банкоматами.

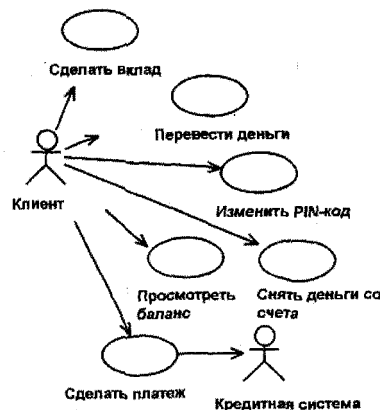


Рис. 2.1. Пример диаграммы вариантов использования

На данной диаграмме человеческие фигурки обозначают действующих лиц, овалы - варианты использования, а линии и стрелки - различные связи между действующими лицами и вариантами использования.

На этой диаграмме показаны два действующих лица: клиент и кредитная система. Существуют также шесть основных действий, выполняемых моделируемой системой: перевести деньги, сделать вклад, снять деньги со счета, просмотреть баланс, изменить PIN-код и сделать платеж.

На диаграмме вариантов использования показано взаимодействие между вариантами использования и действующими лицами. Она отражает требования к системе с точки зрения пользователя. Таким образом, варианты использования - это функции, выполняемые системой, а действующие лица - это заинтересованные лица (stakeholders) по отношению к создаваемой системе. Такие диаграммы показывают, какие действующие лица инициируют варианты использования. Из них также видно, когда

действующее лицо получает информацию от варианта использования. Данная диаграмма, например, отражает взаимодействие между вариантами использования и действующими лицами банковской системы. В сущности, диаграмма вариантов использования иллюстрирует требования к системе. В нашем примере клиент банка инициирует большое количество различных вариантов использования: «Снять деньги со счета», «Перевести деньги», «Сделать вклад», «Просмотреть баланс» и «Изменить PIN-код». От варианта использования «Сделать платеж» стрелка указывает на «Кредитную систему». Действующими лицами могут быть внешние системы, и потому в данном случае «Кредитная система» показана именно как действующее лицо - она внешняя для банковской системы. Направленная от варианта использования к действующему лицу стрелка показывает, что вариант использования предоставляет некоторую информацию, используемую действующим лицом. В данном случае вариант использования «Сделать платеж» предоставляет «Кредитной системе» информацию об оплате по кредитной карточке.

Все варианты использования так или иначе связаны с внешними требованиями к функциональности системы. Варианты использования всегда следует анализировать вместе с действующими лицами системы, определяя при этом реальные задачи пользователей и рассматривая альтернативные способы решения этих задач.

Действующие лица могут играть различные роли по отношению к варианту использования. Они могут пользоваться его результатами или могут сами непосредственно в нем участвовать. Значимость различных ролей действующего лица зависит от того, каким образом используются его связи.

Конкретная цель диаграмм вариантов использования - это документирование вариантов использования (все, входящее в сферу применения системы), действующих лиц (все вне этой сферы) и связей между ними. Разрабатывая диаграммы вариантов использования, старайтесь придерживаться следующих правил:

- * не моделируйте связи между действующими лицами. По определению действующие лица *находятся* вне сферы действия системы. Это означает, что связи между ними также не относятся к ее компетенции;

- * не соединяйте стрелкой два варианта использования непосредственно. Диаграммы данного типа описывают только, какие варианты использования доступны системе, а не порядок их вы-

полнения. Для отображения порядка выполнения вариантов использования применяют диаграммы деятельности;

- * каждый вариант использования должен быть иницирован действующим лицом. Это означает, что всегда имеется стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Хорошим источником для идентификации вариантов использования служат внешние события. Следует начать с перечисления всех событий, происходящих во внешнем мире, на которые система должна каким-то образом реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать чисто Пользовательскую реакцию. Идентификация событий, на которые необходимо реагировать, помогает идентифицировать варианты использования.

Варианты использования начинают описывать, что должна Система делать. Для того чтобы фактически разработать систему, однако потребуются более конкретные детали. Эти детали описываются в документе, называемом «поток событий» (flow of events). Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы и что - сама система.

Хотя поток событий и описывается подробно, он также не должен зависеть от реализации. Цель - описать то, что будет делать система, а не как она будет делать это. Обычно поток событий включает:

- краткое описание;
- предусловия (pre-conditions);
- основной поток событий;
- альтернативный поток событий;
- постусловия (post-conditions).

Последовательно рассмотрим эти составные части.

Краткое описание

Каждый вариант использования должен иметь краткое описание того, что он будет делать. Например, вариант использования «Перевести деньги» может содержать следующее описание:

Вариант использования «Перевести деньги» позволяет клиенту или служащему банка переводить деньги с одного счета на другое требование или сберегательного счета на другой.

Предусловия

Предусловия варианта использования - это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет выполняться сам. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска этого. Не у всех вариантов использования бывают предусловия.

Ранее упоминалось, что диаграммы вариантов использования не должны отражать порядок их выполнения. С помощью предусловий, однако, можно документировать и такую информацию. Так, предусловием одного варианта использования может быть то, что в это время должен выполняться другой.

Основной и альтернативный потоки событий

Конкретные детали вариантов использования описываются в основном и альтернативных потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, что будет делать система, а не как она будет делать это, причем описывает все это с точки зрения пользователя. Основной и альтернативный потоки событий включают следующее описание:

- каким образом запускается вариант использования;
- различные пути выполнения варианта использования;
- нормальный, или основной, поток событий варианта использования;
- отклонения от основного потока событий (так называемые альтернативные потоки);
- потоки ошибок;
- каким образом завершается вариант использования.

Например, поток событий варианта использования «Снять деньги со счета» может выглядеть следующим образом:

Основной поток

1. Вариант использования начинается, когда клиент вставляет свою карточку в банкомат.
2. Банкомат выводит приветствие и предлагает клиенту ввести свой персональный PIN-код.
3. Клиент вводит PIN-код.
4. Банкомат подтверждает введенный код. Если код не подтвержден, выполняется альтернативный поток событий A1.
5. Банкомат выводит список доступных действий:
 - сделать вклад;
 - снять деньги со счета;
 - перевести деньги.
6. Клиент выбирает пункт «Снять деньги со счета».
7. Банкомат запрашивает, сколько денег надо снять.
8. Клиент вводит требуемую сумму.
9. Банкомат определяет, имеется ли на счете достаточно денег. Если денег недостаточно, выполняется альтернативный поток A2. Если во время подтверждения суммы возникают ошибки, выполняется поток ошибок E1.
10. Банкомат вычитает требуемую сумму из счета клиента.
11. Банкомат выдает клиенту требуемую сумму наличными.
12. Банкомат возвращает клиенту его карточку.
13. Банкомат печатает чек для клиента.
14. Вариант использования завершается.

Альтернативный поток A1. Ввод неправильного PIN-кода

1. Банкомат информирует клиента, что код введен неправильно.
2. Банкомат возвращает клиенту его карточку.
3. Вариант использования завершается.

Альтернативный вариант использования A2. Недостаточно денег на счете

1. Банкомат информирует клиента, что денег на его счете недостаточно.
2. Банкомат возвращает клиенту его карточку.
3. Вариант использования завершается.

Поток ошибок Е1. Ошибка в подтверждении запрашиваемой суммы

1. Банкомат сообщает пользователю, что при подтверждении запрашиваемой суммы произошла ошибка, и дает ему номер телефона службы поддержки клиентов банка.
2. Банкомат заносит сведения об ошибке в журнал ошибок. Каждая запись содержит дату и время ошибки, имя клиента, номер его счета и код ошибки.
3. Банкомат возвращает клиенту его карточку.
4. Вариант использования завершается.

Постусловия

Постусловиями называются такие условия, которые всегда должны быть выполнены после завершения варианта использования. Например, в конце варианта использования можно поместить флажок какой-нибудь переключатель. Информация такого типа входит в состав постусловий. Как и для предусловий, с помощью постусловий можно вводить информацию о порядке выполнения вариантов использования системы. Если, например, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

Связи между вариантами использования и действующими лицами

В языке UML для вариантов использования и действующих лиц поддерживается несколько типов связей. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации - это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показываются с помощью однопольной ассоциации (линии со стрелкой). Направление стрелки позволяет понять, кто инициирует коммуникацию.

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования. С помощью таких

связей обычно моделируют многократно используемую функциональность. В примере банковской системы варианты использования «Снять деньги со счета» и «Сделать вклад» должны опознать (аутентифицировать) клиента и его PIN-код перед осуществлением самой транзакции. Вместо того, чтобы подробно описывать процесс аутентификации для каждого из них, можно поместить эту функциональность в свой собственный вариант использования под названием «Аутентифицировать клиента».

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости применять функциональные возможности другого.

На языке UML связи включения и расширения показывают в виде зависимостей с соответствующими стереотипами (рис. 2.2).

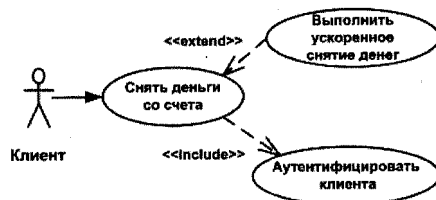


Рис. 2.2. Связи использования и расширения

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты. Например, клиенты могут быть двух типов: корпоративные и индивидуальные. Эту связь можно моделировать с помощью нотации, показанной на рис. 2.3.

Нет необходимости всегда создавать связи этого типа. В общем случае они нужны, если поведение действующего лица одного типа отличается от поведения другого постольку, поскольку это затрагивает систему. Если оба подтипа применяют одни и те же варианты использования, показывать обобщение действующего лица не требуется.

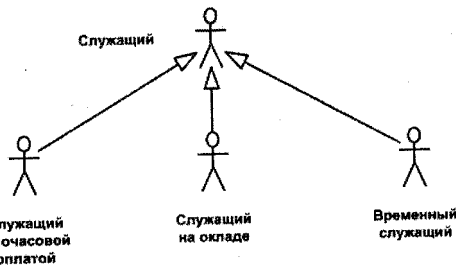


Рис. 2.3. Обобщение действующего лица

Варианты использования являются необходимым средством на стадии формирования требований к ПО. Каждый вариант использования - это потенциальное требование к системе, и пока оно не выявлено, невозможно запланировать его реализацию.

2.1.3. ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображаются ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) - средство, с помощью которого объект-отправитель запрашивает у объекта-получателя выполнение одной из его операций.

Информационное (informative) сообщение - сообщение, снабжающее объект-получатель информацией для обновления его состояния.

Сообщение-запрос (interrogative) - сообщение, запрашивающее выдачу информации об объекте-получателе.

Императивное (imperative) сообщение - сообщение, запрашивающее у объекта-получателя выполнение действий.

Существуют два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

Диаграммы последовательности

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. Например, вариант использования «Снять деньги со счета» предусматривает несколько возможных последовательностей, таких, как снятие денег, попытка снять деньги, не имея их достаточного количества на счете, попытка снять деньги по неправильному PIN-коду и некоторым другим. Нормальный сценарий снятия некоторой суммы денег со счета показан на рис. 2.4. Под сценарием понимается конкретный экземпляр потока событий.

Эта диаграмма последовательности отображает поток событий в рамках варианта использования «Снять деньги со счета». Все действующие лица показаны в верхней части диаграммы; в приведенном выше примере изображено действующее лицо Клиент (Customer). Объекты, требуемые системе для выполнения варианта использования «Снять деньги со счета», также представлены в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника на вершине пунктирной вертикальной линии. Эта вертикальная линия называется *линией жизни (lifeline)* объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое Сообщение изображается в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице, сверху вниз. Каждое сообщение Помечается как минимум именем сообщения; при желании можно добавить также аргументы и некоторую управляющую информа-

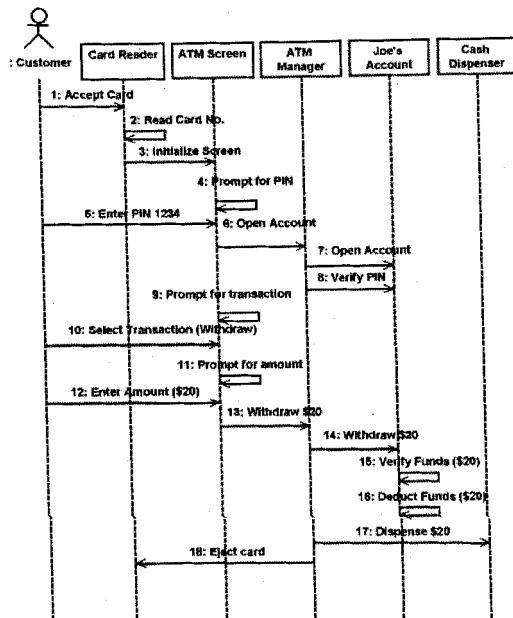


Рис. 2.4. Диаграмма последовательности

цию и, кроме того, показать самоделегирование (self-delegation) - сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

Хороший способ первоначального обнаружения некоторых объектов - это изучение имен существительных в потоке событий. Можно также прочитать документы, описывающие конкрет-

ный сценарий. Поток событий для варианта использования «Снять деньги со счета» говорит о человеке, снимающем некоторую сумму денег со счета с помощью банкомата.

Не все объекты, показанные на диаграмме, явно присутствуют в потоке событий. Там, например, может не быть форм для Заполнения, но их необходимо показать на диаграмме, чтобы Позволить действующему лицу ввести новую информацию в систему или просмотреть ее. В потоке событий скорее всего также не будет и управляющих объектов (control objects). Эти объекты управляют последовательностью событий в варианте использования.

Кооперативные диаграммы

Вторым видом диаграммы взаимодействия является кооперативная диаграмма.

Подобно диаграммам последовательности, кооперативные диаграммы отображают поток событий через конкретный сценарий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы заостряют внимание на связях между объектами. На рис. 2.5 приведена кооперативная диаграмма, описывающая, как клиент снимает деньги со счета.

Как видно из рисунка, здесь представлена вся та информация, которая была и на диаграмме последовательности, но кооперативная диаграмма по-другому описывает поток событий. Из нее легче понять связи между объектами, однако труднее уяснить последовательность событий.

По этой причине часто для какого-либо сценария создают диаграммы обоих типов. Хотя они служат одной и той же цели и содержат одну и ту же информацию, но представляют ее с разных точек зрения.

На кооперативной диаграмме, так же как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность, однако, указывается путем Нумерации сообщений.

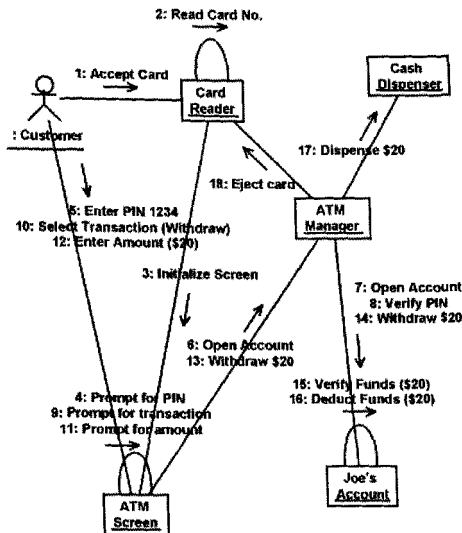


Рис. 2.5. Кооперативная диаграмма

2.1.4. ДИАГРАММЫ КЛАССОВ

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Диаграмма классов для варианта использования «Снять деньги со счета» показана на рис. 2.6.

Применение объектно-ориентированного подхода

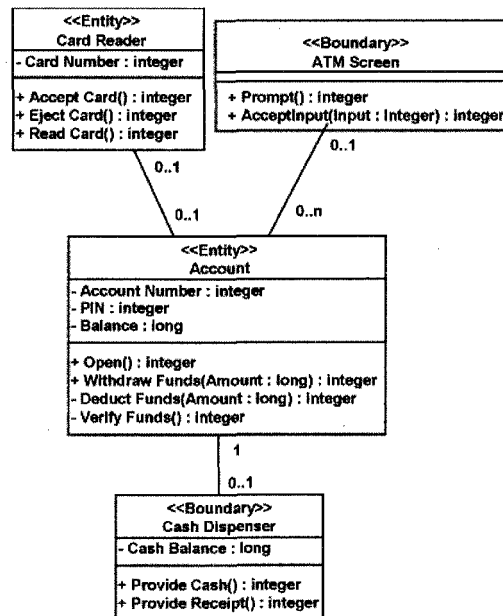


Рис. 2.6. Диаграмма классов для варианта использования «Снять деньги со счета»

На этой диаграмме классов показаны связи между классами, реализующими вариант использования «Снять деньги со счета». В этом процессе задействованы четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран ATM) и Cash Dispenser (кассовый аппарат). Каждый класс на диаграмме выглядит в виде прямоугольника, разделенного на три

части. В первой содержится имя класса, во второй - его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом).

Связывающие классы линии отражают взаимодействие между классами. Так, класс Account связан с классом ATM Screen, потому что они непосредственно сообщаются и взаимодействуют друг с другом. Класс Card Reader не связан с классом Cash Dispenser, поскольку они не сообщаются друг с другом непосредственно.

Стереотипы классов

Стереотипы - это механизм, позволяющий разделять классы на категории. В языке UML основными стереотипами являются: Boundary (граница), Entity (сущность) и Control (управление).

Граничные классы (boundary classes) - это классы, которые расположены на границе системы и окружающей среды. Они включают все формы, отчеты, интерфейсы с аппаратурой (такой, как принтеры или сканеры) и интерфейсы с другими системами.

Для того чтобы найти граничные классы, надо исследовать диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать по крайней мере один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

Классы-сущности (entity classes) отражают основные понятия (абстракции) предметной области и, как правило, содержат хранящую информацию. Обычно для каждого класса-сущности создается таблица в базе данных.

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности - остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например класс Security Manager (менеджер безопасности), отвечающий за конт-

роль событий, связанных с безопасностью. Класс Transaction-Manager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими, как разделение ресурсов, распределенная обработка данных или обработка ошибок.

Помимо упомянутых выше стереотипов можно создавать и свои собственные.

Механизм пакетов

Пакеты применяются, чтобы сгруппировать классы, обладающие некоторой общностью. Существует несколько наиболее распространенных подходов к группировке. Во-первых, можно группировать их по стереотипу. В таком случае получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход может быть полезен. С точки зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах компоненты с граничными Классами уже оказываются в одном пакете.

Другой подход заключается в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам. Если для группировки классов не применять некоторые эвристики, то она становится весьма произвольной. Одна из них, которая в основном используется в UML, - это зависимость. Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость. Таким образом, *диаграмма пакетов* (рис. 2.7) представляет собой диаграмму, содержащую пакеты Классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, т.е. диаграммы пакетов - это форма диаграммы классов.

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины

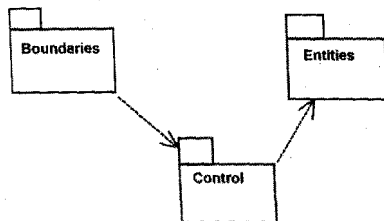


Рис. 2.7. Диаграмма пакетов

для зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в вашей системе, однако они помогают выделить эти зависимости, а после того поработать над снижением их количества. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится нечитаемой.

Атрибуты

Атрибут - это элемент информации, связанный с классом. Например, у класса Company (Компания) могут быть атрибуты Name (Название), Address (Адрес) и NumberOfEmployees (Число служащих).

Атрибуты содержатся внутри класса, поэтому они скрыты от других классов. В связи с этим иногда требуется указать, каюк классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attribute visibility).

У атрибута можно определить четыре возможных значения этого параметра. Рассмотрим каждый из них в контексте примера (рис. 2.8). Пусть у нас имеется класс Employee с атрибутом Address и класс Company:

Employee
-Employee ID : Integer = 0
#SSN:String
#Salary:float
+Address : String
+City: String
+State: String
+Zip Code: long
+Departament: String
+Hire()
+Fire()
+Promote()
+Demote()
+Transfer()

Рис. 2.8. Видимость атрибутов

- *Public (общий, открытый)*. Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В таком случае класс Company может изменить значение атрибута Address класса Employee. В соответствии с нотацией UML общему атрибуту предшествует знак «+».

- *Private (закрытый, секретный)*. Соответствующий атрибут не виден никаким другим классам. Класс Employee будет знать значение атрибута Address и сможет изменять его, но класс Company не сможет его ни увидеть, ни редактировать. Если это понадобится, он должен попросить класс Employee просмотреть или изменить значение этого атрибута, что обычно делается с помощью общих операций. Закрытый атрибут обозначается знаком «-» в соответствии с нотацией UML.

- *Protected* (защищенный). Такой атрибут доступен только самому классу и его потомкам. Допустим, что у нас имеются два типа сотрудников - с почасовой оплатой и на окладе. Таким образом мы получаем два других класса *HourlyEmp* и *SalariedEmp*, являющихся потомками класса *Employee*. Защищенный атрибут *Address* можно просмотреть или изменить из классов *Employee*, *HourlyEmp* и *SalariedEmp*, но не из класса *Company*. Нотация UML для защищенного атрибута - это знак «#».

- *Package or Implementation* (пакетный). Предполагает, что данный атрибут является общим, но только в пределах его пакета. Допустим, что атрибут *Address* имеет пакетную видимость. В таком случае он может быть изменен из класса *Company*, только если этот класс находится в том же пакете. Этот тип видимости не обозначается никаким специальным значком.

В общем случае атрибуты рекомендуется делать закрытыми или защищенными. Это позволяет лучше контролировать сам атрибут и код, а также избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут. Задаваемые параметры видимости повлияют на генерируемый код.

Операции

Операции реализуют связанное с классом поведение. Операция включает три части - имя, параметры и тип возвращаемого значения. Параметры - это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать как имена операций, так и имена операций вместе с их параметрами и типом возвращаемого значения. Для того чтобы уменьшить загруженность диаграммы, полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент1 : тип данных аргумента1, аргумент2: тип данных аргумента2,...) : тип возвращаемого значения

Рассмотрим четыре различных типа операций.

Операции реализации (implementor operations) реализуют некоторые бизнес-функции. Такие операции можно найти, исследуя диаграммы взаимодействия. Диаграммы этого типа фокуси-

руются на бизнес-функциях, и каждое сообщение диаграммы скорее всего можно соотнести с операцией реализации.

Каждая операция реализации должна быть легко прослеживаема до соответствующего требования. Это достигается на различных этапах моделирования. Операция выводится из сообщения на диаграмме взаимодействия, сообщения исходят из подробного описания потока событий, который создается на основе варианта использования, а последний - на основе требований. Возможность проследить всю эту цепочку позволяет гарантировать, что каждое требование будет реализовано в коде, а каждый фрагмент кода реализует какое-то требование.

Операции управления (manager operations) управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.

Операции доступа (access operations) существуют для того, чтобы просматривать или изменять значения атрибутов других классов.

Пусть, например, у нас имеется атрибут *Salary* класса *Employee*. Мы не хотим, чтобы все остальные классы могли изменять этот атрибут. Вместо этого к классу *Employee* мы добавляем две операции доступа - *GetSalary* и *SetSalary*. К *первой* из них, являющейся общей, могут обращаться и другие классы. Она просто получает значение атрибута *Salary* и возвращает его вызывавшему ее классу. Операция *SetSalary* также является общей, она помогает вызывавшему ее классу установить новое значение атрибута *Salary*. Эта операция может содержать любые правила и условия проверки, которые необходимо выполнить, чтобы зарплата могла быть изменена.

Такой подход дает возможность безопасно инкапсулировать атрибуты внутри класса, защитив их от других классов, но все же позволяет осуществить к ним контролируемый доступ. Создание операций *Get* и *Set* (получения и изменения значения) для каждого атрибута класса является стандартом.

Вспомогательные операции (helper operations) - это такие операции класса, которые необходимы ему для выполнения его обязанностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Чтобы идентифицировать операции, выполните следующие действия:

1. Изучите диаграммы последовательности и кооперативные диаграммы. Большая часть сообщений на этих диаграммах явля-

ется операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.

2. Рассмотрите управляющие операции. Может потребоваться добавить конструкторы и деструкторы.

3. Рассмотрите операции доступа. Для каждого атрибута класса, с которым должны будут работать другие классы, надо создать операции Get и Set.

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме последовательности или кооперативной диаграмме, между ними должна существовать связь.

Имеются четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

Ассоциации (association) - это семантическая связь между классами. Ее рисуют на диаграмме классов в виде обыкновенной линии (рис. 2.9).

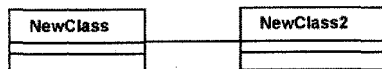


Рис. 2.9. Ассоциация

Ассоциации могут быть двунаправленными, как в примере, или однонаправленными. На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление.

Направление ассоциации можно определить, изучая диаграммы последовательности и кооперативные диаграммы. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, а не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

Зависимости (dependency) также отражают связь между классами, но они всегда однонаправлены и показывают, что один класс зависит от определений, сделанных в другом. Зависимости изображают в виде стрелки, проведенной пунктирной линией (рис. 2.10).



Рис. 2.10. Зависимость

При генерации кода для этих классов к ним не будут добавляться новые атрибуты. Однако будут созданы специфические для языка операторы, необходимые для поддержки связи. Например, на языке C++ в код войдут необходимые операторы `#include`.

Агрегации (aggregations) представляют собой более тесную форму ассоциации. Агрегация - это связь между целым и его частью. Например, у вас *может быть класс* Автомобиль, а также классы Двигатель, Покрышки и классы для других частей автомобиля. В результате объект класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышки и т.д. Агрегации визуализируют в виде линии с ромбиком у класса, являющегося целым (рис. 2.11).

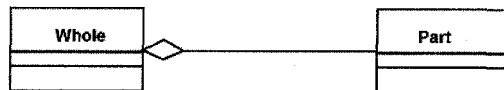
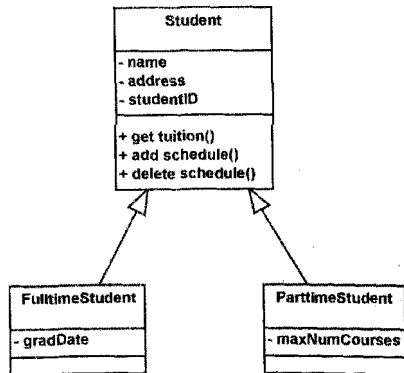


Рис. 2.11. Агрегация

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции объект-часть может принадлежать только единственному целому, и, кроме того, зачастую жизненный цикл частей совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Такое каскадное удаление нередко рассматривается как часть определения агрегации, однако оно всегда подразумевается в том случае, когда множественность роли составляет 1..1; например, если необходимо удалить Клиента, то это удаление должно распространиться и на Заказы (и, в свою очередь, на Строки_заказа).

Обобщения (generalization) показывают связи наследования между двумя классами. Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. На языке UML связи наследования называют обобщениями и изображают в виде стрелок от класса-потомка к классу-предку (рис. 2.12).



Помимо наследуемых каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

Множественность (multiplicity) показывает, сколько экземпляров одного класса взаимодействуют с помощью этой связи с одним экземпляром другого класса в данный момент времени.

Например, при разработке системы регистрации курсов в университете можно определить классы Course (Курс) и Student (Студент). Между ними установлена связь: у курсов могут быть студенты, а у студентов - курсы. Вопрос, на который должен ответить параметр множественности: «Сколько курсов студент может посещать в данный момент?» и «Сколько студентов может за раз посещать один курс?»

Так как множественность дает ответ на оба эти вопроса, ее индикаторы устанавливаются на обоих концах линии связи. В примере регистрации курсов мы решили, что один студент может посещать от 0 до 4 курсов, а один курс может слушать от 10 до 20 студентов. На диаграмме классов это можно изобразить следующим образом (рис. 2.13).

Цифры 0..4 обозначают, что один студент может посещать от 0 до 4 курсов, а цифры 10..20 - что на одном курсе могут заниматься от 10 до 20 студентов.

В языке UML приняты следующие нотации для обозначения множественности (табл. 2.1).

Имена связей. Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи - это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. В связи с этим возникает вопрос, является ли объект класса Person клиентом компании, ее сотрудником или владельцем? Для того чтобы определить это, ассоциацию можно назвать «employs» (нанимает). На рис. 2.14 приведено имя связи.

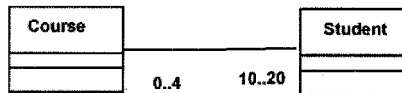


Рис. 2.13. Множественность

Таблица 2.1

Значения множественности

Множественность	Значение
*	Много
0	Нуль
1	Один
0..*	Нуль или больше
1..*	Один или больше
0..1	Нуль или один
1..1	Ровно один

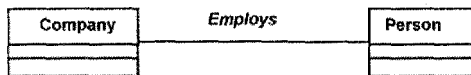


Рис. 2.14. Имя связи

Имена у связей определять необязательно. Обычно это делают, если причина создания связи не очевидна. Имя показывают около линии соответствующей связи.

Роли. Рольевые имена применяют в связях ассоциации или агрегации *вместо имен для описания того, зачем эти связи нужны*. Возвращаясь к примеру с классами *Person* и *Company*, можно сказать, что класс *Person* играет роль сотрудника класса *Company*. Рольевые имена -- это обычно имена существительные или включающие их фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или рольевым именем, или именем связи, но не обоими сразу. Как и имена связей, рольевые имена не обязательны, их дают, только если цель связи не очевидна. Пример рольевых имен приводится на рис. 2.15.

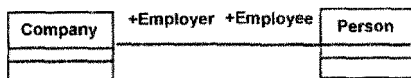


Рис. 2.15. Рольевые имена

2.1.5. ДИАГРАММЫ СОСТОЯНИЙ

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На рис. 2.16 приводится пример диаграммы состояний для банковского счета (*account*). Можно также наблюдать процесс перехода счета из одного состояния в другое. Например, если клиент требует закрыть счет, он переходит в состояние «закрыт». Требование клиента называется событием (*event*), именно такие события и вызывают переход из одного состояния в другое.

Если клиент снимает деньги со счета, он может перейти в состояние «Превышение кредита». Это происходит только в том случае, если баланс по счету меньше нуля, что отражено условием [отрицательный баланс] на нашей диаграмме. Заключенное в квадратных скобках условие (*guard condition*) определяет, когда может произойти переход из одного состояния в другое.

На диаграмме имеются два специальных состояния - начальное (*start*) и конечное (*stop*). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. В нашем примере при превышении кредита клиенту посылается соответствующее сообщение. Процессы, происходящие в этот момент, когда объект находится в определенном состоянии, называются действиями (*actions*).

С состоянием можно связывать следующие данные: деятельность, входное действие, выходное действие и событие. Рассмотрим каждый из них в контексте диаграммы состояний для класса *Account* банковской системы.

Деятельность (activity) - это поведение, реализуемое объектом, пока он находится в данном состоянии. Например, когда счет находится в состоянии «Закрыт», происходит возврат кредитной карточки пользователю. Деятельность - это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния; ее обозначению должно предшествовать слово *do* (делать) и двоеточие.

Входное действие (entry action) - это поведение, которое выполняется, когда объект переходит в данное состояние. Как только счет в банке переходит в состояние «Превышен счет» (см. рис. 2.16),

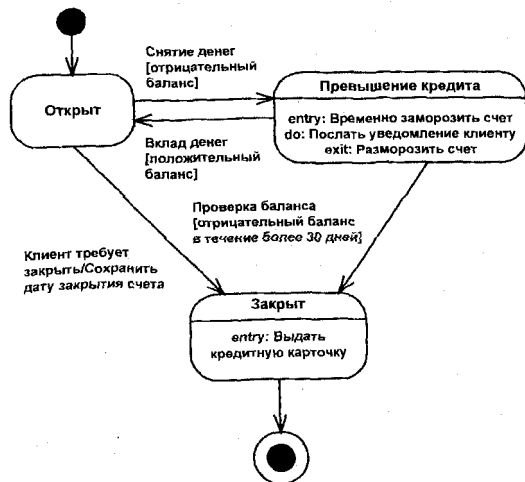


Рис. 2.16. Диаграмма состояний для класса Account

выполняется действие «Временно заморозить счет» независимо от того, откуда объект перешел в это состояние. Таким образом, данное действие осуществляется не после того, как объект перешел в это состояние, а скорее как часть этого перехода. В отличие от деятельности входное действие рассматривается как непрерываемое.

Входное действие также показывают внутри состояния, его обозначению предшествуют слово *entry* (вход) и двоеточие.

Выходное действие (exit action) подобно входному. Однако оно осуществляется как составная часть процесса выхода из данного состояния. В нашем примере при выходе объекта account из состояния «Превышен счет» выполняется действие «Разморозить счет». Оно является частью *процесса* такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, его описанию предшествуют слово *exit* (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях, может включать отправку события другому объекту. Например, объект account (счет) может посылать событие объекту card reader (устройство чтения карты). В этом случае описанию деятельности, входного или выходного действия предшествует знак «^». Соответствующая строка на диаграмме выглядит как

Do: ^Цель.Событие(Аргументы)

Здесь Цель - это объект, получающий событие, Событие - это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

Деятельность может также выполняться в результате получения объектом некоторого события. Например, объект Account может быть в состоянии «Открыто». При получении некоторого События выполняется определенная деятельность.

Переходом (transition) называется перемещение объекта из одного состояния в другое. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события. Рассмотрим каждое из них в контексте примера банковской системы.

Событие (event) – это то, что вызывает переход из одного состояния в другое. В нашем примере событие «Клиент требует закрыть» вызывает переход счета из открытого в закрытое состояние. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу. В нашем примере события описаны обычными фразами. Если нужно использовать операции, то событие «Клиент требует закрыть» можно было бы назвать RequestClosure().

У событий могут быть аргументы. Так, событие «Сделать вклад», вызывающее переход счета из состояния «Превышен счет» в состояние «Открыто», может иметь аргумент Amount (Количество), описывающий сумму депозита.

Большинство переходов должны иметь события, так как именно они прежде всего заставляют переход осуществиться. Тем не менее бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

Ограждающие условия (guard conditions) определяют, когда переход может или не может осуществиться. В нашем примере событие «Сделать вклад» переведет счет из состояния «Превышение счета» в состояние «Открыто», но только при условии, если баланс будет больше нуля. В противном случае переход не осуществится.

Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

Действие (action), как уже говорилось, является непрерываемым поведением, осуществляющимся как часть перехода. Вход-

ные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из состояния. Большую часть действий, однако, изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния.

Например, при переходе счета из открытого в закрытое состояние выполняется действие «Сохранить дату закрытия счета». Это непрерываемое поведение осуществляется только во время перехода из состояния «Открыто» в состояние «Закрыто».

Действие рисуют вдоль линии перехода после имени события, его изображению предшествует косая черта.

Событие или действие может быть поведением внутри объекта, а может представлять собой сообщение, *посылаемое* другому объекту. Если событие или действие посылается другому объекту, перед ним на диаграмме помещают знак «^».

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться диаграмма состояний.

2.1.6. ДИАГРАММЫ ДЕЯТЕЛЬНОСТЕЙ

Диаграммы деятельности особенно полезны в описании поведения, включающего большое количество параллельных процессов. Самым большим достоинством диаграмм деятельности является поддержка параллелизма. Благодаря этому они являются мощным средством моделирования потоков работ и, по существу, параллельного программирования. Самый большой их недостаток заключается в том, что связи между действиями и объектами просматриваются не слишком четко.

Эти связи можно попытаться определить, используя для деятельности метки с именами объектов, но этот способ не обладает такой же простотой, как у диаграмм взаимодействия. Диаграммы деятельности предпочтительнее использовать в следующих ситуациях:

- анализ варианта использования. На этой стадии нас не интересует связь между действиями и объектами, а нужно только понять, какие действия должны иметь место и каковы зависимости в поведении системы. Связывание методов и объектов выполняется позднее с помощью диаграмм взаимодействия;

- анализ потоков работ (workflow) в различных вариантах использования. Когда варианты использования взаимодействуют друг с другом, диаграммы деятельности являются мощным средством представления и анализа их поведения.

2.1.7. ДИАГРАММЫ КОМПОНЕНТОВ

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

На рис. 2.17 изображена одна из диаграмм компонентов для банковской системы.

На этой диаграмме показаны компоненты для клиентской части системы. В данном случае система разрабатывается на языке C++. У каждого класса имеется свой собственный заголовочный файл и файл с расширением .CPP, так что каждый класс преобразуется в свои собственные компоненты на диаграмме. Например, класс ATM Screen преобразуется в компонент ATM Screen диаграммы. Он преобразуется также и во второй компонент ATM Screen. Вместе эти два компонента представляют тело и заголовок класса ATM Screen. Выделенный темным компонент называется спецификацией пакета (package specification) и соответствует файлу тела класса ATM Screen на языке C++ (файл с расширением .CPP). Невыделенный компонент также называется спецификацией пакета, но соответствует заголовочному файлу класса языка C++ (файл с расширением .H).

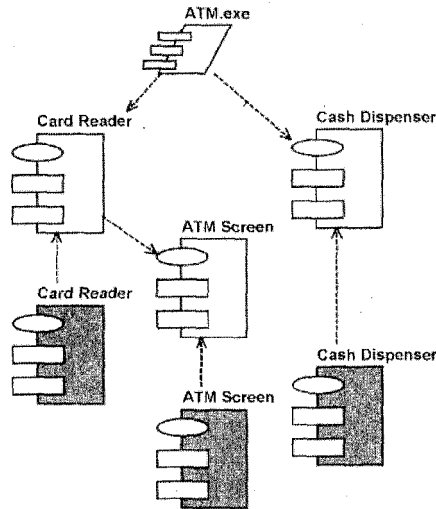


Рис. 2.17. Диаграмма компонентов для клиентской части системы

Компонент ATM.exe является спецификацией задачи и представляет поток обработки информации (thread of processing). В данном случае поток обработки является исполняемой программой.

Компоненты соединены штриховой линией, что соответствует зависимостям между ними. Например, класс Card Reader зависит от класса ATM Screen. Это означает, что для того, чтобы класс Card Reader мог быть скомпилирован, класс ATM Screen должен уже существовать. После компиляции всех классов может быть создан исполняемый файл ATMClient.exe.

Пример банковской системы содержит два потока обработки, и таким образом получаются два исполняемых файла. Один из них - это клиентская часть системы, она содержит компоненты Cash Dispenser, Card Reader и ATM Screen. Второй файл - это сервер, включающий в себя компонент Account. Диаграмма компонентов для сервера показана на рис. 2.18.

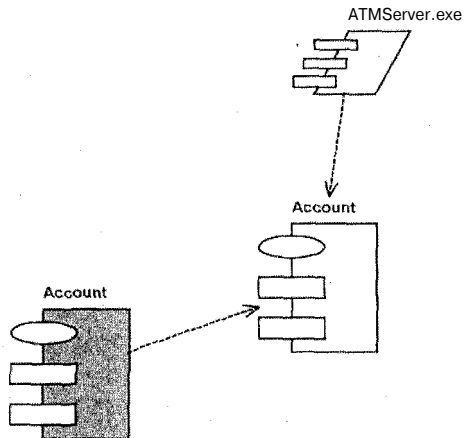


Рис. 2.18. Диаграмма компонентов для сервера

Как видно из примера, у системы может быть несколько диаграмм компонентов в зависимости от числа подсистем или исполняемых файлов. Каждая подсистема является пакетом компонентов. В общем случае пакеты - это совокупности компонентов. Пример банковской системы содержит два пакета: клиентская часть и сервер.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

2.1.8. ДИАГРАММЫ РАЗМЕЩЕНИЯ

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она показывает размещение объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства - в большинстве случаев часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. В нашем примере банковская система состоит из большого количества подсистем, выполняемых на отдельных физических устройствах, или узлах (node). Диаграмма размещения для банковской системы показана на рис. 2.19.

Из данной диаграммы можно узнать о физическом размещении системы. Клиентские программы будут работать в нескольких местах на различных сайтах. Через закрытые сети будет осуществляться их сообщение с региональным сервером системы, с работающим программным обеспечением. В свою очередь, региональный сервер посредством локальной сети будет сообщаться с сервером банковской базы данных, работающим под управлением Oracle. Наконец, с региональным сервером соединен принтер.

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение ее отдельных подсистем.

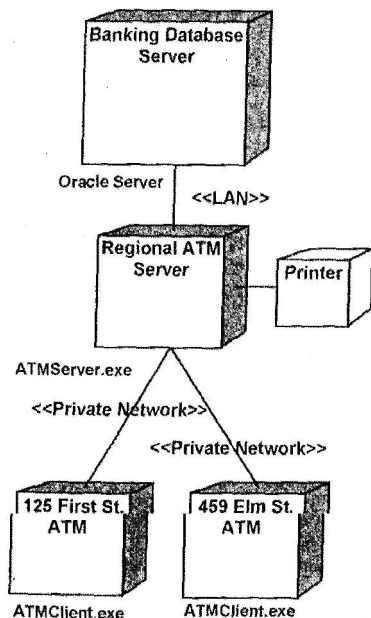


РИС. 2.19. Диаграмма размещения
для банковской системы

2.2. ОСНОВНЫЕ СВЕДЕНИЯ О CASE-СРЕДСТВЕ RATIONAL ROSE

2.2.1. ВВЕДЕНИЕ В RATIONAL ROSE

Rational Rose - семейство объектно-ориентированных CASE-средств фирмы Rational Software Corporation предназначено для автоматизации процессов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует метод объектно-ориентированного анализа и проектирования, основанный на языке UML. Текущая версия Rational Rose реализует генерацию кодов программ для C++, Visual C++, Visual Basic, Java, PowerBuilder, CORBA Interface Definition Language (IDL), генерацию описаний баз данных для ANSI SQL, Oracle, MS SQL Server, IBM DB2, Sybase, а также позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций. Кроме того, Rational Rose содержит средства реверсного инжиниринга программ и баз данных, обеспечивающие повторное использование программных компонентов в новых проектах.

Структура и функции. В основе работы Rational Rose лежит построение диаграмм и спецификаций UML, определяющих архитектуру системы, ее статические и динамические аспекты. В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (браузер), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор для C++, обеспечивающий реверсный инжиниринг.

Репозиторий представляет собой базу данных проекта. Браузер обеспечивает «навигацию» по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т.д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по

мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке C++, используя информацию, содержащуюся в диаграммах классов и компонентов, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом «скелет» программы может быть уточнен путем прямого программирования на языке C++. Анализатор кодов C++ реализован в виде отдельного программного модуля. Его назначение - создавать модули проектов Rational Rose на основе информации, содержащейся в определяемых пользователем исходных текстах на C++. В процессе работы анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Модель, полученная в результате его работы, может целиком или фрагментарно использоваться в различных проектах. Анализатор обладает широкими возможностями настройки по входу и выходу. Например, можно определить типы исходных файлов, базовый компилятор, задать, какая информация должна быть включена в формируемую модель, и какие элементы выходной модели следует выводить на экран. Таким образом, Rational Rose/C++ обеспечивает возможность повторного использования программных компонентов.

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

- диаграммы UML, в совокупности представляющие собой модель разрабатываемой программной системы;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ.

Тексты программ являются заготовками для последующей работы программистов. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Взаимодействие с другими средствами и организация групповой работы. Для поддержки командной работы над проектом на каждой стадии жизненного цикла ПО имеется интегрированный набор продуктов Rational Suite. Rational Suite существует в следующих вариантах:

- Rational Suite AnalystStudio - предназначен для определения и управления полным набором требований к разрабатываемой системе;
- Rational Suite DevelopmentStudio - предназначен для проектирования и реализации ПО;
- Rational Suite TestStudio - представляет собой набор продуктов, предназначенных для автоматического тестирования приложений;
- Rational Suite Enterprise - обеспечивает поддержку полного жизненного цикла ПО и предназначен как для менеджеров Проекта, так и отдельных разработчиков, исполняющих несколько функциональных ролей в команде разработчиков.

В состав Rational Suite, кроме Rational Rose, входят следующие компоненты:

- Rational Requisite Pro - средство управления требованиями, предназначенное для организации совместной работы группы разработчиков. Оно позволяет команде разработчиков создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающих на любом этапе разработки компонентов приложения;
- Rational ClearCase - средство управления конфигурацией ПО;
- Rational SoDA - средство автоматической генерации проектной документации;
- Rational ClearQuest - средство для управления изменениями и отслеживания дефектов в проекте на основе средств e-mail и Web;
- Rational TeamTest - средство автоматического обнаружения ошибок во время выполнения программы и генерации сценариев для проведения регрессионного тестирования;
- Rational Robot - средство для создания, модификации и автоматического запуска тестов;
- Rational Purify - средство для локализации трудно обнаруживаемых ошибок времени выполнения программы;
- Rational PureCoverage - средство идентификации участков кода, пропущенных при тестировании;
- Rational Quantify - средство количественного определения узких мест, влияющих на общую эффективность работы программы;

• Rational Suite PerformanceStudio - средство нагрузочного тестирования приложений «клиент-сервер» и Web-приложений.

Для организации групповой работы в Rational Rose возможно разбиение модели на управляемые подмодели. Каждая из них независимо сохраняется на диске или загружается в модель. В качестве подмодели может выступать пакет или подсистема.

Среда функционирования. Rational Rose функционирует на различных платформах: IBM PC (Windows 95/98/NT), Sun SPARCstations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

2.2.2.

РАБОТА В СРЕДЕ RATIONAL ROSE

Элементы экрана интерфейса Rose - это браузер, окно документации, панели инструментов, окно диаграммы и журнал. Их назначение заключается в следующем:

- браузер (browser) - используется для быстрой навигации по модели;
- окно документации (documentation window) - применяется для работы с текстовым описанием элементов модели;
- панели инструментов (toolbars) - применяются для быстрого доступа к наиболее распространенным командам;
- окно диаграммы (diagram window) - используется для просмотра и редактирования одной или нескольких диаграмм UML;
- журнал (log) - применяется для просмотра ошибок и отчетов о выполнении различных команд.

На рис. 2.20 показаны различные части интерфейса Rose.

Браузер - это иерархическая структура, позволяющая осуществлять навигацию по модели. Все, что добавляется к ней - действующие лица, варианты использования, классы, компоненты, будет показано в окне браузера.

С помощью браузера можно:

- добавлять к модели элементы;
- просматривать существующие элементы модели;
- просматривать существующие связи между элементами модели;
- перемещать элементы модели;

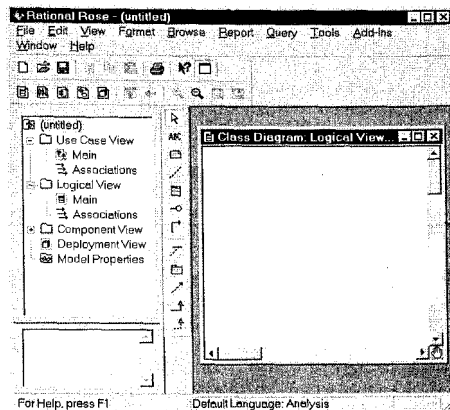


Рис. 2.20. Интерфейс Rose

- переименовывать эти элементы;
- добавлять элементы модели к диаграмме;
- связывать элемент с файлом или адресом Интернета;
- группировать элементы в пакеты;
- работать с детализированной спецификацией элемента;
- открывать диаграмму.

Браузер поддерживает четыре представления (view): представление вариантов использования, компонентов, размещения и логическое представление. Все они и содержащиеся в них элементы модели описаны ниже.

Организация браузера представляет собой древовидную структуру. Каждый элемент модели может содержать другие элементы, находящиеся ниже его в иерархии. Знак «-» около элемента означает, что его ветвь полностью раскрыта. Знак «+» - что его ветвь свернута.

Окно документации. С его помощью можно документировать элементы модели Rose. Например, можно сделать краткое описание каждого действующего лица. При документировании класса все, что будет написано в окне документации, появится затем как комментарий в сгенерированном коде, что избавляет от необходимости впоследствии вносить эти комментарии вручную. Документация будет выводиться также в отчетах, создаваемых в среде Rose.

Панели инструментов Rose обеспечивают быстрый доступ к наиболее распространенным командам. В этой среде существуют два типа панелей инструментов: стандартная панель и панель диаграммы. Стандартная панель видна всегда, ее кнопки соответствуют командам, которые могут использоваться для работы с любой диаграммой. Панель диаграммы своя для каждого типа диаграмм UML.

Все панели инструментов могут быть изменены и настроены пользователем. Для этого выберите пункт меню Tools > Options, затем вкладку Toolbars.

Показать или скрыть стандартную панель инструментов (или панели инструментов диаграммы) можно следующим образом.

1. Выберите пункт Tools > Options.
2. Выберите вкладку Toolbars.

3. Чтобы сделать видимой или невидимой стандартную панель инструментов, пометьте (или снимите пометку) контрольный переключатель Show Standard ToolBar (или Show Diagram ToolBar).

Для того чтобы увеличить размер кнопок на панели инструментов:

1. Щелкните правой кнопкой мыши по требуемой панели.
2. Выберите во всплывающем меню пункт Use Large Buttons (Использовать большие кнопки).

Для настройки панели инструментов:

1. Щелкните правой кнопкой мыши по требуемой панели.
2. Выберите пункт Customize (настроить).
3. Чтобы добавить или удалить кнопки, выберите соответствующую кнопку и затем щелкните мышью по кнопке Add (добавить) или Remove (удалить), как показано на рис. 2.21.

Окно диаграммы. В нем видно, как выглядит одна или несколько диаграмм UML-модели. При внесении в элементы диаграммы изменений Rose автоматически обновит браузер. Аналогично при

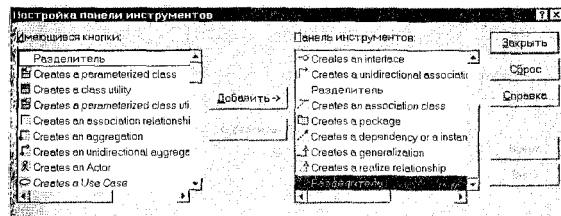


Рис. 2.21. Настройка стандартной панели инструментов

внесении изменений в элемент с помощью браузера Rose автоматически обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.

Журнал. По мере работы над вашей моделью определенная информация будет направляться в окно журнала. Например, туда помещаются сообщения об ошибках, возникающих при генерации кода. Не существует способа закрыть журнал совсем, но его окно может быть минимизировано.

Четыре представления модели Rose

В модели Rose поддерживаются четыре представления - это представление вариантов использования, логическое представление, представление компонентов и представление размещения. Каждое из них предназначено для своих целей и для соответствующей аудитории. В последующих разделах этой главы мы кратко рассмотрим каждое из указанных представлений, а в оставшейся части книги детально обсудим содержащиеся в них элементы модели.

Представление вариантов использования содержит всех действующих лиц, все варианты использования и их диаграммы для конкретной системы. Оно может также содержать некоторые диаграммы последовательности и кооперативные диаграммы. На рис. 2.22 изображено представление вариантов использования в браузере Rose.

Представление вариантов использования содержит:

- Действующих лиц.
- Варианты использования.

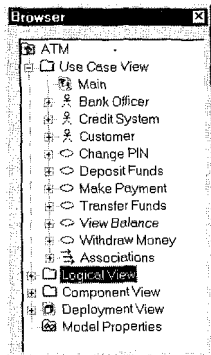


Рис. 2.22. Представление вариантов использования

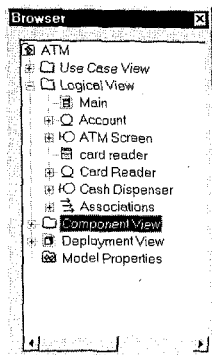


Рис. 2.23. Логическое представление системы

- Документацию по вариантам использования, детализирующую происходящие в них процессы (потоки событий), включая обработку ошибок. Эта пиктограмма соответствует внешнему файлу, прикрепленному к модели Rose. Вид пиктограммы зависит от приложения, используемого для документирования потока событий.

- Диаграммы вариантов использования. Обычно у системы бывает несколько таких диаграмм, каждая из которых показывает подмножество действующих лиц и/или вариантов использования.

- Пакеты, являющиеся группами вариантов использования и/или действующих лиц.

Логическое представление (рис. 2.23) концентрируется на том, как система будет реализовывать поведение, описанное в вариантах использования. Оно дает подробную картину составных частей системы и описывает взаимодействие этих частей. Логическое представление включает, помимо прочего, конкретные требуемые классы, диаграммы классов и диаграммы состояний. С их помощью конструируется детальный проект создаваемой системы.

Логическое представление содержит:

- Классы.
- Диаграммы классов. Как правило, для описания системы используется несколько диаграмм классов, каждая из которых отображает некоторое подмножество всех классов системы.

- Диаграммы взаимодействия, применяемые для отображения объектов, участвующих в одном потоке событий варианта использования.

- Диаграммы состояний.

- Пакеты, являющиеся группами взаимосвязанных классов.

Представление компонентов содержит:

- Компоненты, являющиеся физическими модулями кода.

- Диаграммы компонентов.

- Пакеты, являющиеся группами связанных компонентов.

Представление размещения - это последнее представление Rose. Оно соответствует физическому размещению системы, которое может отличаться от ее логической архитектуры.

В представление размещения входят:

- Процессы, являющиеся потоками (threads), исполняемыми в отведенной для них области памяти.

- Процессоры, включающие любые компьютеры, способные обрабатывать данные. Любой процесс выполняется на одном или нескольких процессорах.

- Устройства, т.е. любая аппаратура, не способная обрабатывать данные (например, терминалы ввода-вывода и принтеры).

- Диаграмма размещения.

Параметры настройки отображения (изображение атрибутов и операций на диаграммах классов)

В Rose имеется возможность настроить диаграммы классов так, чтобы:

- Показывать все атрибуты и операции.
- Скрыть операции.
- Скрыть атрибуты.
- Показывать только некоторые атрибуты или операции.
- Показывать операции вместе с их полными сигнатурами или только их имена.
- Показывать или не показывать видимость атрибутов и операций.
- Показывать или не показывать стереотипы атрибутов и операций.

Значения каждого параметра по умолчанию можно задать с помощью окна, открываемого при выборе пункта меню Tools > Options.

У данного класса на диаграмме можно:

- Показать все атрибуты.
- Скрыть все атрибуты.
- Показать только выбранные вами атрибуты.
- Подавить вывод атрибутов.

Подавление вывода атрибутов приведет не только к исчезновению атрибутов с диаграммы, но и к удалению линии, показывающей место расположения атрибутов в классе.

Существуют два способа изменения параметров представления атрибутов на диаграмме. Можно установить нужные значения у каждого класса индивидуально. Можно также изменить значения нужных параметров по умолчанию до начала создания диаграммы классов. Внесенные таким образом изменения повлияют только на вновь создаваемые диаграммы.

Для того чтобы показать все атрибуты класса:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.

3. В контекстно-зависимом меню выберите Options > Show All Attributes.

Для того чтобы показать у класса только избранные атрибуты:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выберите Options > Select Compartment Items.
4. Укажите нужные вам атрибуты в окне Edit Compartment.

Для того чтобы подавить вывод всех атрибутов класса диаграммы:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выберите Options > Suppress Attributes.

Для изменения принятого по умолчанию вида атрибута:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки значений параметров отображения атрибутов по умолчанию воспользуйтесь контрольными переключателями Suppress Attributes и Show All Attributes. Изменение этих значений по умолчанию повлияет только на новые диаграммы. Вид существующих диаграмм классов не изменится.

Как и в случае атрибутов, имеется несколько вариантов представления операций на диаграммах.

- Показать все операции.
- Показать только некоторые операции.
- Скрыть все операции.
- Подавить вывод операций.

Кроме того, можно:

• Показать только имя операции. В таком случае на диаграмме будет представлено только имя операции, но не аргументы или тип возвращаемого значения.

• Показать полную сигнатуру операции. На диаграмме будет представлено не только имя операции, но и все ее параметры, типы данных параметров и тип возвращаемого значения операции.

Для того чтобы показать все операции класса:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выберите Options > Show All Operations.

Для того чтобы показать только избранные операции класса:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выберите Options > Select Compartment Items.
4. Укажите нужные вам операции в окне Edit Compartment.

Для того чтобы подавить вывод всех операций класса диаграммы:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.

3. В контекстно-зависимом меню выберите Options > Suppress Operations.

Для того чтобы показать на диаграмме классов сигнатуру операции:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.

3. В контекстно-зависимом меню выберите Options > Show Operation Signature.

Для того чтобы изменить принятый по умолчанию вид операции:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки значений параметров отображения операций по умолчанию воспользуйтесь контрольными переключателями Suppress Operations, Show All Operations и Show Operation Signatures.

Для того чтобы показать видимость атрибута или операции класса:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните по нему правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выберите Options > Show Visibility.

Для изменения принятого по умолчанию значения параметра показа видимости:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки параметров отображения видимости по умолчанию воспользуйтесь контрольным переключателем Show Visibility.

Для переключения между нотациями видимости Rose и UML:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Notation.
3. Для переключения между нотациями воспользуйтесь переключателем Visibility as Icons. Если этот переключатель помечен, будет использоваться нотация Rose, в противном случае - нотация UML. Изменение этого параметра повлияет только на новые диаграммы. Существующие диаграммы классов останутся прежними.

2.3. ВЫПОЛНЕНИЕ УЧЕБНОГО ПРОЕКТА

2.3.1. ПОСТАНОВКА ЗАДАЧИ СОЗДАНИЯ СИСТЕМЫ РЕГИСТРАЦИИ ДЛЯ УЧЕБНОГО ЗАВЕДЕНИЯ

Перед руководителем информационной службы университета ставится задача разработки новой клиент-серверной системы регистрации студентов взамен старой системы на мейнфрейме. Новая система должна позволять студентам регистрироваться на курсы и просматривать свои таблицы успеваемости с персональных компьютеров, подключенных к локальной сети университета. Профессора должны иметь доступ к онлайн-системе, чтобы указать курсы, которые они будут читать, и проставить оценки за курсы.

Из-за недостатка средств университет не в состоянии заменить сразу всю существующую систему. База данных, содержащая всю информацию о курсах (каталог курсов), остается функционировать в прежнем виде. Эта база данных поддерживается реляционной СУБД. Новая система будет работать с существующей БД в режиме доступа, без обновления.

В начале каждого семестра студенты могут запросить каталог курсов, содержащий список курсов, предлагаемых в данном семестре. Информация о каждом курсе должна включать имя профессора, наименование кафедры и требования к предварительному уровню подготовки (прослушанным курсам).

Новая система должна позволять студентам выбирать 4 курса в предстоящем семестре. В дополнение каждый студент может указать 2 альтернативных курса на тот случай, если какой-либо из выбранных им курсов окажется уже заполненным или отменным. На каждый курс может записаться не более 10 и не менее 3 студентов (если менее 3, то курс будет отменен). В каждом семестре существует период, когда студенты могут изменить свои планы. В это время студенты должны иметь доступ к системе,

чтобы добавить или удалить выбранные курсы. После того как процесс регистрации некоторого студента завершен, система регистрации направляет информацию в расчетную систему, чтобы студент мог внести плату за семестр. Если курс окажется заполненным в процессе регистрации, студент должен быть извещен об этом до окончательного формирования его личного учебного плана.

В конце семестра студенты должны иметь доступ к системе для просмотра своих электронных таблиц успеваемости. Поскольку эта информация конфиденциальная, система должна обеспечивать ее защиту от несанкционированного доступа.

Профессора должны иметь доступ к онлайн-системе, чтобы указать курсы, которые они будут читать, и просмотреть список студентов, записавшихся на их курсы. Кроме того, профессора должны иметь возможность проставить оценки за курсы.

2.3.2. СОСТАВЛЕНИЕ ГЛОССАРИЯ ПРОЕКТА

Глоссарий предназначен для описания терминологии предметной области. Он может быть использован как неформальный словарь данных системы.

Ниже приведены термины и их значения.

Термин	Значение
Курс	Учебный курс, предлагаемый университетом
Конкретный курс (Course Offering)	Конкретное чтение данного курса в конкретном семестре (один и тот же курс может вестись в нескольких параллельных сессиях). Включает точные дни недели и время
Каталог курсов	Полный каталог всех курсов, предлагаемых университетом
Расчетная система	Система обработки информации об оплате курсов
Оценка	Оценка, полученная студентом за конкретный курс

Профессор	Преподаватель университета
Табель успеваемости (Report Card)	Все оценки за все курсы, полученные студентом в данном семестре
Список курса (Roster)	Список всех студентов, записавшихся на конкретный курс
Студент	Личность, проходящая обучение в университете
Учебный график (Schedule)	Курсы, выбранные студентом в текущем семестре

2.3.3 ОПИСАНИЕ ДОПОЛНИТЕЛЬНЫХ СПЕЦИФИКАЦИЙ

Назначение дополнительных спецификаций - определить требования к системе регистрации курсов, которые не охватывает модель вариантов использования. Вместе они образуют полный набор требований к системе.

Дополнительные спецификации определяют нефункциональные требования к системе, такие, как надежность, удобство использования, производительность, сопровождаемость, а также ряд функциональных требований, являющихся общими для нескольких вариантов использования.

Функциональные возможности. Система должна обеспечивать многопользовательский режим работы.

Если конкретный курс оказывается заполненным в то время, когда студент формирует свой учебный график, включающий данный курс, то система должна известить его об этом.

Удобство использования. Пользовательский интерфейс должен быть Windows 95/98-совместимым.

Надежность. Система должна быть в работоспособном состоянии 24 часа в день 7 дней в неделю, время простоя - не более 10%.

Производительность. Система должна поддерживать до 2000 пользователей, одновременно работающих с центральной базой данных пользователей, и до 500 пользователей, одновременно работающих с локальными серверами.

Безопасность. Система не должна позволять студентам изменять любые учебные графики, кроме своих собственных, а также не должна позволять профессорам модифицировать конкретные курсы, выбранные другими профессорами.

Только профессора имеют право ставить студентам оценки.

Только регистратор может изменять любую информацию о студентах.

Проектные ограничения. Система должна быть интегрирована с существующей системой каталога курсов, функционирующей на основе реляционной СУБД.

2.3.4. СОЗДАНИЕ МОДЕЛИ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Действующие лица:

- Student (Студент) - записывается на курсы.
- Professor (Профессор) - выбирает курсы для преподавания.
- Registrar (Регистратор) - формирует учебный план и каталог курсов, ведет все данные о курсах, профессорах и студентах.
 - Billing System (Расчетная система) - получает от данной системы информацию по оплате курсов.
 - Course Catalog (Каталог курсов) - передает в систему информацию из каталога курсов, предлагаемых университетом.

Упражнение 1. Создание действующих лиц в среде Rational Rose

Для того чтобы поместить действующее лицо в браузер:

1. Щелкните правой кнопкой мыши по пакету представления вариантов использования в браузере.
2. Выберите пункт New > Actor в открывшемся меню.
3. В браузере появится новое действующее лицо под названием NewClass. Слева от его имени вы увидите пиктограмму действующего лица UML.
4. Выделив новое действующее лицо, введите его имя.
5. После создания действующих лиц сохраните модель под именем courserereg(analysis) с помощью пункта меню File > Save.

Результат выполнения упражнения показан на рис. 2.24.

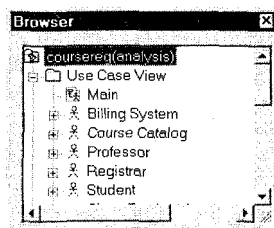


Рис. 2.24. Представление действующих лиц в браузере

Варианты использования

Исходя из потребностей действующих лиц выделяются следующие варианты использования:

- Login (Войти в систему).
- Register for Courses (Зарегистрироваться на курсы).
- View Report Card (Просмотреть таблицу успеваемости).
- Select Courses to Teach (Выбрать курсы для преподавания).
- Submit Grades (Проставить оценки).
- Maintain Professor Information (Вести информацию о профессорах).
- Maintain Student Information (Вести информацию о студентах).
- Close Registration (Закрыть регистрацию).

Упражнение 2. Создание вариантов использования в среде Rational Rose

Для того чтобы поместить вариант использования в браузер:

1. Щелкните правой кнопкой мыши по пакету представления вариантов использования в браузере.
2. Выберите в появившемся меню пункт New > Use Case.

3. Новый вариант использования под названием NewUseCase появится в браузере. Слева от него будет видна пиктограмма варианта использования UML.

4. Выделив новый вариант использования, введите его название.

Результат выполнения упражнения показан на рис. 2.25.

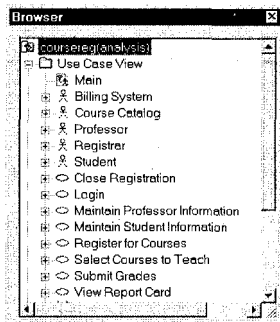


Рис. 2.25. Представление вариантов использования в браузере

Диаграмма вариантов использования

Создайте диаграмму вариантов использования для системы регистрации. Требуемые для этого действия подробно перечислены далее. Готовая диаграмма вариантов использования изображена на рис. 2.26.

В среде Rose диаграммы вариантов использования создаются в представлении вариантов использования. Главная диаграмма (Main) предлагается по умолчанию. Для моделирования системы можно затем разработать необходимое количество дополнительных диаграмм.

Для того чтобы получить доступ к главной диаграмме вариантов использования:

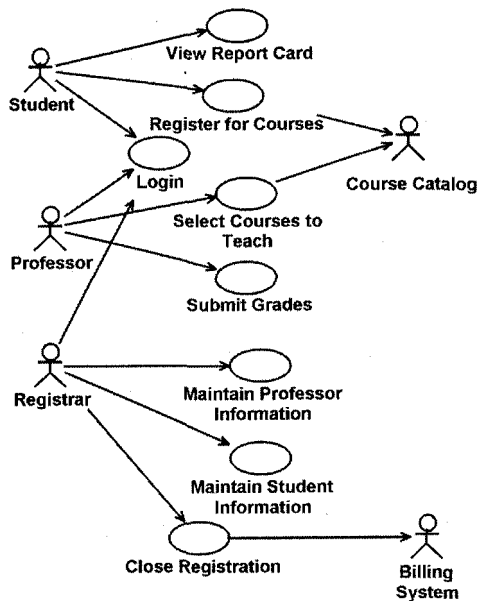


Рис. 2.26. Диаграмма вариантов использования для системы регистрации

1. Откройте данное представление, щелкнув по значку «+» рядом с представлением вариантов использования в браузере.

2. Откройте главную диаграмму, дважды щелкнув мышью. Строка заголовка изменится, включив фразу [Use Case Diagram: Use Case view / Main].

Для создания новой диаграммы вариантов использования:

1. Щелкните правой кнопкой мыши по пакету представления вариантов использования в браузере.
2. Выберите пункт New > Use Case Diagram из всплывающего меню.
3. Выделив новую диаграмму, введите ее имя.
4. Дважды щелкните по названию этой диаграммы в браузере, чтобы открыть ее.

Упражнение 3. Построение диаграммы вариантов использования

1. Откройте диаграмму вариантов использования Main.
2. Перетащите действующее лицо или вариант использования мышью из браузера на диаграмму вариантов использования.
3. С помощью кнопки Unidirectional Association (Однонаправленная ассоциация) панели инструментов нарисуйте ассоциации между действующими лицами и вариантами использования.

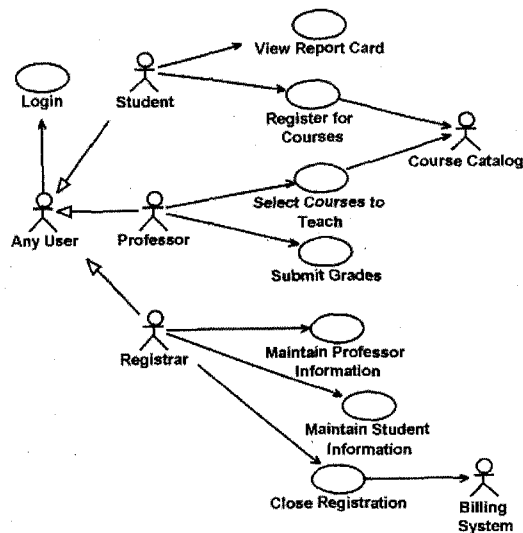
Наличие общего варианта использования Login для трех действующих лиц позволяет обобщить их поведение и ввести новое действующее лицо Any User. Модифицированная диаграмма вариантов использования показана на рис. 2.27.

Упражнение 4. Добавление описаний к вариантам использования

1. Выделите в браузере вариант использования Register for Courses.
2. В окне документации введите следующее описание к этому варианту использования: «This use case allows a student to register for courses in the current semester» («Этот вариант использования дает студенту возможность зарегистрироваться на курсы в текущем семестре»).
3. Создайте с помощью MS Word три текстовых файла с описаниями вариантов использования Login (Войти в систему), Register for Courses (Зарегистрироваться на курсы) и Close Registration (Закреть регистрацию).

Вариант использования Login

Краткое описание. Данный вариант использования описывает вход пользователя в систему регистрации курсов.



Основной поток событий

Данный вариант использования начинает выполняться, когда пользователь хочет войти в систему регистрации курсов.

1. Система запрашивает имя пользователя и пароль.
2. Пользователь вводит имя и пароль.
3. Система проверяет имя и пароль, после чего открывается доступ в систему.

Альтернативные потоки

Неправильное имя/пароль. Если во время выполнения **Основного потока** обнаружится, что пользователь ввел неправильное имя и/или пароль, система выводит сообщение об ошибке. Пользователь может вернуться к началу **Основного потока** или отказаться от входа в систему, при этом выполнение варианта использования завершается.

Предусловия

Отсутствуют.

Постусловия

Если вариант использования выполнен успешно, пользователь входит в систему. В противном случае состояние системы не изменяется.

Вариант использования Register for Courses

Краткое описание. Данный вариант использования позволяет студенту зарегистрироваться на конкретные курсы в текущем семестре. Студент может изменить свой выбор (обновить или удалить курсы), если изменение выполняется в установленное время в начале семестра. Система каталога курсов предоставляет список всех конкретных курсов текущего семестра.

Основной поток событий

Данный вариант использования начинает выполняться, когда студент хочет зарегистрироваться на конкретные курсы или изменить свой график курсов.

1. Система запрашивает требуемое действие (создать, обновить, удалить график).
2. Когда студент указывает действие, выполняется один из Подчиненных потоков (создать, обновить, удалить или принять график).

Создать график

1. Система выполняет поиск в каталоге курсов доступных конкретных курсов и выводит их список.
2. Студент выбирает из списка 4 основных и 2 альтернативных курса.
3. После выбора система создает график студента.
4. Выполняется подчиненный поток «Принять график».

Обновить график

1. Система выводит текущий график студента.
2. Система выполняет поиск в каталоге курсов доступных конкретных курсов и выводит их список.
3. Студент может обновить свой выбор курсов, удаляя или добавляя конкретные курсы.
4. После выбора система обновляет график.
5. Выполняется подчиненный поток «Принять график».

Удалить график

1. Система выводит текущий график студента.
2. Система запрашивает у студента подтверждения удаления графика.
3. Студент подтверждает удаление.
4. Система удаляет график. Если график включает конкретные курсы, на которые записался студент, он должен быть удален из списков этих курсов.

Принять график

Для каждого выбранного, но еще не «зафиксированного» конкретного курса в графике система проверяет выполнение студентом предварительных требований (прохождение определенных курсов), факт открытия конкретного курса и отсутствие конфликтов графика. Затем система вносит данные о студенте в список выбранного конкретного курса. Курс фиксируется в графике, и график сохраняется в системе.

Альтернативные потоки

Сохранить график

В любой момент студент может вместо принятия графика сохранить его. В этом случае шаг «Принять график» заменяется на следующий:

1. «Незафиксированные» конкретные курсы помечаются в графике как «выбранные».
2. График сохраняется в системе.

Не выполнены предварительные требования, курс заполнен или имеют место конфликты графика.

Если во время выполнения подчиненного потока «Принять график» система обнаружит, что студент не выполнил необходимые

предварительные требования, или выбранный им конкретный курс заполнен, или имеют место конфликты графика, то выдается сообщение об ошибке. Студент может либо выбрать другой конкретный курс и продолжить выполнение варианта использования, либо сохранить график, либо отменить операцию, после чего основной поток начнется с начала.

График не найден. Если во время выполнения подчиненных потоков «Обновить график» или «Удалить график» система не может найти график студента, то выдается сообщение об ошибке. После того как студент подтвердит это сообщение, основной поток начнется с начала.

Система каталога курсов недоступна. Если окажется, что невозможно установить связь с системой каталога курсов, то будет выдано сообщение об ошибке. После того как студент подтвердит это сообщение, вариант использования завершится.

Регистрация на курсы закончена. Если в самом начале выполнения варианта использования окажется, что регистрация на текущий семестр закончена, будет выдано сообщение, и вариант использования завершится.

Удаление отменено. Если во время выполнения подчиненного потока «Удалить график» студент решит не удалять его, удаление отменяется, и основной поток начнется сначала.

Предусловия

Перед началом выполнения данного варианта использования студент должен войти в систему.

Постусловия

Если вариант использования завершится успешно, график студента будет создан, обновлен или удален. В противном случае состояние системы не изменится.

Вариант использования Close Registration

Краткое описание. Данный вариант использования позволяет регистратору закрывать процесс регистрации. Конкретные курсы, на которые не записалось достаточного количества студентов, отменяются. В расчетную систему передается информация о каждом студенте по каждому конкретному курсу, чтобы студенты могли оплатить курсы.

Основной поток событий

Данный вариант использования начинает выполняться, когда регистратор запрашивает прекращение регистрации.

1. Система проверяет состояние процесса регистрации. Если регистрация еще выполняется, выдается сообщение, и вариант использования завершается.

2. Для каждого конкретного курса система проверяет, ведет ли его какой-либо профессор и записалось ли на него не менее трех студентов. Если эти условия выполняются, система фиксирует конкретный курс в каждом графике, который включает данный курс.

3. Для каждого студенческого графика проверяется наличие в нем максимального количества основных курсов; если их недостаточно, система пытается дополнить альтернативными курсами из списка данного графика. Выбирается первый доступный альтернативный курс. Если таких курсов нет, то никакое дополнение не происходит.

4. Система закрывает все конкретные курсы. Если в каком-либо конкретном курсе оказывается менее трех студентов (с учетом добавлений, сделанных в п.3), система отменяет его и исключает из каждого содержащего его графика.

5. Система рассчитывает плату за обучение для каждого студента в текущем семестре и направляет информацию в расчетную систему. Расчетная система посылает студентам счета для оплаты с копией их окончательных графиков.

Альтернативные потоки

Конкретный курс никто не ведет. Если во время выполнения основного потока обнаруживается, что некоторый конкретный курс не ведется никаким профессором, то этот курс отменяется. Система исключает данный курс из каждого содержащего его графика.

Расчетная система недоступна. Если невозможно установить связь с расчетной системой, спустя некоторое установленное время система вновь попытается связаться с ней. Попытки будут повторяться до тех пор, пока связь не установится.

Предусловия

Перед началом выполнения данного варианта использования регистратор должен войти в систему.

Постусловия

Если вариант использования завершится успешно, регистрация закрывается. В противном случае состояние системы не изменится.

Упражнение 5. Прикрепление файла к варианту использования

1. Щелкните правой кнопкой мыши по варианту использования.
2. В открывшемся меню выберите пункт Open Specification.
3. Перейдите на вкладку файлов.
4. Щелкните правой кнопкой мыши по белому полю и из открывшегося меню выберите пункт Insert File.
5. Укажите созданный ранее файл и нажмите на кнопку Open, чтобы Прикрепить файл к варианту использования.

В результате представление вариантов использования в браузере примет следующий вид (рис. 2.28).

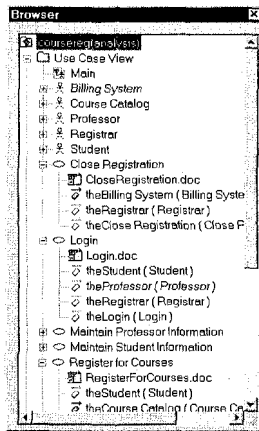


Рис. 2.28. Представление вариантов использования в браузере

Удаление вариантов использования и действующих лиц. Существуют два способа удалить элемент модели - из одной диаграммы или из всей модели. Для удаления элемента модели из диаграммы:

1. Выделите элемент на диаграмме.
2. Нажмите на клавишу Delete.
3. Обратите внимание, что хотя элемент и удален с диаграммы, он остался в браузере и на других диаграммах системы.

Для удаления элемента из модели:

1. Выделите элемент на диаграмме.
2. Выберите пункт меню Edit > Delete from Model или нажмите сочетание клавиш CTRL + D.

2.3.5. АНАЛИЗ СИСТЕМЫ

Архитектурный анализ

Принятие соглашений по моделированию включает:

- используемые диаграммы и элементы модели;
- правила их применения;
- соглашения по именованию элементов;
- организацию модели (пакеты).

Пример соглашений моделирования

- Имена вариантов использования должны быть короткими глагольными фразами.
- Для каждого варианта использования должен быть создан пакет Use-Case Realization, включающий: по крайней мере одну реализацию варианта использования; диаграмму «View Of Participating Classes» (VOPC).
- Имена классов должны быть существительными, соответствующими по возможности понятиям предметной области.
- Имена классов должны начинаться с заглавной буквы.
- Имена атрибутов и операций должны начинаться со строчной буквы.
- Составные имена должны быть сплошными, без подчеркиваний, каждое отдельное слово должно начинаться с заглавной буквы.

Реализация варианта использования (Use-Case Realization)

Описывает реализацию конкретного варианта использования в терминах взаимодействующих объектов и представляется с помощью набора диаграмм (диаграмм классов, реализующих вариант использования, и диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм), отражающих взаимодействие объектов в процессе реализации варианта использования (рис. 2.29).

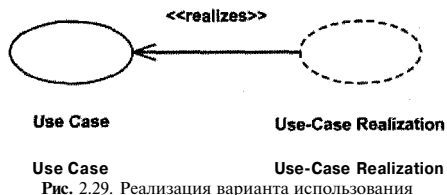


Рис. 2.29. Реализация варианта использования

Идентификация ключевых абстракций. Заключается в предварительном определении классов системы (классов анализа), Источники - знание предметной области, требования к системе, глоссарий. Классы анализа для системы регистрации показаны на рис. 2.30.

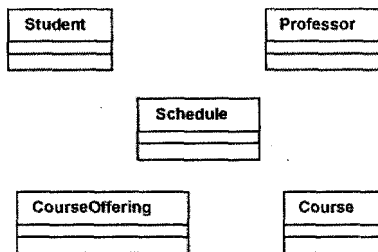


Рис. 2.30. Классы анализа для системы регистрации

Упражнение 6. Создание структуры модели и классов анализа в соответствии с требованиями архитектурного анализа

Структура логического представления браузера должна иметь следующий вид (рис. 2.31).

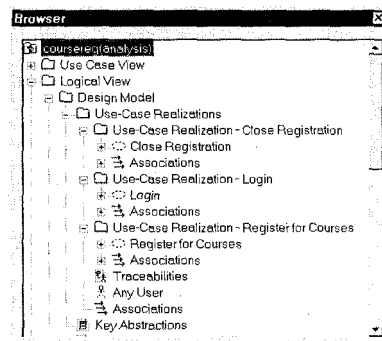


Рис. 2.31. Структура логического представления браузера

Создание пакетов и диаграммы Traceabilities:

1. Щелкните правой кнопкой мыши по логическому представлению браузера.
2. Выберите пункт New > Package в открывшемся меню.
3. Назовите новый пакет Design Model.
4. Создайте аналогичным образом пакеты Use-Case Realizations, Use-Case Realization - Close Registration, Use-Case Realization - Login и Use-Case Realization - Register for Courses.
5. В каждом из пакетов типа Use-Case Realization создайте соответствующие кооперации Close Registration, Login и Register for Courses (каждая кооперация представляет собой вариант использования со стереотипом «use-case realization», который задается в спецификации варианта использования).

6. Создайте в пакете Use-Case Realization новую диаграмму вариантов использования с названием Traceabilities и постройте ее в соответствии с рис. 2.32.

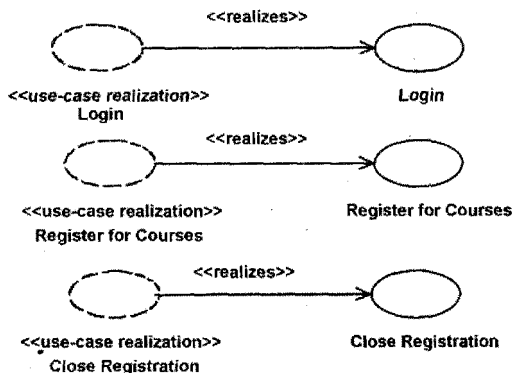


Рис. 2.32. Диаграмма Traceabilities

Создание классов анализа и соответствующей диаграммы Key Abstractions:

1. Щелкните правой кнопкой мыши по пакету Design Model.
2. Выберите пункт New > Class в открывшемся меню. Новый класс под названием NewClass появится в браузере.
3. Выделите его и введите имя Student.
4. Создайте аналогичным образом классы Professor, Schedule, Course и CourseOffering.
5. Щелкните правой кнопкой мыши по пакету Design Model.
6. Выберите пункт New > Class Diagram в открывшемся меню.
7. Назовите новую диаграмму классов Key Abstractions.
8. Чтобы расположить вновь созданные классы на диаграмме классов, откройте ее и перетащите классы на открытую диаграмму мышью. Диаграмма классов должна выглядеть, как на рис. 2.30.

Анализ вариантов использования

Идентификация классов, участвующих в реализации потоков событий варианта использования. В потоках событий варианта использования выявляются классы трех типов:

- **граничные классы (Boundary)** – служат посредниками при взаимодействии внешних объектов с системой. Как правило, для каждой пары «действующее лицо - вариант использования» определяется один граничный класс. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса - кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы, без деталей их реализации);
- **классы-сущности (Entity)** - представляют собой ключевые абстракции (понятия) разрабатываемой системы. Источники выявления классов-сущностей: ключевые абстракции, созданные в процессе архитектурного анализа, глоссарий, описание потоков событий вариантов использования;
- **управляющие классы (Control)** - обеспечивают координацию поведения объектов в системе. Могут отсутствовать в некоторых вариантах использования, ограничивающихся простыми манипуляциями с хранимыми данными. Как правило, для каждого варианта использования определяется один управляющий класс. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

Пример набора классов, участвующих в реализации варианта использования Register for Courses, приведен на рис. 2.33.

Упражнение 7. Создание классов, участвующих в реализации варианта использования Register for Courses, и диаграммы классов «View Of Participating Classes» (VOPC)

1. Щелкните правой кнопкой мыши по пакету Design Model.
2. Выберите пункт New > Class в открывшемся меню. Новый класс под названием NewClass появится в браузере.
3. Выделите его и введите имя RegisterForCoursesForm.
4. Щелкните правой кнопкой мыши по классу RegisterForCoursesForm.
5. Выберите пункт Open Specification в открывшемся меню.
6. В поле стереотипа выберите Boundary и нажмите на кнопку ОК.

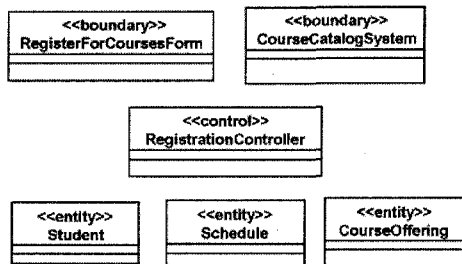


Рис. 2.33. Классы, участвующие в реализации варианта использования Register for Courses

7. Создайте аналогичным образом классы CourseCatalogSystem со стереотипом Boundary и RegistrationController со стереотипом Control.

8. Назначьте классам Schedule, CourseOffering и Student стереотип Entity.

9. Щелкните правой кнопкой мыши по кооперации Register for Courses в пакете Use-Case Realization - Register for Courses.

10. Выберите пункт New > Class Diagram в открывшемся меню.

11. Назовите новую диаграмму классов VOPC (classes only).

12. Откройте ее и перетащите классы на открытую диаграмму в соответствии с рис. 2.33.

Распределение поведения, реализуемого вариантом использования, между классами. Реализуется с помощью диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм). В первую очередь строится диаграмма (одна или более), описывающая основной поток событий и его подчиненные потоки. Для каждого альтернативного потока событий строится отдельная диаграмма.

Примеры:

- обработка ошибок;
- контроль времени выполнения;
- обработка неправильных вводимых данных.

Нечелесообразно описывать тривиальные потоки событий (например, в потоке участвует только один объект).

Упражнение 8. Создание диаграмм взаимодействия

Создадим диаграммы последовательности и кооперативные диаграммы для основного потока событий варианта использования Register for Courses. Готовые диаграммы последовательности должны иметь вид, как на рис. 2.34 - 2.38.

Настройка

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку диаграмм.
3. Контрольные переключатели Sequence Numbering, Collaboration Numbering должны быть помечены, а Focus of Control - нет.
4. Нажмите OK, чтобы выйти из окна параметров.

Создание диаграммы последовательности

1. Щелкните правой кнопкой мыши по кооперации Register for Courses в пакете Use-Case Realization - Register for Courses.
2. Выберите пункт New > Sequence Diagram в открывшемся меню.
3. Назовите новую диаграмму Register for Courses - Basic Flow.
4. Дважды щелкните по ней, чтобы открыть ее.

Добавление на диаграмму действующего лица, объектов и сообщений:

1. Перетащите действующее лицо Student из браузера на диаграмму.
2. Перетащите классы RegisterForCoursesForm и RegistrationController из браузера на диаграмму.
3. На панели инструментов нажмите кнопку Object Message (Сообщение объекта).
4. Проведите мышью от линии жизни действующего лица Student к линии жизни объекта RegisterForCoursesForm.
5. Выделив сообщение, введите его имя: // register for courses.
6. Повторите действия 3 - 5, чтобы поместить на диаграмму остальные сообщения, как показано на рис. 2.34 (для рефлексивного сообщения 3 используется кнопка Message to Self).

Соотнесение сообщений с операциями

1. Щелкните правой кнопкой по сообщению 1, // register for courses.

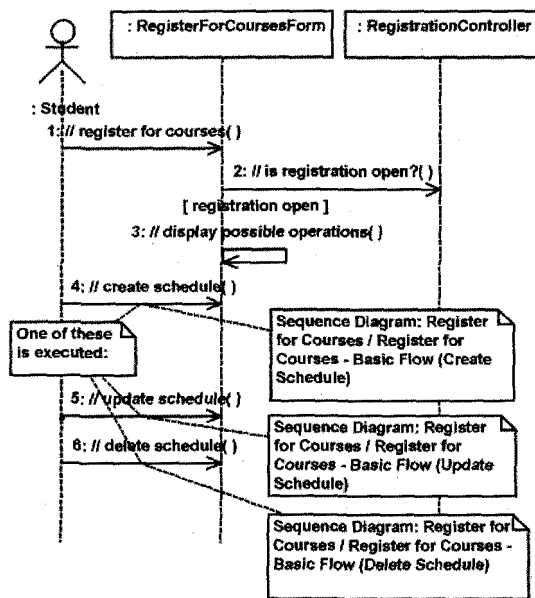
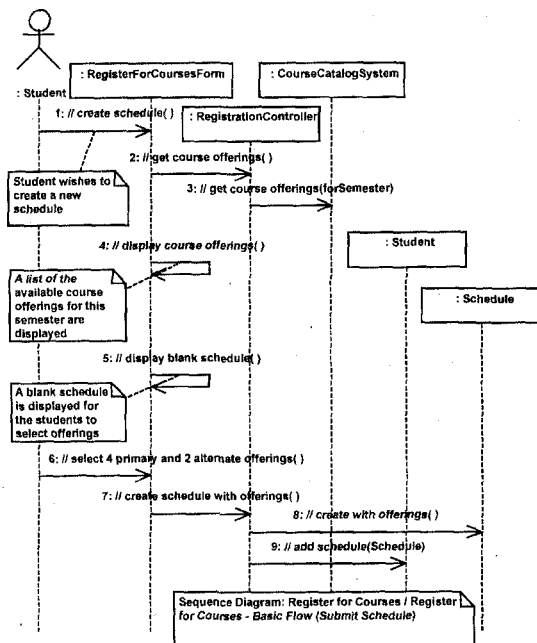


Рис. 2.34. Диаграмма последовательности
Register for Courses - Basic Flow



At this point the Submit Schedule subflow is executed.

Рис. 2.35. Диаграмма последовательности
Register for Courses - Basic Flow (Create Schedule)

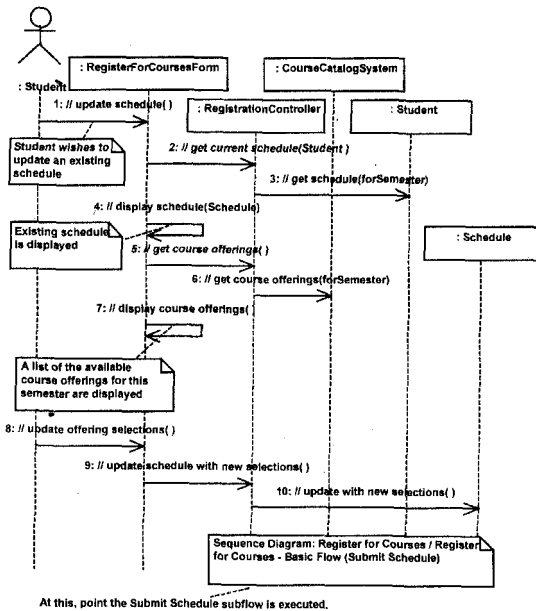


Рис. 2.36. Диаграмма последовательности
Register for Courses - Basic Flow (Update Schedule)

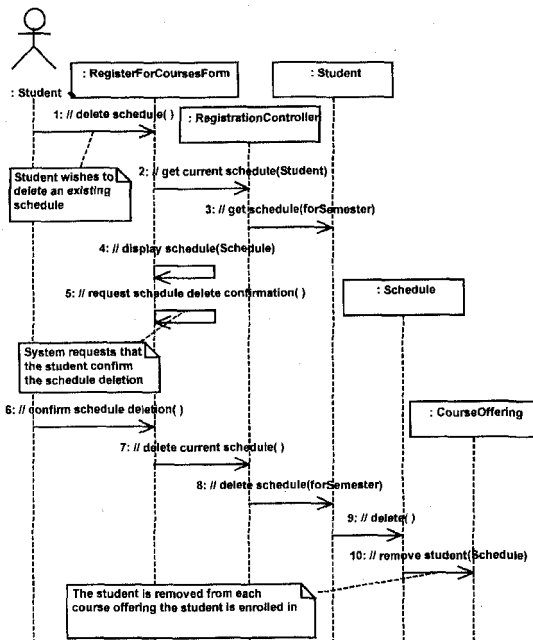


Рис. 2.37. Диаграмма последовательности
Register for Courses - Basic Flow (Delete Schedule)

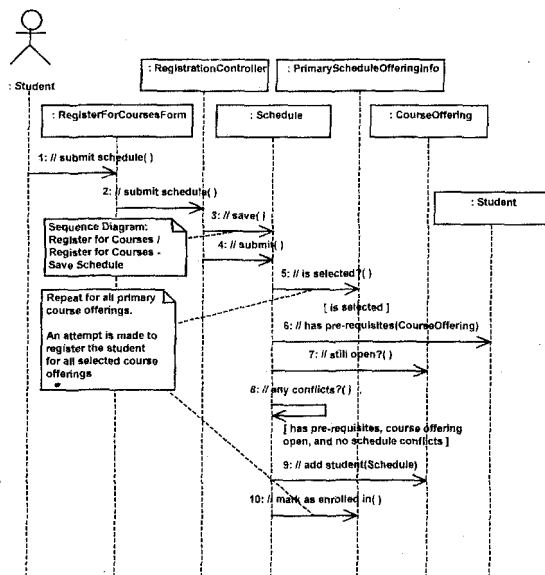


Рис. 2.38. Диаграмма последовательности
Register for Courses - Basic Flow (Submit Schedule)

2. Выберите пункт <new operation> в открывшемся меню. Появится окно спецификации операции.
3. В поле имени оставьте имя сообщения - // register for courses.
4. Нажмите на кнопку OK, чтобы закрыть окно спецификации операции и вернуться на диаграмму.
5. Повторите действия 1-4, пока не соотнесете с операциями все остальные сообщения.

Выполните аналогичные действия для создания диаграмм последовательности, показанных на рис. 2.35 - 2.38. Обратите внимание, что на диаграмме рис. 2.38 появился объект нового класса PrimaryScheduleOfferingInfo (класса ассоциаций, описывающего связь между классами Schedule и OfferingInfo), который нужно предварительно создать.

Создание примечаний

Для того чтобы поместить на диаграмму примечание:

1. Нажмите на панели инструментов кнопку Note.
2. Щелкните мышью по тому месту диаграммы, куда собираетесь поместить примечание.
3. Выделив новое примечание, введите туда текст.
4. Чтобы прикрепить примечание к элементу диаграммы, на панели инструментов нажмите кнопку Anchor Notes To Item (Прикрепить примечания к элементу).
5. Нажав левую кнопку мыши, проведите указатель от примечания до элемента диаграммы, с которым оно будет связано. Между примечанием и элементом возникнет штриховая линия.
6. Чтобы создать примечание-ссылку на другую диаграмму (как это сделано на диаграмме рис. 2.34 и др.), создайте пустое примечание (без текста) и перетащите на него из браузера новую диаграмму.

Кроме примечаний на диаграмму можно поместить также и текстовую область. С ее помощью можно, например, добавить к диаграмме заголовков.

Для того чтобы поместить на диаграмму текстовую область:

1. На панели управления нажмите кнопку Text Box.
2. Щелкните мышью внутри диаграммы, чтобы поместить туда текстовую область.
3. Выделив эту область, введите в нее текст.

Создание кооперативной диаграммы

Для создания кооперативной диаграммы достаточно открыть диаграмму последовательности и нажать клавишу F5.

Определение обязанностей, атрибутов и ассоциаций классов. Обязанность (responsibility) - действие, которое объект обязан выполнять по запросу других объектов. Обязанность преобразуется в одну или более операций класса на шаге проектирования. Обязанности определяются исходя из сообщений на диаграммах взаимодействия и документируются в классах в виде операций «анализа», которые появляются там автоматически в процессе построения диаграмм взаимодействия (соотнесения сообщений с операциями).

Так, диаграмма классов VOPC (classes only) после построения диаграмм взаимодействия в упражнении 8 должна принять следующий вид (рис. 2.39).

Атрибуты классов анализа определяются исходя из знаний о предметной области, требований к системе и глоссария.

Упражнение 9. Добавление атрибутов к классам

Настройка

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Убедитесь, что переключатель Show All Attributes *помечен*.
4. Убедитесь, что переключатели Suppress Attributes и Suppress Operations не помечены.

Добавление атрибутов

1. Щелкните правой кнопкой мыши по классу Student.
2. Выберите пункт New Attribute в открывшемся меню.
3. Введите новый атрибут address.
4. Нажмите клавишу Enter.
5. Повторите шаги 1 - 4, добавив атрибуты name и studentID.
6. Добавьте атрибуты к классам CourseOffering, Schedule и PrimaryScheduleOfferingInfo, как показано на рис. 2.40.

Связи между классами (ассоциации) определяются на основе диаграмм взаимодействия. Если два объекта взаимодействуют

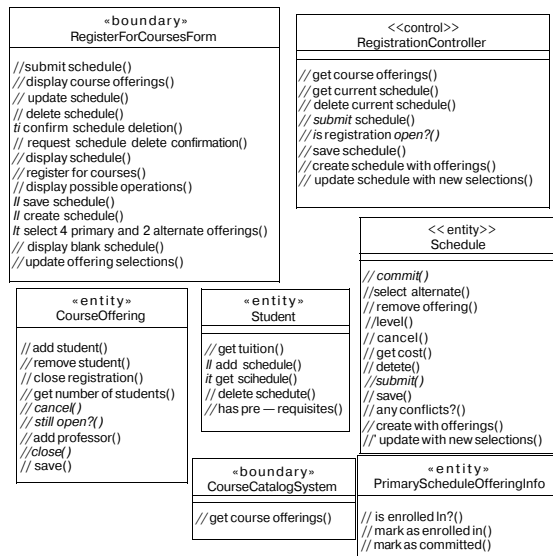


Рис. 2.39. Диаграмма классов VOPC (classes only) с операциями «анализа»

(обмениваются сообщениями), между ними должна существовать связь (путь взаимодействия). Для ассоциаций задаются множественность и, возможно, направление навигации. Могут использоваться множественные ассоциации, агрегации и классы ассоциаций.

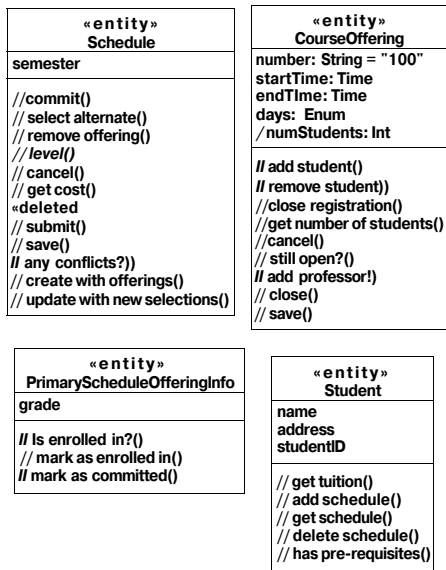


Рис. 2.40. Классы с операциями «анализа» и атрибутами

Упражнение 10. Добавление связей

Добавим связи к классам, принимающим участие в варианте использования Register for Courses. Для отображения связей между классами построим три новые диаграммы классов в кооперации Register for Courses пакета Use-Case Realization - Register for Courses (рис. 2.41 - 2.43).

Добавлены два новых класса - подклассы FulltimeStudent (Студент очного отделения) и ParttimeStudent (Студент вечернего отделения).

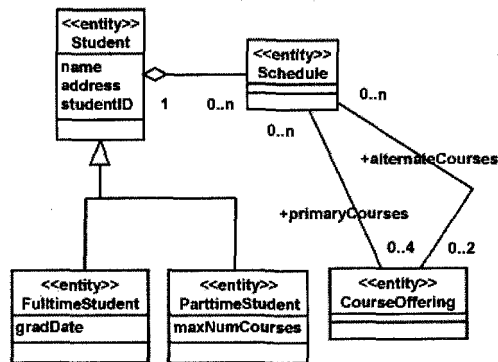


Рис. 2.41. Диаграмма Entity Classes (классы-сущности)

На данной диаграмме показаны классы ассоциаций, описывающие связи между классами Schedule и CourseOffering, и добавлен суперкласс ScheduleOfferingInfo. Данные и операции, содержащиеся в этом классе (status - курс включен в график или отменен), относятся как к основным, так и к альтернативным курсам, в то время как оценка (grade) и окончательное включение курса в график могут иметь место только для основных курсов.

Создание ассоциаций

Ассоциации создают непосредственно на диаграмме классов. Панель инструментов диаграммы классов содержит кнопки для создания как одно-, так и двусторонних ассоциаций. Для создания на диаграмме классов ассоциации сделайте следующее:

1. Нажмите на панели инструментов кнопку Association.
2. Проведите мышью линию ассоциации от одного класса к другому.

С целью задать возможности навигации по ассоциации необходимо выполнить следующие действия:

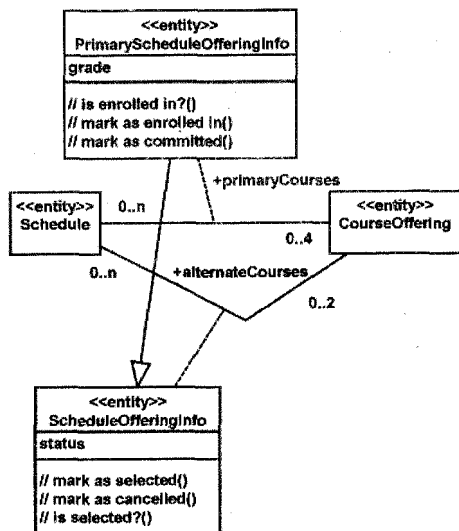


Рис. 2.42. Диаграмма CourseOfferingInfo

1. Щелкните правой кнопкой мыши по связи с того конца, на котором хотите показать стрелку.

2. Выберите пункт Navigable в открывшемся меню.

Для того чтобы создать рефлексивную ассоциацию:

1. На панели инструментов диаграммы нажмите кнопку Association.

2. Проведите линию ассоциации от класса до какого-нибудь места вне класса.

3. Отпустите кнопку мыши.

4. Проведите линию ассоциации назад к классу.

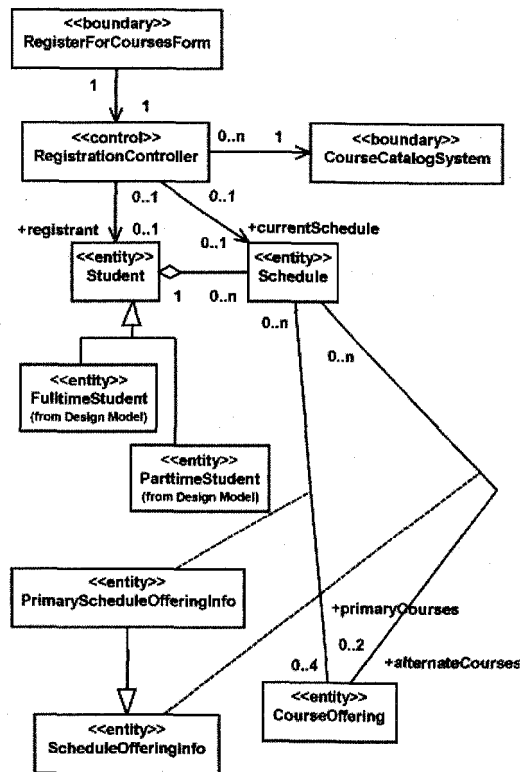


Рис. 2.43. Полная диаграмма классов VOPC (без атрибутов и операций)

Создание агрегаций

1. Нажмите кнопку Aggregation панели инструментов.

2. Проведите линию агрегации от класса-части к целому.

Для того чтобы поместить на диаграмму классов рефлексивную агрегацию:

1. На панели инструментов диаграммы нажмите кнопку Aggregation.

2. Проведите линию агрегации от класса до какого-нибудь места вне класса.

3. Отпустите кнопку мыши.

4. Проведите линию агрегации назад к классу.

Создание обобщений

При создании обобщения может потребоваться перенести некоторые атрибуты или операции из одного класса в другой. Если, например, понадобится перенести их из подкласса в суперкласс Employee, в браузере для этого достаточно просто перетащить атрибуты или операции из одного класса в другой. Не забудьте удалить другую копию атрибута из второго подкласса, если он имеется.

Чтобы поместить обобщение на диаграмму классов:

1. Нажмите кнопку Generalization панели инструментов.

2. Проведите линию обобщения от подкласса к суперклассу.

Спецификации связей

Спецификации связей касаются имен ассоциаций, ролевых имен, множественности и классов ассоциаций.

Для того чтобы задать множественность связи:

1. Щелкните правой кнопкой мыши на одном конце связи.

2. Выберите пункт Multiplicity в открывшемся меню.

3. Укажите нужную множественность.

4. Повторите то же самое для другого конца связи.

Для того чтобы задать имя связи:

1. Выделите нужную связь.

2. Введите ее имя.

Для того чтобы задать связи ролевое имя:

1. Щелкните правой кнопкой мыши на ассоциации с нужного

2. Выберите пункт role Name в открывшемся меню.

3. Введите ролевое имя.

Для того чтобы задать элемент связи (класс ассоциаций):

1. Откройте окно спецификации требуемой связи.

2. Перейдите на вкладку Detail.

3. Задайте элемент связи в поле **Link Element**.

Задание для самостоятельной работы

Выполнить анализ варианта использования Close Registration и построить соответствующие диаграммы взаимодействия и классов.

2.3.6. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Проектирование архитектуры

Цели проектирования архитектуры системы:

- анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;
- уточнение архитектуры с учетом возможностей повторного использования;
- идентификация архитектурных решений и механизмов, необходимых для проектирования системы.

Вводятся глобальные пакеты:

- базисные (foundation) классы (списки, очереди и т.д.);
- обработчики ошибок (error handling classes);
- математические библиотеки;
- утилиты;
- библиотеки других поставщиков.

Определяются проектные классы (design classes):

- класс анализа отображается в проектный класс, если он простой или представляет единственную логическую абстракцию;
- сложный класс анализа может быть разбит на несколько классов, преобразован в пакет или в подсистему.

Примеры возможных подсистем:

- классы, обеспечивающие сложный комплекс услуг (например, обеспечение безопасности и защита);
- граничные классы, реализующие сложный пользовательский интерфейс, или интерфейс с внешними системами;
- различные продукты: коммуникационное ПО (middleware, поддержка COM/CORBA), доступ к базам данных, типы и структуры данных (стеки, списки, очереди), общие утилиты (математические библиотеки), различные прикладные продукты.

Принятие решения о преобразовании класса в подсистему определяется опытом и знаниями архитектора проекта.

Соглашения по проектированию интерфейсов:

- имя интерфейса: короткое (одно-два слова), отражающее его роль в системе;
- описание интерфейса: должно отражать его обязанности (размер - небольшой абзац);
- описание операций: имя, отражающее результат операции, ключевые алгоритмы, возвращаемое значение, параметры с типами;
- документирование интерфейса: характер использования операций и порядок их выполнения (показывается с помощью диаграмм последовательности), тестовые планы и сценарии и т.д. Вся эта информация объединяется в специальный пакет со стереотипом <<subsystem>, который содержит элементы, образующие подсистему, диаграммы последовательности и/или кооперативные диаграммы, описывающие взаимодействие элементов при реализации операций интерфейса, и другие диаграммы;
- класс «subsystem proxy» непосредственно реализует интерфейс и управляет реализацией его операций;
- все интерфейсы должны быть полностью определены в процессе проектирования архитектуры, поскольку они будут служить в качестве точек синхронизации при параллельной разработке.

Выделение архитектурных уровней:

Application Layer - содержит элементы прикладного уровня (пользовательский интерфейс);

Business Services Layer - содержит элементы, реализующие бизнес-логику приложений (наиболее устойчивая часть системы);

Middleware Layer - обеспечивает сервисы, не зависящие от платформы.

Пример выделения архитектурных уровней и объединения элементов модели в пакеты для системы регистрации приведен на рис. 2.44.

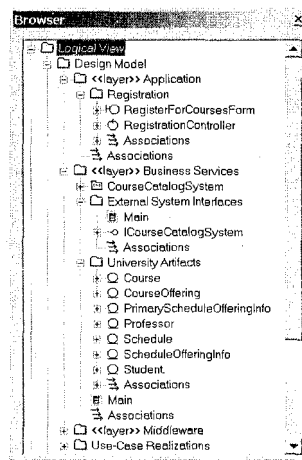


Рис. 2.44. Структура логического представления модели на шаге проектирования

Для того чтобы поместить класс в пакет, достаточно просто перетащить его в браузер на нужный пакет.

Данное представление отражает следующие решения, принятые архитектором:

- выделены три архитектурных уровня (созданы три пакета со стереотипом <<layer>>);
- в пакете Application создан пакет Registration, куда включены граничные и управляющие классы;

- граничный класс CourseCatalogSystem преобразован в подсистему (пакет CourseCatalogSystem со стереотипом «subsystem»);
- в пакет Business Services, помимо подсистемы CourseCatalogSystem, включены еще два пакета: пакет External System Interfaces включает интерфейс с подсистемой CourseCatalogSystem (класс ICourseCatalogSystem со стереотипом «Interface»>>), а пакет University Artifacts - все классы-сущности.

Структура и диаграммы пакета (подсистемы) CourseCatalogSystem показаны на рис. 2.45 - 2.49.

Для того чтобы поместить зависимость между пакетами на диаграмму классов:

1. Нажмите кнопку Dependency панели инструментов.
2. Проведите линию зависимости от зависимого пакета к тому, от которого он зависит.

Класс DBCourseOffering отвечает за взаимодействие с БД каталога курсов (рис. 2.48, 2.49).

Моделирование распределенной конфигурации системы

Распределенная конфигурация системы моделируется с помощью диаграммы размещения. Ее основные элементы:

- узел (node) - вычислительный ресурс (процессор или другое устройство (дисковая память, контроллеры различных устройств и т.д.). Для узла можно задать выполняющиеся на нем процессы;

- соединение (connection) - канал взаимодействия узлов (сеть).

Пример: сетевая конфигурация системы регистрации (без процессов) (рис. 2.50).

Распределение процессов по узлам сети производится с учетом следующих факторов:

- используемые образцы распределения (трехзвенная клиент-серверная конфигурация, «толстый» клиент, «тонкий» клиент, равноправные узлы (peer-to-peer) и т.д.);

- время отклика;
- минимизация сетевого трафика;
- мощность узла;
- надежность оборудования и коммуникаций.

Пример: распределение процессов по узлам (рис. 2.51).

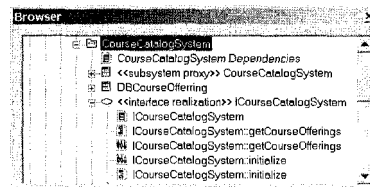


Рис. 2.45. Структура пакета CourseCatalogSystem

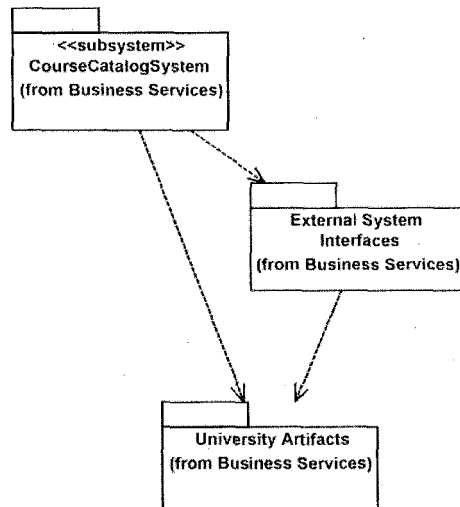


Рис. 2.46. Зависимости между подсистемой и другими пакетами (диаграмма классов CourseCatalogSystem Dependencies)

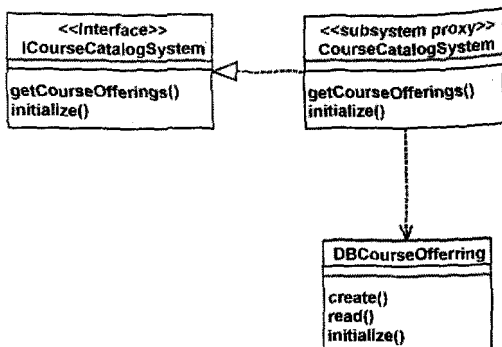


Рис. 2.47. Классы, реализующие интерфейс подсистемы (диаграмма классов ICourseCatalogSystem)

Упражнение 11. Создание диаграммы размещения системы регистрации

Для того чтобы открыть диаграмму размещения, надо дважды щелкнуть мышью по представлению Deployment View (представлению размещения) в браузере.

Для того чтобы поместить на диаграмму процессор:

1. На панели инструментов диаграммы нажмите кнопку Processor.

2. Щелкните по диаграмме размещения в том месте, куда хотите поместить процессор.

3. Введите имя процессора.

В спецификациях процессора можно ввести информацию о его стереотипе, характеристиках и планировании. Стереотипы применяются для классификации процессоров (например, компьютеров под управлением UNIX или ПК).

Характеристики процессора - это его физическое описание. Оно может, в частности, включать скорость процессора и объем памяти.

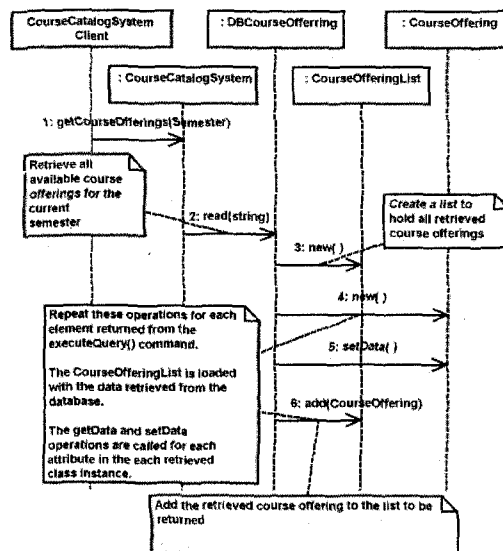


Рис. 2.48. Диаграмма последовательности ICourseCatalogSystem: getCourseOfferings, описывающая взаимодействие элементов при реализации операции интерфейса getCourseOfferings

Поле планирования (scheduling) процессора содержит описание того, как осуществляется планирование его процессов:

- **Preemptive** (с приоритетом). Высокоприоритетные процессы имеют преимущество перед низкоприоритетными.
- **Non preemptive (без приоритета)**. У процессов не имеется приоритета. Текущий процесс выполняется до его завершения, после чего начинается следующий.

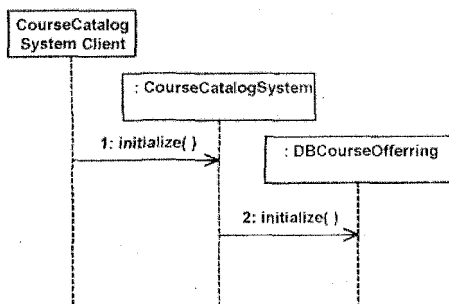


Рис. 2.49. Диаграмма последовательности ICourseCatalogSystem::initialize, описывающая взаимодействие элементов при реализации операции интерфейса initialize

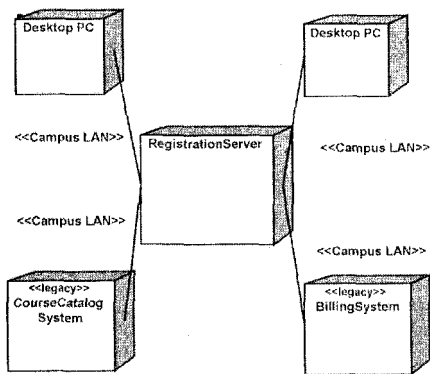


Рис. 2.50. Сетевая конфигурация системы регистрации

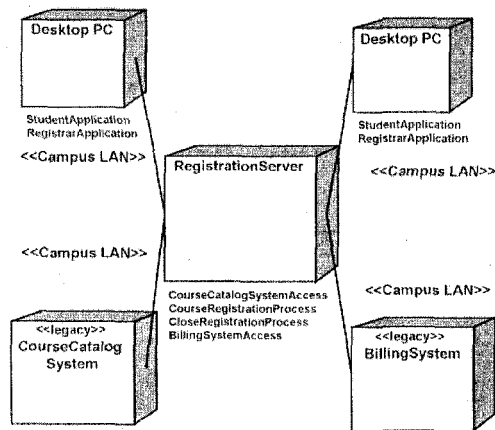


Рис. 2.51. Сетевая конфигурация системы регистрации с распределением процессов по узлам

- **Cyclic (циклический).** Управление передается между процессами по кругу. Каждому процессу дается определенное время на его выполнение, затем управление переходит к следующему процессу.
- **Executive (исполнительный).** Существует некоторый вычислительный алгоритм, который и управляет планированием процессов.
- **Manual (вручную).** Процессы планируются пользователем. Для того чтобы назначить процессору стереотип:
 1. Откройте окно спецификации процессора.
 2. Перейдите на вкладку General.
 3. Введите стереотип в поле Stereotype.

Для введения характеристик и планирования процессора:

1. Откройте окно спецификации процессора.
2. Перейдите на вкладку Detail.
3. Введите характеристики в поле характеристик.
4. Укажите один из типов планирования.

Для того чтобы показать планирование на диаграмме:

1. Щелкните правой кнопкой мыши по процессору.
2. Выберите пункт Show Scheduling в открывшемся меню.

Для того чтобы добавить связь на диаграмму:

1. На панели инструментов нажмите кнопку Connection.
2. Щелкните по узлу диаграммы.
3. Проведите линию связи к другому узлу.

Для того чтобы назначить связи стереотипа:

1. Откройте окно спецификации связи.
2. Перейдите на вкладку General.
3. Введите стереотип в поле Stereotype (Стереотип).

Для того чтобы добавить процесс:

1. Щелкните правой кнопкой мыши по процессору в браузере.
2. Выберите пункт New > Process в открывшемся меню.
3. Введите имя нового процесса.

Для того чтобы показать процессы на диаграмме:

1. Щелкните правой кнопкой мыши по процессору.
2. Выберите пункт Show Processes в открывшемся меню.

Проектирование классов

Классы анализа преобразуются в проектные классы:

- Проектирование граничных классов - зависит от возможностей среды разработки пользовательского интерфейса (GUI Builder).

- Проектирование классов-сущностей - с учетом соображений производительности (выделение в отдельные классы атрибутов с различной частотой использования).

- Проектирование управляющих классов - удаление классов, реализующих простую передачу информации от граничных классов к сущностям.

- Идентификация устойчивых (persistent) классов, содержащих хранимую информацию.

Обязанности классов, определенные в процессе анализа, преобразуются в операции. Каждой операции присваивается имя, характеризующее ее результат. Определяется полная сигнатура операции: `operationName(parameter:class,...):returnType`. Создается краткое описание операции, включая смысл всех ее параметров. Определяется видимость операции: `public`, `private`, `protected`. Определяется область действия (scope) операции: экземпляр или классификатор.

Определяются (уточняются) атрибуты классов:

- Кроме имени, задаются тип и значение по умолчанию (необязательное): `attributeName:Type = Default`.
- Учитываются соглашения по именованию атрибутов, принятые в проекте и языке реализации.
- Задается видимость атрибутов: `public`, `private`, `protected`.
- При необходимости определяются производные (вычисляемые) атрибуты.

Пример определения операций и атрибутов (рис. 2.52).

«entity» Student
- name: string - address: siting «class» - nextAvailID : int - studentID: int - dateofBirth: Date
+ getTurtion(): double + addSchedule(theSchedule : Schedule) + getSchedule(forSemester: Semester): Schedule + deleteSchedule(forSemester: Semester) + hasPrerequisites(forCourseOffering: CourseOffering): boolean # passed(theCourseOffering : CourseOffering): boolean «class» + getNextAvailID(): int + getStudentID(): int + getName(): string + getAddress(): string

Рис. 2.52. Класс Student с полностью определенными операциями и атрибутами

Упражнение 12. Определение атрибутов и операций для класса Student

Задать тип данных, значение по умолчанию и видимость атрибута можно с помощью следующих действий:

1. Щелкните правой кнопкой мыши по атрибуту в браузере.
2. Выберите пункт Open Specification в открывшемся меню.
3. Укажите тип данных в раскрывающемся списке типов или введите собственный тип данных.
4. В поле Initial Field (Первоначальное значение) введите значение атрибута по умолчанию.

5. В поле Export Control выберите видимость атрибута: Public, Protected, Private или Implementation. По умолчанию видимость всех атрибутов соответствует Private.

С целью изменить нотацию для обозначения видимости:

1. Выберите пункт Tools > Options в меню модели.
2. Перейдите на вкладку Notation.
3. Поставьте контрольный переключатель Visibility as Icons, чтобы использовать нотацию Rose, или снимите пометку, чтобы использовать нотацию UML.

Примечание. Изменение значения этого параметра приведет к смене нотации только для новых диаграмм и не затронет уже существующие диаграммы.

Для того чтобы задать тип возвращаемого значения стереотип и видимость операции:

1. Щелкните правой кнопкой мыши по операции в браузере.
2. Откройте окно спецификации класса этой операции.
3. Укажите тип возвращаемого значения в раскрывающемся списке или введите свой тип.
4. Укажите стереотип в соответствующем раскрывающемся списке или введите новый.
5. В поле Export Control укажите значение видимости операции: Public, Protected, Private или Implementation. По умолчанию видимость всех операций установлена в public.

Для того чтобы добавить к операции аргумент:

1. Откройте окно спецификации операции.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши по области аргументов, в открывшемся меню выберите пункт Insert.

4. Введите имя аргумента.
 5. Щелкните по колонке Data type и введите туда тип данных аргумента.
 6. При необходимости щелкните по колонке default и введите значение аргумента по умолчанию.
- Определение состояний для классов: моделируется с помощью диаграмм состояний.

Диаграммы состояний создаются для описания объектов с высоким уровнем динамического поведения.

В качестве примера рассмотрим поведение объекта класса CourseOffering. Он может находиться в открытом состоянии (возможно добавление данных о новом студенте) или в закрытом состоянии (максимальное количество студентов уже записалось на курс). Таким образом, конкретное состояние зависит от количества студентов, связанных с объектом CourseOffering. Рассматривая каждый вариант использования, можно выделить еще два состояния: инициализацию (до начала регистрации студентов на курс) и отмену (курс исключается из расписания).

Диаграмма состояний для класса CourseOffering приведена на рис. 2.53.

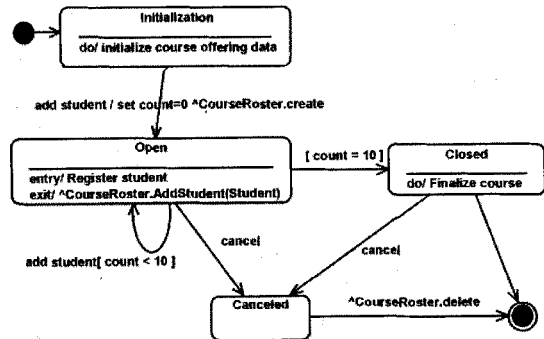


Рис. 2.53. Диаграмма состояний для класса CourseOffering

Упражнение 13. Создание диаграммы состояний для класса CourseOffering

Для создания диаграммы состояний:

1. Щелкните правой кнопкой мыши в браузере по нужному классу.
2. Выберите пункт New > Statechart Diagram в открывшемся меню.

Для того чтобы добавить состояние:

1. На панели инструментов нажмите кнопку State.
2. Щелкните мышью на диаграмме состояний по тому месту, куда хотите поместить состояние.

Все элементы состояния можно добавить с помощью вкладки Detail окна спецификации состояния.

Для того чтобы добавить деятельность:

1. Откройте окно спецификации требуемого состояния.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши по окну Actions.
4. Выберите пункт Insert в открывшемся меню.
5. Дважды щелкните по новому действию.
6. Введите действие в поле Actions.
7. В окне When укажите Do, чтобы сделать новое действие деятельностью.

Для того чтобы добавить входное действие, в окне When укажите On Entry.

Для того чтобы добавить выходное действие, в окне When укажите On Exit.

Послать событие можно с помощью следующих операций:

1. Откройте окно спецификации требуемого состояния.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши по окну Actions.
4. Выберите пункт Insert в открывшемся меню.
5. Дважды щелкните по новому действию.
6. В качестве типа действия укажите Send Event.
7. В соответствующие поля введите событие (event), аргументы (arguments) и целевой объект (Target).

Для того чтобы добавить переход:

1. Нажмите кнопку Transition панели инструментов.
2. Щелкните мышью по состоянию, откуда осуществляется переход.

3. Проведите линию перехода до того состояния, где он завершается.

Чтобы добавить рефлексивный переход:

1. Нажмите кнопку Transition to Self панели инструментов.
2. Щелкните мышью по тому состоянию, где осуществляется рефлексивный переход.

Для того чтобы добавить событие, его аргументы, ограждающее условие и действие:

1. Дважды щелкните по переходу, чтобы открыть окно его спецификации.
2. Перейдите на вкладку General.
3. Введите событие в поле Event.
4. Введите аргументы в поле Arguments.
5. Введите ограждающее условие в поле Condition.
6. Введите действие в поле Action.

Для отправки события:

1. Дважды щелкните по переходу, чтобы открыть окно его спецификации.
2. Перейдите на вкладку Detail.
3. Введите событие в поле Send Event.
4. Введите аргументы в поле Send Arguments.
5. Задайте цель в поле Send Target.

Для указания начального или конечного состояния:

1. На панели инструментов нажмите кнопку Start State или End State.
2. Щелкните мышью на диаграмме состояний по тому месту, куда хотите поместить состояние.

Уточнение ассоциаций: некоторые ассоциации (семантические, структурные, устойчивые связи по данным) могут быть преобразованы в зависимости (неструктурные, временные связи отражают видимость), а агрегации - в композиции (рис. 2.54).

Для преобразования агрегации в композицию:

1. Щелкните правой кнопкой мыши по тому концу агрегации, который упирается в класс-часть (см. рис.2.54 - Schedule).
2. Выберите пункт Containment в открывшемся меню.
3. Укажите метод включения By Value.

Примечание. Значение By Value предполагает, что целое и часть создаются и разрушаются одновременно, что соответствует композиции. Агрегация (By Reference) предполагает, что целое и часть создаются и разрушаются в разное время.

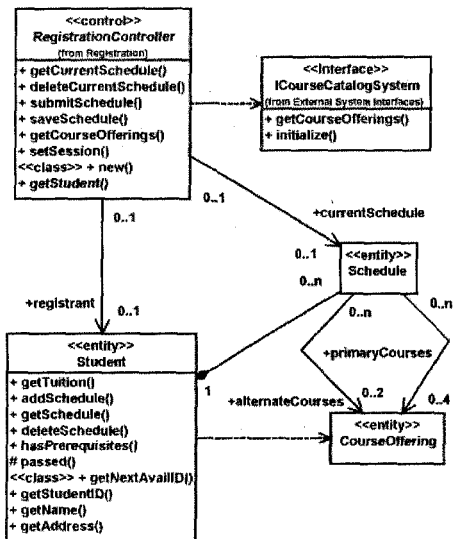


Рис. 2.54. Пример преобразования ассоциаций и агрегаций

Уточнение обобщений: в случае ситуации с миграцией под-классов (студент может переходить с очной формы обучения на вечернюю) иерархия наследования реализуется так, как показано на рис. 2.55. Такое решение повышает устойчивость системы (не нужно модифицировать описание объекта).

Проектирование баз данных

Проектирование реляционных баз данных выполняется с использованием средства Data Modeler. Его работа основана на

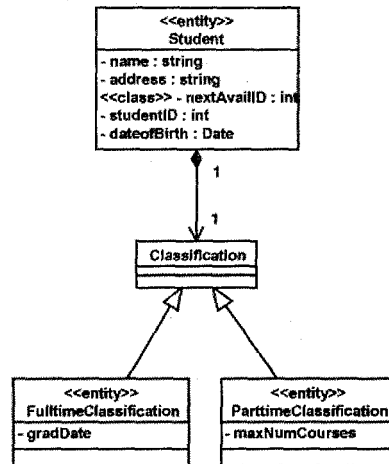


Рис. 2.55. Преобразование обобщения

известном механизме отображения объектной модели в реляционную. Результатом являются построение диаграммы «сущность-связь» и последующая генерация описания БД на SQL.

Упражнение 14. Проектирование реляционной базы данных

Проектирование БД состоит из следующих шагов.

Создание нового компонента - базы данных:

1. Щелкните правой кнопкой мыши на представлении компонентов.
2. Выберите пункт Data Modeler > New > Database в открывшемся меню.
3. Откройте окно спецификации вновь созданного компонента DB_0 и в списке Target выберите Oracle 8.x.

Определение устойчивых (persistent) классов:

1. Откройте окно спецификации класса Student в пакете University Artifacts.

2. Перейдите на вкладку Detail.

3. Установите значение переключателя Persistence в Persistent.

4. Прodelайте такие же действия для классов Classification, FulltimeClassification и ParttimeClassification.

5. Откройте класс Student в браузере, нажав «+».

6. Щелкните правой кнопкой мыши по атрибуту studentID.

7. Выберите пункт Data Modeler > Part of Object Identity (указание атрибута в качестве части первичного ключа) в открывшемся меню.

Примечание. Шаги 5,6 и 7 можно выполнять в Rational Rose, начиная с версии 2001.

Создание схемы БД:

1. Щелкните правой кнопкой мыши по пакету University Artifacts.

2. Выберите пункт Data Modeler > Transform to Data Model в открывшемся меню.

3. В появившемся окне в списке Target Database укажите DB_0 и нажмите ОК. В результате в логическом представлении появится новый пакет Schemas.

4. Откройте пакет Schemas и щелкните правой кнопкой мыши по пакету «Schema» S_0.

5. Выберите пункт Data Modeler > New > Data Model Diagram в открывшемся меню.

6. Откройте пакет, затем откройте вновь созданную диаграмму «сущность-связь» NewDiagram и перенесите на нее все классы-таблицы, находящиеся в пакете «Schema» S_0. Получившаяся диаграмма показана на рис. 2.56.

Генерация описания БД на SQL:

1. Щелкните правой кнопкой мыши по пакету «Schema» S_0.

2. Выберите пункт Data Modeler > Forward Engineer в открывшемся меню.

3. В открывшемся окне мастера Forward Engineering Wizard нажмите Next.

4. Отметьте все флажки генерации DDL и нажмите Next.

5. Укажите имя и расположение текстового файла с результатами генерации и нажмите Next.

6. После завершения генерации откройте созданный текстовый файл и просмотрите результаты.

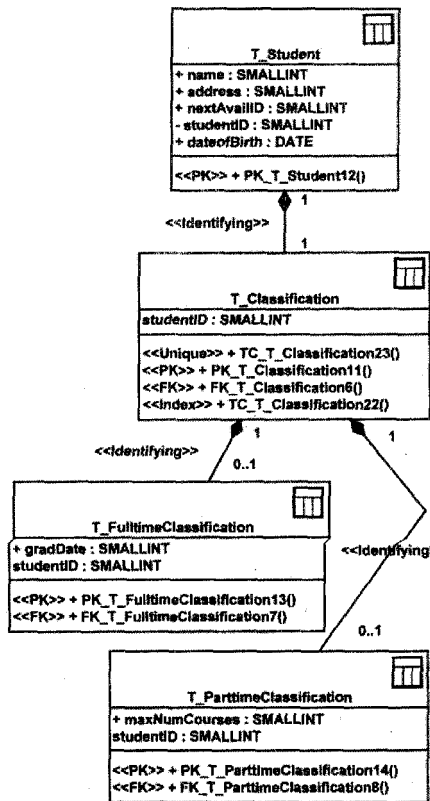


Рис. 2.56. Диаграмма «сущность-связь»

2.3.7. РЕАЛИЗАЦИЯ СИСТЕМЫ

Создание компонентов

В Rational Rose диаграммы компонентов создаются в представлении компонентов системы. Отдельные компоненты можно создавать непосредственно на диаграмме, или перетаскивать их туда из браузера.

Упражнение 15. Создание компонентов

Выберем в качестве языка программирования C++ и для класса Student создадим соответствующие этому языку компоненты. Создание диаграммы компонентов:

1. Дважды щелкните мышью по главной диаграмме компонентов в представлении компонентов.
2. На панели инструментов нажмите кнопку Package Specification.
3. Поместите спецификацию пакета на диаграмму.
4. Введите имя спецификации пакета Student и укажите в окне спецификации язык C++.
5. На панели инструментов нажмите кнопку Package Body.
6. Поместите тело пакета на диаграмму.
7. Введите имя тела пакета Student и укажите в окне спецификации язык C++.
8. На панели инструментов нажмите кнопку Dependency.
9. Проведите линию зависимости от тела пакета к спецификации пакета.

Результат показан на рис. 2.57.

Соотнесение классов с компонентами:

1. В логическом представлении браузера найдите класс Student.
2. Перетащите этот класс на спецификацию пакета компонента Student в представлении компонентов браузера. В результате класс Student будет соотнесен со спецификацией пакета компонента Student.
3. Перетащите класс Student на тело пакета компонента Student в представлении компонентов браузера. В результате класс Student будет соотнесен с телом пакета компонента Student.

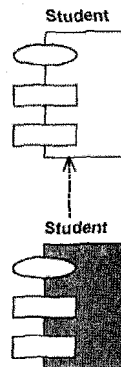


Рис. 2.57. Диаграмма компонентов

Генерация кода

Процесс генерации кода состоит из четырех основных шагов:

1. Проверка корректности модели.
2. Установка свойств генерации кода.
3. Выбор класса, компонента или пакета.
4. Генерация кода.

Для проверки модели:

1. Выберите в меню Tools > Check Model.
2. Проанализируйте все найденные ошибки в окне журнала.

К наиболее распространенным ошибкам относятся такие, например, как сообщения на диаграмме последовательности или кооперативной диаграмме, не соотнесенные с операцией, либо объекты этих диаграмм, не соотнесенные с классом.

С помощью пункта меню Check Model можно выявить большую часть неточностей и ошибок в модели. Пункт меню Access Violations позволяет обнаруживать нарушения правил доступа, возникающие тогда, когда существует связь между двумя классами разных пакетов, но связи между самими пакетами нет.

Для того чтобы обнаружить нарушение правил доступа:

1. Выберите в меню Report > Show Access Violations.
2. Проанализируйте все нарушения правил доступа в окне.

Можно установить несколько параметров генерации кода для классов, атрибутов, компонентов и других элементов модели. Этими свойствами определяется способ генерации программ. Для каждого языка в Rose предусмотрен ряд определенных свойств генерации кода. Перед генерацией кода рекомендуется анализировать эти свойства и вносить необходимые изменения.

Для анализа свойств генерации кода выберите Tools > Options, а затем вкладку соответствующего языка. В окне списка можно выбрать класс, атрибут, операцию и другие элементы модели. Для каждого языка в этом списке указаны свои собственные элементы модели. При выборе разных значений на экране появляются разные наборы свойств.

Любые изменения, вносимые в набор свойств в окне Tools > Options, воздействуют на все элементы модели, для которых используется данный набор.

Изменить свойства генерации кода для одного класса, атрибута, одной операции и т.д. можно, открыв окно спецификации элемента модели. Выберите вкладку языка (C++, Java,...) и измените свойства. Все изменения, вносимые в окне спецификации элемента модели, оказывают влияние только на этот элемент.

При генерации кода за один раз можно создать класс, компонент или целый пакет. Код генерируется с помощью диаграммы или браузера. При генерации кода из пакета можно выбрать или пакет логического представления на диаграмме классов, или пакет представления компонентов на диаграмме компонентов. При выборе пакета логического представления генерируются все классы этого пакета. При выборе пакета представления компонентов генерируются все компоненты этого пакета.

После выбора класса или компонента на диаграмме выберите в меню соответствующий вариант генерации кода. Сообщения об ошибках, возникающих в процессе генерации кода, будут появляться в окне журнала.

Во время генерации кода Rose выбирает информацию из логического и компонентного представлений модели и генерирует большой объем «скелетного» (skeletal) кода:

- **Классы.** Генерируются все классы модели.

- **Атрибуты.** Код включает атрибуты каждого класса, в том числе видимость, тип данных и значение по умолчанию.
- **Сигнатуры операций.** Код содержит определения операций со всеми параметрами, типами данных параметров и типом возвращаемого значения операции.
- **Связи.** Некоторые из связей модели вызывают создание атрибутов при генерации кода.
- **Компоненты.** Каждый компонент реализуется в виде соответствующего файла с исходным кодом.

Упражнение 16. Генерация кода C++

1. Откройте диаграмму компонентов системы.
2. Выберите все объекты на диаграмме компонентов.
3. Выберите Tools > C++ > Code Generation в меню.
4. Выполните генерацию кода.
5. Просмотрите результаты генерации (меню Tools > C++ > Browse Header и Tools > C++ > Browse Body).

2.4. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА

Задание 1. Построить диаграмму вариантов использования для системы обработки заказов

Компания - торговый посредник, продающая товары различных производителей, разрабатывает систему обработки заказов. Дважды в год компания публикует каталог продуктов, который рассылается клиентам и другим заинтересованным лицам.

Клиенты приобретают товары, направляя в компанию перечень продуктов с информацией об оплате. Компания выполняет заказы и отправляет товары по адресам клиентов.

Система должна отслеживать заказ от момента его получения до отправки товара.

Клиенты могут возвращать товары, оплачивая, возможно, при этом некоторые издержки.

Часть клиентов заказывает товары через Интернет.

Компания пользуется услугами различных транспортных и страховых компаний.

Задача 2. Построить диаграмму вариантов использования для системы кредитования коммерческого банка

Небольшой банк автоматизирует деятельность, связанную с кредитованием физических и юридических лиц (индивидуальных клиентов и организаций).

В настоящее время кандидат на получение кредита заполняет бумажную форму, прикладывает необходимые документы (финансовый отчет, перспективную оценку финансового состояния и др.) и отправляет в банк. Референт по кредитованию анализирует запрос на предмет возможных ошибок и подтверждает его достоверность.

Затем референт запрашивает отчет о кредитных операциях клиента в отделе кредитования. Копия отчета просматривается банковским служащим, а референт проверяет финансовое положение и доход клиента. Служащий также обращается к существующей системе управления счетами клиентов, чтобы получить необходимую информацию о состоянии счета и предыдущих кредитах клиента.

Вся информация комплектуется в кредитный запрос и направляется для оценки инспектору по кредитам. Если запрос утверждается, инспектор определяет наилучшие условия кредитования и уведомляет об этом клиента. Если клиент принимает условия, то кредит оформляется.

На обработку запроса обычно уходит минимум две недели (как для индивидуальных клиентов, так и для организаций).

Цель автоматизации - сократить время обработки запроса до 48 часов для индивидуальных клиентов и 72 часов для организаций, сократить количество сотрудников, занятых в процессе обработки, и увеличить количество запросов, обрабатываемых в заданный период.

Задача 3. Выполнить учебный проект аналогично п. 2.3 для системы начисления зарплаты

Перед информационной службой компании поставлена задача создания новой системы начисления зарплаты взамен морально устаревшей существующей системы. Новая система должна

предоставлять служащим возможность записывать электронным способом информацию из карточки учета рабочего времени и автоматически формировать чеки на оплату, учитывающие количество отработанных часов и общий объем продаж (для служащих, получающих комиссионное вознаграждение).

Новая система должна предоставлять служащим возможность вводить информацию из карточки учета рабочего времени, вводить заказы на поставку, изменять свои параметры (такие, как способ оплаты за работу) и формировать различные отчеты. Система должна работать на персональных компьютерах служащих всей компании. В целях обеспечения безопасности и аудита служащие должны иметь возможность доступа и редактирования только своих собственных карточек учета рабочего времени и заказов на поставку.

В системе должна храниться информация обо всех служащих компании в различных странах. Система должна обеспечивать правильную и своевременную оплату работы каждого служащего в соответствии с указанным им способом. Компания из соображений экономии желает сохранить без изменений одну из существующих баз данных (БД управления проектами), которая содержит всю информацию относительно проектов и тарифов. БД управления проектами функционирует в среде DB2 на мейнфрейме IBM. Новая система может читать данные из БД управления проектами, но не может обновлять их.

Часть служащих получает почасовую оплату. Она начисляется на основе карточек учета рабочего времени, каждая из которых содержит дату и количество часов, отработанных в соответствии с конкретным тарифом. Если какой-либо служащий отработал в день более 8 часов, сверхурочное время оплачивается с коэффициентом 1,5. Служащие-почасовики получают зарплату каждую пятницу.

Некоторые служащие получают фиксированный оклад, однако они тоже представляют свои карточки учета рабочего времени. Благодаря этому система может вести учет количества часов, отработанных в соответствии с конкретными тарифами. Такие служащие получают зарплату в последний рабочий день месяца.

Некоторые из служащих с фиксированным окладом также получают комиссионное вознаграждение, учитывающее объем продаж. Они представляют заказы на поставку, отражающие дату

и объем продаж. Процент комиссионного вознаграждения определяется индивидуально для каждого служащего и может составлять 10, 15, 25 или 35%.

Одной из наиболее часто используемых возможностей новой системы является формирование различных отчетов: запросить количество отработанных часов, суммарную зарплату, оставшееся время отпуска и т.д.

Служащие имеют право выбирать способ оплаты за работу. Они могут получать свои чеки на оплату по почте, на счет в банке или на руки в офисе.

Администратор системы курирует информацию о служащих. В его обязанности входят ввод данных о новых служащих, удаление данных и изменение любой информации о служащем, такой, как имя, адрес и способ оплаты, а также формирование различных отчетов для руководства.

Приложение начисления зарплаты запускается автоматически каждую пятницу и в последний рабочий день месяца, рассчитывая в эти дни зарплату соответствующих служащих. Начисление зарплаты должно производиться автоматически, без ручной обработки.

Система не должна позволять служащим изменять любые карточки учета рабочего времени, кроме своих собственных. Только администратор системы может изменять любую информацию о служащих, за исключением способа доставки оплаты.

Система должна взаимодействовать с существующей банковской системой посредством интерфейса электронных транзакций.

Система должна обеспечивать Windows-совместимый пользовательский интерфейс.

Задание 4. Выполнить учебный проект для постановки задачи, приведенной в п. 1.1.

МЕТОДИКИ ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. МЕТОДИКА ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ НА ОСНОВЕ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Данная методика основана на материалах компании Rational Software.

3.1.1. ОПРЕДЕЛЕНИЕ ВЕСОВЫХ ПОКАЗАТЕЛЕЙ ДЕЙСТВУЮЩИХ ЛИЦ

Все действующие лица системы делятся на три типа: простые, средние и сложные.

- Простое действующее лицо представляет внешнюю систему с четко определенным программным интерфейсом.
- Среднее действующее лицо представляет либо внешнюю систему, взаимодействующую с данной системой посредством протокола наподобие TCP/IP, либо личность, пользующуюся текстовым интерфейсом (например, алфавитно-цифровым терминалом).
- Сложное действующее лицо представляет личность, пользующуюся графическим пользовательским интерфейсом.

Общее количество действующих лиц каждого типа умножается на соответствующий весовой коэффициент, затем вычисляется общий весовой показатель (табл. 3.1).

Таблица 3.1

Весовые коэффициенты действующих лиц

Тип действующего лица	Весовой коэффициент
Простое	1
Среднее	2
Сложное	3

В качестве примера рассмотрим систему регистрации для учебного заведения, описанную в главе 2 (табл. 3.2).

Таблица 3.2

Типы действующих лиц

Действующее лицо	Тип
Студент	Сложное
Профессор	Сложное
Регистратор	Сложное
Расчетная система	Простое
Каталог курсов	Простое

Таким образом, общий весовой показатель равен:

$$A = 2 * 1 + 3 * 3 = 11.$$

3.1.2.

ОПРЕДЕЛЕНИЕ ВЕСОВЫХ ПОКАЗАТЕЛЕЙ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Все варианты использования делятся на три типа: простые, средние и сложные в зависимости от количества транзакций в потоках событий (основных и альтернативных). В данном случае под транзакцией понимается атомарная последовательность действий, которая выполняется полностью или отменяется.

Общее количество вариантов использования каждого типа умножается на соответствующий весовой коэффициент, затем вычисляется общий весовой показатель (табл. 3.3).

Таблица 3.3

Весовые коэффициенты вариантов использования

Тип варианта использования	Описание	Весовой коэффициент
Простой	3 или менее транзакций	5
Средний	От 4 до 7 транзакций	10
Сложный	Более 7 транзакций	15

Другой способ определения сложности вариантов использования заключается в подсчете количества классов анализа, участвующих в их реализации (табл. 3.4).

Таблица 3.4

Весовые коэффициенты вариантов использования

Тип варианта использования	Описание	Весовой коэффициент
Простой	Менее 5 классов	5
Средний	От 5 до 10 классов	10
Сложный	Более 10 классов	15

Для системы регистрации сложность вариантов использования определяется следующим образом (табл. 3.5).

Таблица 3.5

Сложность вариантов использования

Вариант использования	Тип
Войти в систему	Простой
Зарегистрироваться на курсы	Средний

Продолжение

Вариант использования	Тип
Просмотреть таблицу успеваемости	Простой
Выбрать курсы для преподавания	Средний
Проставил оценки	Простой
Курировать информацию о профессорах	Простой
Курировать информации? о студентах	Простой
Закрыть регистрацию	Средник

Таким образом, общий весовой показатель равен:

$$UC = 5 * 5 + 10 * 3 = 55.$$

В результате получаем показатель UUCP (Unadjusted Use Case Points):

$$UUCP = A + UC = 66.$$

3.1.3. ОПРЕДЕЛЕНИЕ ТЕХНИЧЕСКОЙ СЛОЖНОСТИ ПРОЕКТА

Техническая сложность проекта (TCF - Technical Complexity Factor) вычисляется с учетом показателей технической сложности (табл. 3.6).

Каждому показателю присваивается значение T_i в диапазоне от 0 до 5 (0 означает отсутствие значимости показателя для данной проекта, 5 - высокую значимость). Значение TCF вычисляется по формуле

$$TCF = 0,6 + (0,01 * (\sum T_i * Вес_i)).$$

Вычислим TCF для системы регистрации (табл. 3.7).

$$TCF = 0,6 + (0,01 * 40) = 1,0.$$

Таблица 3.6

Показатели технической сложности проекта TCF

Показатель	Описание	Вес
T1	Распределенная система	2
T2	Высокая производительность (пропускная способность)	1
T3	Работа конечных пользователей в режиме он-лайн	1
T4	Сложная обработка данных	1
T5	Повторное использование кода	1
T6	Простота установки	0,5
T7	Простота использования	0,5
T8	Переносимость	2
T9	Простота внесения изменений	1
T10	Паралелизм	1
T11	Специальные требования к безопасности	1
T12	Непосредственный доступ к системе со стороны внешних пользователей	1
T13	Специальные требования к обучению пользователей	1

Таблица 3.7

Показатели технической сложности системы регистрации

Показатель	Вес	Значение	Значение с учетом веса
T1	2	4	8
T2	1	3	3
T3	1	5	5
T4	1	1	1
T5	1	0	0

Продолжение

Показатель	Вес	Значение	Значение с учетом веса
T6	0,5	5	2,5
T7	0,5	5	2,5
T8	2	0	0
T9	1	4	4
T10	1	5	5
T11	1	3	3
T12	1	5	5
T13	1	1	1
Σ			40

3.1.4. ОПРЕДЕЛЕНИЕ УРОВНЯ КВАЛИФИКАЦИИ РАЗРАБОТЧИКОВ

Уровень квалификации разработчиков (EF - Environmental Factor) вычисляется с учетом следующих показателей (табл. 3.8).

Таблица 3.8

Показатели уровня квалификации разработчиков

Показатель	Описание	Вес
F1	Знакомство с технологиями	1,5
F2	Опыт разработки приложений	0,5
F3	Опыт использования объектно-ориентированного подхода	1
F4	Наличие ведущего аналитика	0,5
F5	Мотивация	1

Продолжение

Показатель	Описание	Вес
F6	Стабильность требований	2
F7	Частичная занятость	-1
F8	Сложные языки программирования	-1

Каждому показателю присваивается значение в диапазоне от 0 до 5. Для показателей F1 - F4 0 означает отсутствие, 3 - средний уровень, 5 - высокий уровень. Для показателя F5 0 означает отсутствие мотивации, 3 - средний уровень, 5 - высокий уровень мотивации. Для F6 0 означает высокую нестабильность требований, 3 - среднюю, 5 - стабильные требования. Для F7 0 означает отсутствие специалистов с частичной занятостью, 3 - средний уровень, 5 - все специалисты с частичной занятостью. Для показателя F8 0 означает простой язык программирования, 3 - среднюю сложность, 5 - высокую сложность.

Значение EF вычисляется по формуле

$$EF = 1,4 + (-0,03 \cdot (\sum F_i \cdot \text{Вес}_i)).$$

Вычислим EF для системы регистрации (табл. 3.9).

Таблица 3.9

Показатели уровня квалификации разработчиков
системы регистрации

Показатель	Вес	Значение	Значение с учетом веса
F1	1,5	1	1,5
F2	0,5	1	0,5
F3	1	1	1
F4	0,5	4	2
F5	1	5	5
F6	2	3	6

Продолжение

Показатель	Вес	Значение	Значение с учетом веса
F7	-1	0	0
F8	-1	3	-3
Σ			13

$$EF = 1,4 + (-0,03 * 13) = 1,01.$$

В результате получаем окончательное значение UCP (Use Case Points):

$$UCP = UUCP * TCF * EF = 56 * 1,0 * 1,01 = 56,56.$$

3.1.5. ОЦЕНКА ТРУДОЕМКОСТИ ПРОЕКТА

В качестве начального значения предлагается использовать 20 чел.-ч на одну UCP. Эта величина может уточняться с учетом опыта разработчиков. Приведем пример возможного уточнения.

Рассмотрим показатели F1 - F8 и определим, сколько показателей F1 - F6 имеют значение меньше 3 и сколько показателей F7 - F8 имеют значение больше 3. Если общее количество меньше или равно 2, следует использовать 20 чел.-ч на одну UCP, если 3 или 4 - 28. Если общее количество равно 5 или более, следует внести изменения в сам проект, в противном случае риск провала слишком высок.

Для системы регистрации получаем 28 чел.-ч на одну UCP, таким образом, общее количество человеко-часов на весь проект равно $56,56 * 28 = 1583,68$, что составляет 40 недель при 40-часовой рабочей неделе. Допустим, что команда разработчиков состоит из четырех человек, и добавим 3 недели на различные непредвиденные ситуации, тогда в итоге получим 13 недель на весь проект.

3.2. МЕТОДИКА ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ НА ОСНОВЕ ФУНКЦИОНАЛЬНЫХ ТОЧЕК

3.2.1. ОБЩИЕ СВЕДЕНИЯ

Рассматриваемый в данном разделе сокращенный вариант методики оценки трудоемкости разработки ПО основан на материалах консорциума IFPUG (International Function Point User Group) и компании SPR (Software Productivity Research), которая является одним из лидеров в области методов и средств оценки характеристик ПО.

Составляющие оценки трудоемкости разработки ПО:

- оценка размера разрабатываемого продукта. Для ПО в прежнее время основной мерой оценки являлось количество строк кода (LOC - Lines of Code), а в настоящее время является количество функциональных точек (FPs - Function Points);
- оценка трудоемкости в человеко-месяцах или человеко-часах;
- оценка продолжительности проекта в календарных месяцах;
- оценка стоимости проекта.

Согласно данной методике трудоемкость вычисляется на основе функциональности разрабатываемой системы, которая в свою очередь определяется на основе выявления функциональных типов - логических групп взаимосвязанных данных, используемых и поддерживаемых приложением, а также элементарных процессов, связанных с вводом и выводом информации (рис. 3.1).

Порядок расчета трудоемкости разработки ПО:

- определение количества и сложности функциональных типов приложения;
- определение количества связанных с каждым функциональным типом элементарных данных (DET), элементарных записей (RET) и файлов типа ссылок (FTR);
- определение сложности (в зависимости от количества DET, RET и FTR);
- подсчет количества функциональных точек приложения;

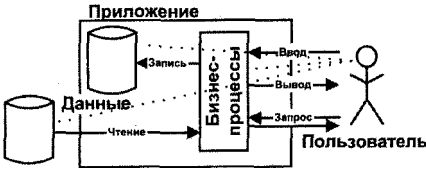


Рис. 3.1. Выявление функциональных типов

- подсчет количества функциональных точек с учетом общих характеристик системы (рис. 3.2);

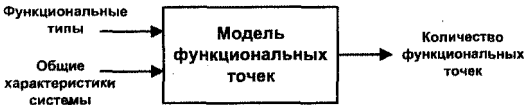


Рис. 3.2. Определение количества функциональных точек

- оценка трудоемкости разработки (с использованием различных статистических данных).

В состав функциональных типов (function type) включаются следующие элементы разработкой разрабатываемой системы:

1. *Внутренний логический файл (Internal Logical File, ILF)* – идентифицируемая совокупность логически взаимосвязанных записей данных, поддерживаемая внутри приложения посредством элементарного процесса (рис. 3.3).
2. *Внешний интерфейсный файл (External Interface File, EIF)* – идентифицируемая совокупность логически взаимосвязанных записей данных, передаваемых другому приложению или получаемых от него и поддерживаемых вне данного приложения (рис. 3.4).
3. *Входной элемент приложения (External Input, EI)* – элементарный процесс, связанный с обработкой входной информации

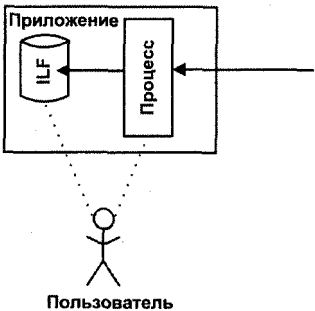


Рис. 3.3. Внутренний логический файл

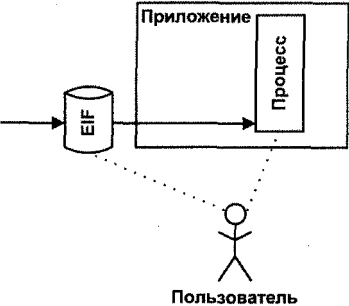


Рис. 3.4. Внешний интерфейсный файл

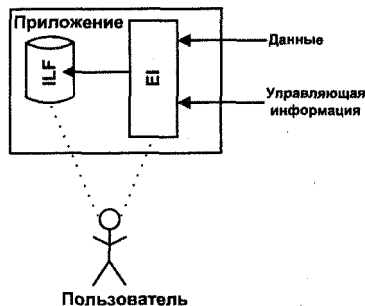


Рис. 3.5. Входной элемент приложения

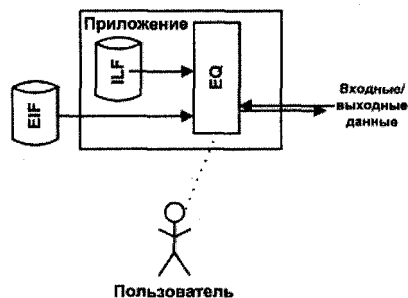


Рис. 3.7. Внешний запрос

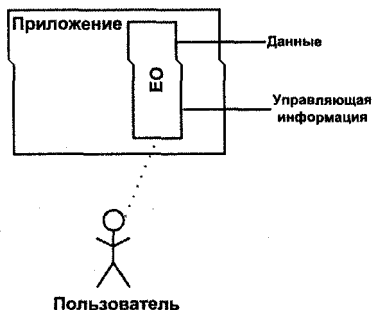


Рис. 3.6. Выходной элемент приложения

приложения - входного документа или экранной формы. Обрабатываемые данные могут соответствовать одному или более ILF (рис. 3.5).

4. *Выходной элемент приложения (External Output, EO)* - элементарный процесс, связанный с обработкой выходной информации приложения - выходного отчета, документа, экранной формы (рис. 3.0.).

5. *Внешний запрос (External Query, EQ)* - элементарный процесс, состоящий из комбинации «запрос/ответ», не связанной с вычислением производных данных или обновлением ILF (базы данных) (рис. 3.7).

3.2.2. ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА И СЛОЖНОСТИ ФУНКЦИОНАЛЬНЫХ ТИПОВ ПО ДАННЫМ

Количество функциональных типов по данным (внутренних логических файлов и внешних интерфейсных файлов) определяется на основе диаграмм «сущность-связь» (для структурного под-

хода) и диаграмм классов (для объектно-ориентированного подхода). В последнем случае в расчете участвуют только устойчивые (persistent) классы.

Устойчивый класс соответствует ILF (если его объекты обязательно создаются внутри самого приложения) или EIF (если его объекты не создаются внутри самого приложения, а получаются в результате запросов к базе данных).

Примечание. Если операции класса являются операциями-запросами, то это характеризует его принадлежность к EIF.

Далее для каждого выявленного функционального типа (ILF и EIF) определяется его сложность (низкая, средняя или высокая). Она зависит от количества связанных с этим функциональным типом элементарных данных (Data Element Types, DET) и элементарных записей (Record Element Types, RET), которые, в свою очередь, определяются следующим образом:

- **DET** - уникальный идентифицируемый нерекурсивный элемент данных (включая внешние ключи), входящий в ILF или EIF;
- **RET** - идентифицируемая подгруппа элементов данных, входящая в ILF или EIF. На диаграммах «сущность-связь» такая подгруппа обычно представляется в виде сущности-подтипа в связи «супертип-подтип».

Один DET соответствует отдельному атрибуту или связи класса. Количество DET не зависит от количества объектов класса или количества связанных объектов. Если данный класс связан с некоторым другим классом, который обладает явно заданным идентификатором, состоящим более чем из одного атрибута, то для каждого такого атрибута определяется один отдельный DET (а не один DET на всю связь). Производные атрибуты могут игнорироваться. Повторяющиеся атрибуты одинакового формата рассматриваются как один DET.

Одна RET на диаграмме устойчивых классов соответствует либо абстрактному классу в связи обобщения (generalization), либо классу - «части целого» в композиции, либо классу с рекурсивной связью «родитель-потомок» (агрегацией).

Зависимость сложности функциональных типов от количества DET и RET определяется следующей таблицей (табл. 3.10).

Таблица 3.10

Зависимость сложности ILF и EIF от количества DET и RET

Количество RET	Количество DET		
	1-19	20 - 50	51 +
1	Низкая	Низкая	Средняя
2 - 5	Низкая	Средняя	Высокая
6 +	Средняя	Высокая	Высокая

3.2.3. ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА И СЛОЖНОСТИ ТРАНЗАКЦИОННЫХ ФУНКЦИОНАЛЬНЫХ ТИПОВ

Количество транзакционных функциональных типов (входных элементов приложения, выходных элементов приложения и внешних запросов) определяется на основе выявления входных и выходных документов, экранных форм, отчетов, а также по диаграммам классов (в расчете участвуют граничные классы).

Далее для каждого выявленного функционального типа (EI, EO или EQ) определяется его сложность (низкая, средняя или высокая). Она зависит от количества связанных с этим функциональным типом DET, RET и файлов типа ссылок (File Type Referenced, FTR) - ILF или EIF, читаемых или модифицируемых функциональным типом.

Правила расчета DET для EI:

- каждое нерекурсивное поле, принадлежащее (поддерживаемое) ILF и обрабатываемое во вводе;
- каждое поле, которое через процесс ввода поддерживается в ILF;
- логическое поле, которое физически представляет собой множество полей, но воспринимается пользователем как единый блок информации;

- группа полей, которые появляются в ILF более одного раза, но в связи с особенностями алгоритма их использования воспринимаются как один DET;
- группа полей, которые фиксируют ошибки в процессе обработки или подтверждают, что обработка закончилась успешно;
- действие, которое может быть выполнено во вводе.

Правила расчета DET для EO:

- каждое распознаваемое пользователем нерекурсивное поле, участвующее в процессе вывода;
- поле, которое физически отображается в виде нескольких полей, его составляющих, но используемое как единый информационный элемент;
- каждый тип метки и каждое значение числового эквивалента при графическом выводе;
- текстовая информация, которая может содержать одно слово, предложение или фразу;
- литералы не могут считаться элементами данных;
- переменные, определяющие номера страниц или генерируемые системой логотипы, не являются элементами данных.

Правила расчета DET для EQ

Правила определения DET для вводной части:

- каждое распознаваемое пользователем нерекурсивное поле, появляющееся во вводной части запроса;
- каждое поле, которое определяет критерий выбора данных;
- группа полей, в которых выдаются сообщения о возникающих ошибках в процессе ввода информации в DET или подтверждающих успешное завершение процесса ввода;
- группа полей, которые позволяют выполнять запросы.

Правила определения DET для выводной части:

- каждое распознаваемое пользователем нерекурсивное поле, которое появляется в выводной части запроса;
- логическое поле, которое физически отображается как группа полей, однако воспринимается пользователем как единое поле;
- группа полей, которые в соответствии с методикой обработки могут повторяться в ILF;
- литералы не могут считаться DET;

- колонтитулы или генерируемые системой пиктограммы не могут считаться DET.
- Зависимость сложности функциональных типов от количества DET, RET или FTR определяется по табл. 3.11 и 3.12.

Таблица 3.11

Зависимость сложности EI от количества DET, RET или FTR

Количество FTR	Количество DET		
	1 - 4	5 - 15	16 +
0 - 1	Низкая	Низкая	Средняя
2	Низкая	Средняя	Высокая
3 +	Средняя	Высокая	Высокая

Таблица 3.12

Зависимость сложности EO от количества DET и FTR

Количество FTR	Количество DET		
	1 - 5	6 - 19	20 +
0 - 1	Низкая	Низкая	Средняя
2 - 3	Низкая	Средняя	Высокая
4 +	Средняя	Высокая	Высокая

Сложность EQ определяется как максимальная суммирование значений EI и EO, связанных с данным запросом.

3.2.4. ПОДСЧЕТ КОЛИЧЕСТВА ФУНКЦИОНАЛЬНЫХ ТОЧЕК

Для каждого функционального типа подсчитывается количество входящих в его состав функциональных точек (Function Point, FP) - условных элементарных единиц. Этот подсчет выполняется в соответствии с табл. 3.13.

Таблица 3.13

Зависимость количества FP от сложности функционального типа

Функциональный тип	Сложность		
	низкая	средняя	высокая
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

В результате суммирования количества FP по всем функциональным типам получается общее количество FP (**UFP, Unadjusted Function Points**) без учета поправочного коэффициента. Значение поправочного коэффициента (**VAF, Value Adjustment Factor**) определяется набором из 14 общих характеристик системы (**GSC, General System Characteristics**) и вычисляется по формуле

$$VAF = (0,65 + (\text{sum GSC} * 0,01)).$$

Значения GSC варьируются в диапазоне от 0 до 5 и определяются по табл. 3.14 - 3.27.

Таблица 3.14

Коммуникации данных

0	Полностью пакетная обработка на локальном ПК
1	Пакетная обработка, удаленный ввод данных или удаленная печать
2	Пакетная обработка, удаленный ввод данных и удаленная печать
3	Сбор данных в режиме «он-лайн» или дистанционная обработка, связанная с пакетным процессом
4	Несколько внешних интерфейсов, один тип коммуникационного протокола
5	Несколько внешних интерфейсов, более одного типа коммуникационного протокола

Таблица 3.15

Распределенная обработка данных

0	Передача данных или процессов между компонентами системы отсутствует
1	Приложение готовит данные для обработки на ПК конечного пользователя
2	Данные готовятся для передачи, затем передаются и обрабатываются на другом компоненте системы (не на ПК конечного пользователя)
3	Распределенная обработка и передача данных в режиме «он-лайн» только в одном направлении
4	Распределенная обработка и передача данных в режиме «он-лайн» в обоих направлениях
5	Динамическое выполнение процессов в любом подходящем компоненте системы

Таблица 3.16

Производительность

0	К системе не предъявляется специальных требований, касающихся производительности
1	Требования к производительности определены, но не требуется никаких специальных действий
2	Время реакции или пропускная способность являются критическими в пиковые периоды. Не требуется никаких специальных решений относительно использования ресурсов процессора. Обработка может быть завершена в течение следующего рабочего дня
3	Время реакции или пропускная способность являются критическими в обычное рабочее время. Не требуется никаких специальных решений относительно использования ресурсов процессора. Время обработки ограничено взаимодействующими системами
4	То же, что в случае 3, кроме того, пользовательские требования к производительности достаточно серьезны, чтобы ее необходимо было анализировать на стадии проектирования
5	То же, что в случае 4, кроме того, на стадиях проектирования, разработки и/или реализации для удовлетворения пользовательских требований к производительности используются специальные средства анализа

Таблица 3.17

Эксплуатационные ограничения

0	Какие-либо явные или неявные ограничения отсутствуют
1	Эксплуатационные ограничения присутствуют, но не требуют никаких специальных усилий
2	Должны учитываться некоторые ограничения, связанные с безопасностью или временем реакции
3	Должны учитываться конкретные требования к процессору со стороны конкретных компонентов приложения
4	Заданные эксплуатационные ограничения требуют специальных ограничений на выполнение приложения в центральном или выделенном процессоре
5	То же, что в случае 4, кроме того, специальные ограничения затрагивают распределенные компоненты системы

Таблица 3.18

Частота транзакций

0	Пиковых периодов не ожидается
1	Ожидаются пиковые периоды (ежемесячные, ежеквартальные, ежегодные)
2	Ожидаются еженедельные пиковые периоды
3	Ожидаются ежедневные пиковые периоды
4	Высокая частота транзакций требует анализа производительности на стадии проектирования
5	То же, что в случае 4, кроме того, на стадиях проектирования, разработки и/или внедрения необходимо использовать специальные средства анализа производительности

Таблица 3.19

Ввод данных в режиме «он-лайн»

0	Все транзакции обрабатываются в пакетном режиме
1	От 1 до 7% транзакций требуют интерактивного ввода данных
2	От 8 до 15% транзакции требуют интерактивного ввода данных
3	От 16 до 23% транзакций требуют интерактивного ввода данных
4	От 24 до 30% транзакций требуют интерактивного ввода данных
5	Более 30% транзакций требуют интерактивного ввода данных

Таблица 3.20

Эффективность работы конечных пользователей *

0	Ни одной из перечисленных функциональных возможностей *
1	От одной до трех функциональных возможностей
2	От четырех до пяти функциональных возможностей
3	Шесть или более функциональных возможностей при отсутствии конкретных пользовательских требований к эффективности
4	То же, что в случае 3, кроме того, пользовательские требования к эффективности включают специальные проектные решения для учета эргономических факторов (например, минимизации нажатий клавиш, максимизации значений по умолчанию, использования шаблонов)
5	То же, что в случае 4, кроме того, пользовательские требования к эффективности включают применение специальных средств и процессов, демонстрирующих их выполнение

* Эффективность работы конечных пользователей определяется наличием следующих функциональных возможностей:

- средств навигации (например, функциональные клавиши, динамически генерируемые меню);
- меню;
- онлайн-подсказок и документации;
- автоматического перемещения курсора;
- скроллинга;
- удаленной печати;
- предварительно назначенных функциональных клавиш;
- выбора данных на экране с помощью курсора;
- использования видеоэффектов, цветового выделения, подчеркивания и других индикаторов;
- всплывающих окон;
- минимизации количества экранов, необходимых для выполнения бизнес-функций;
- поддержкой двух и более языков.

Таблица 3.21

Онлайновое обновление

0	Отсутствует
1	Онлайновое обновление от одного до трех управляющих файлов. Объем обновлений незначителен, восстановление несложно
2	Онлайновое обновление четырех или более управляющих файлов. Объем обновлений незначителен, восстановление несложно
3	Онлайновое обновление основных внутренних логических файлов
4	То же, что в случае 3, плюс необходимость специальной защиты от потери данных
5	То же, что в случае 4, кроме того, большой объем данных требует учета затрат на процесс восстановления. Требуется автоматизированные процедуры восстановления с минимальным вмешательством оператора

Таблица 3.22

Сложная обработка *

0	Ни одной из перечисленных функциональных возможностей *
1	Любая одна из возможностей
2	Любые две из возможностей
3	Любые три из возможностей
4	Любые четыре из возможностей
5	Все пять возможностей

* Сложная обработка характеризуется наличием у приложения следующих функциональных возможностей:

- повышенной реакцией на внешние воздействия и/или специальной защитой от внешних воздействий;
- экстенсивной логической обработкой;
- экстенсивной математической обработкой;
- обработкой большого количества исключительных ситуаций;
- поддержкой разнородных типов входных/выходных данных.

Таблица 3.23

Повторное использование

0	Отсутствует
1	Повторное использование кода внутри одного приложения
2	Не более 10% приложений будут использоваться более чем одним пользователем
3	Более 10% приложений будут использоваться более чем одним пользователем
4	Приложение оформляется как продукт и/или документируется для облегчения повторного использования. Настройка приложения выполняется пользователем на уровне исходного кода
5	То же, что в случае 4, с возможностью параметрической настройки приложений

Таблица 3.24

Простота установки

0	К установке не предъявляется никаких специальных требований
1	Для установки требуется специальная процедура
2	Заданы пользовательские требования к конвертированию (переносу существующих данных и приложений в новую систему) и установке, должны быть обеспечены и проверены соответствующие руководства. Конвертированию не придается важное значение
3	То же, что и в случае 2, однако конвертированию придается важное значение
4	То же, что и в случае 2, плюс наличие автоматизированных средств конвертирования и установки
5	То же, что и в случае 3, плюс наличие автоматизированных средств конвертирования и установки

Таблица 3.25

Простота эксплуатации

0	К эксплуатации не предъявляется никаких специальных требований, за исключением обычных процедур резервного копирования
1-4	Приложение обладает одной, несколькими или всеми из перечисленных далее возможностей. Каждая возможность, за исключением второй, обладает единичным весом: 1) наличие процедур запуска, копирования и восстановления с участием оператора; 2) то же, без участия оператора; 3) минимизируется необходимость в монтировании носителей для резервного копирования; 4) минимизируется необходимость в средствах подачи и укладки бумаги при печати
5	Вмешательство оператора требуется только при запуске и завершении работы системы. Обеспечивается автоматическое восстановление работоспособности приложения после сбоев и ошибок

Таблица 3.26

Количество возможных установок на различных платформах

0	Приложение рассчитано на установку у одного пользователя
1	Приложение рассчитано на много установок для строго стандартной платформы (технические средства плюс программное обеспечение)
2	Приложение рассчитано на много установок для платформ с близкими характеристиками
3	Приложение рассчитано на много установок для различных платформ
4	То же, что в случаях 1 или 2, плюс наличие документации и планов поддержки всех установленных копий приложения
5	То же, что в случае 3, плюс наличие документации и планов поддержки всех установленных копий приложения

Таблица 3.27

Гибкость *

0	Ни одной из перечисленных возможностей *
1	Любая одна из возможностей
2	Любые две из возможностей
3	Любые три из возможностей
4	Любые четыре из возможностей
5	Все пять возможностей

* Гибкость характеризуется наличием у приложения следующих возможностей:

- поддержкой простых запросов, например логики *и/или* в применении только к одному ILF (вес - 1);
- поддержкой запросов средней сложности, например логики *и/или* в применении более чем к одному ILF (вес - 2);
- поддержкой сложных запросов, например комбинации логических связей *и/или* в применении к одному или более ILF (вес - 3);
- управляющая информация хранится в таблицах, поддерживаемых пользователем в интерактивном режиме, однако эффект от ее изменений проявляется на следующий рабочий день;
- то же, что в предыдущем случае, но эффект проявляется немедленно (вес - 2).

После определения всех значений GSC и вычисления поправочного коэффициента VAF вычисляется итоговая оценка количества функциональных точек (Adjusted Function Points, AFP):

$$AFP = UFP * VAF.$$

3.2.5. ОЦЕНКА ТРУДОЕМКОСТИ РАЗРАБОТКИ

Вариант 1

По таблице 3.28 (данные SPR) определяется количество строк кода (SLOC) на одну функциональную точку в зависимости от используемого языка программирования.

Таблица 3.28

Количество строк кода на одну функциональную точку

Язык (средство)	Количество SLOC на FP
ABAP/4	16
Access	38
ANSI SQL	13
C++	53
Clarion	58
Data base default	40
Delphi 5	18
Excel 5	6
FoxPro 2.5	34
Oracle Developer	23
PowerBuilder	16
Smalltalk	21
Visual Basic 6	24
Visual C++	34
HTML4	14
Java 2	46

Умножая AFP на количество SLOC на FP, получаем количество SLOC в приложении.

Далее используется один из вариантов известной модели оценки трудоемкости разработки ПО под названием COCOMO (Constructive Cost Model), опубликованной в книге Б.У. Бозма*, и ее современной версии COCOMO II.

В табл. 3.29 приведены значения линейного коэффициента производительности (LPF), полученные в COCOMO.

* См.: Бозм Б. У. Инженерное проектирование программного обеспечения: Пер. с англ. - М.: Радио и связь, 1985.

Таблица 3.29

Линейный коэффициент производительности

Тип проекта	LPF
COCOMO II Default	2,94
Встроенное ПО	2,58
Электронная коммерция	3,60
Web-приложения	3,30
Военные разработки	2,77

Трудоемкость разработки (количество человеко-месяцев) вычисляется по следующей формуле:

$$\text{Трудоемкость} = \text{LPF} \cdot \text{KSLOC},$$

где KSLOC - количество тысяч строк кода в приложении.

Приведенная формула применяется для проектов малого размера, а для больших проектов учитывается ESPF, значения которого приведены в табл. 3.30.

Таблица 3.30

Экспоненциальный коэффициент размера

Тип проекта	ESPF
COCOMO II Default	1,052
Встроенное ПО	1,110
Электронная коммерция	1,030
Web-приложения	1,030
Военные разработки	1,072

С учетом данного коэффициента

$$\text{Трудоемкость} = \text{LPF} \cdot \text{KSLOC}^{\text{ESPF}}.$$

Подсчитанная таким образом трудоемкость подлежит дальнейшему уточнению с учетом поправок на характеристики среды разработки. Эти характеристики учитываются в двух поправочных коэффициентах: нелинейном коэффициенте среды (NEF) и линейном коэффициенте среды (LEF), значения которых приведены в табл. 3.31 и 3.32.

Таблица 3.31

Нелинейный коэффициент среды

Фактор	Значение		
	низкое	номинальное	высокое
Архитектурный риск	0,0423	0,014	-0,0284
Гибкость среды разработки	0,0223	0,002	-0,0284
Уровень знания новых технологий и предметной области	0,0336	0,0088	-0,0284
Зрелость процессов в организации	0,0496	0,0814	-0,0284
Сплоченность проектной команды	0,0264	0,0045	-0,0284

Таблица 3.32

Линейный коэффициент среды

Фактор	Значение		
	низкое	номинальное	высокое
Квалификация аналитиков	1,42	1,00	0,71
Опыт разработки приложений	1,22	1,00	0,81
Опыт работы с языками и инструментальными средствами	1,20	1,00	0,84
Преемственность персонала	1,29	1,00	0,81
Квалификация руководства	1,18	1,00	0,87
Опыт руководства	1,11	1,00	0,90
Опыт работы с данной платформой	1,19	1,00	0,85
Квалификация программистов	1,34	1,00	0,76

Продолжение

Фактор	Значение		
	низкое	номинальное	высокое
Ограничения на время выполнения	1,00	1,00	1,63
Ограничения на объем памяти	1,00	1,00	1,46
Нестабильность платформы	0,87	1,00	1,30
Эффективность средств управления	1,22	1,00	0,84
Распределенная разработка	1,22	1,00	0,80
Эргономика офиса	1,19	1,00	0,82
Использование CASE-средств	1,17	1,00	0,78
Размер базы данных	0,90	1,00	1,28
Требуемая степень документированности	0,81	1,00	1,23
Интернационализация	0,97	1,00	1,35
Сложность продукта	0,75	1,00	1,66
Степень повторного использования	0,95	1,00	1,24
Требуемая надежность	0,82	1,00	1,26
Графика и мультимедиа	0,95	1,00	1,35
Интеграция с унаследованным ПО	1,00	1,00	1,18
Уровень безопасности	0,92	1,00	1,40
Необходимость выбора инструментальных средств	0,95	1,00	1,14
Интенсивность транзакций	0,96	1,00	1,59
Разработка для использования в Web	0,88	1,00	1,45

Сумма значений NEF добавляется к значению ESPF.

Значение трудоемкости, подсчитанное с учетом NEF, последовательно умножается на все; выбранные значения LEF.

Срок разработки можно определить с помощью коэффициента преобразования (CF) по следующей формуле:

$$\text{Срок разработки} = CF \cdot \text{Трудоемкость}^{0,33},$$

где значение CF определяется по табл. 3.33.

Таблица 3.33

Коэффициент преобразования

Тип проекта	CF
COCOMO II Default	3,67
Встроенное ПО	• 4,00
Электронная коммерция	3,20
Web-приложения	3,10
Военные разработки	3,80

Вариант 2

Используется промежуточная модель COCOMO, в соответствии с которой номинальную трудоемкость (без учета коэффициентов затрат труда, стоимостных факторов и сложности) можно вычислить по формуле

Трудоемкость = N1 * KSLOC^{N2}.

Значения N1 и N2 определяются по табл. 3.34.

Таблица 3.34

Коэффициенты N1 и N2

Тип ПО	N1	N2
Распространенное	3,2	1,05
Полунезависимое	3,0	1,12
Встроенное	2,8	1,20

Распространенное ПО - ПО небольшого объема (не более 50 KSLOC), разрабатываемое относительно небольшой группой опытных специалистов в стабильных условиях.

Полунезависимое ПО - ПО среднего объема (не более 300 KSLOC), разрабатываемое неоднородной группой специалистов средней квалификации.

Встроенное ПО - ПО с жесткими ограничениями (система резервирования авиабилетов, система управления воздушным движением и т.п.).

Время разработки вычисляется по формуле

Время = 2,5 * Трудоемкость^{N3}

Значения N3 приведены в табл. 3.35.

Таблица 3.35

Коэффициент N3

Тип ПО	N3
Распространенное	0,38
Полунезависимое	0,35
Встроенное	0,32

Время разработки может быть изменено с учетом статистических данных, накопленных в реальных проектах и отраженных в табл. 3.36, где сопоставлены планируемый и реальный сроки выполнения проекта в зависимости от его размера, выраженного в количестве функциональных точек. Частично это отставание объясняется неточной оценкой, частично - ростом количества требований к системе после того, как выполнена начальная оценка.

Таблица 3.36

Статистические данные

Размер проекта	<100FP	100-1000 FP	1000- 10000 FP	> 10000 FP
Планируемый срок (мес.)	6	12	18	24
Реальный срок (мес.)	8	16	24	36
Отставание	2	4	6	12

3.3. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА

Задание 1. Выполнить оценку трудоемкости разработки для системы обработки заказов (п. 2.4) с использованием методики оценки на основе вариантов использования.

Задание 2. Выполнить оценку трудоемкости разработки для системы кредитования коммерческого банка (п. 2.4) с использованием методики оценки на основе вариантов использования.

Задание 3. Выполнить оценку трудоемкости разработки для системы регистрации курсов с использованием обеих методик и сравнить полученные результаты.

Боггс У., Боггс М. UML и Rational Rose: Пер. с англ. - М.: Лори, 2000.

Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. - 2-е изд.: Пер. с англ. - М.: Бином, СПб.: Невский диалект, 1999.

Буч Г., Рамбо Д., Джекобсон А. Язык UML, Руководство пользователя: Пер. с англ. - М.: ДМК, 2000.

Вендров А.М. Проектирование программного обеспечения экономических информационных систем. - М.: Финансы и статистика, 2000.

Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования: Пер. с англ. - М.: ДМК, 2000.

Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. - М.: Мир, 1999.

Ларман К. Применение UML и шаблонов проектирования: Пер. с англ. - М.: Вильямс, 2001.

Предисловие.....	3
ГЛАВА 1. ПРИМЕНЕНИЕ СТРУКТУРНОГО ПОДХОДА К АНАЛИЗУ И ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	5
1.1. ОСНОВНЫЕ СВЕДЕНИЯ О СТРУКТУРНЫХ МЕТОДАХ АНАЛИЗА И ПРОЕКТИРОВАНИЯ.....	5
1.1.1. Общие сведения о моделировании потоков данных (процессов).....	6
1.1.2. Общие сведения о моделировании данных.....	15
1.1.3. Подход, используемый в CASE-средстве Silverrun.....	16
1.1.4. Общие сведения о функциональных моделях, используемых на стадии проектирования.....	24
1.2. ВЫПОЛНЕНИЕ УЧЕБНОГО ПРОЕКТА.....	26
1.2.1. Постановка задачи.....	26
1.2.2. Описание контекста системы приема пациентов в больнице и построение начальной контекстной диаграммы.....	29
1.2.3. Спецификация структур данных.....	30
1.2.4. Построение начального варианта концептуальной модели данных.....	31
1.2.5. Построение диаграмм потоков данных нулевого и последующих уровней.....	33
1.2.6. Уточнение концептуальной модели данных.....	39
1.2.7. Построение диаграмм системных процессов и диаграмм последовательностей экранных форм.....	39
1.3. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА.....	44

ГЛАВА 2. ПРИМЕНЕНИЕ ОБЪЕКТНО- ОРИЕНТИРОВАННОГО ПОДХОДА К АНАЛИЗУ И ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	47
2.1. ОСНОВНЫЕ СВЕДЕНИЯ О ЯЗЫКЕ UML.....	47
2.1.1. Средства UML.....	47
2.1.2. Диаграммы вариантов использования.....	48
2.1.3. Диаграммы взаимодействия.....	56
2.1.4. Диаграммы классов.....	60
2.1.5. Диаграммы состояний.....	73
2.1.6. Диаграммы деятельностей.....	77
2.1.7. Диаграммы компонентов.....	78
2.1.8. Диаграммы размещения.....	81
2.2. ОСНОВНЫЕ СВЕДЕНИЯ О CASE-СРЕДСТВЕ RATIONAL ROSE.....	83
2.2.1. Введение в Rational Rose.....	83
2.2.2. Работа в среде Rational Rose.....	86
2.3. ВЫПОЛНЕНИЕ УЧЕБНОГО ПРОЕКТА.....	95
2.3.1. Постановка задачи создания системы регистрации для учебного заведения.....	95
2.3.2. Составление глоссария проекта.....	96
2.3.3. Описание дополнительных спецификаций.....	97
2.3.4. Создание модели вариантов использования.....	98
2.3.5. Анализ системы.....	109
2.3.6. Проектирование системы.....	129
2.3.7. Реализация системы.....	148
2.4. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА.....	151

ГЛАВА 3. МЕТОДИКИ ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	155
3.1. МЕТОДИКА ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ НА ОСНОВЕ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ.....	155
3.1.1. Определение весовых показателей действующих лиц.....	155
3.1.2. Определение весовых показателей вариантов использования.....	156
3.1.3. Определение технической сложности проекта.....	158
3.1.4. Определение уровня квалификации разработчиков.....	160
3.1.5. Оценка трудоемкости проекта.....	162
3.2. МЕТОДИКА ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ НА ОСНОВЕ ФУНКЦИОНАЛЬНЫХ ТОЧЕК.....	163
3.2.1. Общие сведения.....	163
3.2.2. Определение количества и сложности функциональных типов по данным.....	167
3.2.3. Определение количества и сложности транзакционных функциональных типов.....	169
3.2.4. Подсчет количества функциональных точек.....	171
3.2.5. Оценка трудоемкости разработки.....	179
3.3. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ ПРОЕКТА.....	186
Дополнительная литература.....	187

Учебное издание

Вендров Александр Михайлович

ПРАКТИКУМ ПО ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЭКОНОМИЧЕСКИХ ИНФОРМАЦИОННЫХ СИСТЕМ

Заведующая редакцией Л. А. Табакова

Редактор И.А. Кузнецова

Младший редактор Н.А. Федорова

Художественный редактор Ю.И. Артюхов

Технический редактор В.Ю. Фомева

Корректоры Т. М. Колпакова, Г. В. Хлопцева

Компьютерная верстка Е. Ф. Тимохиной

Художник обложки Е.К. Самойлов

ИБ № 4337

Подписано в печать 29.03.2002

Формат 60х88/16. Гарнитура «Таймс». Печать офсетная

Усл. п.л. 11,76. Уч. изд.-л. 10,73

Тираж 5000 экз. Заказ 1250. «С» 091

Издательство «Финансы и статистика»

101000, Москва, ул. Покровка, 7

Телефон (095) 925-35-02, факс (095) 925-09-57

E-mail: mail@finstat.ru http: // www.finstat.ru

ГУП «Великолукская городская типография»
Комитета по средствам массовой информации Псковской области
182100, Великие Луки, ул. Полиграфистов, 78/12

Тел./факс: (811-53) 3-62-95

E-mail: VTL@MART.RU

Издательство
ФИНАНСЫ И СТАТИСТИКА"
предлагает учебник

Г.Н.Смирновой, А.А. Сорокина, Ю.Ф.Тельнова

Проектирование экономических информационных систем



Раскрываются основы проектирования экономических информационных систем на различных стадиях жизненного цикла. Рассматриваются методы и средства канонического и индустриального проектирования экономических информационных систем, а также управления процессом проектирования. Особое внимание уделяется методологии реинжиниринга бизнес-процессов, CASE-, RAD- и компонентных технологий.

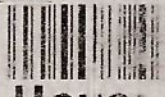
Для студентов, обучающихся по специальностям «Прикладная информатика в экономике», «Прикладная информатика в менеджменте», «Прикладная информатика в юриспруденции», а также для специалистов в области разработки экономических информационных систем.

Книгу можно приобрести в киоске издательства
или заказать по почте

Адрес: 101000, Москва, ул.Покровка, 7
(метро "Китай-город", выход на ул.Маросейка)
Тел.: (095)925-35-02, 923-18-68
Факс: (095)925-09-57
E-mail: mail@finstat.ru
<http://www.finstat.ru>

А. М. Вендров – известный специалист в области CASE-технологий. Является одним из авторов справочника "Системы управления базами данных" ("Финансы и статистика", 1991), автором монографии "CASE-технологии. Современные методы и средства проектирования информационных систем" ("Финансы и статистика", 1998) и учебника "Проектирование программного обеспечения экономических информационных систем" ("Финансы и статистика", 2000), переводчиком книг М. Фаулера "UML в кратком изложении. Применение стандартного языка объектного моделирования" ("Мир", 1999) и Э. Йордана "Путь камикадзе" ("ЛОРИ", 2000)

Цель практикума – формирование навыков самостоятельного практического применения современных методов и средств проектирования программного обеспечения, основанных на использовании визуального моделирования и CASE-средств. Практикум дополняет выпущенный ранее учебник "Проектирование программного обеспечения экономических информационных систем": содержит задания и упражнения на построение моделей программного обеспечения с использованием как структурного, так и объектно-ориентированного подхода (с применением стандартного языка моделирования UML). Построение моделей и диаграмм UML выполняется с помощью CASE-средства Rational Rose.

ДОМ КНИГИ
"МОЛОДАЯ ГВАРДИЯ"
Вендров А. М. Практикум
ISBN 5-279-02440-6

Цена: 43 р.

ISBN 5-279-02440-6



9 785279 024407

