Федеральное агентство по образованию РФ
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав.кафедрой ИиУС
_____А.В.Бушманов
«____»_____2007г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

ПРОЕКТИРОВАНИЕ ИС

для специальности
230201 – Информационные системы и технологии

Составитель: А.В.Бушманов

2007г.

Печатается по решению
редакционно-издательского совета
факультета математики
и информатики
Амурского государственного
университета

Проектирование информационных систем для специальности 230201 «Информационные системы и технологии»: учебно-методический комплекс дисциплины. /Бушманов А.В. – Благовещенск. Изд-во Амурского гос. ун-та, 2006г.

ОГЛАВЛЕНИЕ

1. Выписка из государственного образовательного стандарта высшего профессионального образования ……4
2. Рабочая программа ……...5
3. График самостоятельной работы студентов ……19
4. Методические рекомендации по проведению самостоятельной работы студентов ……19
5. Перечень учебников, учебных пособий ……43
6. Краткий конспект лекций ……...44
7. Методические указания по выполнению курсового проекта …….69
8. Методические рекомендации по выполнению лабораторных работ ……75
9. Перечень программных продуктов, используемых в практике выпускников и учебно-методическое пособие …….93
10. Методические указания по применению современных ИТ для преподавания учебной дисциплины ……98
11. Методические указания по организации межсессионного и экзаменационного контроля знаний студентов ……127
12. Комплекты заданий для лабораторных работ и курсовых проектов ……128
13. Фонд тестовых и контрольных заданий для оценки качества заданий по дисциплине ……129
14. Комплект экзаменационных билетов ……135
15. Карта кадровой обеспеченности дисциплины ……..140

1. ВЫПИСКА ИЗ ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО СТАНДАРТА ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

Направление подготовки дипломированного специалиста
654600 – Информатика и вычислительная техника

Специальность

071900 (230201) – Информационные системы в технике и технологиях

Квалификация – *инженер*.

Индекс	**Наименование дисциплин и их основные разделы**	Всего часов
СД.07	**Проектирование информационных систем** Общая характеристика процесса проектирования ИС; структура информационно-логической модели ИС; разработка функциональной модели; исходные данные для проектирования; разработка модели и защита данных; разработка пользовательского интерфейса; разработка проекта распределенной обработки. Структура программных модулей; разработка алгоритмов; логический анализ структур ИС; анализ и оценка производительности ИС; управление проектом ИС; проектная документация; инструментальные средства проектирования ИС; типизация проектных решений; графические средства представления проектных решений; эксплуатация ИС.	204

2. РАБОЧАЯ ПРОГРАММА

по дисциплине «Проектирование ИС»
для специальности 230201 «Информационные системы и технологии»

Курс 4, 5. Семестр 8, 9.

Лекции - 45 часов: 8 сем. – 15 (час.)
9 сем. – 30 (час.)

Зачет – 8 семестр.
Экзамен – 9 семестр.

Практические (семинарские) занятия: нет

Лабораторные занятия 60 (час.): 8 сем. – 30 (час.)
9 сем. – 30 (час.)

Самостоятельная работа: 100 (час.)

Всего – 205(час.)

1. Цель и задачи дисциплины, ее место в учебном процессе

1.1. Целью изучения дисциплины является получение студентами знаний и приобретение практических навыков по проектированию Информационных систем.

1.2. Основные разделы дисциплины определены в **Государственном образовательном стандарте** высшего профессионального образования: Общая характеристика процесса проектирования ИС; структура информационно-логической модели ИС; разработка функциональной модели; исходные данные для проектирования; разработка модели и защита данных; разработка пользовательского интерфейса; разработка проекта распределенной обработки**.** Структура программных модулей; разработка алгоритмов; логический анализ структур ИС; анализ и оценка производительности ИС; управление проектом ИС; проектная документация; инструментальные средства проектирования ИС; типизация проектных решений; графические средства представления проектных решений; эксплуатация ИС.

1.3. Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности информационных систем (ИС), создаваемых в различных областях экономики.

Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

- сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
- наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);
- отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;
- необходимость интеграции существующих и вновь разрабатываемых приложений;
- функционирование в неоднородной среде на нескольких аппаратных платформах;
- разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;
- существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта, объект проектирования (ИС) должен быть прежде всего адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели ИС.

1.4. Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

С целью приобретения навыков в решении основных вопросов проектирования ИС, при их качественной формулировке при изучении дисциплины, необходимы практические шаги в виде курсового проектирования.

1.5. Перечень разделов (тем) необходимых дисциплин

1.5.1. Информатика: Алгоритмизация и программирование; языки программирования высокого уровня; базы данных; программное обеспечение и технологии программирования; локальные и глобальные сети ЭВМ; основы защиты информации и сведений, составляющих государственную тайну; методы защиты информации.

1.5.2. Метрология, стандартизация и сертификация: Теоретические основы метрологии. Основные понятия, связанные с объектами измерения: свойство, величина, количественные и качественные проявления свойств объектов материального мира. Основные понятия, связанные со средствами измерений (СИ). Закономерности формирования результата измерения, понятие погрешности, источники погрешностей.

1.5.3. Информационные технологии: Модели процессов передачи, обработки, накопления данных в информационных системах; системный подход к решению функциональных задач и к организации информационных процессов в системах; глобальная, базовая и конкретные информационные технологии; особенности информационных технологий.

1.5.4. Теория информационных процессов и систем: Системный анализ; качественные и количественные методы описания информационных систем; кибернетический подход; динамическое описание информационных систем; каноническое представление информационной системы; агрегатное описание информационных систем. Операторы входов и выходов; принципы минимальности информационных связей агрегатов; агрегат как случайный процесс; информация и управление. Модели информационных систем.

1.5.5. Управление данными: Информация и данные; предметная область банка данных; роль и место банков данных в информационных системах; пользователи банков данных; преимущества централизованного управления данными; база данных как информационная модель предметной области; система управления базой данных (СУБД); администратор базы данных.

1.5.6. Информационные сети: Модели и структуры информационных сетей; информационные ресурсы сетей; теоретические основы современных информационных сетей; базовая эталонная модель Международной организации стандартов; компоненты информационных сетей; коммуникационные подсети; моноканальные подсети; циклические подсети; узловые подсети; методы маршрутизации информационных потоков; методы коммутации информации; протокольные реализации; сетевые службы; модель распределенной обработки информации.

1.5.7. Основы теории управления: Управление и информатика; общие принципы системной организации; устойчивость, управляемость и наблюдаемость; инвариантность и чувствительность систем управления; математические модели объектов и систем управления; формы представления моделей;

методы анализа и синтеза систем управления. Цифровые системы управления; использование микропроцессоров и микро-ЭВМ в системах управления.

1.5.8. Моделирование систем: Классификация видов моделирования; имитационные модели информационных процессов; математические методы моделирования информационных процессов и систем; планирование имитационных экспериментов с моделями.

1.5.9. Архитектура ЭВМ и систем: Области применения ЭВМ различных классов; функциональная и структурная организация процессора; организация памяти ЭВМ; основные стадии выполнения команды; организация прерываний в ЭВМ; организация ввода-вывода; периферийные устройства; архитектурные особенности организации ЭВМ различных классов; параллельные системы.

1.5.10. Операционные системы: Вычислительный процесс и его реализация с помощью ОС; основные функции ОС; обзор современных ОС и операционных оболочек; стандартные сервисные программы; машинно-зависимые свойства ОС; управление вычислительными процессами, вводом-выводом, реальной памятью; управление виртуальной памятью.

1.5.11. Технология программирования: Основные этапы решения задач на ЭВМ; критерии качества программы; диалоговые программы; дружественность, жизненный цикл программы; постановка задачи и спецификация программы; способы записи алгоритма; программа на языке высокого уровня; стандартные типы данных. Представление основных структур программирования: итерация, ветвление, повторение; процедуры; типы данных, определяемые пользователем; записи; файлы; динамические структуры данных.

1.5.12. Информационная безопасность и защита информации: Защита информации при реализации информационных процессов (ввод, вывод, передача, обработка, накопление, хранение); организационное обеспечение информационной безопасности; защита информации от несанкционированного доступа; математические и методические средства защиты.

1.5.13. Корпоративные информационные системы: Структура корпораций и предприятий; архитектура корпоративных информационных систем (КИС); КИС для автоматизированного управления; КИС для административного управления; информационные технологии управления корпорацией; выбор аппаратно программной платформы; транспортные подсистемы; построение локальных и глобальных связей.

1.5.14. Администрирование в информационных системах: Функции, процедуры и службы администрирования; объекты администрирования; программная структура; методы администрирования. Службы управления конфигурацией, контролем характеристик, ошибочными ситуациями, учетам и безопасностью; службы управления общего пользования; информационные службы.

1.5.15. Интеллектуальные информационные системы: Информационные системы, имитирующие творческие процессы; информация и данные; системы интеллектуального интерфейса для информационных систем; интеллектуальные информационно-поисковые системы; экспертные системы. Ин-

формационные модели знаний; логико-лингвистические и функциональные семантические сети.

1.5.16. Мультимедиа технология: Классификация и области применения мультимедиа приложений; мультимедиа продукты учебного назначения; аппаратные средства мультимедиа технологии; типы и форматы файлов; текстовые файлы; растровая и векторная графика; гипертекст; звуковые файлы; трехмерная графика и анимация; видео; виртуальная реальность; программные средства для создания и редактирования элементов мультимедиа.

1.5.17. Надежность информационных систем: Основные определения теории надежности; классификация отказов информационных систем; характеристики надежности при внезапных и постепенных отказах; показатели надежности при хранении информации; комплексные показатели надежности информационных систем; факторы, влияющие на надежность информационных систем.

2. Содержание дисциплины

2.1. Федеральный компонент

СД Цикл специальных дисциплин

2.2. Лекционные занятия

2.2.1. Введение. Основные направления развития проектирования Информационных систем. Проблемы проектирования ИС. Мировые концепции управления ИС.(3 часа)

2.2.2. Тема 1. Этапы проектирования ИС, состав работ и проектной документации. (2 часа)

2.2.3. Тема 2. Функциональные и обеспечивающие подсистемы ИС. (4 часа)

2.2.4. Тема 3. Методологические основы проектирования ИС. (4 часа)

2.2.5. Тема 4. Состав стадий и этапов канонического проектирования ИС. Состав и содержание работ на предпроектной стадии создания ИС. (2 часа)

2.2.6. Тема 5. Состав и содержание работ на стадии техно-рабочего проектирования. Состав и содержание работ на стадиях внедрения, эксплуатации и сопровождения проекта. (2 часа)

2.2.7. Тема 6. Понятие унифицированной системы документации. Проектирование унифицированной системы документации. (2 часа)

2.2.8. Тема 7. Проектирование экранных форм электронных документов. Понятие информационной базы и способы ее организации. Проектирование информационной базы при различных способах организации. (4 часа)

2.2.9. Тема 8. Основные понятия и классификация технологических процессов обработки данных. Показатели оценки эффективности и выбор варианта организации технологических процессов. (2 часа)

2.2.10. Тема 9. Проектирование процессов получения первичной информации. Проектирование процесса загрузки и ведения информационной базы. (2 часа)

2.2.11. Тема 10. Проектирование технологических процессов обработки данных в пакетном режиме. Проектирование технологически процессов обработки данных в диалоговом режиме. (2 часа)

2.2.12. Тема 11. Реинжиниринг бизнес-процессов и проектирование корпоративной ЭИС. (4 часа)

2.2.13. Тема 12. Проектирование клиент-серверных корпоративных ИС. (4 часа)

2.2.14. Тема 13. Основные понятия и классификация CASE – технологий. Функционально-ориентированное проектирование ИС. (2 часа)

2.2.15. Тема 14. Объектно-ориентированное проектирование ИС. Прототипное проектирование ИС (RAD-технология). (2 часа)

2.2.16. Тема 15. Типовое проектирование ИС. (4 часа)

2.3. Лабораторные занятия

2.3.1. Теоретическое введение в предметную область. (2 часа)

2.3.2. Методология IDEF0. (2 часа)

2.3.3. Дополнение моделей процессов диаграммами DFD и WorkFlow (IDEF3). (2 часа)

2.3.4. Разработка отчетов в BPWin. (2 часа)

2.3.5. Методология IDEF1X. (4 часа)

2.3.6. Создание логической модели, ERD-диаграммы. (4 часа)

2.3.7. Нормализация. Создание физической модели, используя CASE-средства ERWin. (4 часа)

2.3.8. Отчеты в ERWin. (4 часа)

2.3.9. Изучение основных этапов проведения проектирования в Rational Rose 2000. (4 часа)

2.3.10. Диаграммы вариантов использования, язык UML. (4 часа)

2.3.11. Диаграммы классов. (4 часа)

2.3.12. Диаграммы взаимодействия. (4 часа)

2.3.13. Диаграммы состояний, (4 часа)

2.3.14. Диаграммы пакетов, компонентов и размещения. (4 часа)

2.3.15. Генерация исходных текстов диаграмм. (4 часа)

2.3.16. Обратное проектирование. (4 часа)

2.3.17. Сравнение объектно-ориентированного и структурного методов проектирования. (4 часа)

2.4. Курсовое проектирование

2.4.1. Методология построения (реорганизации) информационно-управляющей системы включает несколько этапов:

- Осуществляется обследование и анализ структурных подразделений организации в целях определения: функциональных задач, функционального взаимодействия, внутреннего документооборота, информационных потоков и информационного взаимодействия, применяемых средств автоматизации.
- Разрабатываются функционально-информационные модели технологии работы подразделений. Выполняется объединение в единую функциональную модель технологии работы организации, а также создается информационная модель единого документооборота.
- В результате выполнения комплекса работ формируются предложения по совершенствованию организационной структуры, технологии, автоматизированной системы, компьютерной сети.

2.4.2. Примерные темы для КП:

2.4.2.1 Разработка системы описания и хранения слабоформализованных документов.

2.4.1.1. Проектирование информационной системы Учреждения юстиции по государственной регистрации прав на недвижимое имущество и сделок с ним.

2.4.1.2. Проектирование информационной системы «Учет и анализ дорожно-транспортных происшествий».

2.4.1.3. Информационная система «Учет и трудоустройство несовершеннолетних» Департамента Государственной службы занятости населения».

2.4.1.4. Проектирование базы данных статистического учета пациентов, выбывших из стационара АОКБ.

2.4.1.5. Проектирование системы автоматизированного документооборота для отдела «Канцелярии».

2.4.1.6. Проектирование подсистемы учета эксплутационного отдела.

2.4.1.7. Проектирование информационной системы отдела снабжения и сбыта.

2.4.1.8. Проектирование информационной системы планово-экономического отдела строительной компании.

2.4.1.9. Информационная система управления потоками данных между отделами.

2.4.1.10. Разработка комплекса программных средств, направленного на автоматизацию токарно-фрезерного и слесарного цехов.

2.4.1.11. Проектирование автоматизированной биллинговой системы для сети Wi-Fi.

2.4.1.12. Разработка системы электронного документооборота контрольного отдела.

2.4.1.13. Проектирование информационной web-системы Администрации города.
2.4.1.14. Проектирование системы управления содержимым информационного портала.
2.4.1.15. Автоматизация документооборота среднего учебного заведения.
2.4.1.16. Организация решения задач анализа выпуска и реализации продукции с применением глобальных сетей.

2.5. Самостоятельная работа студентов

Самостоятельная работа студента, это работа над рекомендованной литературой, контроль осуществляется оценкой курсового проекта.

Самостоятельная работа включает в себя изучение следующих разделов.

2.5.1. Информационный процесс управления производством.
2.5.2. Организация информационного обеспечения задач оперативного управления.
2.5.3. Разработка моделей организации и ИС.
2.5.4. Процесс проектирования и жизненный цикл продукта.
2.5.5. Построение ИС с помощью CASE-средств.
2.5.6. Информационные системы и сети.
2.5.7. Планирование сетей.
2.5.7.1. Физическая среда передачи данных.
2.5.7.2. Сетевые модели.
2.5.7.3. Сетевые архитектуры.
2.5.7.4. Функционирование сети.
2.5.8. Администрирование сетей.

2.6. Вопросы на экзамен

2.6.1. Современные принципы развития ИС.
2.6.2. Основные проблемы в проектировании ИС.
2.6.3. Мировые концепции в управлении ИС.
2.6.4. Этапы проектирования ИС.
2.6.5. Состав работ при проектировании ИС.
2.6.6. Состав проектной документации.
2.6.7. Поддержка процесса проектирования ИС и документирование.
2.6.8. Цели проектирования ИС.
2.6.9. Качество проектирования ИС.
2.6.10. Эффективность методик разработки ИС.
2.6.11. Жизненный цикл ИС.
2.6.12. Эффективность технологий проектирования ИС.
2.6.13. Методы и средства автоматизации учрежденческой деятельности.
2.6.14. Методы автоматизации работы отделов, учреждений, фирм, предприятий.

2.6.15.Средства офисной автоматизации и коллективной работы в сети.
2.6.16.Средства работы и управления электронными документами.
2.6.17.Средства автоматизации документооборота.
2.6.18.Структурный подход к проектированию ИС.
2.6.19.Моделирование потоков данных.
2.6.20.Методология функционального проектирования.
2.6.21.Моделирование данных.
2.6.22.Инструментальные средства проектирования.
2.6.23.Информационная поддержка управленческой деятельности.
2.6.24.Компоненты пользовательского интерфейса.
2.6.25.Стратегия разработки интерфейса.
2.6.26.Математическое обеспечение Информационных систем.
2.6.27. Организационное и правовое обеспечение Информационных систем.
2.6.28.Техническое и эргономическое обеспечение ИС.
2.6.29.Лингвистическое обеспечение ИС.
2.6.30.Программное обеспечение ИС.
2.6.31.Классификация прикладного программного обеспечения ИС.
2.6.32.Основные параметры качества ПО.
2.6.33.Информационное обеспечение.
2.6.34.Система классификации и кодирования.
2.6.35.Унифицированная система документации.
2.6.36.Особенности разработки прикладных ИС на основе ПЭВМ.
2.6.37.Структурирование программ на уровне модулей.
2.6.38.Реорганизация АИиУС предприятия.
2.6.39. Реорганизация деятельности предприятия при проектировании ИС.
2.6.40.Основные подходы к созданию ИС.
2.6.41.Типовой состав функциональных подсистем ИС.
2.6.42.Типовой состав обеспечивающих подсистем ИС
2.6.43.Использование архитектуры «клиент-сервер».
2.6.44.Единая система управления базами данных и преимущества ее использования.
2.6.45.Жизненный цикл проектируемой ИС.
2.6.46. Разработка моделей организации информационных потоков, анализ.
2.6.47.Автоматизированные информационные системы и сети.
2.6.48.Планирование сетей для предприятий.
2.6.49.Физическая среда передачи данных.
2.6.50.Сетевые модели.
2.6.51.Сетевые архитектуры.
2.6.52.Функционирование сетей.
2.6.53.Администрирование сетей.
2.6.54.Методы защиты информации в ИС.
2.6.55.Защита информации в ПЭВМ.

2.6.56.Защита информации в ЛВС.
2.6.57.Защита информации в глобальных сетях.
2.6.58.Информация как объект частной собственности.
2.6.59.Информация как коммерческая тайна.
2.6.60.Правовые вопросы и ИС.

Экзамен предусматривает два теоретических вопроса.

Экзаменуемый студент должен подтвердить знание фундаментальных основ:

- Проектирование ИС

знание и использование методологии проектирования ИС;

умение строить модели различных этапов жизненного цикла программного продукта.

2.7. Оценочные критерии

При оценке знаний на экзамене учитывается:

правильность и осознанность изложения содержания ответа на вопросы, полнота раскрытия понятий и закономерностей, точность употребления и трактовки общенаучных и специальных терминов;

степень сформированности интеллектуальных и научных способностей экзаменуемого;

- самостоятельность ответа;
- речевая грамотность и логическая последовательность ответа.

Оценка "отлично":

полно раскрыто содержание вопросов в объеме программы и рекомендованной литературы;

четко и правильно даны определения и раскрыто содержание концептуальных понятий, закономерностей, корректно использованы научные термины;

для доказательства использованы различные теоретические знания, выводы из наблюдений и опытов;

ответ самостоятельный, исчерпывающий, без наводящих дополнительных вопросов, с опорой на знания, приобретенные в процессе специализации по выбранному направлению информатики.

Оценка "хорошо":

раскрыто основное содержание вопросов;

в основном правильно даны определения понятий и использованы научные термины;

ответ самостоятельный;

определения понятий неполные, допущены нарушения последовательности изложения, небольшие неточности при использовании научных терминов или в выводах и обобщениях, исправляемые по дополнительным вопросам экзаменаторов.

Оценка "удовлетворительно":

усвоено основное содержание учебного материала, но изложено фрагментарно, не всегда последовательно;

определение понятий недостаточно четкое;

не использованы в качестве доказательства выводы из наблюдений и опытов или допущены ошибки при их изложении;

допущены ошибки и неточности в использовании научной терминологии, определении понятий.

Оценка "неудовлетворительно":

ответ неправильный, не раскрыто основное содержание программного материала;

не даны ответы на вспомогательные вопросы экзаменаторов;

допущены грубые ошибки в определении понятий, при использовании терминологии.

3. Литература

3.1. Основная литература:

1.1.1. Автоматизация управления предприятием. / В.В.Баронов, Г.Н.Калянов, Ю.Н.Попов и др. –М.:Инфра-М, 2000.

1.1.2. Автоматизированные информационные технологии в экономике: Учебник / Под ред.проф.Г.А.Титоренко. –М.:Компьютер, ЮНИТИ, 1998.

1.1.3. Алан Р. Саймон. Стратегические технологии баз данных: менеджмент на 2000 год /Пер. с англ. и предисл. М.Р.Когаловского. –М.:Финансы и статистика, 1999.

1.1.4. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. –М.:Конкорд, 1996.

1.1.5. Вендеров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. –М.:Финансы и статистика, 2000.

3.2. Дополнительная литература:

3.2.1. Гост 19.001-77. Единая система программной документации: Общие положения. –М.:Изд.-во стандартов, 1994.

3.2.2. Гост 19.101-77. Единая система программной документации: Виды программ и программных документов. –М.:Изд.-во стандартов, 1994.

3.2.3. Гост 19.102-77. Единая система программной документации: Стадии разработки. –М.:Изд.-во стандартов, 1994.

3.2.4. Гост 19.105-78. Единая система программной документации: Общие требования к программным документам. –М.:Изд.-во стандартов, 1994.

3.2.5. Гост 19.201-78. Единая система программной документации: Техническое задание. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.6. Гост 19.202-78. Единая система программной документации: Спецификация. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.7. Гост 19.502-78. Единая система программной документации: Описание применения. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.8. Гост 19.404-79. Единая система программной документации: Пояснительная записка. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.9. Гост 19.503-79. Единая система программной документации: Руководство системного программиста. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.10. Гост 19.504-79. Единая система программной документации: Руководство программиста. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.11. Гост 19.505-79. Единая система программной документации: Руководство оператора. Требования к содержанию и оформлению. –М.:Изд.-во стандартов, 1994.

3.2.12. Гост 19.507-79. Единая система программной документации: Ведомость эксплуатационных документов. –М.:Изд.-во стандартов, 1994.

3.2.13. Гост 3.11.09-82. Система технологической документации: Термины и определения основных понятий. –М.:Изд.-во стандартов, 1994.

3.2.14. Гост 20.886-85. Организация баз данных в системах обработки данных: Термины и определения. –М.:Изд.-во стандартов, 1994.

3.2.15. Гост 6.61.1-87. Единая система классификации и кодирования технико-экономической информации. Основные положения. –М.:Изд.-во стандартов, 1994.

3.2.16. Гост 24.402-88. Организация баз данных в системах обработки данных: Термины и определения. –М.:Изд.-во стандартов, 1994.

3.2.17. Гост 28.147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. –М.:Изд.-во стандартов, 1991.

3.2.18. Гост 34.201-89. Виды, комплектность и обозначение документов при создании автоматизированных систем. –М.:Изд.-во стандартов, 1991.

3.2.19. Гост 34.602-89. Техническое задание на создание автоматизированной системы. –М.:Изд.-во стандартов, 1991.

3.2.20. Гост 15.971-90. Системы обработки информации. Термины и определения. –М.:Изд.-во стандартов, 1991.

3.2.21. Гост 19.701-90. Единая система программной документации: Схемы алгоритмов, программ данных и систем. Условные обозначения и правила выполнения. –М.:Изд.-во стандартов, 1994.

3.2.22. Гост 19.781-90. Обеспечение систем обработки информации программное: Термины и определения. –М.:Изд.-во стандартов, 1994.

3.2.23. Гост 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы: Автоматизированные системы: Термины и определения. –М.:Изд.-во стандартов, 1991.

3.2.24. Гостехкоммисия России. Руководящий документ. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от НСД к информации. –М., 1992.

3.2.25. Диго С.М. Проектирование и эксплуатация баз данных. –М.:Финансы и статистика, 1995.

3.2.26. Зиндер Е.З. Новое системное проектирование: информационные технологии и бизнес-реинжиниринг// СУБД. -1996. -№4.

3.2.27. Козлов В.А. Открытые информационные системы. –М.:Финансы и статистика, 1999.

3.2.28. Липаев В.В. Системное проектирование сложных программных средств для информационных систем. –М.:Синтег, 1999.

3.3. Литература для самостоятельного изучения

3.3.1. Маклаков С.В. BPWin и ERWin. CASE-средства разработки информационных систем. –М.: ДИАЛОГ-МИФИ, 2000.

3.3.2.Тельнов Ю.Ф. Интеллектуальные информационные системы в экономике: Учеб. Пособие. –М.:СИНТЕГ, 1999.

3.3.3. Фаулер М., Скотт К. UML в кратком изложении: Применение стандартного языка объектного моделирования: Пер. с англ. –М.: Мир, 1999.

4. Учебно – методическая (технологическая) карта дисциплины

Номер недели	Номер темы	изучаемые на лекцииВопросы,	Занятия		и методические пособияИспользуемые наглядные	Самостоятельная работа студентов		Форма контроля
			Практические	Лабораторные		Содержание	Часы	
1	2	3	4	5	6	7	8	9
8 семестр								
1	Вв[1]	2.2.1	-	2.3.2-2.3.3	3.3.1	2.5.1	4	злр [2]
2	1	2.2.2	-	2.3.4-2.3.5	3.3.1	2.5.9.1	4	
3	2	2.2.3	-	2.3.5	3.3.1	2.5.9.1	4	злр
4	2	2.2.3	-	2.3.6	3.3.1	2.5.9.2	5	
5	3	2.2.4	-	2.3.7	3.3.1	2.5.9.2	5	
6	3	2.2.4	-	2.3.8	3.3.1	2.5.9.3	4	

7	4	2.2.5	-	2.3.9	3.3.3	2.5.9.3	4	зач.[3]
9 семестр								
1	5	2.2.6	-	2.3.10	3.3.3	2.5.9.3	4	
2	6	2.2.7	-	2.3.10	3.3.3	2.5.9.3	5	злр
3	7	2.2.8	-	2.3.11	3.3.3	2.5.2	5	
4	7	2.2.8	-	2.3.11	3.3.3	2.5.2	5	злр,
5	8	2.2.9	-	2.3.12	3.3.3	2.5.3	5	сб.[4]
6	9	2.2.10	-	2.3.12	3.3.3	2.5.3	5	
7	10	2.2.11	-	2.3.13	3.3.3	2.5.4	5	злр
8	11	2.2.12	-	2.3.13	3.3.3	2.5.4	5	
9	11	2.2.12	-	2.3.14	3.3.3	2.5.5	5	
10	12	2.2.13	-	2.3.14	3.3.3	2.5.5	5	сб.
11	12	2.2.13	-	2.3.15	3.3.3	2.5.6	5	
12	13	2.2.14	-	2.3.15	3.3.3	2.5.7	5	
13	14	2.2.15	-	2.3.16	3.3.3	2.5.8	5	злр
14	15	2.2.16	-	2.3.16	3.3.3		5	
15	15	2.2.16	-	2.3.17	3.3.3		5	зач.[4]

1. Введение,
2. Защита отчета о выполнении лабораторной работы,
3. Зачет по лабораторным работам,
4. Собеседование по результатам.

3. ГРАФИК САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Содержание	Объем в часах	Сроки и форма контроля
Изучение основных этапов проведения проектирования в Rational Rose 2000. 1. Взаимодействие объектов: Создать диаграмму Последовательностей; Создать диаграмму Кооперации; Сохранить файл модели, составить отчет. 2. Классы и пакеты: Создать диаграмму Классов; Сохранить файл модели, составить отчет. Атрибуты и операции: Модифицировать диаграмму Классов; Сохранить файл модели, составить отчет. Связи:	15 час. 15 час.	Собеседование (8 неделя) Собеседование (8 неделя)

Модифицировать диаграмму Классов; Сохранить файл модели, составить отчет.		

4. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Выполнение самостоятельной работы по курсу «Проектирование ИС», имеет целью выработку у студентов навыков, в том числе и в следующих направлениях:

1. Применение соответствующих методологий для разработки информационных систем и программного обеспечения;
2. Применение языка UML для моделирования и проектирования информационных систем;
3. Применение соответствующего программного инструментария - Rational Rose 2001.

Компания Rational Software является лидирующей в области создания методологий и программных решений, ориентированных на программистов, аналитиков, тестировщиков. Спектр выпускаемого программного обеспечения целиком покрывает потребность всех участников проекта: от аналитиков до разработчиков и внедренцев. Все программно-методологические решения - плод многолетнего труда аналитиков и разработчиков как самой Rational, так и ее партнеров. В итоге все решения были собраны воедино. Так появился RUP - Rational Unified Process - методологическая энциклопедия, в которой описаны все шаги, необходимые для создания качественного программного продукта. Пользуясь подобной энциклопедией и применяя соответствующие инструменты, рекомендуемые Rational, команда будет создавать обеспечение качественно и в срок. "Строй быстрей и качественней!" - вот лозунг, выдвигаемый Rational.

Особое место в RUP занимают проектирование и конфигурационное управление. Особо выделяются они потому, что те два инструмента, которые поддерживаются на данных этапах (Rational Rose и Rational ClearCase), используются на протяжении всего жизненного цикла разработки программного обеспечения. Если Rose со своей стороны используют и проектировщики, и разработчики, и аналитики (практически добрая половина коллектива), то ClearCase используют все, поскольку результатом любой деятельности в мире информационных технологий является файл, который где-то необходимо хранить и не просто хранить, а знать все изменения, которые были в него внесены на каждом этапе разработки информационной системы.

Rational Rose, являясь объектно-ориентированным средством проектирования, способна моделировать ситуации любой сложности: от проектиро-

вания банковской системы до разработки кода на C++. В умелых руках Rose неоценимый инструмент! Rational Rose в отличие от подобных средств проектирования способна проектировать системы любой сложности, то есть инструментарий программы допускает как высокоуровневое (абстрактное) представление (например, схема автоматизации предприятия), так и низкоуровневое проектирование (интерфейс программы, схема базы данных, частичное описание классов). Вся мощь программы базируется всего на 7 диаграммах, которые в зависимости от ситуации способны описывать различные действия.

Давайте попробуем разобраться, что же реально даст данный инструмент отдельным участникам проекта: проектировщикам, аналитикам, разработчикам.

Проектировщикам. В большинстве случаев подрядчик не может написать качественное программное обеспечение в установленный срок, поскольку заранее не были оговорены с заказчиком наиболее важные моменты в системе. Например, заказчик забыл рассказать об очень важной составляющей своей системы, а вспомнил об этом, когда увидел (запустил) предложенную готовую программу. Соответственно, проблема на данном этапе - взаимопонимание двух сторон. Значит, если проектировщик с заказчиком посредством моделирования предметной области в Rational Rose четко и скрупулезно описали каждую деталь и увидели ее в виде понятных диаграмм, то проблему взаимопонимания можно отбросить. Таким образом, при разработке современных информационных систем много времени уделяется проектированию (моделированию предметной области), поскольку необходимо изначально договориться с заказчиком обо всех нюансах, а не вносить в режиме "пожарной команды" изменения на более поздних этапах. То есть Rational Rose поможет на концептуальном уровне разобраться с генеральным планом автоматизации. Для улучшения взаимопонимания обеих сторон совместно с Rose применяют инструмент SoDA, позволяющий на основе построенной модели дать полный отчет по ее состоянию, соответствующий всем общепризнанным мировым стандартам (в частности ISO 9000). Как видим, внедрение Rose на предприятии позволяет, в дополнение к вышеописанному, структурировать сопроводительную документацию, привести ее к необходимому стандарту с минимальными девиациями.

Разработчикам. Не меньше возможностей Rose дает и разработчикам. Давайте снова повторим очевидную вещь: информационные системы конца 90 гг. вышли на такой уровень сложности, что справиться с ними уже под силу только крупным компаниям с большим количеством узкоспециализированных разработчиков. Времена программистов-одиночек ушли в небытие. В современных условиях механизм "сам все сделаю" дает явный сбой. В качестве второй проблемы можно отметить некоторую текучесть кадров на отдельно взятом предприятии. Каждый раз, при включении нового сотрудника в проект, необходимо посвящать его во все детали проекта, на что уходит драгоценное время коллег, отрываемых от основной работы. При наличии же Rose достаточно показать все диаграммы проекта и предоставить проектную

документацию, сгенерированную на основе полученной модели, как все станет на свои места. Разработчик увидит как весь проект в целом, так и свою часть. Конкретно же в плане разработки Rose поддерживает проектирование, основанное на двух способах: прямом и обратном. В режиме прямого проектирования разработчик рисует определенным образом диаграммы классов и их взаимодействия, а на выходе получает сгенерированный код на определенном языке программирования. В режиме же обратного проектирования возможно построение модели на базе имеющегося исходного кода. Из этого следует самая главная возможность для разработчиков: повторное проектирование (Round-trip), когда разработчик описывает классы в Rose, генерирует код, дописывает необходимую полнофункциональную часть и снова закачивает в Rose, для представления того, что же система получила в результате его действий.

Важнейшим свойством Rational Rose принято считать открытость архитектуры, что позволяет дополнять имеющийся в ней инструментарий новыми функциями. Например, в стандартном варианте Rose не поддерживает кодогенерацию на Ассемблере: Путем написания дополнительного модуля подобную проблему можно решить.

Вот список включенных стандартных модулей: C++, ADA, CORBA, Visual Basic, XML, COM, Oracle). То есть Rational Rose способна проводить прямое и обратное проектирование в данных системах.

Подведем первые итоги того, что может делать Rational Rose

- Проектировать системы любой сложности
- Давать развернутое представление о проекте в сочетании со средствами документирования (SoDA)
- Проводить кодогенерацию
- Проводить обратное проектирование имеющихся систем
- Имеет открытый для дополнений интерфейс
- Интегрируется со средствами разработки (Visual Studio)
- Поддержка языка UML
- Наличие средств автоматического контроля, в том числе проверки соответствия двух моделей
- Удобный для пользователя графический интерфейс
- Многоплатформенность
- Интегрируемость с другими инструментальными средствами, поддерживающими жизненный цикл программных систем, в том числе со средством управления требованиями (Requisite Pro), со средствами тестирования (SQA Suite, Performance Studio), со средствами конфигурационного управления (ClearCase, PVCS).

Итак, от общих тем перейдем непосредственно к тому, что умеет делать CASE Rational Rose. Являясь объектно-ориентированным инструментом моделирования, Rose базируется на UML (Universal Modeling Language) - универсальном языке моделирования, который был разработан компанией Rational именно с целью создания наиболее оптимального и универсального языка для описания как предметной области, так и конкретной задачи в про-

граммировании. Любая задача программируется при помощи определенных диаграмм. UML поддерживает построение следующих диаграмм:

- Activity diagram (диаграммы описаний технологий, процессов, функций).
- Use case diagram (диаграммы функций).
- Class diagram (диаграммы классов).
- State diagram (диаграммы состояний);
- Sequence diagram (диаграммы последовательностей действий);
- Collaboration diagram (диаграммы взаимодействий
- Component diagram (диаграммы компонент);
- Deployment diagram (диаграммы топологии).

Соответственно, Rational Rose 2000 является инструментом, который позволяет строить указанные диаграммы при проектировании программных систем. К сожалению, объем статьи не позволяет описать назначение всех диаграмм и спецификаций! Но мы попробуем разобраться в инструменте с точки зрения разработчика, для простоты используя только один тип диаграмм - Class Diagramm.

Все разработчики сталкиваются с ситуацией, когда приходится проектировать большие классы. При ручном вводе и объявлении имеется ряд подводных камней: во-первых, постановщик задач, как правило, описывает "что нужно" на словах, в крайнем случае, с минимальным бумажным сопровождением; во-вторых, разработчик, создающий систему, опять-таки в большинстве случаев игнорирует все комментарии, которыми необходимо сопровождать программный код. Что же получается в итоге? Постановщик задач путается в программе, разработчик сам не помнит, что к чему, а если на его место взят новый сотрудник: Тут на ум приходит еще одно традиционное для России высказывание разработчика: "мне проще все написать заново". И ведь пишут: Тормозя производство программного продукта. Дело в том, что к разработке ПО относятся как к искусству, а необходимо относиться, как к производственному процессу со строгим распределением ролей, полномочий и пр.

4.1. Создание вариантов использования.

Пример выполнения упражнения, пример Диаграммы Вариантов использования.

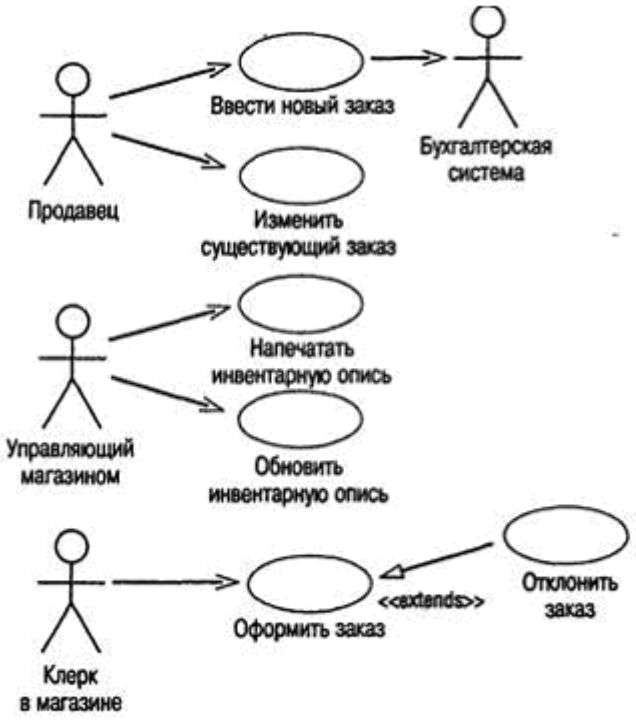


Рис. 4.1. Диаграмма Вариантов Использования для системы обработки заказов

Создание диаграммы Вариантов Использования, вариантов использования и действующих лиц.

1. Дважды щелкнув мышью на Главной диаграмме Вариантов Использования (Main) в браузере, откройте ее.
2. С помощью кнопки Use Case (Вариант использования) панели инструментов поместите на диаграмму новый вариант использования.
3. Назовите его "Ввести новый заказ".
4. Повторив этапы 2 и 3, поместите на диаграмму остальные варианты использования: Изменить существующий заказ, Напечатать инвентарную опись, Обновить инвентарную опись, Оформить заказ, Отклонить заказ.
5. С помощью кнопки Actor (Действующее лицо) панели инструментов поместите на диаграмму новое действующее лицо.
6. Назовите его "Продавец".
7. Повторив шаги 5 и 6, поместите на диаграмму остальных действующих лиц: Управляющий магазином, Клерк магазина, Бухгалтерская система.

Создание абстрактного варианта использования.

1. Щелкните правой кнопкой мыши на варианте использования "Отклонить заказ" на диаграмме.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).

3. Установите флажок Abstract (Абстрактный), чтобы сделать этот вариант использования абстрактным.

Добавление ассоциаций.

1. С помощью кнопки Unidirectional Association (Однонаправленная ассоциация) панели инструментов нарисуйте ассоциацию между действующим лицом Продавец и вариантом использования "Ввести новый заказ".
2. Повторив шаг 1, поместите на диаграмму остальные ассоциации.

Добавление связи расширения.

1. С помощью кнопки Generalization (Обобщение) панели инструментов нарисуйте связь между вариантом! использования "Отклонить заказ" и вариантом использования "Оформить заказ". Стрелка должна идти. от первого варианта использования ко второму. Связь расширения означает, что вариант использования "Отклонить заказ" при необходимости дополняет функциональные возможности варианта использования "Оформить заказ".
2. Щелкните правой кнопкой мыши на новой связи между вариантами использования "Отклонить заказ" и "Оформить заказ".
3. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
4. В раскрывающемся списке стереотипов введите слово extends (расширение), затем нажмите ОК.
5. Надпись «extends» появится на линии данной связи.

Добавление описаний к вариантам использования.

1. Выделите в браузере вариант использования "Ввести новый заказ".
2. В окне документации введите следующее описание: "Этот вариант использования дает клиенту возможность ввести новый заказ в систему".
3. С помощью окна документации добавьте описания ко всем остальным вариантам использования.

Добавление описаний к действующему лицу.

1. Выделите в браузере действующее лицо Продавец.
2. В окне документации введите следующее описание: "Продавец — это служащий, старающийся продать товар".
3. С помощью окна документации добавьте описания к остальным действующим лицам.

Прикрепление файла к варианту использования.

1. Создайте свой собственный файл OrderFlow.doc и внесите в него поток событий, как показано ниже.
   Основной поток событий для варианта использования "Ввести новый заказ'
   1. *Продавец выбирает в имеющемся меню пункт “Создать новый заказ”.*
   2. *Система выводит форму “Детали заказа”.*
   3. *Продавец вводит номер заказа, заказчика и то, что заказано.*
   4. *Продавец сохраняет заказ.*
   5. *Система создаёт новый заказ и сохраняет его в базе данных.*
2. Щелкните правой кнопкой мыши на варианте использования "Ввести новый заказ".
3. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
4. Перейдите на вкладку Files (Файлы).
5. Щелкните правой кнопкой мыши в белой области и в открывшемся меню выберите пункт Insert File (Вставить файл).
6. Укажите файл OpenFlow. doc и нажмите на кнопку Open (Открыть), чтобы прикрепить файл к варианту использования.

4.2. Взаимодействие объектов.

Пример выполнения упражнения, создание диаграммы Последовательностей и Кооперации.

Постановка задачи, описание сценариев:

- Продавец вводит новый заказ.
- Продавец пытается ввести заказ, но товара нет на складе.
- Продавец пытается ввести заказ, но при его сохранении в базе данных возникает ошибка.

Создание диаграмм Последовательности и Кооперативных диаграмм для сценария "Ввести новый заказ".

Создание диаграмм Взаимодействия

Создайте диаграмму Последовательности и Кооперативную диаграмму, отражающую ввод нового заказа в систему обработки заказов. Это только одна из диаграмм, необходимых для моделирования варианта использования "Ввести новый заказ". Она соответствует успешному варианту хода событий. Для описания того, что случится, если возникнет ошибка или если пользователь выберет другие действия из предложенных, придется разработать дополнительные диаграммы. Каждый "альтернативный поток варианта использования может быть промоделирован с помощью собственных диаграмм Взаимодействия.

Этапы выполнения

Настройка

1. В меню модели выберите пункт Toots > Options (Инструменты > Параметры).
2. Перейдите на вкладку Diagram (Диаграмма).
3. Установите флажки Sequence numbering, Collaboration numbering и Focus of control.
4. Нажмите OK, чтобы выйти из окна параметров.

Создание диаграммы Последовательности

1. Щелкните правой кнопкой мыши на Логическом представлении браузера.
2. В открывшемся меню выберите пункт New > Sequence Diagram (Создать > Диаграмма Последовательности).
3. Назовите новую диаграмму Add order (Ввод заказа).
4. Дважды щелкнув на этой диаграмме, откройте ее.

Добавление на диаграмму действующего лица и объектов

1. Перетащите действующее лицо Salesperson (Продавец) из браузера на диаграмму.
2. Нажмите кнопку Object (Объект) панели инструментов.
3. Щелкните мышью в верхней части диаграммы, чтобы поместить туда новый объект.
4. Назовите объект Order Options Form (Выбор варианта заказа).
5. Повторив шаги 3 и 4, поместите на диаграмму объекты:
    - Order Detail Form (Форма деталей заказа)
    - Order N1234 (Заказ №1234)

Добавление сообщений на диаграмму

1. На панели инструментов нажмите кнопку Object Message (Сообщение объекта).
2. Проведите мышью от линии жизни действующего лица Salesperson (Продавец) к линии жизни объекта Order Options Form (Выбор варианта заказа).
3. Выделив сообщение, введите его имя — Create new order (Создать новый заказ).
4. Повторив шаги 2 и 3, поместите на диаграмму сообщения:
    - Open form (Открыть форму) — между Order Options Form и Order Detail Form

- Enter order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — между Salesperson и Order Detail Form
- Save the order (Сохранить заказ) — между Salesperson и Order Detail Form
- Create new, blank order (Создать пустой заказ) — между Order Detail Form и Order N1234
- Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — между Order Detail Form и Order N1234
- Save the order (Сохранить заказ) — между Order Detail Form и Order N1234

Завершен первый этап работы. Готовая диаграмма Последовательности представлена на рис. 2. Теперь нужно позаботиться об управляющих объектах и о взаимодействии с базой данных. Как. видно из диаграммы, объект Order Detail Form имеет множество ответственностей, с которыми лучше всего мог бы справиться управляющий объект. Кроме того, новый заказ должен сохранять себя в базе данных сам. Вероятно, эту обязанность лучше было бы переложить на другой объект.

Добавление на диаграмму дополнительных объектов

1. Нажмите кнопку Object панели инструментов.
2. Щелкните мышью между объектами Order Detail Form и Order N1234, чтобы поместить туда новый объект.
3. Введите имя объекта — Order Manager (Управляющий заказами).
4. Нажмите кнопку Object панели инструментов.
5. Новый объект расположите справа от Order N1234.
6. Введите его имя — Transaction Manager (Управляющий транзакциями).

Назначение ответственностей объектам

1. Выделите сообщение 5: Create new, blank order (Создать пустой заказ).
2. Нажав комбинацию клавиш CTRL+D, удалите это сообщение.
3. Повторите шаги 1 и 2 для удаления двух последних сообщений:
   - Set the order number, customer, order items (Вести номер заказа, заказчика и число заказываемых предметов)
   - Save the order (Сохранить заказ)
4. . Нажмите кнопку Object Message панели инструментов.
5. Поместите на диаграмму новое сообщение, расположив его под сообщением 4 между Order Detail Form и Order Manager.
6. Назовите его Save the order (Сохранить заказ).
7. Повторите шаги 4 — 6, добавив сообщения с шестого по девятое и назвав их:

    - Create new, blank order (Создать новый заказ) — между Order Manager и Order N1234
    - Set the order number, customer, order items (Вести номер заказа, заказчика и число заказываемых предметов) — между Order Manager и Order N1234
    - Save the order (Сохранить заказ) — между Order Manager и Transaction Manager
    - Collect order information (Информация о заказе) — между Transaction Manager и Order N1234
8. На панели инструментов нажмите кнопку Message to Self (Сообщение себе).
9. Щелкните на линии жизни объекта Transaction Manager (Управляющий транзакциями) ниже сообщения 9, добавив туда рефлексивное сообщение.
10. Назовите его Save the order information to the database (Сохранить информацию о заказе в базе данных).

Соотнесение объектов с классами

1. Щелкните правой кнопкой мыши на объекте Order Options Form (Выбор варианта заказа).
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт (Создать). Появится окно спецификации классов.
4. В поле Name введите OrderOptions (Выбор заказа).
5. Щелкните на кнопке ОК. Вы вернетесь в окно спецификации объекта.
6. В списке классов выберите класс OrderOptions.
7. Щелкните на кнопке ОК, чтобы вернуться к диаграмме. Теперь объект называется Order Options Form : OrderOptions.
8. Для соотнесения остальных объектов с классами повторите шаги с 1 по 7:
    - Класс OrderDetail соотнесите с объектом Order Detail Form
    - Класс OrderMgr — с объектом Order Manager
    - Класс Order — с объектом Order N1234
    - Класс TransactionMgr — с объектом Transaction Manager.

Соотнесение сообщений с операциями

1. Щелкните правой кнопкой мыши на сообщении 1: Create new order (Создать новый заказ).
2. В открывшемся меню выберите пункт (создать операцию). Появится окно спецификации операции.
3. В поле Name введите имя операции — Create (Создать).

4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться к диаграмме.
5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
6. В открывшемся меню выберите новую операцию CreateO.
7. Повторите шаги с 1 по 6, чтобы соотнести с операциями все остальные сообщения:
    - Сообщение 2: Open form (Открыть форму) соотнесите с операцией Open()
    - Сообщение 3: Enter order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией Submitlnfo()
    - Сообщение 4: Save the order (Сохранить заказ) — с операцией Save()
    - Сообщение 5: Save the order (Сохранить заказ) — с операцией SaveOrderQ
    - Сообщение 6: Create new, blank order (Создать пустой заказ) — с операцией Create()
    - Сообщение 7: Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией Settrrfo()
    - Сообщение 8: Save the order (Сохранить заказ) — с операцией SaveOrderQ
    - Сообщение 9: Collect order information (Информация о заказе) -- с операцией GetlnfoQ
    - Сообщение 10: Save the order information to the database (Сохранить информацию о заказе в базе данных) — с операцией Commit()

Создание Кооперативной диаграммы

Для создания Кооперативной диаграммы достаточно нажать клавишу F5. Если же вы хотите сами проделать все требуемые операции, воспользуйтесь приводимым далее планом.

Создание Кооперативной диаграммы

1. Щелкните правой кнопкой мыши на Логическом представлении в браузере
2. В открывшемся меню выберите пункт New > Collaboration Diagram (Создать > Кооперативная диаграмма).
3. Назовите эту диаграмму Add order (Ввод заказа).
4. Дважды щелкнув мышью на диаграмме, откройте ее.

Добавление действующего лица и объектов на диаграмму

1. Перетащите действующее лицо Salesperson (Продавец) из браузера на диаграмму.
2. Нажмите кнопку Object (Объект) панели инструментов.
3. Щелкните мышью где-нибудь внутри диаграммы, чтобы поместить туда новый объект.
4. Назовите объект Order Options Form (Выбор варианта заказа).
5. Повторив шаги 3 и 4, поместите на диаграмму объекты:
    - Order Detail Form (Форма деталей заказа)
    - Order N1234 (Заказ Ns1234)

Добавление сообщений на диаграмму

1. На панели инструментов нажмите кнопку Object Link (Связь объекта).
2. Проведите мышью от действующего лица Salesperson (Продавец) к объекту Order Options Form (Выбор варианта заказа).
3. Повторите шаги 1 и 2, соединив связями следующие объекты:
    - Действующее лицо Salesperson и объект Order Detail Form
    - Объект Order Options Form и объект Order Detail Form
    - Объект Order Detail Form и объект Order N1234
4. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
5. Щелкните мышью на связи между Salesperson и Order Options Form.
6. Выделив сообщение, введите его имя - Create new order (Создать новый заказ).
7. Повторив шаги с 4 по 6, поместите на диаграмму сообщения:
    - Open form (Открыть форму) - между Order Options Form и Order Detail Form
    - Enter order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) - между Salesperson и Order Detail Form
    - Save the order (Сохранить заказ) - между Salesperson и Order Detail Form
    - Create new, blank order (Создать пустой заказ) - между Order Detail Form и Order N1234
    - Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) - между Order Detail Form и Order N1234
    - Save the order (Сохранить заказ) - между Order Detail Form и Order N1234

Теперь нужно поместить на диаграмму дополнительные элементы, а также рассмотреть ответственности объектов.

Добавление на диаграмму дополнительных объектов.

1. Нажмите кнопку Object панели инструментов.
2. Щелкните мышью где-нибудь на диаграмме, чтобы поместить туда новый объект.
3. Введите имя объекта - Order Manager (Управляющий заказами).
4. На, панели инструментов нажмите кнопку Object.
5. Поместите на диаграмму еще один объект.
6. Введите его имя - Transaction Manager (Управляющий транзакциями).

Назначение ответственностей объектам

1. Выделите сообщение 5: Create new, blank order (Создать пустой заказ). Выделяйте слова, а не стрелку.
2. Нажав комбинацию клавиш CTRL+D, удалите это сообщение.
3. Повторите шаги 1 и 2 для удаления сообщений 6 и 7:
    - Set the order number, customer, order items
    - Save the order
4. Выделите связь между объектами Order Detail Form и Order N1234.
5. Нажав комбинацию клавиш CTRL+D. удалите эту связь.
6. На панели инструментов нажмите кнопку Object Link (Связь объекта).
7. Нарисуйте связь между Order Detail Form и Order Manager.
8. На панели инструментов нажмите кнопку Object Link (Связь объекта).
9. Нарисуйте связь между Order Manager и Order N1234.
10. На панели инструментов нажмите кнопку Object Link (Связь объекта).
11. Нарисуйте связь между Order N1234 и Transaction Manager.
12. На панели инструментов нажмите кнопку Object Link (Связь объекта).
13. Нарисуйте связь между Order Manager и Transaction Manager.
14. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
15. Щелкните мышью на связи между объектами Order Detail Form и Order Manager, чтобы ввести новое сообщение.
16. Назовите это сообщение Save the order (Сохранить заказ).
17. Повторите шаги 14 - 16, добавив сообщения с шестого по девятое и назвав их:
    - Create new, blank order (Создать новый заказ) - между Order Manager и Order N1234
    - Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) - между Order Manager и Order N1234
    - Save the order (Сохранить заказ) — между Order Manager и Transaction Manager
    - Collect order information (Информация о заказе) — между Transaction Manager и Order N1'234
    - На панели инструментов нажмите кнопку Link to Self (Связь с собой).

- Щелкнув на объекте Transaction Manager, добавьте к нему рефлексивное сообщение.
- На панели инструментов нажмите кнопку Link Message (Сообщение связи).
- Щелкните мышью на рефлексивной связи Transaction Manager, чтобы ввести туда сообщение.
- Назовите новое сообщение Save the order information to the database (Сохранить информацию о заказе в базе данных).

Соотнесение объектов с классами (если классы были созданы при разработке описанной выше диаграммы Последовательности)

1. Найдите в браузере класс Order Options.
2. Перетащите его на объект Order Options Form (Выбор варианта заказа) на диаграмме.
3. Повторите шаги 1 и 2, соотнеся остальные объекты и соответствующие им классы:
   - Класс OrderDetail соотнесите с объектом Order Detail Form
   - Класс OrderMgr — с объектом Order Manager
   - Класс Order — с объектом Order N1234
   - Класс TransactionMgr — с объектом Transaction Manager

Соотнесение объектов с классами (если вы не создавали описанную выше диаграмму Последовательности)

1. Щелкните правой кнопкой мыши на объекте Order Options Form.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт (Создать). Появится окно спецификации классов.
4. В поле имени введите OrderOptions (Выбор заказа).
5. Щелкните на кнопке OK. Вы вернетесь в окно спецификации объекта.
6. В списке классов выберите класс OrderOptions.
7. Щелкните на кнопке OK, чтобы вернуться к диаграмме. Теперь объект называется Order Options Form : OrderOptions.
8. Для соотнесения остальных объектов с классами повторите шаги с 1 по 7:
   - Класс OrderDetail соотнесите с объектом Order Detail Form
   - Класс OrderMgr — с объектом Order Manager
   - Класс Order — с объектом Order N1234
   - Класс TransactionMgr -- с объектом Transaction Manager .

Соотнесение сообщений с операциями (если операции были созданы при разработке описанной выше диаграммы Последовательности)

1. Щелкните правой кнопкой мыши на сообщении 1: Create new order (Создать новый заказ).
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке имен укажите имя операции — Create() (Создать).
4. Нажмите на кнопку ОК.
5. Повторите шаги 1 — 4 для соотнесения с операциями остальных сообщений:
    - Сообщение 2: Open form (Открыть форму) соотнесите с операцией Open()
    - Сообщение 3: Enter order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией SubmitlnfoO
    - Сообщение 4: Save the order (Сохранить заказ) — с операцией Save()
    - Сообщение 5: Save the order (Сохранить заказ) — с операцией Save0rder()
    - Сообщение 6: Create new, blank order (Создать пустой заказ) - с операцией Create()
    - Сообщение 7: Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией Setlnfo()
    - Сообщение 8: Save the order (Сохранить заказ) — с операцией SaveOrder()
    - Сообщение 9: Collect order information (Информация о заказе) - с операцией Getlnfo()
    - Сообщение 10: Save the order information to the database (Сохранить информацию о заказе в базе данных) — с операцией Commit()

Соотнесение сообщений с операциями (если вы не создавали описанную выше диаграмму Последовательности)

1. Щелкните правой кнопкой мыши на сообщении 1: Create new order (Создать новый заказ).
2. В открывшемся меню выберите пункт (создать операцию). Появится окно спецификации операции.
3. В поле имени введите имя операции — Create() (Создать).
4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться к диаграмме.
5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
6. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
7. В раскрывающемся списке Name <Имя> укажите имя новой операции.

8. Нажмите на кнопку ОК.
9. Повторите шаги 1 — 8, чтобы создать новые операции и соотнести с ними остальные сообщения:
    - Сообщение 2: Open form (Открыть форму) соотнесите с операцией Open()
    - Сообщение 3: Enter order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией Submitlnfo()
    - Сообщение 4: Save the order (Сохранить заказ) — с операцией Save()
    - Сообщение 5: Save the order (Сохранить заказ) — с операцией SaveOrderQ
    - Сообщение 6: Create new, blank order (Создать пустой заказ) - с операцией CreateO
    - Сообщение 7: Set the order number, customer, order items (Ввести номер заказа, заказчика и число заказываемых предметов) — с операцией SettnfoQ
    - Сообщение 8: Save the order (Сохранить заказ) — с операцией SaveOrderQ
    - Сообщение 9: Collect order information (Информация о заказе) - с операцией GetinfoO
    - Сообщение 10: Save the order information to the database (Сохранить информацию о заказе в базе данных) — с операцией Commit()

4.3. Классы и пакеты.

Необходимо сгруппировать в пакеты классы, созданные при выполнении предыдущего упражнения. Затем нужно будет построить несколько диаграмм Классов и показать на них классы и пакеты системы.

Объедините обнаруженные нами классы в пакеты. Создайте диаграмму Классов для отображения пакетов, диаграммы Классов для представления классов в каждом пакете и диаграмму Классов для представления всех классов варианта использования “Ввести новый заказ”.

Этапы выполнения упражнения

Настройка

1. В меню модели выберите пункт Tools > Options (Инструменты > Параметры).
2. Перейдите на вкладку Diagram (Диаграмма).
3. Убедитесь, что установлен флажок Show stereotypes (Показать стереотипы).

4. Убедитесь, что установлены флажки Show All Attributes (Показать все атрибуты) и Show All Operations (Показать все операции).
5. Убедитесь, что сброшены флажки Suppress Attributes (Подавить вывод атрибутов) и Suppress Operations (Подавить вывод операций).

Создание пакетов

1. Щелкните правой кнопкой мыши на Логическом представлении браузера.
2. В открывшемся меню выберите пункт New > Package (Создать > Пакет).
3. Назовите новый пакет Entities (Сущности).
4. Повторив шаги 1—3, создайте пакеты Boundaries (Границы) и Control (Управление).

Создание Главной диаграммы Классов

1. Дважды щелкнув мышью на Главной диаграмме Классов, находящейся под Логическим представлением браузера, откройте ее.
2. Перетащите пакет Entities из браузера на диаграмму.
3. Перетащите пакеты Boundaries и Control из браузера на диаграмму.

Создание диаграммы Классов для сценария "Ввести новый заказ с отображением всех классов*

1. Щелкните правой кнопкой мыши на Логическом представлении брауэера.
2. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмма Классов).
3. Назовите новую диаграмму Классов Add New Order (Ввод нового заказа).
4. Дважды щелкнув мышью на этой диаграмме в браузере; откройте ее.
5. Перетащите из браузера все классы (OrderOptions, OrderDetail, Order, OrderMgr и TransactionMgr).

Добавление стереотипов к классам

1. Щелкните правой кнопкой мыши на классе OrderOptions диаграммы.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В поле стереотипа введите слово Boundary.
4. Нажмите на кнопку ОК.
5. Щелкните правой кнопкой мыши на классе OrderDetail диаграммы.
6. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).

7. В раскрывающемся списке поля стереотипов будет указан стереотип Boundary. Выделите его
8. Нажмите на кнопку ОК.
9. Повторив шаги 1—4, свяжите классы OrderMgr и TransactionMgr со стереотипом Control, а класс Order — со стереотипом Entity

Объединение классов в пакеты

1. В браузере перетащите класс OrderOpttons-на пакет Boundaries. ^
2. Перетащите класс OrderDetail на пакет Boundaries.
3. Перетащите классы OrderMgr и TransactionMgr на пакет Control.
4. Перетащите класс Order на пакет Entities.

Добавление диаграмм Классов к каждому пакету

1. В браузере Щелкните правой кнопкой мыши на пакете Boundaries.
2. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмма Классов).
3. Введите имя новой диаграммы — Main (Главная).
4. Дважды щелкнув мышью на этой диаграмме, откройте ее. Перетащите на нее из браузера классы OrderOpttons и OrderDetail..
   Закройте диаграмму.
5. В браузере щелкните правой кнопкой мыши на пакете Entities.
6. В открывшемся меню выберите пункт New >- Class Diagram (Создать >- Диаграмма Классов).
7. Введите имя новой диаграммы — Main (Главная).
8. Дважды щелкнув мышью на этой диаграмме, откройте ее.
9. Перетащите на нее из браузера класс Order.
10. Закройте диаграмму.
11. В браузере щелкните правой кнопкой мыши на пакете Control.
12. В открывшемся меню выберите пункт New > Class Diagram (Создать > Диаграмма Классов).
13. Введите имя новой диаграммы — Main (Главная).
14. Дважды щелкнув мышью на этой диаграмме, откройте ее
15. Перетащите на нее из браузера классы OrderMgr и TransactionMgr
16. Закройте диаграмму.

4.4. Атрибуты и операции.

В созданную диаграмму Классов для своего индивидуального задания добавьте атрибуты и операции.

Пример выполнения.

Добавим атрибуты и операции к классам диаграммы Классов "Ввод нового заказа". При этом используем специфические для языка особенности.

Установим параметры так, чтобы показывать все атрибуты, все операции и их сигнатуры. Применим нотацию UML.

Настройка

1. В меню модели выберите пункт Tools -> Options (Инструменты -> Параметры).
2. Перейдите на вкладку Diagram.
3. Убедитесь, что флажок Show visibility (Показать видимость) установлен.
4. Убедитесь, что флажок Show stereotypes (Показать стереотипы) установлен.
5. Убедитесь, что флажок Show operation signatures (Показать сигнатуры операций) установлен.
6. Убедитесь, что флажки Show all attributes (Показать все атрибуты) и Show all operations (Показать все операции) установлены.
7. Убедитесь, что флажки Suppress attributes (Подавить атрибуты) и Suppress operations {Подавить операции) сброшены.
8. Перейдите на вкладку Notation (Нотация).
9. Убедитесь, что флажок Visibility as icons (Отображать пиктограммы) сброшен.

Добавление нового класса

1. Найдите в браузере диаграмму Классов варианта использования "Ввести новый заказ".
2. Дважды щелкнув мышью на диаграмме, откройте ее.
3. Нажмите кнопку Class панели инструментов.
4. Щелкните мышью внутри диаграммы? чтобы поместить туда новый класс.
5. Назовите его OrderItem.
6. Назначьте этому классу стереотип Entity.
7. В браузере перетащите класс в пакет Entities.

Добавление атрибутов

1. Щелкните правой кнопкой мыши на классе Order.
2. В открывшемся меню выберите пункт New Attribute (Создать атрибут).
3. Введите новый атрибут:OrderNumber : Integer
4. Нажмите клавишу Enter.
5. Введите следующий атрибут:CustomerName : String.
6. Повторив шаги 4 и 5, добавьте атрибуты:OrderDate : Date OrderFUIDate : Date
7. Щелкните правой кнопкой мыши на классе Orderltem.
8. В открывшемся меню выберите пункт New Attribute (Создать атрибут).
9. Введите новый атрибут:ItemID : Integer.

10. Нажмите клавишу Enter.
11. Введите следующий атрибут:HemDescription : String.

Добавление операций к классу Orderltem

1. Щелкните правой кнопкой мыши на классе Orderltem.
2. В открывшемся меню выберите пункт New Operation (Создать операцию).
3. Введите новую операцию:Create.
4. Нажмите клавишу Enter.
5. Введите следующую операцию:Setlnfo
6. Нажмите клавишу Enter.
7. Введите операцию:Getlnfo

Подробное описание операций с помощью диаграммы Классов

1. Щелкнув мышью на классе Order, выделите его.
2. Щелкните на этом классе еще раз, чтобы переместить курсор внутрь.
3. Отредактируйте операцию Create(), чтобы она выглядела следующим образом:Create() : Boolean
4. Отредактируйте операцию Setlnfo():Setlnfo(OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) : Boolean
5. Отредактируйте операцию Getlnfo(): Getlnfo() : String

Подробное описание операций с помощью браузера

1. Найдите в браузере класс Orderltem.
2. Раскройте этот класс, щелкнув на значке "+" рядом с ним В браузере появятся атрибуты и операции класса
3. Дважды щелкнув мышью на операции Oe11п1o(),откройте окно ее спецификации.
4. В раскрывающемся списке Return class (Возвращаемый класс) укажите String.
5. Щелкнув мышью на кнопке ОК, закройте окно спецификации операции
6. Дважды щелкните в браузере на операции Setlnfo() класса Orderltem, чтобы открыть окно ее спецификации
7. В раскрывающемся списке Return class укажите Boolean.
8. Перейдите на вкладку Detail (Подробно).
9. Щелкните правой кнопкой мыши в области аргументов, чтобы добавить туда новый параметр
10. В открывшемся меню выберите пункт Insert (Вставить). Rose добавит аргумент под названием argname.
11. Щелкнув один раз на этом слове, выделите его и измените имя аргумента на ID.
12. Щелкните на колонке Type (Тип). В раскрывающемся списке типов выберите Integer.

13. Щелкните на колонке Default (По умолчанию), чтобы добавить значение аргумента по умолчанию. Введите число 0.
14. Нажав на кнопку ОК, закройте окно спецификации операции.
15. Дважды щелкните на операции Create() класса Orderltem, чтобы открыть окно ее спецификации.
16. В раскрывающемся списке Return class укажите Boolean.
17. Нажав на кнопку ОК, закройте окно спецификации операции.

Подробное описание операций

1. Используя браузер или диаграмму Классов, введите следующие сигнатуры операций класса OrderDetail: Open() : Boolean Submitlnfo() : Boolean Save() : Boolean
2. Используя браузер или диаграмму Классов, введите сигнатуру операций класса OrderOptions: Create() : Boolean
3. Используя браузер или диаграмму Классов, введите сигнатуру операций класса OrderMgr: SaveOrder(OrderlD : Integer): Boolean
4. Используя браузер или диаграмму Классов, введите сигнатуры операций класса TransactionMgr: SaveOrder(OrderlD : Integer) : Boolean Commit() : Integer

4.5. Связи

Пример выполнения.

Определяются связи между классами, участвующими в варианте использования "Ввести новый заказ".

Постановка задачи

После добавления к классам атрибутов и операций Карен была готова к генерации кода. Но сначала она должна была изучить связи между классами.

Чтобы найти связи, Карен просмотрела диаграммы Последовательности. Все взаимодействующие там классы нуждались в определении соответствующих связей на диаграммах Классов. После обнаружения связей Карен добавила их в модель.

Добавление связей

Добавим связи к классам, принимающим участие в варианте использования "Ввести новый заказ".

Этапы выполнения упражнения

Настройка

1. Найдите в браузере диаграмму Классов "Ввод нового заказа".
2. Дважды щелкнув на диаграмме, откройте ее.
3. Проверьте, имеется ли в панели инструментов диаграммы кнопка Unidirectional Association (Однонаправленная ассоциация). Если ее нет, продолжите настройку, выполнив шаги 4 и 5. Если есть, приступайте к выполнению самого упражнения.
4. Щелкните правой кнопкой мыши на панели инструментов диаграммы и в открывшемся меню выберите пункт Customize (Настроить).
5. Добавьте на панель кнопку Creates A Unidirectional Association (Создать однонаправленную ассоциацию).

Добавление ассоциаций

1. Нажмите кнопку Unidirectional Association панели инструментов.
2. Проведите ассоциацию от класса OrderOptions к классу OrderDetail.
3. Повторите шаги 1 и 2, создав ассоциации:
    - От класса OrderDetail к классу OrderMgr
    - От класса OrderMgr к классу Order
    - От класса OrderMgr к классу TransactionMgr
    - От класса TransactionMgr к классу Order
    - От класса TransactionMgr к классу Orderltem
    - От класса Order к классу Orderitem
4. Щелкните правой кнопкой мыши на однонаправленной ассоциации между классами OrderOptions и OrderDetail со стороны класса OrderOptions.
5. В открывшемся меню выберите пункт Multiplicity > Zero or One (Множественность >- Нуль или один).
6. Щелкните правой кнопкой мыши на другом конце однонаправленной ассоциации.
7. В открывшемся меню выберите пункт Multiplicity > Zero or One (Множественность >- Нуль или один).

4.6. Поведение объектов

Пример выполнения.

Создание диаграммы

1. Найдите в браузере класс Order.
2. Щелкните на классе правой кнопкой мыши и в открывшемся меню укажите пункт Open State Diagram (Открыть диаграмму состояний).

Добавление начального и конечного состояний

1. Нажмите кнопку Start State (Начальное состояние) панели инструментов.

2. Поместите это состояние на диаграмму.
3. Нажмите кнопку End State (Конечное состояние) панели инструментов.
4. Поместите это состояние на диаграмму.

Добавление суперсостояния

1. Нажмите кнопку State (Состояние) панели инструментов.
2. Поместите это состояние на диаграмму.

Добавление оставшихся состояний

1. На панели инструментов нажмите кнопку State (Состояние).
2. Поместите состояние на диаграмму.
3. Назовите состояние Cancelled (Отменен).
4. На панели инструментов нажмите кнопку State (Состояние).
5. Поместите состояние на диаграмму.
6. Назовите состояние Filled (Выполнен).
7. На панели инструментов нажмите кнопку State (Состояние).
8. Поместите состояние на диаграмму внутрь суперсостояния.
9. Назовите состояние Initialization (Инициализация).
10. На панели инструментов нажмите кнопку State (Состояние).
11. Поместите состояние на диаграмму внутрь суперсостояния.
12. Назовите состояние Pending (выполнение заказа приостановлено).

Описание состояний

1. Дважды щелкните мышью на состоянии Initialization (Инициализация).
2. Перейдите на вкладку Detail (Подробно).
3. Щелкните правой кнопкой мыши в окне Actions (Действия).
4. В открывшемся меню выберите пункт Insert (Вставить).
5. Дважды щелкните мышью на новом действии.
6. Назовите его Store order date (Сохранить дату заказа).
7. Убедитесь, что в окне When (Когда) указан пункт On Entry (На входе).
8. Повторив шаги 3—7, добавьте следующие действия:
    - Collect customer info (Собрать клиентскую информацию), в окне When укажите Entry until Exit (Выполнять до завершения)
    - Add order items (Добавить к заказу новые позиции), укажите Entry until Exit (Выполнять до завершения)
9. Нажмите два раза на ОК, чтобы закрыть спецификацию.
10. Дважды щелкните мышью на состоянии Cancelled (Отменен).
11. Повторив шаги 2 — 7, добавьте действие:• Store cancellation data (Сохранить дату отмены), укажите On Exit (На выходе)
12. Нажмите два раза на ОК, чтобы закрыть спецификацию.
13. Дважды щелкните мышью на состоянии Filled (Выполнен).
14. Повторив шаги 2—7, добавьте действие:• Bill customer (Выписать счет), укажите Entry until Exit

15. Нажмите два раза на ОК, чтобы закрыть спецификацию.

Добавление переходов

1. Нажмите кнопку Transition (Переход) панели инструментов.
2. Щелкните мышью на начальном состоянии.
3. Проведите линию перехода к состоянию Initialization (Инициализация).
4. Повторив шаги с первого по третий, создайте следующие переходы:
    - От состояния Initialization (Инициализация) к состоянию Pending (Выполнение заказа приостановлено)
    - От состояния Pending (Выполнение заказа приостановлено) к состоянию Filled (Выполнен)
    - От суперсостояния к состоянию Cancelled (Отменен)
    - От состояния Cancelled (Отменен) к конечному состоянию
    - От состояния Filled (Выполнен) к конечному состоянию
5. На панели инструментов нажмите кнопку Transition to Self (Переход к себе).
6. Щелкните мышью на состоянии Pending (Выполнение заказа приостановлено).

Описание переходов

1. Дважды щелкнув мышью на переходе от состояния Initialization (Инициализация) к состоянию Pending (Выполнение заказа приостановлено), откройте окно спецификации перехода.
2. В доле Event (Событие) введите фразу Finalize order (Выполнить заказ).
3. Щелкнув на кнопке ОК, закройте окно спецификации.
4. Повторив шаги с первого по третий, добавьте событие Cancel Order (Отменить заказ) к переходу между суперсостоянием и состоянием Cancelled (Отменен).
5. Дважды щелкнув мышью на переходе от состояния Pending (Выполнение заказа приостановлено) к состоянию Filled (Выполнен), откройте окно его спецификации.
6. В поле Event (Событие) введите фразу Add order item (Добавить к заказу новую позицию).
7. Перейдите на вкладку Detail (Подробно).
8. В поле Condition (Условие) введите No unfilled items remaining (Не осталось незаполненных позиций).
9. Щелкнув на кнопке ОК, закройте окно спецификации.
10. Дважды щелкните мышью на рефлексивном переходе (Transition to Self) состояния Pending (Выполнение заказа приостановлено).
11. В поле Event (Событие) введите фразу Add order item (Добавить к заказу новую позицию).
12. Перейдите на вкладку Detail (Подробно).

13. В поле Condition (Условие) введите Unfilled items remaining (Остаются незаполненные позиции).
14. Щелкнув на кнопке ОК, закройте окно спецификации.

5. ПЕРЕЧЕНЬ УЧЕБНИКОВ И УЧЕБНЫХ ПОСОБИЙ

1. Леоненков А.В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose./ БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006.
2. Грекул В.И., Денищенко Г.Н., Коровкина Н.Л. Проектирование информационных систем. /Интернет-университет информационных технологий - ИНТУИТ.ру, 2005.

3. Амриш К.И., Ахмед Х.З. Разработка корпоративных Java-приложений с использованием J2EE и UML. /Пер. с англ. -М.: "Вильямс", 2002. - 272 с.

4. Боггс У., Боггс М. UML и Rational Rose./ - М.: "ЛОРИ", 2000. - 582 с.

5. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. - М.: "Бином", СПб: "Невский диалект", 1999. - 560 с.

7. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. /- М.: ДМК, 2000. - 432 с.
8. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. /Паттерны проектирования. - СПб: "Питер", 2001.- 368 с.
9. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений. - М.: "ДМК Пресс", 2002. - 704 с.
10. Грехем И. Объектно-ориентированные методы. Принципы и практика.- М.: "Вильямс", 2004. - 880 с.
11. Коналлен Дж. Разработка Web-приложений с использованием UML. /Пер. с англ. - М.: "Вильямс", 2001. - 288 с.
12. Кьоу Дж., Джеанини М. Объектно-ориентированное программирование. Учебный курс. - СПб: "Питер", 2005.- 238 с.
13. Ларман К. Применение UML и шаблонов проектирования - М.: "Вильямс", 2001. - 496 с.
14. Ларман К. Применение UML и шаблонов проектирования. 2-е издание. - М.: "Вильямс", 2002. - 624 с.
15. Леоненков А.В. Самоучитель UML. - СПб.: "БХВ - Петербург", 2001. - 304 с.
16. Леоненков А.В.Самоучитель UML. 2-е издание. - СПб.: "БХВ-Петербург", 2004. - 432 с.

17. Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. - М.: "Вильямс", 2002. - 448 с.
18. Нейбург Э.Дж., Максимчук Р.А. Проектирование баз данных с помощью UML. - М.: "Вильямс", 2002. - 288 с.
19. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник - СПб: "Питер", 2001. - 656 с.
20. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов. - М.: "ДМК Пресс", 2002. - 160 с.
21. Санблэд С., Санблэд С. Разработка масштабируемых приложений для Microsoft Windows. Мастер-класс. - М.: ИТД "Русская редакция", 2002. - 416 с.
22. Фаулер М., Скотт К. UML. Основы. - СПб: "Символ-Плюс", 2002. - 192 с.
23. Шаллоуей А., Тротт Дж.Р. Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию. - М.: "Вильямс", 2002. - 288 с.
24. Шмуллер Д. Освой самостоятельно UML за 24 часа. - М.: "Вильямс", 2002. - 352 с.
25. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. - СПб: "Питер", 2002. - 496 с.
26. Grand M. Patterns in Java. A Catalog of Reusable Design Patterns Illustrated with UML. - Wiley & Sons, 1998. Vol. 1, - 468 p. Vol. 2, - 356 p.
27. Starr L. Executable UML. How to build class models. - Prentice Hall PTR, 2002. - 418 p.
28. Wampler B.E. The Essence Object-Oriented Programming with Java and UML. - Addison-Wesley, 2002, - 290 p.

6. КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ

Лекция 1. Общее представление об информационной системе.

В зависимости от конкретной области применения информационные системы могут очень сильно различаться по своим функциям, архитектуре, реализации. Можно выделить два свойства, которые являются общими для всех информационных систем. Во-первых, любая информационная система предназначена для сбора, хранения и обработки информации. Поэтому в основе любой информационной системы лежит среда хранения и доступа к данным. Среда должна обеспечивать уровень надежности хранения и эффективность доступа, которые соответствуют области применения информационной системы. В вычислительных программных системах наличие такой среды не является обязательным. Основным требованием к программе, выполняющей численные расчеты, является ее быстродействие. Нужно, чтобы программа произвела достаточно точные результаты за установленное время.

При решении серьезных вычислительных задач даже на суперкомпьютерах это время может измеряться неделями, а иногда и месяцами. Поэтому программисты-вычислители всегда очень скептически относятся к хранению данных во внешней памяти, предпочитая так организовывать программу, чтобы в течение как можно более долгого времени обрабатываемые данные помещались в основной памяти компьютера. Внешняя память обычно используется для периодического сохранения промежуточных результатов вычислений, чтобы в случае сбоя компьютера можно было продолжить работу программы от сохраненной контрольной точки. Во-вторых, информационные системы ориентируются на конечного пользователя, например, банковского клерка. Такие пользователи могут быть очень далеки от мира компьютеров. Поэтому информационная система обязана обладать простым, удобным, легко осваиваемым интерфейсом, который должен предоставить конечному пользователю все необходимые для его работы функции.

Лекция 2. Задачи информационных систем.

Конкретные задачи, которые должны решаться информационной системой, зависят от той прикладной области, для которой предназначена система. Области применения информационных приложений разнообразны: банковское дело, страхование, медицина, транспорт, образование и т.д. Трудно найти область деловой активности, в которой сегодня можно было обойтись без использования информационных систем. С другой стороны, очевидно, что, например, конкретные задачи, решаемые банковскими информационными системами, отличаются от задач, решение которых требуется от медицинских информационных систем. Но можно выделить некоторое количество задач, не зависящих от специфики прикладной области. Такие задачи связаны с общими чертами информационных систем, и прежде всего, это связано с информацией, которая долго накапливается и утрата которой невосполнима.

Уровень надежности и продолжительность хранения информации во многом определяются конкретными требованиями корпорации к информационной системе. Например, можно представить себе малую торговую компанию с быстрым оборотом, в информационной складской системе которой достаточно поддерживать информацию о товарах, имеющихся на складе, и об еще неудовлетворенных заявках от потребителей. Однако неизвестно, не потребуется ли впоследствии полная история работы склада с момента основания компании.

Следующей задачей, которую должно выполнять большинство информационных систем, - это хранение данных, обладающих разными структурами. Трудно представить себе более или менее развитую информационную систему, которая работает с одним однородным файлом данных. Более того, разумным требованием к информационной системе является то, чтобы она могла развиваться. Могут появиться новые функции, для выполнения которых требуются дополнительные данные с новой структурой. При этом вся накоп-

ленная ранее информация должна остаться сохранной. Теоретически можно решить эту задачу путем использования нескольких файлов внешней памяти, каждый из которых хранит данные с фиксированной структурой. В зависимости от способа организации используемой системы управления файлами эта структура может быть структурой записи файла (как, например, в файловых системах DEC VMS) или поддерживаться отдельной библиотечной функцией, написанной специально для данной информационной системы. Ко второму способу решения проблемы пришлось бы прибегать при работе в среде ОС UNIX.

При использовании такого подхода информационная система перегружается деталями организации хранилища данных. При выполнении функций уровня пользовательского интерфейса информационной системе самой приходится выполнять выборку информации из файлов по заданному критерию.

Лекция 3. Понятия классической транзакции.

До сих пор были отмечены те функции информационной системы, которые требуют выборки данных из внешнего хранилища, например, производят отчеты. Но, возникают следующие вопросы, откуда берутся данные во внешнем хранилище? Каким образом поддерживается соответствие хранимой информации состоянию предметной области? Конечно, для этого должны существовать дополнительные функции информационной системы, которые обеспечивают ввод, обновление и удаление данных. Поддержка этих функций существенно повышает уровень требований к СУБД.

Наличие групповых или корпоративных информационных систем, предполагает возможность работы с системой с нескольких рабочих мест. Некоторые из конечных пользователей изменяют содержимое базы данных (вводят, обновляют, удаляют данные). Другие выполняют операции, связанные с выборкой из базы данных. Третьи делают и то, и другое. Такая коллективная работа должна производиться согласованно и желательно, чтобы согласованность действий обеспечивалась автоматически. Под согласованностью действий понимается следующее, оператор, формирующий отчеты, не сможет воспользоваться данными, которые начал, но еще не закончил формировать другой оператор. Оператор, формирующий данные, не сможет выполнить операцию над данными, которыми пользуется другой оператор, начавший, но не закончивший формировать отчет. Оператор, желающий обновить или удалить данные, не сможет выполнить операцию до тех пор, пока не закончится аналогичная операция над теми же данными, которую ранее начал, но еще не закончил другой оператор. При поддержке согласованности действий все результаты, получаемые от информационной системы, будут соответствовать согласованному состоянию базы данных, т.е. будут достоверны и непротиворечивы.

Подобные рассуждения вызвали появления понятия классической транзакции. Под целостным состоянием базы данных информационной системы такое ее состояние, которое соответствует требованиям модели прикладной

области, на основе которой проектировалась информационная система. Классической транзакцией называется последовательность операций изменения базы данных и/или выборки из базы данных, воспринимаемая СУБД как атомарное действие. Это означает, что при успешном завершении транзакции СУБД гарантирует наличие в базе данных результатов всех операций изменения, произведенных при выполнении транзакции. Условием успешного завершения транзакции является то, что база данных находится в целостном состоянии. Если это условие не выполняется, то СУБД производит полный откат транзакции, ликвидируя в базе данных результаты всех операций изменения, произведенных при выполнении транзакции. Отсюда видно, что база данных будет находиться в целостном состоянии при начале любой транзакции и останется в целостном состоянии после успешного завершения любой транзакции.

Лекция 4. Проблемы построения ИС.

Первой проблемой, при построении ИС, является проблема проектирования. Нельзя начинать техническую разработку, не имея тщательно проработанного проекта. Если начинать с решения наиболее очевидных задач, не обращая внимания на потенциально существующие, то такая система будет непрерывно находиться в стадии разработки и переделки.

Первой стадией проектирования должен быть анализ требований корпорации. Для этого на основе экспертных запросов необходимо выявить все актуальные и потенциальные потребности корпорации, которые должны удовлетворяться проектируемой информационной системой, понять какие потоки данных существуют внутри корпорации, оценить объемы информации, которые должны поддерживаться и обрабатываться информационной системой. Эта стадия, как правило, носит неформальный характер, хотя, конечно, очень важно сохранить полученную информацию, поскольку она должна входить в документацию системы. Существуют CASE-средства верхнего уровня, которые помещают полученные данные в общий репозиторий проекта и позволяют использовать их на следующих стадиях проектирования.

Следующая стадия проектирования - выработка концептуальной схемы базы данных, которая будет лежать в основе информационной системы. Сначала придется выбрать систему нотаций, в которой будет представляться концептуальная схема, т.е. выбрать концептуальную модель. Таких нотаций существует много, и хотя практически все они диаграммные, в духе ER-модели, они отличаются одна от другой. Даже в случае использования некоторых CASE-средств, все равно предлагается на выбор несколько нотаций. Выбор нотации - дело вкуса, по возможностям они почти эквивалентны, это ответственный выбор. Концептуальное представление базы данных должно сохраняться как часть документации информационной системы на все время ее существования и будет использоваться при ее сопровождении и развитии.

С большой вероятностью в основе информационной системы будет лежать реляционная база данных. Несмотря на очевидную привлекательность объектно-ориентированных пакетов ObjectStore, Objectivity, O2, Jasmin и т.д., и объектно-реляционных пакетов Illustra, UniSQL СУБД, в ближайшие годы придется работать с хорошо отлаженными, развитыми и сопровождаемыми системами, поддерживающими стандарт SQL-92 (например, Oracle, Informix, CA-OpenIngres, Sybase, DB2). Должно пройти время, чтобы эти системы устоялись, обрели необходимую надежность, стали бы опираться на какие-либо стандарты и т.д.

На следующей стадии проектирования понадобится, на основе имеющейся концептуальной схемы, произвести набор определений схемы реляционной базы данных в терминах языка SQL.

Лекция 5. Архитектура информационной системы.

После того, как выработана общая реляционная схема базы данных, необходимо определиться с архитектурой системы. Очень важно решить, какой будет база данных - централизованной или распределенной, другими словами будет ли использоваться только один сервер баз данных или их будет несколько. Если принимается решение о распределенном характере базы данных, то необходимо произвести соответствующую декомпозицию набора определений схемы базы данных.

Принятие решения об архитектуре системы возможно и до выработки общей реляционной схемы базы данных. Тогда декомпозиция производится на уровне концептуальной схемы, а затем для каждой отдельной части концептуальной схемы создается реляционная схема в терминах языка SQL.

Наиболее простым случаем декомпозиции является тот, когда образующиеся разделы базы данных логически автономны. В терминах концептуальной схемы это означает, что между разделенными сущностями отсутствуют прямые или транзитивные, через другие сущности, связи. В терминах реляционной схемы: ни в одном разделе не присутствует таблица, ссылающаяся на таблицу, которая располагается в другом разделе (рис. 1.3 и 1.4). Если требование логической автономности компонентов распределенной базы данных выполнено, то дальнейшее проектирование можно производить для каждого компонента независимо.

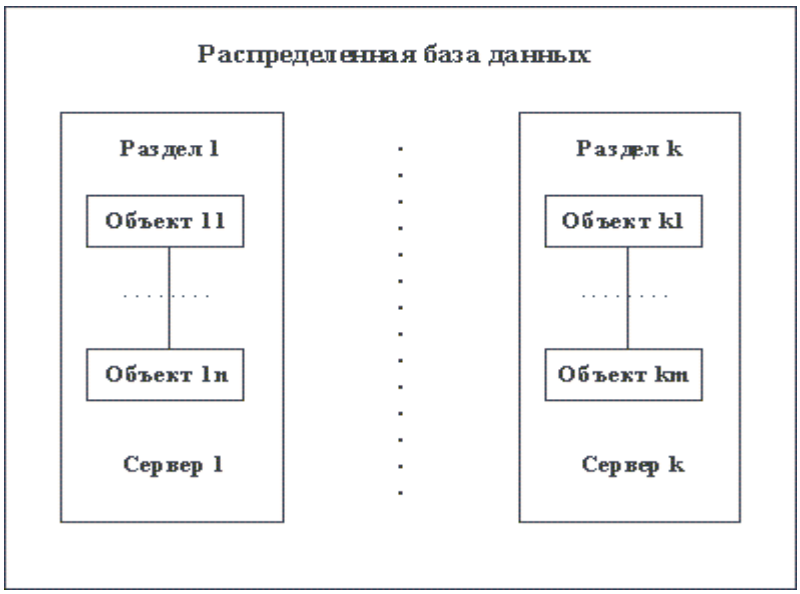


Рис. 1.3. Распределенная база данных с логически автономными разделами

В чем состоит проблема распределенных баз данных с логически неавтономными разделами? В том, что любая связь, отраженная в схеме базы данных, между сущностями в концептуальной модели или между таблицами в реляционной модели, соответствует ограничению целостности, которое впоследствии должно сохраняться в базе данных и поддерживаться СУБД.

Лекция 6. Определения общих ограничений целостности, триггеров и хранимых процедур.

Следующая стадия проектирования состоит в дополнении реляционных схем разделов распределенной базы данных определениями общих ограничений целостности, триггеров и хранимых процедур. На каждом из этих компонентов схемы следует остановиться отдельно. Начнем с общих ограничений целостности.

Язык SQL-92 позволяет определить ограничения целостности, относящиеся к общему состоянию базы данных и включающие ссылки на произвольное число таблиц. Семантика таких ограничений целостности может быть существенно шире, чем ограничения, задаваемые связями *1* к *n* и даже *n* к *m*. Поэтому, часто ограничения общего вида не выводятся автоматически из концептуальной схемы базы данных, и их приходится добавлять к реляционной схеме вручную. Для того чтобы понять, какие ограничения общего вида должны быть включены в реляционные схемы разделов, приходится возвращаться к документу, содержащему анализ требований корпорации. Задача проектировщика состоит в том, чтобы, с одной стороны, выявить все необходимые ограничения целостности и, с другой стороны, не перегрузить базу данных необязательными ограничениями, любое дополнительное ограничение целостности вызывает дополнительные проверки на стороне сервера при выполнении операций изменения базы данных; проверки для ограничений общего вида могут быть весьма громоздкими. Если предполагается использо-

вание распределенной базы данных, то придется учитывать возможности сервера по части ссылок на объекты "чужих" разделов. Это может повлиять на выбор ограничений целостности и/или повлечь создание новой декомпозиции общей базы данных на разделы.

Триггер - это, фактически, хранимая процедура без параметров, содержащая оператор или операторы изменения базы данных и вызываемая сервером баз данных автоматически при совершении некоторого события, обычно под событием понимается выполнение определенного рода операторов изменения базы данных. При определении триггера указывается вид события, возбуждающего триггер, условие его срабатывания и набор операторов, которые должны быть выполнены при выполнении условия. Имеется несколько практических областей применения механизма триггеров. Триггеры позволяют поддерживать логическую целостность базы данных в тех случаях, когда пользовательская транзакция не может не нарушить эту целостность. Вторая область применения триггеров – автоматический мониторинг действий конечных пользователей.

Однако в распространенном на сегодня стандарте SQL-92 механизм триггеров не специфицирован. В СУБД, которые поддерживают этот механизм, соответствующие языковые средства и их семантика различаются. Например, в Oracle V.7 для определения триггеров можно использовать процедурное расширение языка SQL PL/SQL.

Лекция 7. Встраивание операторов языка SQL в программу.

В базе данных могут храниться хранимые процедуры. В стандарте SQL-92 вообще не встречается термин "хранимая процедура". В стандарте специфицированы два способа взаимодействия прикладной программы с сервером баз данных. Первый, наиболее часто используемый способ состоит во встраивании операторов языка SQL в программу, написанную на одном из традиционных языков программирования. В самом стандарте определены правила встраивания SQL в программы, которые написаны на языках Си, Паскаль, Фортран, Ада и т.д. Второй способ основан на специфицированном в стандарте "языке модулей SQL". С использованием этого языка можно определить модуль, содержащий несколько процедур, каждая из которых соответствует некоторому параметризованному оператору SQL. В прикладной программе содержатся не операторы SQL, а лишь вызовы процедур, с указанием фактических параметров, из модуля SQL, с которым эта прикладная программа связана, правила связывания в стандарте не определены. Однако стандарт не обязывает следовать каким-то конкретным правилам при реализации встроенного SQL или языка модулей.

Что касается встроенного SQL, то во всех реализациях прикладная программа, содержащая операторы SQL, подвергается предкомпиляции с помощью соответствующего программного продукта, который входит в состав программного обеспечения сервера баз данных. В результате предкомпиляции всегда формируется текст прикладной программы на используемом язы-

ке программирования, уже не содержащий напрямую текста на языке SQL. Каждое вхождение оператора SQL в исходный текст программы, заменяется на вызов некоторой процедуры или функции в синтаксисе включающего языка программирования. Основная разница между разными реализациями состоит в том, что из себя представляет эта процедура или функция. В большинстве систем, Oracle, Informix, Sybase, CA-OpenIngres, такая процедура представляет собой обращение к клиентской части СУБД с передачей в качестве параметра текста оператора на языке SQL. Вся обработка оператора, включая его грамматический разбор и выработку процедурного плана, производится сервером во время выполнения прикладной программы, при повторном выполнении оператора сервер использует уже подготовленный план. Однако имеется и другой подход, исторически используемый компанией IBM в ее серии продуктов DB2. В DB2 предкомпилятор, выбрав текст очередного оператора SQL, сам обращается к серверу с запросом на компиляцию оператора. Сервер выполняет грамматический разбор и строит процедурный план выполнения оператора, сохраняя этот план в базе данных и привязывая к соответствующей прикладной программе. В качестве ответа от сервера предкомпилятор получает идентификатор, указание которого серверу повлечет выполнение соответствующего плана, имя удаленной процедуры.

Лекция 8. Разработка интерфейса системы.

Параллельно с физическим проектированием базы данных информационной системы может проводиться проектирование и разработка интерфейса системы и ее обрабатывающей части. К этому моменту уже должно быть ясно, что Вы хотите от интерфейса и какие функции должна выполнять система. Основной проблемой этой стадии является выбор инструментальных средств, которые позволят достаточно быстро произвести достаточно эффективную реализацию. Нужно понять, что для организации важно, как можно быстрее ввести информационную систему в действие или добиться требуемого уровня эффективности.

В любом случае неразумно программировать интерфейсные функции вручную. Нужно использовать имеющиеся полуфабрикаты, но здесь имеется большая возможность выбора: базовые библиотеки используемой оконной системы, например, X Toolkit Intrinsics в оконной системе X, универсальные инструментальные средства построения графического пользовательского интерфейса более высокого уровня, например, Motif или Tcl/Tk, языки и системы программирования 4-го поколения, например, PowerBuilder, Jam и т.д. Выбор определяется разработчиком, а также общей ориентацией проекта. Например, если с самого начала проект был ориентирован на использование продуктов компании Oracle и ведется в интегрированной среде Designer/Developer 2000, то нежелательно покидать эту среду при разработке интерфейсов.

Что же касается обрабатывающих частей информационной системы, то все зависит от того, что они должны делать. Во многих информационных си-

стемах вся обработка данных состоит в преобразовании их форматов при вводе и выводе, формировании отчетов и т.д. Такие функции можно фактически не программировать, поскольку любой 4GL может сгенерировать их автоматически по соответствующему описанию. Но если требуется более сложная обработка данных, то в любом случае потребуется явное программирование, и тогда уже неважно, на каком языке это программирование будет вестись. Если Вы используете Delphi для разработки интерфейсов, то для программирования разумно использовать Object Pascal, прекрасный язык, но пока жестко привязанный к Intel-платформам. В других случаях можно применять процедурную часть 4GL, обычно все они похожи на Basic, или Си/Си++ , как правило, стыковка 4GL с этими языками поддерживается.

Описана одна из возможных схем проектирования и разработки информационной системы, весь процесс представлен в виде последовательно выполняемых стадий. Если проект основывается на некотором интегрированном CASE-средстве с единым репозитарием, то эти стадии, могут перемешиваться.

Лекция 9. Требования к техническим средствам, поддерживающим ИС.

Естественно, что требования к техническим средствам определяются требованиями к информационной системе в целом. Какие бы информационные возможности не требовались сотрудникам корпорации, окончательное решение всегда принимается ее руководством, которое корректирует требования к информационной системе и формирует окончательное представление об аппаратной среде. Имеются четыре возможных позиции руководства по поводу места информационной системы в корпорации: пессимистическая, пессимистически-оптимистическая, оптимистически-пессимистическая и оптимистическая. Руководитетель-пессимист рассуждает следующим образом. Корпорации нужно продержаться хотя бы какое-то время. Без информационной системы это невозможно. Нужно выбрать самое дешевое решение, которое может быть реализовано максимально быстро. Руководитель не думает, что будет с корпорацией через два года, вернее, поскольку он пессимист, то думает, что, скорее всего, через два года корпорация просто не будет существовать или у нее сменится руководитель.

При такой позиции наиболее подходящим является некоторое закрытое и законченное техническое решение. Например, это может быть полностью сбалансированная локальная сеть Novell с выделенным файл-сервером и фиксированным числом рабочих станций. Жесткость решения затем закрепляется соответствующим программным обеспечением информационной системы. Возможности расширения системы отсутствуют, реинжиниринг требует практически полной переделки системы. Пессимистически-оптимистический руководитель не ожидает краха корпорации или своего собственного увольнения. Но он не надеется на изменение статуса компании, например, на появление зарубежных филиалов. Возможно, корпорация будет несколько развиваться, возможно, появятся новые виды бизнеса, возможно, увеличится число

сотрудников. Но поскольку руководитель все-таки более пессимист, чем оптимист, то он не очень высоко оценивает шансы на развитие.

Такой позиции руководства больше всего подходит закрытое решение, обладающее ограниченными возможностями расширения. Например, это может быть локальная сеть Novell, в которой пропускная способность превосходит потребности имеющихся рабочих станций, а файл-сервер может быть оснащен дополнительными магнитными дисками. Если пессимизм руководителя окажется неоправданным, то корпорация встретится с потребностью сложного реинжиниринга. Пессимистически-оптимистический руководитель не ожидает краха корпорации или своего собственного увольнения. Но он не надеется на изменение статуса компании, например, на появление зарубежных филиалов. Возможно, корпорация будет несколько развиваться, возможно, появятся новые виды бизнеса, возможно, увеличится число сотрудников.

Лекция 10. Жизненный цикл ПО ИС.

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО). ЖЦ ПО - это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 [1] (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

- основные процессы ЖЦ ПО (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);
- организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение).

Разработка включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения

персонала и т.д. Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО, обучение персонала и т.п.

Лекция 11. Модели жизненного цикла ПО.

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует). Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

К настоящему времени наибольшее распространение получили следующие две основные модели ЖЦ:

- каскадная модель (70-85 г.г.);
- спиральная модель (86-90 г.г.).

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 2.1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Положительные стороны применения каскадного подхода заключаются в следующем [2]:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;

- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений.

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов.

Лекция 12. Методологии и технологии проектирования ИС.

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования определяется как совокупность трех составляющих:

- пошаговой процедуры, определяющей последовательность технологических операций проектирования;
- критериев и правил, используемых для оценки результатов выполнения технологических операций;
- нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

Технологические инструкции, составляющие основное содержание технологии, должны состоять из описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования, разработки и сопровождения ИС должна удовлетворять следующим общим требованиям:

- технология должна поддерживать полный ЖЦ ПО;
- технология должна обеспечивать гарантированное достижение целей разработки ИС с заданным качеством и в установленное время;
- технология должна обеспечивать возможность выполнения крупных проектов в виде подсистем (т.е. возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей). Опыт разработки крупных ИС показывает, что для повыше-

ния эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций;

- технология должна обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек). Это обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;

технология должна обеспечивать минимальное время получения работоспособной ИС. Речь идет не о сроках готовности всей ИС, а о сроках реализации отдельных подсистем. Реализация ИС в целом в короткие сроки может потребовать привлечения большого числа разработчиков.

Лекция 13. Сущность структурного подхода.

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции, разбиении, на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы "снизу-вверх" от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов. В качестве двух базовых принципов используются следующие:

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

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

- принцип абстрагирования - заключается в выделении существенных аспектов системы и отвлечения от несущественных;
- принцип формализации - заключается в необходимости строгого методического подхода к решению проблемы;

- принцип непротиворечивости - заключается в обоснованности и согласованности элементов;
- принцип структурирования данных - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых являются следующие:

- SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы;
- DFD (Data Flow Diagrams) диаграммы потоков данных;
- ERD (Entity-Relationship Diagrams) диаграммы "сущность-связь".

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами.

Лекция 14. Методология функционального моделирования SADT.

Методология SADT разработана Дугласом Россом и получила дальнейшее развитие в работе. На ее основе разработана, в частности, известная методология IDEF0 (Icam DEFinition), которая является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой по инициативе ВВС США.

Методология SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Основные элементы этой методологии основываются на следующих концепциях:

- графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описываются посредством интерфейсных дуг, выражающих "ограничения", которые в свою очередь определяют, когда и каким образом функции выполняются и управляются;
- строгость и точность.

Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика.

Правила SADT включают:

- ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков);
- связность диаграмм (номера блоков);

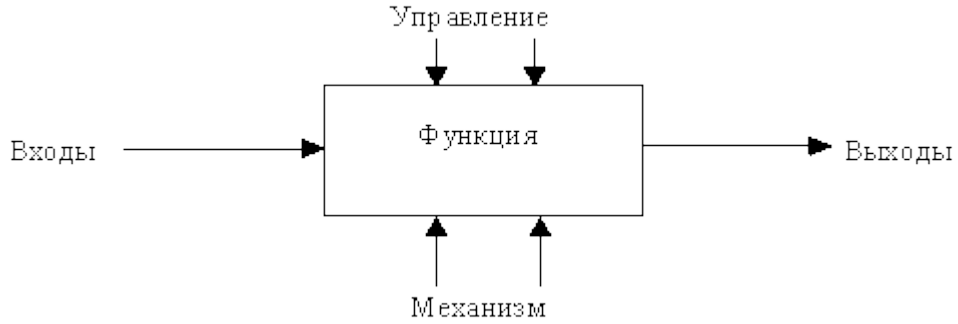
- уникальность меток и наименований (отсутствие повторяющихся имен);
- синтаксические правила для графики (блоков и дуг);
- разделение входов и управлений (правило определения роли данных);
- отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Методология SADT может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы, которая удовлетворяет этим требованиям и реализует эти функции. Для уже существующих систем SADT может быть использована для анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.

Лекция 15. Состав функциональной модели.

Результатом применения методологии SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы - главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показана с левой стороны блока, а результаты выхода показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу.

Одной из наиболее важных особенностей методологии SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.



Функциональный блок и интерфейсные дуги.

На рисунке, где приведены четыре диаграммы и их взаимосвязи, показана структура SADT-модели. Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует "внутреннее строение" блока на родительской диаграмме.

Построение SADT-модели начинается с представления всей системы в виде простейшей компоненты - одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представ-

ляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг - они также представляют полный набор внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления.

Лекция 16. Типы связей между функциями.

Одним из важных моментов при проектировании ИС с помощью методологии SADT является точная согласованность типов связей между функциями. Различают, по крайней мере, семь типов связывания (табл.).

Таблица

Тип связи	Относительная значимость
Случайная	0
Логическая	1
Временная	2
Процедурная	3
Коммуникационная	4
Последовательная	5
Функциональная	6

Ниже каждый тип связи кратко определен и проиллюстрирован с помощью типичного примера из SADT.

(0) Тип случайной связности: наименее желательный.

Случайная связность возникает, когда конкретная связь между функциями мала или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют малую связь друг с другом.

(1) Тип логической связности. Логическое связывание происходит тогда, когда данные и функции собираются вместе вследствие того, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

(2) Тип временной связности. Связанные по времени элементы возникают вследствие того, что они представляют функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

(3) Тип процедурной связности. Процедурно-связанные элементы появляются сгруппированными вместе вследствие того, что они выполняются в течение одной и той же части цикла или процесса.

(4) Тип коммуникационной связности. Диаграммы демонстрируют коммуникационные связи, когда блоки группируются вследствие того, что они используют одни и те же входные данные и/или производят одни и те же выходные данные (рис. 3.10).

(5) Тип последовательной связности. На диаграммах, имеющих последовательные связи, выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем на рассмотренных выше уровнях связок, поскольку моделируются причинно-следственные зависимости.

Лекция 17. Моделирование потоков данных.

В основе данной методологии (методологии Gane/Sarson [10]) лежит построение модели анализируемой ИС - проектируемой или реально существующей. В соответствии с методологией модель системы определяется как иерархия диаграмм потоков данных (ДПД или DFD), описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становятся элементарными и детализировать их далее невозможно.

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. Таким образом, основными компонентами диаграмм потоков данных являются:

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

Внешние сущности.

Внешняя сущность представляет собой материальный предмет 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 она находится за пределами границ анализируемой ИС. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой ИС, если это необходимо, или, наоборот, часть про-

цессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

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

При построении модели сложной ИС она может быть представлена в самом общем виде на так называемой контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем.

Лекция 18. Построение иерархии диаграмм потоков данных.

Первым шагом при построении иерархии ДПД является построение контекстных диаграмм. Обычно при проектировании относительно простых ИС 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.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы. Признаками сложности (в смысле контекста) 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 ИС 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 проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС.

Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры ИС на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в разработке 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 при помощи ДПД. Каждый процесс на ДПД, в свою очередь, может быть детализирован при помощи ДПД или миниспецификации.

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

Лекция 19. Моделирование данных.

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

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

Нотация ERD была впервые введена П. Ченом (Chen) и получила дальнейшее развитие в работах Баркера [11]. Метод Баркера будет излагаться на примере моделирования деятельности компании по торговле автомобилями. Ниже приведены выдержки из интервью, проведенного с персоналом компании.

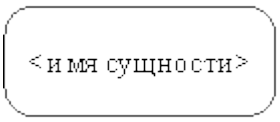
Главный менеджер: одна из основных обязанностей - содержание автомобильного имущества. Он должен знать, сколько заплачено за машины и каковы накладные расходы. Обладая этой информацией, он может установить нижнюю цену, за которую мог бы продать данный экземпляр. Кроме того, он несет ответственность за продавцов и ему нужно знать, кто что продает и сколько машин продал каждый из них.

Продавец: ему нужно знать, какую цену запрашивать и какова нижняя цена, за которую можно совершить сделку. Кроме того, ему нужна основная информация о машинах: год выпуска, марка, модель и т.п.

Администратор: его задача сводится к составлению контрактов, для чего нужна информация о покупателе, автомашине и продавце, поскольку именно контракты приносят продавцам вознаграждения за продажи.

Первый шаг моделирования - извлечение информации из интервью и выделение сущностей.

Сущность (Entity) - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.



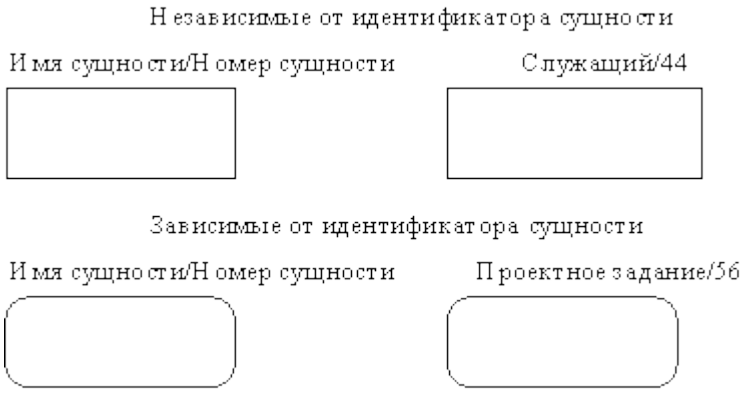
Графическое изображение сущности.

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

Лекция 20. Методология IDEF1.

Метод IDEF1, разработанный Т.Рэмей (T.Ramey), также основан на подходе П.Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии IDEF1 создана ее новая версия - методология IDEF1X. IDEF1X разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF).

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рис.).



Сущности.

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Лекция 21. Общая классификация архитектур информационных приложений.

Организация информационных систем на основе использования выделенных файл-серверов является наиболее распространенной в связи с наличием большого количества персональных компьютеров разного уровня развитости и сравнительной дешевизны связывания PC в локальные сети. Чем привлекает такая организация? Скорее всего, тем, что при опоре на файл-серверные архитектуры сохраняется автономность прикладного и большей части системного программного обеспечения, работающего на каждой PC сети. Фактически, компоненты информационной системы, выполняемые на разных PC, взаимодействуют только за счет наличия общего хранилища файлов, которое хранится на файл-сервере. В классическом случае в каждой PC дублируются не только прикладные программы, но и средства управления базами данных. Файл-сервер представляет собой разделяемое всеми PC комплекса расширение дисковой памяти (рис. 4.1).

Кратко перечислим основные достоинства и недостатки файл-серверных архитектур.

Основным достоинством является простота организации. Проектировщики и разработчики информационной системы находятся в привычных и комфортных условиях IBM PC в среде MS-DOS, Windows или какого-либо облегченного варианта Windows NT. Имеются удобные и развитые средства разработки графического пользовательского интерфейса, простые в использовании средства разработки систем баз данных и/или СУБД. Но во многом эта простота является кажущейся.

Во-первых, информационной системе предстоит работать с базой данных. Следовательно, эта база данных должна быть спроектирована. Почему-то часто разработчики файл-серверных приложений считают, что по причине простоты средств управления базами данных проблемой проектирования базы данных можно пренебречь. Это неправильно. База данных есть база данных. Чем качественнее она спроектирована, тем больше шансов впоследствии эффективно использовать информационную систему. Сложность проектирования базы данных определяется объективной сложностью моделируемой предметной области.

Во-вторых, необходимыми требованиями к базе данных информационной системы являются поддержание ее целостного состояния и гарантиро-

ванная надежность хранения информации. Минимальными условиями, при соблюдении которых можно удовлетворить эти требования, являются:

1. наличие транзакционного управления,
2. хранение избыточных данных (например, с применением методов журнализации),

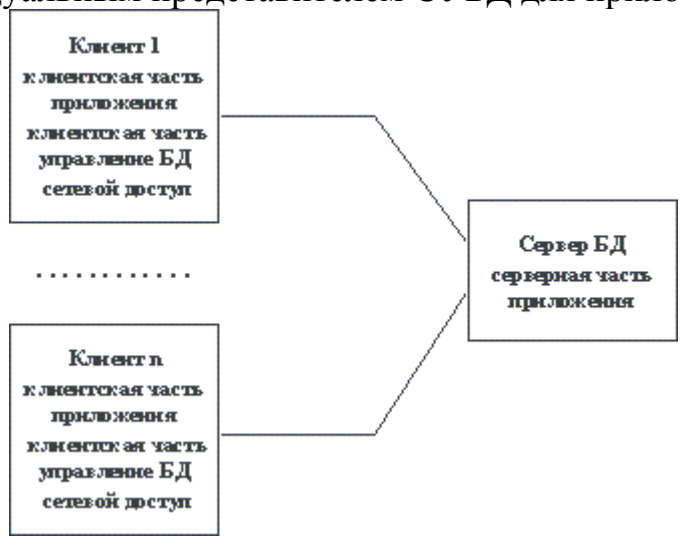
возможность формулировать ограничения целостности и проверять их соблюдение.

Лекция 22. Клиент-серверные приложения.

Под клиент-серверным приложением будем понимать информационную систему, основанную на использовании серверов баз данных. Общее представление информационной системы в архитектуре "клиент-сервер" показано на рисунке.

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных, которая, фактически, является индивидуальным представителем СУБД для приложения.



Общее представление информационной системы
в архитектуре "клиент-сервер".

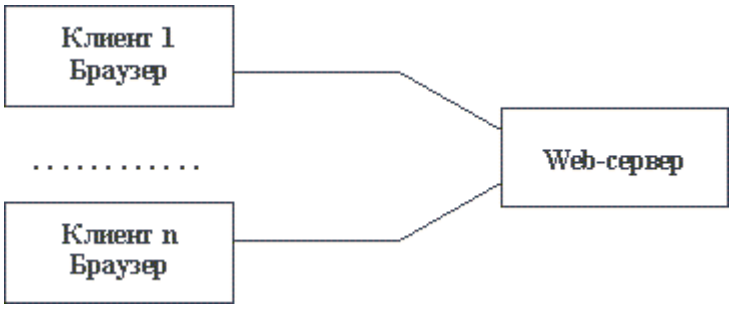
Интерфейс между клиентской частью приложения и клиентской частью сервера баз данных, как правило, основан на использовании языка SQL. Поэтому такие функции, как, например, предварительная обработка форм, предназначенных для запросов к базе данных, или формирование результирующих отчетов выполняются в коде приложения. Клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

Обычно компании, производящие развитые серверы баз данных, стремятся к тому, чтобы обеспечить возможность использования своих продуктов не только в стандартных на сегодняшний день TCP/IP-ориентированных сетях, но в сетях, основанных на других протоколах, например, SNA или IPX/SPX. Поэтому, при организации сетевых взаимодействий между клиентской и серверной частями СУБД, часто используются не стандартные средства высокого уровня.

Лекция 23. Intranet-приложения.

Возникновение и внедрение в широкую практику высокоуровневых служб Всемирной Сети Сетей Internet (e-mail, ftp, telnet, Gopher, WWW и т.д.) естественным образом повлияли на технологию создания корпоративных информационных систем, породив направление, известное теперь под названием Intranet. Информационная Intranet-система - это корпоративная система, в которой используются методы и средства Internet. Такая система может быть локальной, изолированной от остального мира Internet, или опираться на виртуальную корпоративную подсеть Internet. В последнем случае особенно важны средства защиты информации от несанкционированного доступа.

В общем случае в Intranet-системе могут использоваться все возможные службы Internet, наибольшее внимание привлекает гипермедийная служба WWW (World Wide Web - Всемирная Паутина). Для этого имеются две основные причины. Во-первых, с использованием языка гипермедийной разметки документов HTML можно сравнительно просто разработать удобную для использования информационную структуру, которая в дальнейшем будет обслуживаться одним из готовых Web-серверов. Во-вторых, наличие нескольких готовых к использованию клиентских частей - браузеров, или "обходчиков" избавляет от необходимости создавать собственные интерфейсы с пользователями, предоставляя им удобные и развитые механизмы доступа к информации. В ряде случаев такая организация корпоративной информационной системы (рис) оказывается достаточной для удовлетворения потребностей компании.



Простая организация Intranet-системы
с использованием средств WWW.

Однако, при всех своих преимуществах, простота организации, удобство использования, стандартность интерфейсов, эта схема обладает сильными ограничениями. Прежде всего, как видно из рисунка, в информационной системе отсутствует прикладная обработка данных. Все, что может пользователь, это только просмотреть информацию, поддерживаемую Web-сервером. Гипертекстовые структуры трудно модифицируются.

Лекция 24. Склады данных и системы оперативной аналитической обработки данных.

У аналитических отделов организации и у высшего звена управляющего состава имеются следующие задачи: проанализировав поведение корпорации на рынке с учетом сопутствующих внешних факторов и спрогнозировав хотя бы ближайшее будущее, выработать тактику, а возможно, и стратегию организации. Для решения таких задач требуются данные и прикладные программы, отличные от тех, которые используются в оперативных информационных системах. В последние время все более популярным становится подход, основанный на концепциях склада данных и системы оперативной аналитической обработки данных. Трудно производить долговременные прогнозы бизнес-деятельности, но анализ прошлого и краткосрочные прогнозы будущего могут оказаться очень полезными.

Поскольку термины, связанные со складами данных не так давно появились и на английском языке, и смысл их постоянно уточняется, трудно найти правильные русскоязычные эквиваленты. На сегодняшний день "datawarehouse" разными авторами переводится на русский язык как "хранилище данных", "информационное хранилище", "склад данных". Поскольку термин "хранилище" явно перегружен (он соответствует и английским терминам "storage" и "repository"), будем использовать термин "склад данных". Термин "data mart" по смыслу толкования - термин "рынок данных. Постепенно терминология будет согласована, но это произойдет только тогда, когда склады данных будут активно использоваться в России.

Прежде чем перейти к техническим аспектам, рассмотрим соответствующие вопросы на концептуальном уровне. Начнем с того, что главным образом различает оперативные и аналитические информационные приложения с точки зрения обеспечения требуемых данных, речь идет о так называемых OLAP-системах (от On-Line Analitical Processing), т.е. аналитических системах, помогающих принимать бизнес-решения за счет динамически производимых анализа, моделирования и/или прогнозирования данных.

Основным источником информации, поступающей в оперативную базу данных, является деятельность организации. Для проведения анализа данных требуется привлечение внешних источников информации, например, статистических отчетов, тем самым, склад данных должен включать как внутренние корпоративные данные, так и внешние данные, характеризующие рынок в целом.

Если для оперативной обработки, как правило, требуются свежие данные, обычно в оперативных базах данных информация сохраняется не более нескольких месяцев, то в складе данных нужно поддерживать хранение информации о деятельности корпорации и состоянии рынка на протяжении нескольких лет, для проведения достоверных анализа и прогнозирования. Как следствие, аналитические базы данных имеют объем как минимум на порядок больший, чем оперативные.

Лекция 25. Интегрированные распределенные приложения.

Не возникает никаких проблем, если с самого начала информационное приложение проектируется и разрабатывается в духе подхода открытых систем: все компоненты являются мобильными и интероперабельными, общее функционирование системы не зависит от конкретного местоположения компонентов, система обладает хорошими возможностями сопровождаемости и развития. На практике этот идеал является трудно достижимым. По разным причинам возникают потребности в интеграции независимо и по-разному организованных информационно-вычислительных ресурсов. Ни в одной действительно серьезной распределенной информационной системе не удастся обойтись без применения некоторой технологии интеграции. Существует путь решения этой проблемы, который сам лежит в русле открытых систем, - подход, предложенный крупнейшим международным консорциумом OMG (Object Management Group).

Остановимся на некоторых факторах, стимулирующих использование методов интеграции разнородных информационных ресурсов [12].

Неоднородность, распределенность и автономность информационных ресурсов системы. Неоднородность ресурсов может быть синтаксической, при их представлении используются, например, разные модели данных, и/или семантической, используются разные виды семантических правил, детализируются и/или агрегируются разные аспекты предметной области. Возможна и чисто реализационная неоднородность информационных ресурсов, обусловленная использованием разных компьютерных платформ, операционных систем, систем управления базами данных, систем программирования и т.д.

Потребности в интеграционном комплексировании компонентов информационной системы. Наиболее естественным способом организации сложной информационной системы является ее иерархически-вложенное построение. Более сложные функционально-ориентированные компоненты строятся на основе более простых компонентов, которые могли проектироваться и разрабатываться независимо.

Реинжинерия системы. После создания начального варианта информационной системы неизбежно последует процесс ее непрерывных переделок, реинжинерии, обусловленный развитием и изменением соответствующих бизнес-процессов корпорации. Реконструкция системы не должна быть

революционной. Все компоненты, не затрагиваемые процессом реинжиниринга, должны сохранять работоспособность.

Решение проблемы унаследованных систем. Любая компьютерная система со временем становится бременем корпорации. Постоянно, и чем раньше, тем лучше, приходится решать задачу встраивания устаревших информационных компонентов в систему, основанную на новой технологии. Нужно, чтобы эта задача была разрешимой, т.е. чтобы компоненты унаследованных систем сохраняли интероперабельность.

7. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

Настоящие методические указания содержат основные требования и рекомендации по организации, выполнению и защите курсовых проектов (работ) по направлению 654600 - “Информатика и вычислительная техника“ по специальности 230201 -“Информационные системы и технологии”.

В основу этих указаний положены “Инструкция по подготовке курсовых проектов (работ) в высших учебных заведениях” и квалификационная характеристика дипломированного специалиста по направлению 654600 “Информатика и вычислительная техника“ по специальности 230201 - “Информационные системы и технологии”.

Общие положения.

Курсовой проект выполняется при обучении в девятом семестре в соответствии с установленным учебным планом.

Целью выполнения курсового проекта является формирование у студентов навыков:

- самостоятельной научно-исследовательской деятельности;
- практической деятельности;
- грамотного оформления полученных результатов в печатном виде;
- представления результатов своей работы в виде научного доклада;
- защиты полученных результатов в дискуссии.

Выбор темы работы.

Тематика курсовых проектов определяется преподавателями кафедр, осуществляющих руководство научной работой студентов. Перечень предлагаемых тем (названий) работ с указанием научного руководителя доводится до сведения студентов в течение первых трех недель текущего учебного года.

Студент самостоятельно выбирает научного руководителя и тему работы в соответствии со своими интересами, о чем лично сообщает выбранному им научному руководителю. В ходе предварительного обсуждения выбранной темы с научным руководителем и в процессе выполнения работы тема

может быть изменена по согласованию между научным руководителем и студентом.

Выбор темы должен быть сделан студентом в течение четвертой недели текущего учебного года.

Проект выполняется в течение семестра и может быть продолжением ранее начатого исследования или развитием результатов, полученных студентом в течение предшествующих лет обучения.

Содержание курсовой работы.

Курсовой проект, как правило, представляет собой:

1. Исследование актуальной задачи по специальности;
2. Разработку информационной системы (ИС) или иного программного продукта;

Исследование задачи включает следующие разделы:

1. Описание (постановка) задачи.
2. Обоснование актуальности задачи.
3. Обзор информации, содержащейся в открытых источниках, посвященных данной задаче или области исследований.
4. Исследование задачи:

а) классификация задачи, т. е. отнесение ее к некоторому известному классу задач;

б) описание известных методов решения задач этого класса;

в) описание особенностей исследуемой задачи, ее отличительных черт, которые не позволяют применять существующие методы в стандартном виде;

г) предложения по модификации существующих методов для решения задачи (близких задач) или по модификации самой задачи для применения существующих методов;

д) описание предлагаемых методов решения или подходов к решению с обоснованием их применимости к данной задаче;

е) описание возникающих в процессе решения проблем или других побочных, вспомогательных или параллельных задач.

Разработка информационной (программной) системы включает следующие разделы:

1. Постановка задачи:

а) описание предметной области, например, бизнес-процессов, протекающих в предметной области;

б) описание информационных потоков;

в) описание процессов обработки информации, управления и т. п., требующих автоматизации.

2. Обзор существующих программных продуктов, выполняющих аналогичные функции. Их достоинства и недостатки. Сравнение, классификация.
3. Обоснование необходимости разработки.
4. Описание математической задачи, решаемой с помощью ИС:

а) постановка задачи;
б) исследование задачи;
в) математическая модель;
г) метод решения возникающей математической задачи;
д) алгоритм, реализующий метод решения.
5. Проект ИС:
а) архитектура ИС, аппаратная платформа, ОС, СУБД, состав других программных продуктов;
б) описание структуры БД: таблицы, индексы, процедуры, триггеры, события, запросы;
в) описание логики программ и интерфейсов;
г) алгоритмы обработки данных.
6. Реализация ИС:
а) назначение и функции программы, режимы работы программы;
б) описание категорий пользователей программы, разграничения прав пользователей;
в) описание последовательности пользовательских интерфейсов, реализующих каждую функцию ИС;
г) описание входных данных;
д) описание выходных данных;
е) описание методов защиты данных в ИС и информационных систем в целом;
ж) технические характеристики ИС (тип и минимально необходимые аппаратные ресурсы вычислительной системы, требуемое программное обеспечение и т. п.).
7. Анализ:
а) внедрение ИС;
б) области применения ИС;
в) достоинства и недостатки по сравнению с перечисленными ранее в разделе 2 аналогами;
г) описание возникших в процессе разработки проблем, их причины, предложения по решению;
д) направления дальнейшего развития ИС.
Реферативная работа включает следующие разделы:
1. Обоснование актуальности выбранной тематики и описание целей выполнения работы.
2. Систематизация и анализ найденных в научной печати, в сети Интернет и других источниках материалов.
3. Выводы.
4. Предложения по использованию результатов работы в конкретных областях и возможные направления дальнейших исследований.

Порядок выполнения курсовой работы.

Курсовой проект выполняется в течение девятого семестра.

По итогам работы, до начала сессии, студент в установленный кафедрой срок представляет подготовленный в печатном виде материал и по решению кафедры делает сообщение по теме курсового проекта на семинаре. Руководитель на титульном листе выставляет оценку, заверяя ее своей подписью.

На титульном листе фиксируется срок представления отчета. В случае несоблюдения сроков представления или низкого качества отчета, его оценка снижается, как минимум, на один балл.

Студенты, не представившие в срок курсовые проекты, не допускаются к публичной защите. В этом случае защита переносится на начало сессии.

Оценка за курсовую работу складывается из следующих оценок:

• оценки руководителя;
• оценки публичной защиты;
• оценки оформления.

При выставлении оценки учитываются также сроки представления печатного варианта курсового проекта.

Выступление с докладом по промежуточным результатам курсового проекта на студенческой научной конференции может быть засчитано с выставлением заслуженной оценки (отлично, хорошо, удовлетворительно).

В случае отсутствия научного руководителя в период представления курсового проекта (командировка, болезнь и т. п.), курсовой проект сдается на кафедру в установленные сроки. Оценка руководителя выставляется позднее.

Структура курсового проекта.

Работа начинается с титульного листа стандартной формы, за которым следует лист задания, оглавление работы, введение, нескольких разделов, заключения, списка использованных научных источников, приложений.

Введение содержит общий обзор работы, позволяющий составить общее представление об исследуемой проблеме и полученных результатах. Во введении может быть предложена краткая аннотация отдельных разделов работы. Первый раздел должен содержать достаточно подробное описание проблемы, поставленной перед исполнителем с обоснованием ее актуальности и анализ современного состояния исследований и разработок в данной области.

В последующих разделах, число которых произвольно, описываются результаты, полученные по отдельным аспектам исследуемой проблемы. Каждый раздел может разбиваться на подразделы.

Заключение содержит перечень основных результатов, полученных в работе, и сделанных выводов. В него могут включаться рекомендации относительно перспектив продолжения данной работы.

В списке использованных источников указываются использованные автором работы научные публикации, а также другие источники, в том числе электронные, по проблемам разработки аналогичных систем, по средствам

разработки, по методам решения математических задач. На все перечисленные в списке литературы источники в соответствующих местах работы должны быть сделаны ссылки (номер источника заключается в квадратные скобки).

Список использованных источников должен содержать не менее 10 печатных изданий и любого количества непечатных изданий.

Приложения могут содержать дополнительную информацию: графики, таблицы, тексты программ и т. п.

Оформление курсового проекта.

Курсовой проект представляется на кафедру в полностью готовом виде (сшитом, в переплете или в обложке).

Дополнительно к проекту прилагаются специальные (магнитные или иные) носители информации, содержащие программы (тексты и исполняемые файлы), данные или объемные приложения, включение которых в текст работы является нецелесообразным.

Текст курсовой работы оформляется в принятом для научных работ виде.

На странице располагается 30 строк., в строке 60 знаков, включая пробелы.

Следует соблюдать следующие размеры полей: левое – не менее 30мм, правое – не менее 10 мм, верхнее – не менее 15 мм, нижнее – не менее 20 мм.

Нумерация страниц выполняется арабскими цифрами, начиная с титульного листа. На титульном листе, а также на первой странице оглавления номер не ставится. На следующих страницах номер ставят в правом верхнем углу.

Работа должна быть отпечатана. Рукописные работы не принимаются.

Графический материал.

Графический материал проекта должен состоять из чертежей, выполненных в соответствии с требованиями ЕСКД, ЕСПД и других действующих ГОСТов. Чертежи должны быть выполнены или на фолиях или в мультимедийном исполнении и снабжены штампом. Каждый чертеж должен быть снабжен основной надписью, графы которой должны быть заполнены в соответствии с образцом.

Макеты чертежей перед окончательным оформлением необходимо показать руководителю проекта.

К основным ГОСТам, которые необходимо использовать при оформлении курсового проекта, приведенные в работе относятся:

ГОСТ 34.201-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем.

ГОСТ 34.602-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. с. 106-116.

РД 50-682-89 Методические указания. Информационная технология Комплекс стандартов и руководящих документов на автоматизированные системы. Общие положения. с. 157-161.

РД 50-680-88 Методические указания. Автоматизированные системы. Основные положения. с. 152-156.

ГОСТ 34.601-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. с. 100-105.

ГОСТ 34.401-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Средства технические периферийные автоматизированных систем. Типы и технические требования.

РД 50-34.698-90 Методические указания. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов. с. 127-151.

ГОСТ 34.003-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения.

Р50-34.119-90 Рекомендации. Информационная технология. Комплекс стандартов на автоматизированные системы. Архитектура локальных вычислительных сетей в системах промышленной автоматизации. Общие положения.

ГОСТ 24.301-80* Система технической документации на АСУ. Общие требования к выполнению текстовых документов.

ГОСТ 24.302-80 Система технической документации на АСУ. Общие требования к выполнению схем. с. 22-24.

ГОСТ 24.303-80 Система технической документации на АСУ. Обозначения условные графические технических средств. с.25-31.

ГОСТ 24.304-80 Система технической документации на АСУ. Требования к выполнению чертежей с.32-34.

ГОСТ 19.401-78* Единая система программной документации. Текст программы. Требования к содержанию и оформлению.

ГОСТ 19.402-78* Единая система программной документации Описание программы.

Кроме того, графический материал может быть представлен в виде фолий и мультимедийных разработок.

Защита курсового проекта.

Курсовой проект, подписанный исполнителем, представляется на проверку руководителю за 3-4 дня до назначенного срока защиты.

Защита проекта проводится индивидуально каждым студентом на заседании комиссии, состав которой утверждается кафедрой, с обязательным участием руководителя курсового проекта.

Студент, выполнивший курсовой проект, делает доклад (5-7 минут) и отвечает па вопросы комиссии.

В докладе студенту необходимо изложить важнейшие этапы, особенности и результаты работы, не углубляясь в тонкости конкретных технических решений, четко сформулировать конечные результаты.

Вопросы, задаваемые студенту, могут касаться деталей выполненного проекта, а также разделов курсов, по которым выполнялся проект.

Результаты защиты курсового проекта определяются оценками "отлично", "хорошо", "удовлетворительно", "неудовлетворительно", которые проставляются на титульном листе пояснительной записки. При оценке работы учитывается качество выполнения и оформления курсового проекта, уровень защиты проекта и ответов на вопросы, мнение руководителя.

8. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа № 1
Нормализация. Создание физической модели

Цель работы:

- изучить виды нормальных форм,
- освоить роль CASE-средства ERWin при нормализации и денормализации БД,
- построить физическую модель,
- изучить алгоритмы перевода БД в первую, вторую и третью нормальную форму (для самостоятельного изучения).

1. Нормализация

Нормализация - процесс проверки и реорганизации сущностей и атрибутов с целью удовлетворения требований к реляционной модели данных. Нормализация позволяет быть уверенным, что каждый атрибут определен для своей сущности, значительно сократить объем памяти для хранения данных.

Для рассмотрения видов нормальных форм введем понятия функциональной и полной функциональной зависимости.

Функциональная зависимость

Атрибут В сущности Е функционально зависит от атрибута А сущности Е, если и только если каждое значение А и Е связало с ним точно одно значение В в Е. Другими словами, А однозначно определяет В.

Полная функциональная зависимость

Атрибут B сущности E полностью функционально зависит от ряда атрибутов A сущности E. если и только если B функционально зависит от A и не зависит ни от какого подряда A.

Существуют следующие виды нормальных форм:

- Первая нормальная форма (1NF). Сущность E находится в первой нормальной форме, если и только если все атрибуты содержат только атомарные значения. Среди атрибутов не должно встречаться повторяющихся групп, т. е. нескольких значений для каждого экземпляра.
- Вторая нормальная форма. Сущность E находится во второй нормальной форме, если она находится в первой нормальной форме и каждый неключевой атрибут полностью зависит от первичного ключа, т. е. не существует зависимостей от части ключа.
- Третья нормальная форма (3 NF). Сущность E находится в третьей нормальной форме, если она находится во второй нормальной форме и неключевые атрибуты сущности E зависят от других атрибутов E.

После третьей нормальной формы существуют нормальная форма Бойсса-Кодда, четвертая и пятая нормальные формы. На практике ограничиваются приведением к третьей нормальной форме. Часто после проведения нормализации все взаимосвязи данных становятся правильно определены, модель данных становится легче поддерживать. Однако нормализация не ведет к повышению производительности системы в целом, поэтому при создании физической модели в целях повышения производительности приходится сознательно отходить от нормальных форм, чтобы использовать возможности конкретного сервера. Такой процесс называется денормализацией.

1.1. Поддержка нормализации в ERWin

ERWin обеспечивает только поддержку нормализации, но не содержит в себе алгоритмов, автоматически преобразующих модель данных из одной формы в другую.

Поддержка первой нормальной формы

В модели каждая сущность или атрибут идентифицируется с помощью имени. В ERWin поддерживают корректность имен следующим образом:

- отмечает повторное использование имени сущности и атрибута;
- не позволяет внести в сущность более одного внешнего ключа;
- запрещает присвоение неуникальных имен атрибутов внутри одной модели, соблюдая правил в одном месте - один факт».

2. Создание физическое модели

Целью создания физической модели является обеспечение администратора соответствующей информацией для переноса логической модели данных в СУБД.

ERWin поддерживает автоматическую генерацию физической модели данных для конкретной СУБД. При этом логическая модель трансформиру-

ется в физическую по следующему принципу: сущности становятся таблицами, атрибуты становятся столбцами, а ключи становятся индексами.

Таблица 7.1. Сопоставление компонентов логической и физической модели

Логическая модель	Физическая модель
Сущность	Таблица
Атрибут	Столбец
Логический тип (текст, число, дата, blob)	Физический тип (коррективный тип, зависящий от выбранной СУБД)
Первичный ключ	Первичный ключ, индекс PK
Внешний ключ	Внешний ключ, индекс FK
Альтернативный ключ	AK-индекс – уникальный, непервичный индекс
Правило бизнес-логики	Триггер или сохраненная процедура
Взаимосвязи	Взаимосвязи, определяемые использованием FK-атрибутов

3. Денормализация

После нормализации все взаимосвязи данных становятся определены, исключая ошибки при оперировании данными. Но нормализация данных снижает быстродействие БД. Для более эффективной работы с данными, используя возможности конкретного сервера БД, приходится производить процесс, обратный нормализации, - денормализацию.

Для процесса денормализации не существует стандартного алгоритма, поэтому в каждом конкретном случае приходится искать свое решение. Денормализация обычно проводится на физическом уровне модели. ERWin имеет следующие возможности по поддержке процесса денормализации:

- Сущности, атрибуты, группы ключей и домены можно создавать только на логическом уровне модели. В ERWin существует возможность выделения элементов логической модели таким образом, чтобы они не появлялись на физическом уровне.
- Таблицы, столбцы, индексы и домены можно создавать только на физическом уровне. В ERWin существует возможность выделения элементов модели таким образом, что бы они не появлялись на логическом уровне. Эта возможность напрямую поддерживает денормализацию физической модели, так как позволяет проектировщику включать таблицы, столбцы и индексы в физическую модель, ориентированную на конкретную СУБД.
- Разрешение связей «многие-ко-многим». При разрешении этих связей в логической модели ERWin добавляет ассоциированные сущности и позволяет добавить в них атрибуты. При разрешении связей в логической модели автоматически разрешаются связи и в физической модели.

4. Пример

Нормализуем полученную в предыдущей лабораторной работе БД до третьей нормальной формы. Для приведения БД в первую нормальную форму необходимо выполнить условие, при котором все атрибуты содержат атомарные значения. Рассмотрим атрибуты сущности «Студент». Студент может иметь несколько адресов электронной почты и несколько телефонных номеров, что является нарушением первой нормальной формы. Необходимо создать отдельные сущности «E-mail» и «Телефон» и связать их с сущностью «Студент» (рис. 7.1).

Проверим соответствие БД второй нормальной форме. Все неключевые заметить, что это условие выполняется для всех сущностей БД; следовательно, можно сделать вывод о том, что она находится во второй нормальной форме.

Для приведения БД к третьей нормальной форме необходимо обеспечить отсутствие транзитивных зависимостей неключевых атрибутов. Такая зависимость наблюдается у атрибутов «Специальность» и «Специализация» у сущности «Студент»: специализация зависит от специальности и от группы, в которой обучается студент. Создадим новую независимую сущность «Специальность», перенеся в нее атрибут «Специализация» и создав новый атрибут «Группа», являющийся ключевым и определяющий атрибуты «Специальность» и «Специализация». Проведем неидентифицирующую связь от сущности «Специальность» к сущности «Студент», при этом ключевой атрибут «Группа» мигрирует в сущность «Студент». Получим БД в третьей нормальной форме, так как других транзитивных зависимостей неключевых атрибутов нет (рис.7.2).

Перейдем к построению физической модели.

Перед построением физической модели выбрать сервер (меню Server/Target Server). Выберем в качестве сервера Microsoft Access 97. получив физическую модель, сгенерированную ERWin по умолчанию (рис. 7.3).

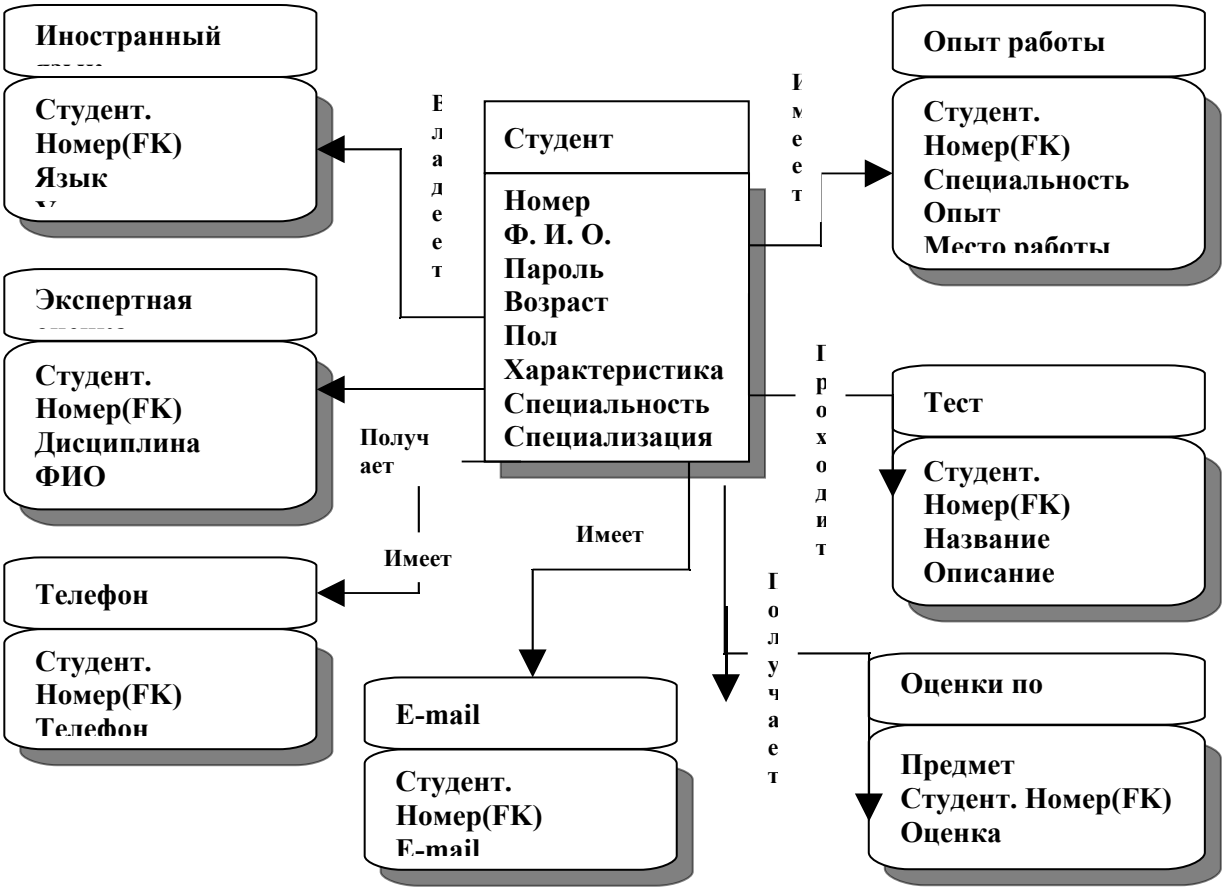


Рис. 7.1. ERD-диаграмма БД студентов в первой нормальной форме

В полученной модели необходимо скорректировать типы и размеры полей. Кроме того, на этапе создания физической модели данных вводятся правила валидации колонок, определяющие списки допустимых значений и значения по умолчанию.

После установки правил валидации в диалоговом окне Validation Rule Editor должны получиться следующие правила (рис. 7.3).

После установки правил валидации в диалоговом окне Column Editor необходимо присвоить соответствующим колонкам таблиц установленные для них правила (рис. 7.4).

Таким образом, проделав все вышеописанные действия, мы получили модель БД, готовую для помещения в СУБД. Для генерации кода создания БД необходимо выбрать пункт меню Tasks->Forward Engineer/Schema Generation, после чего откроется окно установки свойств генерируемой схемы данных. Для предварительного просмотра SQL-скрипта служит кнопка Preview, для генерации схемы - Generate. В процессе генерации ERWin связывается с БД, выполняя SQL-скрипт. Если в процессе генерации возникают какие-либо ошибки, то она прекращается, открывается окно с сообщениями об ошибках.

5. Задания

1. Нормализовать БД до третьей нормальной формы.

2. Построить физическую модель БД системы службы занятости.

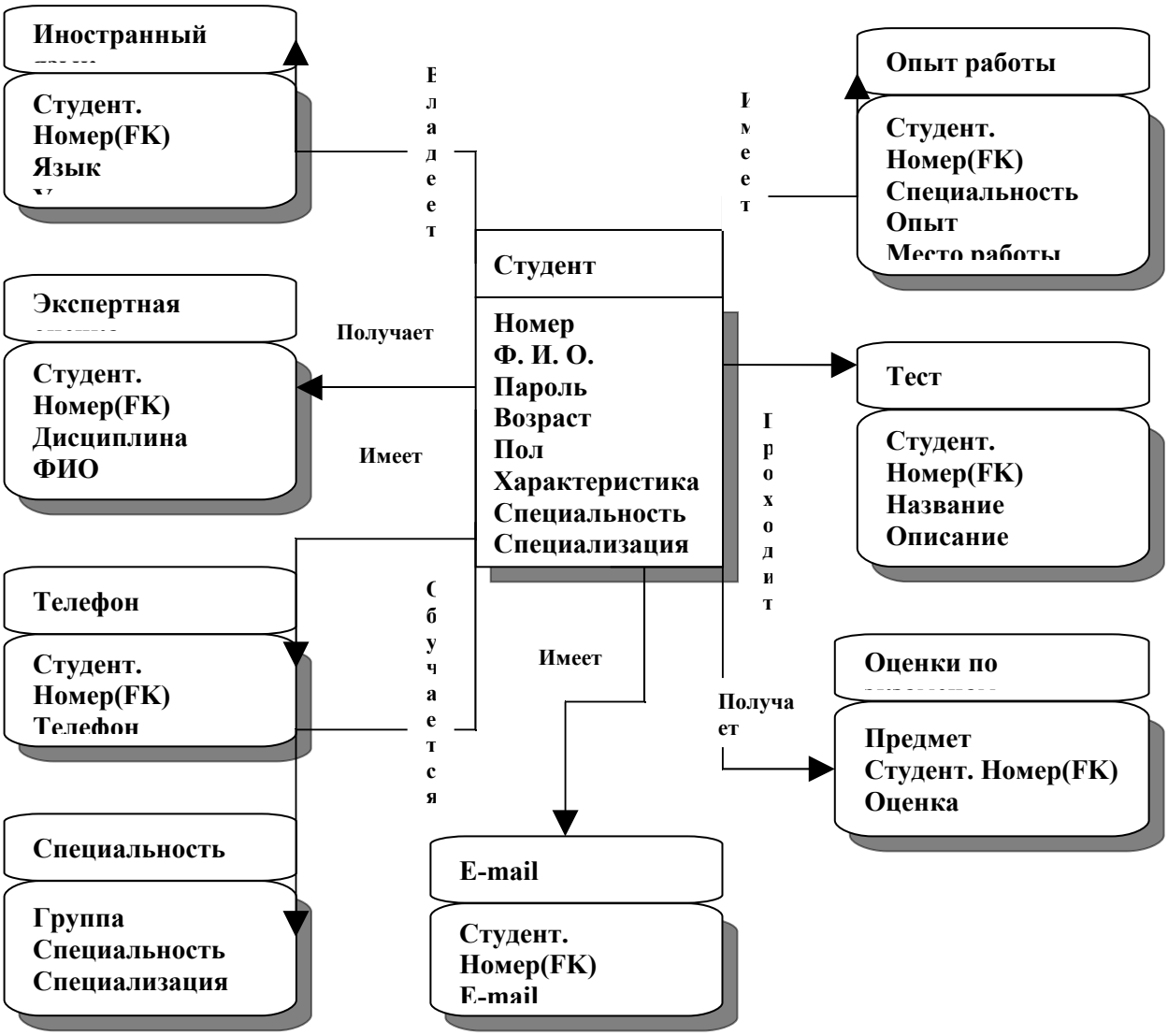


Рис. 7.2. ERD-диаграмма БД студентов в третьей нормальной форме

Таблица 7.2. Свойства колонок таблиц физической модели БД студентов

Колонка	Тип	Размер	Правило валидации
Номер	Long Integer		
Группа	Text	7	
Ф.И.О.	Text	64	

Таблица 7.2. Свойства колонок таблиц (физической модели БД студентов)

Колонка	Тип	Размер	Правило валидации
Пароль	Text	15	> 10 и < 100
Возраст	Number		

Пол	Text	1	М или Ж
Характеристика	Memo		
E-mail	Text	40	
Специальность	Text	20	
Специализация	Text	20	
Опыт	Number		> 0
Место работы	Text	20	
Язык	Text	25	
Уровень владения	Number		≥ 2 и ≤ 5
Название	Text	30	
Описание	Memo		
Оценка	Number		≥ 2 и ≤ 5
Дисциплина	String	30	
Ф.И.О. преподавателя	Text	64	
Предмет	Text	30	

Проверка опыта **> = 0**
Client/Server
Проверка оценки **BETWEEN 2 AND 5**
Client/Server

Рис. 7.3. Правила валидации

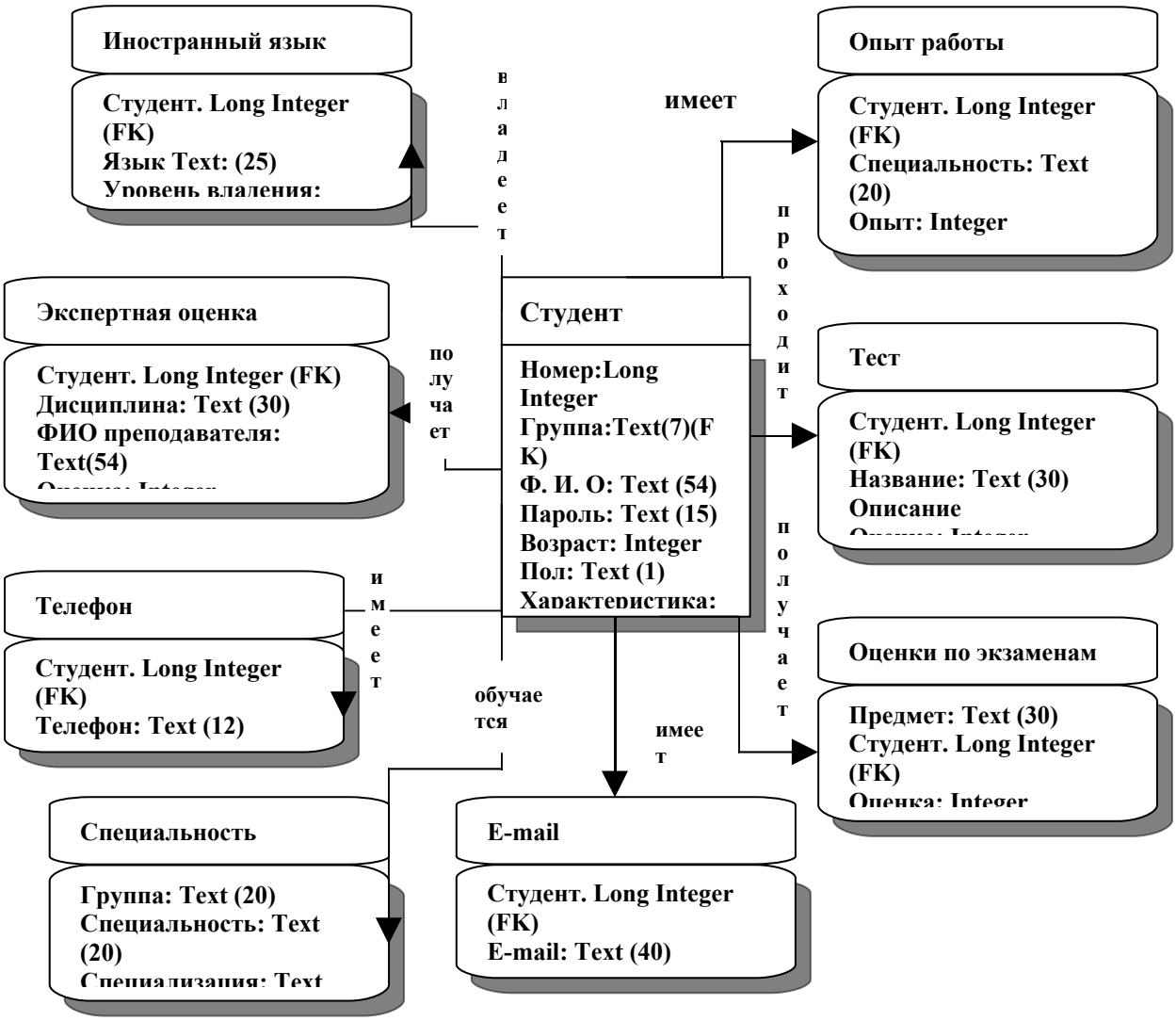


Рис. 7.4. Физическая модель БД студентов

Лабораторная работа № 2
Отчеты в ERWin

Цель работы:

- изучить виды отчетов,
- освоить процедуру создания отчетов,
- изучить экспортирование, сохранение и печать отчетов.

Для формирования отчетов в ERWin имеется простой инструмент -Report Browser. Он позволяет создавать предопределенные отчеты сохранять результаты, печатать и экспортировать в различные форматы. Кроме того, Report Browser позволяет создавать собственные типы отчетов.

Диалоговое окно Report Browser вызывается из панели инструментов главного окна нажатием кнопки  .

В левой части окна, в виде дерева, отображаются предварительно определенные отчеты, позволяющие представлять информацию об основных объектах логической и физической модели. Для выполнения отчета необхо-

димо выделить его в окне и нажать соответствующую кнопку на панели инструментов. Результат выполнения отчета отобразится в правой части окна. При этом в дерево отчетов будет добавлена иконка образованного отчета.

Отчеты группируются в папках, при этом каждый отчет может включать в себя несколько результирующих наборов данных, каждое из которых создается при выполнении отчета. Все элементы помечены одной из следующих иконок:

 - папка,

 - отчет,

 - изменяемый отчет,

 - результирующий набор данных,

 - представление.

Диалоговое окно имеет собственное меню и панели инструментов (табл. 1 ,2).

Таблица 1. Кнопки панели инструментов диалогового, окна Report Browser

Кнопка	Назначение
	Создание нового отчета или папки
	Печать отчета
	Просмотр результата выполнения отчета
	Выполнение отчета
	Фиксация изменений (для редактируемого отчета)
	Поиск элементов отчета: задание условий поиска, поиск следующей строки и поиск другого отчета, соответствующего строке
	Включение и выключение дерева отчетов
	Показывает список отчетов в том порядке, в котором они создавались
	Переход к следующему отчету
	Выбор кнопок и сортировка полученного отчета

Таблица 2. Кнопки нижней панели инструментов

Кнопка	Назначение
	Редактировать выделенный отчет
	Удалить отчет
	Показать только верхний уровень дерева
	Сделать выбранную папку корнем дерева (показать только выбранную ветвь дерева)
	Сделать корнем дерева родительскую папку (по отношению к выбранной)

1. Создание отчета

Для создания нового, не предопределенного отчета необходимо:

1. Выбрать в меню пункт File->New или щелкнуть на кнопке  панели инструментов.
2. В появившемся диалоговом окне ERWin Report Editor в поле Name ввести имя отчета. Поле Category предназначено для указания категории отчета, т. е. типа объектов, по которым будет создаваться отчет (атрибуты, диаграммы, сущности, домены, связи и т. д.).
3. Указать категории, которые будут включены в отчет, при помощи ж иерархического списка, расположенного на закладке Options. Иконка  показывает, что соответствующую колонку в полученном отчете можно будет изменять. Папка, помеченная иконкой  , позволяет выбрать условия фильтрации данных отчета.
4. Щелкнуть по кнопке ОК, после чего отчет будет добавлен в диалоговое окно Report Browser.
5. Выполнить отчет, нажав на кнопку  на панели инструментов.

Полученный в результате выполнения отчета результирующий набор данных можно отформатировать, распечатать, экспортировать или сохранишь в виде представления.

Редактирование отчета производится выбором пункта Edit Report format во всплывающем меню, вызываемом на иконке результирующего набора. В появившемся диалоговом окне Report format можно изменить порядок сортировки данных, очередность колонок, сделать колонку невидимой, а также задать ее стиль.

Для полученного отчета необходимо выбрать во всплывающем меню пункт Export result set. Результирующий набор данных можно экспортировать в следующие форматы:

- CSV,
- HTML,
- DDE,
- RPTWin - специализированный генератор отчетов.

После окончания форматирования и настройки результирующего набора данных можно сохранить его в виде именованного представления. Для этого необходимо щелкнуть по кнопке  на панели инструментов и в открывшемся диалоговом окне указать имя представления. Представления служат для сохранения всех настроек результирующего набора и позволяют использовать их несколько раз, что значительно облегчает работу с отчетами.

2. Пример

Рассмотрим группу отчетов, проверяющих правильность построения модели. Эти отчеты в диалоговом окне Report Browser носят название Model Validation Reports, исполнение, которых может быть полезным для нахождения ошибок в моделях.

Выполним некоторые из них и рассмотрим полученные результаты, сведя их в таблицу.

Таблица 3. Отчеты

Отчет	Результат
Отчет «Сущности без атрибутов» (Entities without attributes)	Пустой отчет, т.е. сущности без атрибутов в модели нет
Отчет «Таблицы без первичного ключа» (Tables without PK)	Пустой отчет, т.е. модели таблицы в физической модели имеют первичный ключ
Отчет «Сущности без первичного ключа» (Entities without PK)	То же
Отчет «Колонки с различным типом внешнего ключа» (Columns with different FK datatype)	Найдена колонка «Группа», являющаяся внешним ключом сущности «Студент», отличающаяся от колонки «Группа» в сущности «Специальность»

Скорректируем модель согласно найденным ошибкам (рис. 1.)

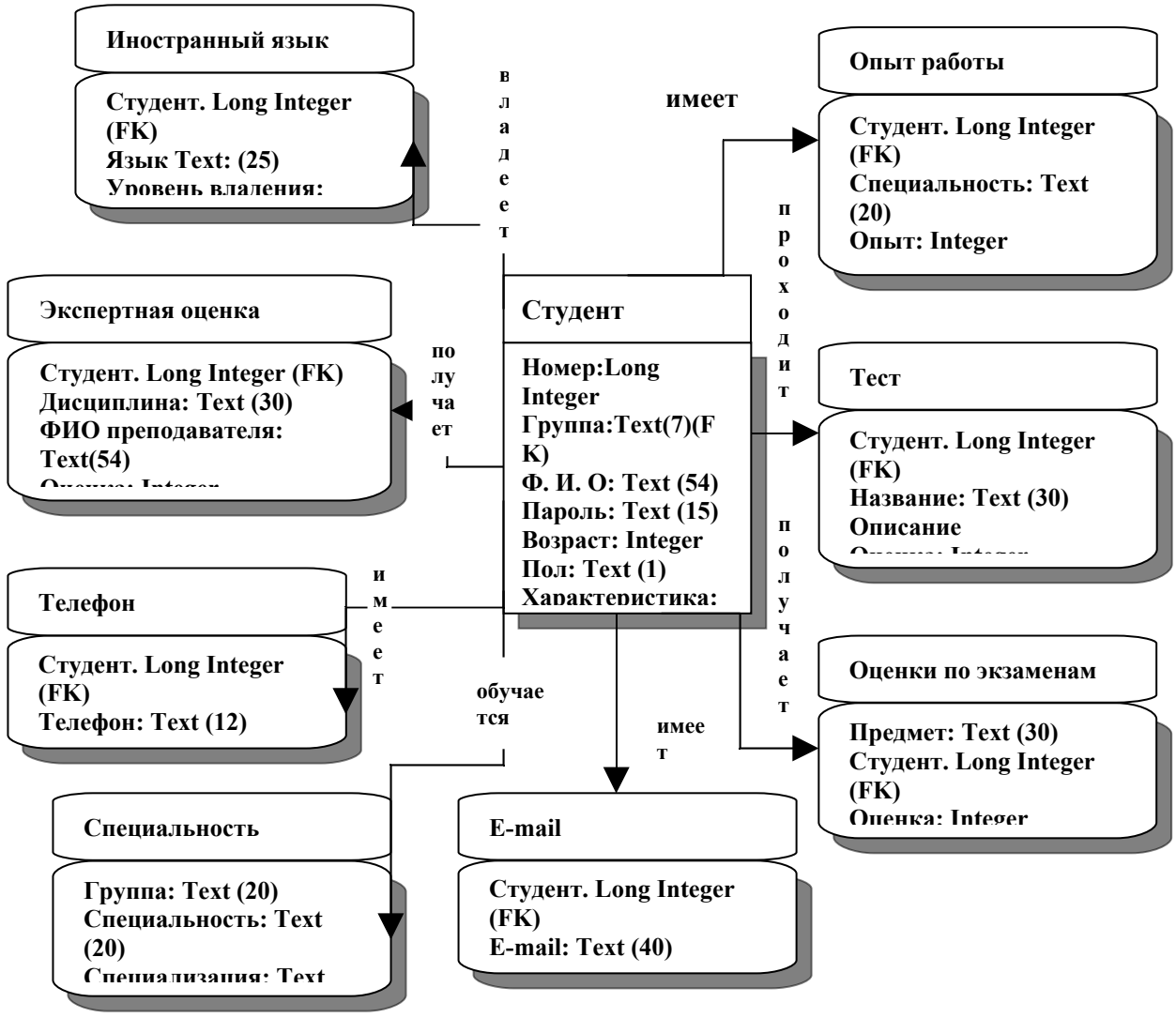


Рис. 1. Скорректированная физическая модель БД студентов

3. Задания

1. Создать отчет о таблицах физической модели, созданной на лабораторной работе № 7.
2. Создать отчет по всем сущностям и их атрибутам.
3. Сохранить полученные отчеты в формате HTML.
4. Изменить порядок сортировки в полученных отчетах и сохранить отредактированные отчеты в виде представлений.
5. Назначить полученным отчетам иконки.
6. Сформировать новый отчет из категории Model Validation, задав в нем все опции проверки корректности модели.
7. Выполнить полученный отчет и убедиться в отсутствии ошибок в модели данных.

Лабораторная работа № 3
Введение в CASE-пакет Rational Rose 98

Цель работы:

- изучение основных этапов проведения проектирования в Rational Rose 98,
- изучение основных элементов нотации, применяемых в CASE-пакете Rational Rose 98,
- изучение интерфейса Rational Rose и принципов работы с ним,
- создание нового проекта в Rational Rose.

1. Этапы проведения моделирования в Rational Rose 98

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

Концептуальная модель выражается в виде диаграмм вариантов использования (use-case diagram). Этот тип диаграмм служит для проведения итерационного цикла общей постановки задачи вместе с заказчиком. Часто можно услышать следующее: «Заказчик и раньше не знал, и теперь не знает, и в будущем не будет точно знать, что ему нужно». Диаграммы вариантов использования как раз и служат основой для достижения взаимопонимания между программистами-профессионалами, разрабатывающими проект, и заказчиками проекта.

Внутри каждого прецедента могут быть определены:

- вложенная диаграмма вариантов использования,
- диаграмма взаимодействия объектов (collaboration diagram),
- диаграмма последовательности взаимодействий (sequence diagram),
- диаграмма классов (class diagram),
- диаграмма перехода состояний (state diagram).

Логическая модель позволяет определить два различных взгляда им системы: статический и динамический. Статический подход выражается диаграммами классов (class diagram). Именно диаграммы классов служат основой для генерации программного кода на целевом языке программирования.

Возможна очень гибкая настройка генерации кода, позволяющая учитывать конкретные соглашения (например, по префиксам имен идентификаторов), принятые в команде разработчиков проекта.

Динамический подход описывается двумя типами диаграмм:

- диаграммами взаимодействия объектов,
- диаграммами последовательности взаимодействий.

В текущей версии Rational Rose 98 эти диаграммы не влияют на генерируемый код, однако фирмы-партнеры Rational Software применяют эти диаграммы в своих приложениях. Так, диаграммы последовательности взаимодействий используются в пакете SQA Suite для автоматизированного тестирования компонентов, разработанных в Rational Rose 98. Классы, введенные на этих диаграммах, попадают в список классов модели и могут использоваться при конструировании диаграмм классов.

Динамика конкретного класса может быть выражена с помощью диаграмм перехода состояний, определяющих модель конечного автомата, описывающего поведение класса. Каждое состояние задается своей вершиной; определены входное и выходные состояния, а также условия перехода из состояния в состояние.

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

После построения первого/последующего слоя статической модели с использованием диаграмм классов можно провести генерацию кода на целевом языке программирования. На уровне кода можно ввести новые уточняющие классы, изменить атрибуты и методы классов модели и затем синхронизировать код и модель, выполнив обратное проектирование, т. е. по модифицированному коду Rational Rose 98 позволяет построить новую логическую модель взаимосвязи классов между собой. Повторение такой процедуры несколько раз называется итерационным моделированием (round-trip modeling), которое составляет основу мягкого и постепенного уточнения постановки задачи и согласования требований заказчика с имеющимися ресурсами (вычислительными, временными, финансовыми и т. п.).

Создание нового проекта в Rational Rose производится сбором меню File/New. При этом создается несколько пустых диаграмм вечного уровня: диаграмма вариантов использования, диаграмма классов и др. Каждую диаграмму можно выбрать для редактирования, при этом на инструментальной панели отображаются элементы, доступные для данного вида диаграмм. Выбор типа текущей диаграммы производится в меню Browse.

Таблица 1. Описание элементов управления основной панели инструментов Rational Rose

Элемент управления	Описание	Соответствующий пункт меню
	Создать новую модель	File→ New
	Открыть модель	File→Open
	Сохранить модель	File→Save
	Напечатать модель	File→Print
	Переключение между типами диаграмм	Browse →Diagram…
	Получение справки	Help
	Открытие окна для ввода комментариев	View → Documentation
	Навигация по диаграммам	Browse → Previous Diagram
	Масштабирование	View → Zoom

2. Количественная оценка диаграмм

Методика количественной оценки и сравнение диаграмм UML строится на присвоении элементам диаграмм оценок, зависящих от их информационной ценности, а также от вносимой ими в диаграмму дополнительной сложности. Ценность отдельных элементов меняется в зависимости от типа диаграммы, на которой они находятся.

Словарь языка UML включает два вида строительных блоков: сущности и отношения. Сущности - это абстракции, являющиеся основными элементами модели. Отношения связывают различные сущности.

Количественную оценку диаграммы можно провести по следующей формуле:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}}, \qquad (1)$$

где S - оценка диаграммы; S_{Obj} - оценки для элементов диаграммы; S_{Lnk} -оценки для связей на диаграмме; Obj - число объектов на диаграмме; T_{Obj} - число типов объектов на диаграмме; T_{Lnk}- число типов связей на диаграмме.

Если диаграмма содержит большое число связей одного типа (например, модель БД), то число и тип связей можно не учитывать и формула расчета приводится к виду

$$S = \frac{\Sigma S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}}, \qquad (2)$$

Если на диаграмме показаны атрибуты и операции классов, можно учесть их при расчете, при этом оценка прибавляется к оценке соответствующего класса:

$$S_{cls} = \frac{\sqrt{Op} + \sqrt{Art}}{0{,}3 * \left(Op + Art\right)}, \quad (3)$$

где S_{cls} - оценка операций и атрибутов для класса; Op - число операций у класса, Art - число атрибутов у класса.

При этом учитываются только атрибуты и операции, отображенные на диаграмме.

Далее в табл. 2 и 3 приводятся оценки для различных типов элементов и связей.

Таблица 9.2. Основные элементы языка UML

Тип элемента	Цифра элемента
Класс (class)	5
Интерфейс (interface)	4
Прецедент (use case)	2
Компонент (component)	4
Узел (node)	3
Процессор (processor)	2
Взаимодействие (interaction)	6
Пакет (package)	4
Состояние (state)	4
Примечание (node)	2

Таблица 3. Основные типы связей языка UML

Тип связи	Оценка для связи
Зависимость (dependency)	2
Ассоциация (association)	1
Агрегирование (aggregation)	2
Композиция (composition)	3
Обобщение (generalization)	3
Реализация (realization)	2

Остальные типы связей должны рассматриваться как ассоциации.

Недостатком диаграммы является как слишком низкая оценка (при этом диаграмма недостаточно информативна), так и слишком высокая оценка (при этом диаграмма обычно слишком сложна для понимания). В табл. 4 приведены диапазоны оптимальных оценок для основных типов диаграмм.

Таблица 4. Диапазоны оценок для диаграмм UML

Тип диаграммы	Диапазон оценок
Классов (class)– с атрибутами и операциями	5 – 5,5
Классов (class) – без атрибутов и операций	3 – 3,5
Компонентов (component)	3,5 – 4
Вариантов использования (use case)	2,5 – 3
Развертывание (deployment)	2 –2,5
Последовательности (sequences)	3 – 3,5
Кооперативная (cooperative)	3,5 – 4
Пакетов (package)	3,5 – 4
Состояний (state)	2,5 – 3

Далее приведен пример оценки простой диаграммы классов по данной методике.

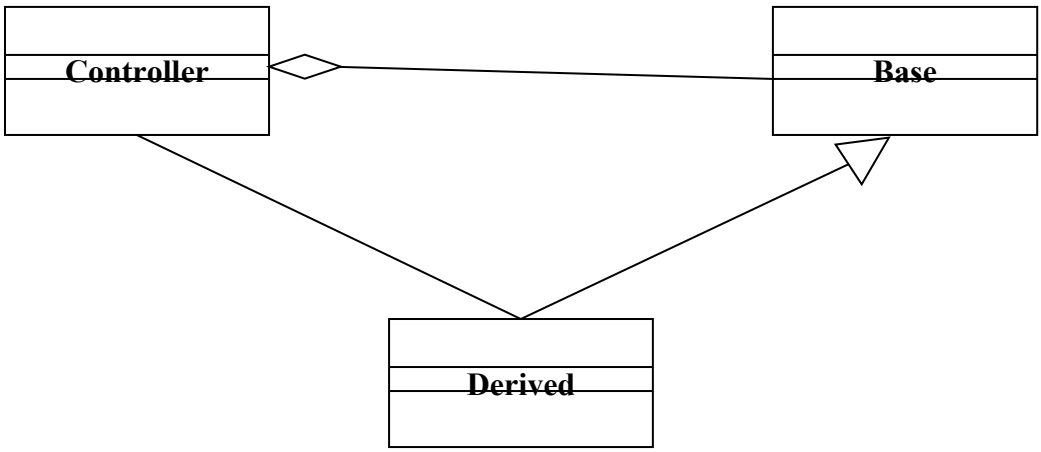


Рис. 2.

Диаграмма содержит три класса без операций и атрибутов; следовательно, $T_{Obj} = 1$, $\Sigma S_{Obj} = 15$, и Obj = 3. В качестве связей используются ассоциация, агрегирование и обобщение; следовательно, $\Sigma S_{Lnk} = 6$ и $T_{Lnk} = 3$.

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{15 + 6}{1 + 3 + 2} = 3,5$$

То есть численная оценка для данной диаграммы равна 3,5.

3. Пример

На рис. 9.3 и 9.4 приведены диаграммы классов модели подсистемы «Служба занятости в рамках вуза» системы «Дистанционное обучение». Эти диаграммы реализуют один и тот же фрагмент полсистемы «Службы занятости в рамках вуза», но первая из них более полно реализует принципы объектно-ориентированного подхода.

Найдем численную оценку для каждой из диаграмм.

Диаграмма 1

Проведем расчет оценки атрибутов и операций для классов «Работодатель», «БД студентов» и «Студент».

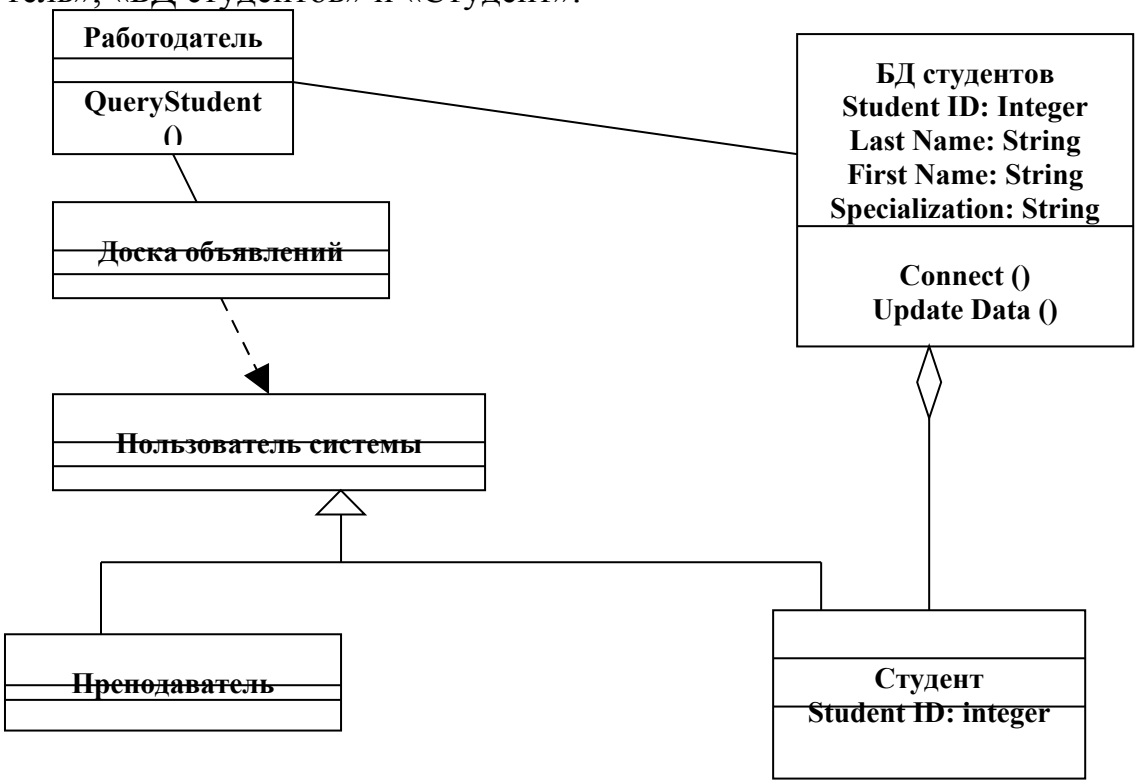


Рис. 9.3. Диаграмма 1. «Работодатель»

$$S_{cls} = \frac{\sqrt{Op} + \sqrt{Art}}{0{,}3 * (Op + Art)} = \frac{\sqrt{2} + \sqrt{0}}{0{,}3 * (2 + 0)} = 2{,}36$$

Аналогично для класса «БД студентов» получаем 2,53; для класса «Студент» - 3,33.

Рассчитаем полное значение для диаграммы:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{38{,}33 + 9}{1 + 6 + \sqrt{5}} = 5{,}11$$

Диаграмма 2

Проведем расчет оценки атрибутов и операций для классов «Деканат», «Группа» и «Пользователь системы». Для класса «Деканат» получаем 2,36; для класса «Группа» - 3,33; для класса «Пользователь системы» - 1,11.

Рассчитаем полное значение для диаграммы:

$$S = \frac{\Sigma S_{Obj} + \Sigma S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{31{,}8 + 8}{1 + 5 + 2} = 4{,}85$$

В результате оценка для диаграммы 1 попадает в середину оптимального диапазона для диаграмм классов, а оценка для диаграммы 2 оказывается ниже оптимального диапазона.

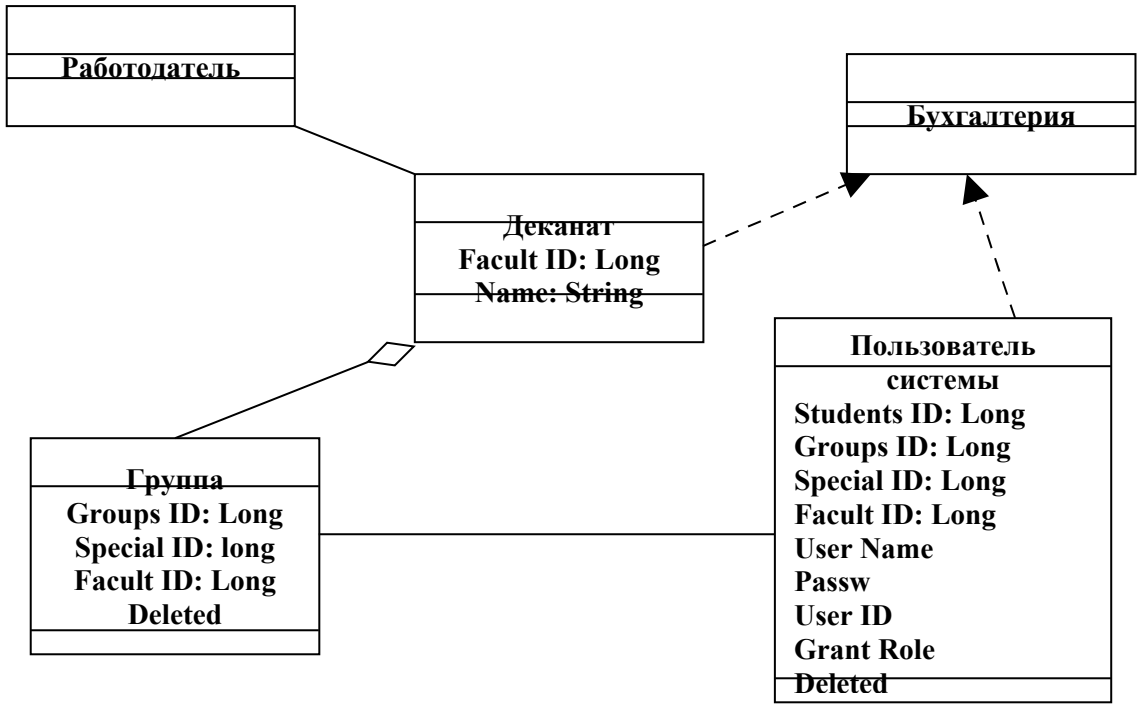


Рис. 4. Диаграмма 2.

Такой результат можно объяснить следующими причинами:

1. Диаграмма 2 содержит излишне детализированный класс «Пользователь системы», тогда как в диаграмме 1 он упрощен с помощью построения иерархии классов.
2. Класс «Деканат» на диаграмме 2 берет на себя слишком много функций, следствием чего является избыток связей.
3. Класс «Бухгалтерия» на диаграмме 2 не относится напрямую к фрагменту, смоделированному на диаграмме, т. е. усложняет модель, не внося при этом полезной информации.

Задания

1. Создать новый проект в Rational Rose.
2. Добавить в проект диаграмму классов.
3. Разместить на ней класс с произвольным именем.
4. Редактировать имя класса на диаграмме.
5. Добавить на диаграмму метку с комментарием.
6. Открыть спецификацию класса, изменить тип класса.
7. Удалить класс, используя браузер диаграмм.
8. Сохранить проект.

9. ПЕРЕЧЕНЬ ПРОГРАММНЫХ ПРОДУКТОВ, ИСПОЛЬЗУЕМЫХ В ПРАКТИКЕ ВЫПУСКНИКОВ И УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Программный продукт – пакет объектно-ориентированной технологии разработки ПО, Rational Rose корпорации Rational Software.

Сегодня лидирующей в мире CASE-системой считается Rational Rose корпорации Rational Software. Система Rational Rose нацелена на создание моделей с использованием языка Unified Modeling Language (UML). UML стал стандартным языком объектно-ориентированной разработки не без подачи Rational Software, которая не только выпускает программные продукты, где используется UML, но и активно принимает участие в организации Object Management Group (OMG), занятой созданием и обновлением спецификаций языка UML, технологии распределенных вычислений CORBA и т. д. В компании Rational работают три создателя и евангелиста объектно-ориентированной разработки и языка UML. Это Гради Буч (Grady Booch), Айвар Джекобсон (Ivar Jacobson) и Джим Рамбаух (Jim Rumbaugh).

Своеобразным признанием заслуг Rational Software стало включение усеченного варианта продукта Rational Rose в знаменитый пакет разработчика Microsoft Visual Studio. Правда, название поменялось с Rational Rose на Visual Modeler. Многие, используя Visual Modeler, так и не знают, с чем же на самом деле они работают.

Последняя версия CASE-системы компании Rational Software – Rational Rose 98 уже вовсю применяется для создания коммерческих ПО и поддерживает популярные языки программирования Java, Cu++, Смолток, Ада, Visual Basic, PowerBuilder и Forte. Кроме того, пакет Rose 98 способен генерировать описания на языках Interface Definition Language (IDL) для приложений CORBA и Data Definition Language (DDL) для приложений доступа к базам данных, в том числе и Oracle 8. Разумеется, поддержка того или иного языка программирования зависит от того, о какой редакции пакета Rational Rose 98 идет речь. К примеру, нельзя требовать многого от самого простого варианта пакета — Rose 98 Modeler Edition. Зато Rose 98 Enterprise Edition оснащен от души.

Интегрированная среда

Среда Rational Rose 98 — типичное окно в стиле Microsoft MFC со швартуемыми панелями (рис.1).

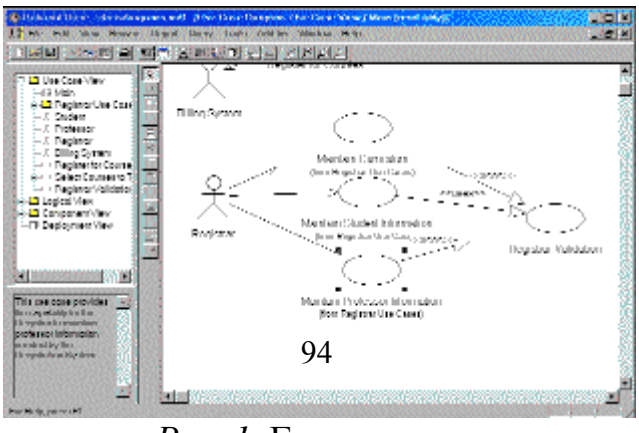


Рис. 1. Главное окно

По умолчанию программа выводит окна браузера, документации и Рабочего стола. Между этими окнами уютно расположилась палитра с элементами, из которых строятся различные диаграммы. Окно браузера отображает дерево элементов текущей открытой в Rational Rose модели. Это могут быть сценарии поведения (use cases), действующие лица (actors), пакеты (packages) и прочие диаграммы. Пользуясь перетаскиванием, можно перемещать отображаемые в браузере элементы в различные места «дерева» модели, изменяя структуру модели по своему усмотрению. Чуть ниже, под окошком браузера, представлена документация к текущему выделенному элементу. Разработчик может добавить свое описание любого объекта модели, и оно тут же появится в окне документации. Рабочий стол служит для того, чтобы «раскладывать» диаграммы. Если в окне браузера дважды щелкнуть на значке какой-либо диаграммы, то последняя будет открыта в отдельном окне, а окно в свою очередь будет помещено на Рабочий стол Rose. Там же находится постоянно свернутое окно протокола работы. Контекстное меню, вызываемое нажатием правой кнопки мыши, играет в окне браузера важную роль. Оно дублирует множество пунктов главного меню Rose, в то же время отслеживая контекст вызова и организуя пункты меню наилучшим для этого контекста образом. Умело оперируя контекстным меню, можно и вовсе обойтись без главного меню пакета, по крайней мере для вызовов всех основных операций. Стартуя, Rational Rose 98 показывает на экране диалоговую панель с заготовками типичных проектов (рис. 2).

Рис. 2. Рутинного моделирования можно избежать, если воспользоваться заготовками Rose

Как пользоваться Rational Rose.

Пользователь, впервые встретившийся с Rational Rose 98, вряд ли сразу же поймет идеологию пакета. Но что сделаешь, практически все CASE-продукты «грешат» ориентацией на специалистов. И все же стать таким специалистом, работая с Rose, не так и сложно. Тем паче, что время, затраченное на изучение пакета, в дальнейшем многократно окупится. А чтобы было понят-

нее, как работают с Rational Rose, рассмотрим небольшой пример построения диаграммы сценариев поведения для модели, которую, кстати, можно найти среди примеров, поставляемых с Rose 98. Предположим, требуется создать некую систему регистрации студентов на учебные курсы в университете.

Сценарий проекта таков. Сначала каждый профессор университета заполняет специальную форму, в которой указывает, какие учебные курсы он намерен вести в следующем семестре. Данные из формы помещаются в университетский компьютер работником регистратуры. После этого из полученных данных формируется каталог курсов, который раздается студентам. Студенты выбирают из каталога те курсы, на которых они собираются учиться, и подают заявки на обучение в регистратуру. Все эти данные также попадают в компьютер, где происходит их обработка и формирование списков курсов и студентов. В задачи создаваемой системы входит, в частности, такое комплектование учебных курсов, чтобы каждый курс посещало бы от трех до десяти студентов. Если на какой-то курс не набирается трех студентов, он отменяется. После формирования курсов профессора получают списки студентов, которых им предстоит обучать, а каждый студент получает подтверждение о зачислении на курс и счет на оплату.

Первое, что требуется при построении модели, — определить действующие лица системы и сценарии поведения. Действующих лиц в создаваемой системе четыре: профессор, студент, регистратура и биллинговая программа. Первые три выбраны действующими лицами, поскольку они активно взаимодействуют с создаваемой системой. Биллинговая же программа чаще всего является отдельным программным продуктом, а в нашем случае она получает информацию для своей работы от создаваемой курсовой системы, поэтому может считаться самостоятельным действующим лицом.

Теперь выделим сценарии использования для нашего примера. Каждый из них описывает некоторое требование к функциям системы:

- регистрировать на курс;
- выбирать курс для чтения;
- затребовать распечатку списка студентов курса;
- обрабатывать информацию о курсе;
- обрабатывать информацию о профессоре;
- обрабатывать информацию о студенте;
- создавать каталог курсов.

Теперь перенесем все это на диаграмму. Для этого нужно создать пустой проект, переключиться на папку Use Case View и открыть контекстное меню нажатием правой кнопки мыши. Если теперь выбрать пункт New•Actor (рис. 3), то вы получите действующее лицо; если же выбрать New•Use Case, то будет создан сценарий поведения. Все введенные нами действующие лица и сценарии поведения немедленно появляются в окне браузера, откуда вы можете перетаскивать их мышью на диаграммы.

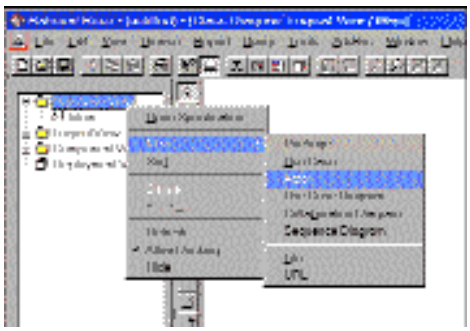


Рис. 3. Создаем новое действующее лицо модели

Далее, как правило, строится диаграмма сценариев поведения. Для этого двойным щелчком на пиктограмме Main из папки Use Case View открывается главная диаграмма сценариев. В нее из окна браузера перетаскиваются все действующие лица и сценарии поведения, которые были созданы в рамках модели. После размещения эти компоненты нужно связать между собой, чтобы отобразить взаимосвязи. В нашей модели наилучшим образом подойдут связи типа «однонаправленная ассоциация» (Unidirectional Association). Для реализации связей также применяется метод перетаскивания. Сначала в палитре выбирается тип связи «однонаправленная ассоциация», после чего нужно протянуть линию между действующим лицом и сценарием поведения. В результате на диаграмме возникнет стрелка. Аналогичным образом поступают со всеми компонентами диаграммы. Готовая диаграмма показана на рис. 4.

Примечательно, что теперь все компоненты можно перемещать по диаграмме, не прерывая связей, — Rational Rose позаботится об их корректной отрисовке.

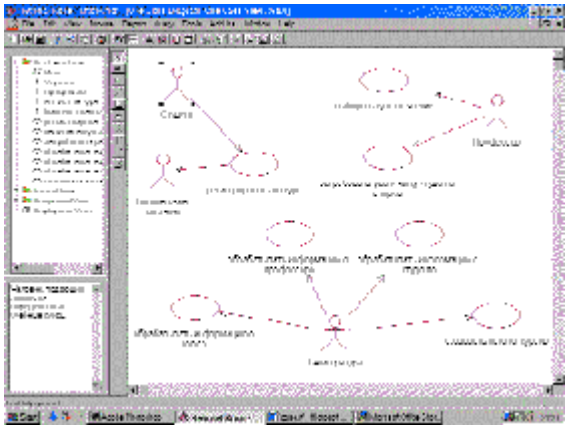


Рис. 4. Диаграмма сценариев поведения

Теперь вы можете задать дополнительные параметры для компонентов диаграммы. Нужно лишь дважды щелкнуть на выбранном объекте, после чего вы увидите диалоговую панель, которая позволяет задать текст документации для выделенного объекта и различные опции (рис. 5).

Рис. 5. Документация к объекту.

В среде Rational Rose 98 так же легко создаются и диаграммы других видов, необходимые для генерации исходных текстов на выбранном языке программирования. И для каждого объекта диаграммы существует масса тонких настроек.

Нельзя не отметить, что...

... система Rose — признанный лидер среди средств визуального моделирования, и, используя ее, вы сможете интерактивно разрабатывать архитектуру создаваемого приложения, генерировать его исходные тексты и параллельно работать над документированием разрабатываемой системы. С помощью Rational Rose можно создавать новые модели на базе обратного разбора двоичных COM-модулей или исходные тексты прикладных программ и библиотек классов.

Преимущества от применения Rational Rose 98 значительны:

- сокращение цикла разработки приложения;
- увеличение продуктивности работы программистов;
- улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;
- способность вести большие проекты и группы проектов;
- возможность повторного использования уже созданного ПО за счет упора на разбор их архитектуры и компонентов;
- язык UML служит универсальным «мостиком» между разработчиками из разных отделов.

Кроме того, в рамках специально проводимой программы RoseLink объединены более 40 компаний, поддерживающих технологию Rose и создающих собственные расширения к пакету Rational Rose. Число пользователей Rational Rose превысило 50 тыс., причем в самых различных областях. Возможно, это и не так много для программного пакета, но для CASE-систем эта цифра очень внушительная. Тем более что среди клиентов Rational такие всемирно известные компании, как NASA, Boeing, Lockheed Martin, Honeywell,

NBC, Reuters, AT&T и др., имена которых, выстроенные в столбик, заняли бы целую колонку журнальной страницы.

10. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРИМЕНЕНИЮ СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ДЛЯ ПРЕПОДАВАНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ.

Итак, Вы завершили установку пакета и ввели лицензионную информацию, как указано в сопроводительном буклете. Среда Rational Rose – типичное окно в стиле Microsoft MFC со швартуемыми панелями. В верхней части экрана, как и у большинства редакторов в стиле Windows, находится меню и строка инструментов (Tool Bar), которую можно перетащить мышкой в любое удобное место.

Слева находится окно Browser для быстрого доступа к диаграммам. Это окно позволяет быстро перемещаться по дереву диаграмм, буксировать диаграммы мышкой и изменять структуру по своему усмотрению.

Под окном Browser находится окно Documentation (документация). В этом окне появляется описание, которое введено разработчиком для выделенного в текущий момент элемента.

В правой части экрана находятся те диаграммы, которые открыты в текущий момент, обычно это поле называется рабочим столом Rational Rose.

При создании новой модели на рабочем столе открывается Class Diagram (диаграмма классов), пока эта диаграмма пуста. Однако Вы можете воспользоваться мастером для создания модели.

Между окном Browser и окном Diagram находится строка инструментов текущей диаграммы, которая изменяется в зависимости от выбранной диаграммы.

Внизу рабочего стола находится свернутое окно Log (протокол). В нем Rational Rose постоянно фиксирует все действия, произведенные над диаграммами.

Таким образом, UML – набор диаграмм, которые позволяют проектировать и создавать сложные программные системы.

В распоряжение проектировщика системы Rational Rose предоставляет следующие типы диаграмм, последовательное создание которых позволяет получить полное представление обо всей проектируемой системе и об отдельных ее компонентах:

- Use case diagram (диаграммы сценариев);
- Deployment diagram (диаграммы топологии);
- Activity diagram (диаграммы активности);
- Interaction diagram (диаграммы взаимодействия);
- Sequence diagram (диаграммы последовательностей действий);
- Collaboration diagram (диаграммы сотрудничества);
- Class diagram (диаграммы классов);
- Component diagram (диаграммы компонент).

Use case diagram.

Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто этот вид диаграмм называют диаграммой функций, потому что на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

Каждая такая диаграмма или, как обычно ее называют, каждый Use case – это описание сценария поведения, которому следуют действующие лица (Actors).

Данный тип диаграмм полезен при замысле системы и используется для того, чтобы создать требования к системе, определить действующие в системе объекты и основные задачи, выполняемые этими объектами.

Deployment diagram.

Этот вид диаграмм предназначен для анализа аппаратной части системы, то есть «железа», а не программ. В прямом переводе с английского Deployment означает «развертывание», но термин «топология» точнее отражает сущность этого типа диаграмм.

Для каждой модели создается только одна такая диаграмма, отражающая процессоры «Processor», устройства «Device» и их соединения.

State diagram.

Диаграммы состояний «State» предназначены для отображения состояний системы, имеющих сложную модель поведения.

Каждый объект системы, обладающий определенным поведением, может находиться в определенных состояниях и переходить из состояния в состояние в процессе реализации сценария поведения. Поведение большинства объектов реальных систем можно представить с точки зрения конечных автоматов, т.е. поведение объекта отражается в его состояниях, и данный тип диаграмм позволяет отобразить это графически.

Activity diagram.

Диаграмма активности, это дальнейшее развитие диаграммы состояний. Фактически этот тип диаграмм может использоваться и для отражения состояний моделируемого объекта, однако, основное назначение Activity diagram не в том, чтобы отражать состояния, а в том, чтобы отражать бизнес-процессы объекта.

Этот тип диаграмм позволяет проектировать алгоритмы поведения объектов любой сложности, в том числе может использоваться для составления блок-схем.

Interaction diagram.

Этот тип диаграмм включает в себя диаграммы Sequence diagram (диаграммы последовательностей действий) и Collaboration diagram (диаграммы сотрудничества). Эти диаграммы позволяют с разных точек зрения рассмотреть взаимодействие объектов в создаваемой системе.

Sequence diagram.

Взаимодействие объектов в системе происходит посредством приема и передачи сообщений объектами-клиентами и обработки этих сообщений объектами серверами. При этом в разных ситуациях одни и те же объекты могут выступать и в качестве клиентов и в качестве серверов.

Данный тип диаграмм позволяет отразить последовательность передачи сообщений между объектами.

Collaboration diagram.

Этот тип диаграмм позволяет описать взаимодействие объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта, и типы этих сообщений.

Class diagram.

Этот тип диаграмм позволяет создавать логическое представление системы, на основе которого создается исходный код описываемых классов.

Значки диаграмм позволяют отображать сложную иерархию систем, взаимосвязи классов и интерфейсов. Данный тип диаграмм противоположен по содержанию диаграмме Collaboration, на которой отображаются объекты системы. Rational Rose позволяет создавать классы при помощи данного типа диаграмм в различных нотациях.

Component diagram.

Этот тип диаграмм предназначен для распределения классов и объектов по компонентам при физическом проектировании системы. Часто данный тип диаграмм называют диаграммами модулей.

При проектировании больших систем может оказаться, что система должна быть разложена на несколько сотен или даже тысяч компонентов, и этот тип диаграмм позволяет не потеряться в обилие модулей и их связей.

Общий порядок работы.

Для создания систем различных предметных областей общий порядок работы может несколько отличаться от приведенного, поэтому при разработке другой системы необходимо внести в него соответствующие изменения.

В первую очередь необходимо произвести анализ списка операций, которые будет выполнять система, и определить список объектов системы, которые данные функции выполняют. Таким образом, необходимо определить

требования к системе и границы предметной области. Для этого используйте диаграммы Use case.

Затем, еще до окончательной детализации сценариев поведения, проведите анализ аппаратной части системы при помощи Deployment диаграммы. Это скорее задача системного анализа, чем практическая. В общем случае она позволит определится в таких вопросах как технологичность и стоимость системы, а также определить набор аппаратных средств, на которых предстоит эксплуатировать систему.

Анализ аппаратной части системы может ограничится перечислением устройств, которые будут работать под управлением используемого программного обеспечения. Возможно, именно аппаратные средства внесут свои коррективы, ограничения или дополнительные требования к создаваемому программному обеспечению.

Затем определяется список классов, которые должны присутствовать в системе, пока без конкретной детализации и подробного описания действий. Для этого будем использовать диаграмму классов (Class diagram).

После заведения в системе необходимых классов определим поведение конкретных классов при помощи диаграмм State diagram (диаграммы состояний) и Activity diagram (диаграммы активности).

Дальнейшая детализация взаимодействия классов будет производится при помощи Sequence diagram (диаграммы последовательностей действий), Collaboration diagram (диаграммы сотрудничества).

На основании производимых классами действий создадим окончательную иерархию классов системы при помощи диаграммы классов (Class diagram) и определим компоненты, в которые эти классы необходимо включить при помощи диаграммы компонентов (Component diagram).

Необходимо понимать, что разработка – это итерационный процесс. Нельзя за один раз создать полный проект системы. Придется многократно возвращаться к уже созданным диаграммам и вносить в них изменения.

Работа в среде состоит из написания диаграмм. После составления диаграммы идет либо согласование ее с другим типом диаграмм, либо если это Class – диаграмма, выполняется преобразование кода в тот язык программирования, который предпочитает пользователь.

Рабочее окно Rational Rose выглядит следующим образом (рис. 1):

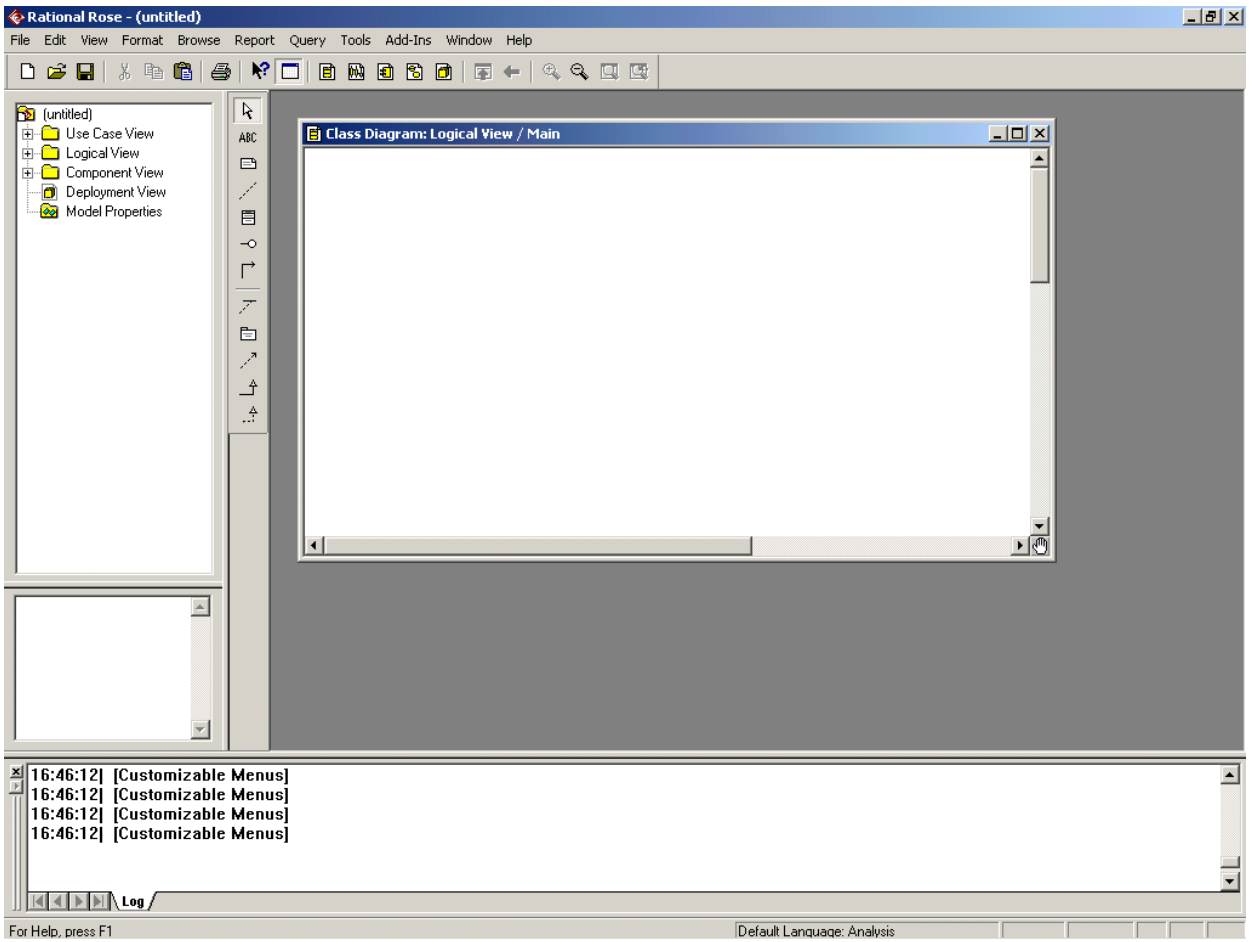


Рис.1 Главное рабочее окно Rational Rose.

С него все начинается. По умолчанию Rational Rose открывает Class-диаграмму. На панели инструментов ищем значок Class (рис.2). При нажатии на него на рабочем поле диаграммы выведется следующий рисунок (рис.3).

Это и есть заготовка класса. В верхней строке - указывается имя класса, в средней - его атрибуты, а в нижней – выполняемые им операции.

Начинаем заполнять поля класса. Выбранный нами класс характеризует какой-то некоторый товар на складе. Так и назовем класс Tovar. Для работы внутри класса (задания названия, атрибутов, операций и т.д.), нажимаем правой клавишей на изображении класса. Выбираем пункт “Open Specification…

Рис.2. Панель инструментов (повернуто).

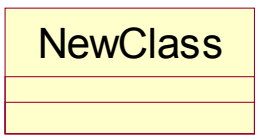


Рис.3. Изображение класса.

В результате чего появляется поле преобразования класса со множеством вкладок (рис.4).

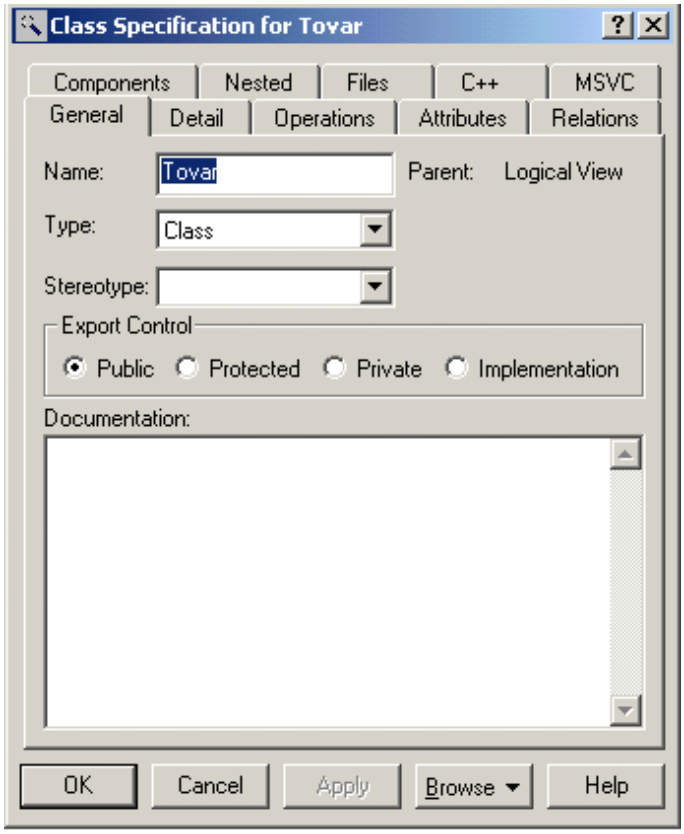


Рис.4. Окно свойств класса Tovar.

В строке Name пишем Tovar – это название класса, далее задается тип (в нашем случае это будет - Class), и отмечаем **область видимости**: либо **Public**, либо **Protected**, либо **Private,** либо Implementation. Область видимости Public, означает, что элементы внутри класса могут быть доступными вне класса, Protected – элементы доступны, только внутри данного модуля, а Private – элементы доступны не только данному классу, но и производным тоже. Это уже в зависимости от того, какой класс нужен в программе.

Следующим нашим шагом будет задание атрибутов (свойств) класса.

В кладке Attributes в самом большом поле вызываем правой клавишей мыши дополнительное меню и нажимаем insert (добавить атрибут). В результате в этом поле появиться новый атрибут с названием (по умолчанию) – name. Дважды нажав на него, выходим в свойства отдельного (данного) атрибута (рис.5).

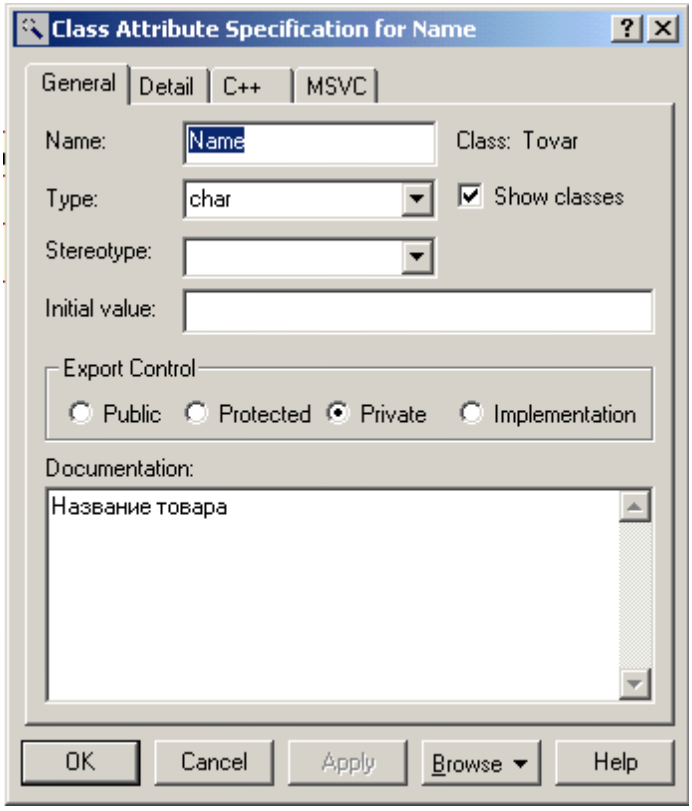


Рис.5. Окно свойств атрибута класса.

Здесь уже пишем название (например, Name, quantity, или что-нибудь другое), определяем тип: целочисленный, символьный или другой любой на выбор в сплывающем меню. Одним из достоинств Rational Rose является поле для комментариев, что намного упрощает работу в дальнейшем не только программисту, но и остальным. Самое главное, что все эти комментарии отображаются при кодогенерации, и вы уже не запутаетесь, что означает каждый атрибут или операция.

После задания атрибутов переходим к операциям. Здесь все точно так же как и во вкладке атрибутов, только помимо названия, типа и область видимости, нужно описывать так называемые детали операции (т.е. параметры функции).

Наш класс будет выглядеть примерно так (рис.6):

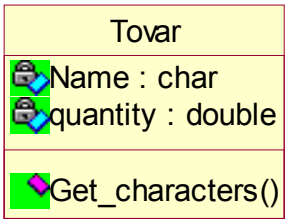


Рис.6. Изображение класса Tovar.

Как видно из рисунка в классе Tovar присутствует два атрибута – Name и quantity, и функция - Get_characters, которая выводит все характеристики данного класса.

Хотелось бы сказать, что в нашей работе мы задумали показать структуру хранения товаров на складе. Для этого мы используем три класса класс

Tovar, который мы уже описали, класс Line и класс Sklad. А теперь поподробнее об оставшихся двух классах.

Как мы уже выяснили класс Tovar, это как бы физический класс, здесь переменные хранятся непосредственно, т.е. внутри класса будут не указатели, а уже описанные внутри этого класса переменные. Этот класс характеризует сам товар.

Следующий класс – Line. Этот класс характеризует строку накладной, которая присутствует на каждом классе. Так вот Line – это одна из строк накладной в ней содержатся характеристики, какого то одного товара, который мы можем определить по имени.

Попробуем его построить.

Для этого вначале проделываем те же действия, которые мы применяли для построения предыдущего класса. Название класса – Line, атрибуты: Name и Quantity, только на этот раз у второго атрибута (Quantity), тип будет указатель, так как данные о количестве данного товара мы берем из класса Tovar, и следовательно данный атрибут является указателем на физический параметр класса Tovar.

Следующее нововведение класса Line, это функция Summ. Она подсчитывает общую сумму товара в зависимости от его количества. То есть у этой функции будут два параметра, количество (того товара, который выделит пользователь) и цена (того же товара). И в итоге мы получим общую цену на данные товары.

Следующим нашим шагом будет определение роли класса по отношению друг к другу. *Роль (**association role**)* - это неотделимая часть ассоциации, описывающая некоторые свойства её "соединения" с классом (роль класса в данной ассоциации). Естественно, что классы должны быть связаны между собой. Опять же на панели инструментов можно увидеть несколько типов связей, помимо этого можно добавить еще несколько видов связей, который просто не стоят по умолчанию. После нажатия на панели правой клавишей мыши появиться маленькое меню. На нем выбираем Customize… После чего появиться меню, в котором есть два окна: *Имеющиеся кнопки* и *Панель инструментов*. В этом то меню и можно добавить еще дополнительные функции, которые не содержит панель инструментов.

Для связи представленных двух классов мы использовали агрегирование (***aggregation*** - показывает, что ассоциация является отношением типа целое/часть). Далее определим свойства роли. Например, множественность. *Множественность **(multiplicity)*** - количество представителей (конкретных объектов), которые могут быть связаны с одним партнером ассоциации.

Нажимаем правой клавишей мыши на составленную нами связь ищем слово ***multiplicity,*** и выбираем для обоих классов свое. В данном случае нажимаем на цифру – 1, как для первого, так и для второго класса. Следовательно, отношение классов получится один к одному. На нашем примере это значит, что каждой строчке накладной (класс - Line), соответствует один товар из класса – Tovar. Ведь мы же не можем для одной строки накладной, имеющей одно название товара задать, их несколько. Но можно поставить отношение один

ко многим или другие, но об этом попозже. Итак, мы установили отношение классов, и теперь они связаны. Теперь наша диаграмма приобрела следующий вид (рис.7):

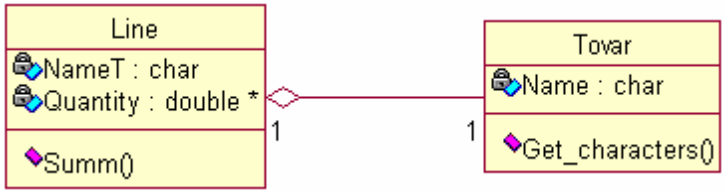


Рис.7. Связь между двумя классами.

Следующий класс – Sklad. Здесь будет присутствовать обращение к классу Line, поэтому в атрибутах укажем line1, который является указателем на параметры предыдущего класса. Таким образом, получается, что в классе склад будут присутствовать все данные предыдущих классов. Для их просмотра введем функцию Get1, которая непосредственно будет обращаться к классу Tovar, и выводить все его характеристики, а из класса Line, будем получать сумму цен товара относительно количества.

Теперь зададим отношение между классами Line и Sklad. Так как в одном классе Sklad, мы будем получать сведения о любых товарах, то есть у нас будет доступ к любому параметру, следовательно, проводим подобные действия, что и в первый раз, только ставим отношение один ко многим. У класса Sklad, ставим единицу, а класса Line – звездочку.

И в результате мы получили готовую диаграмму (рис. 8):

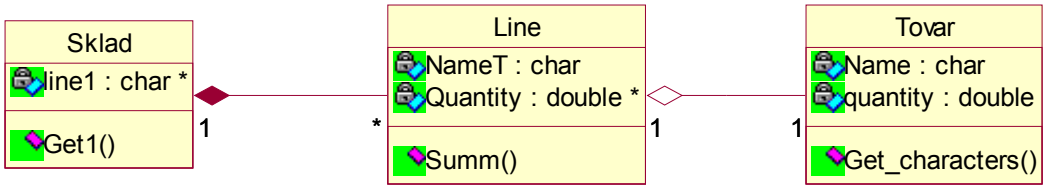


Рис.8. Полная диаграмма.

Если вы хотите, чтобы на диаграммах выводилось минимум информации, ну, например, только имя класса, то Rational Rose, позволяет сделать следующее. Предварительно выделив либо всю диаграмму, либо один класс, зайдите в Menu=>Format=>***, вместо трех звездочек в меню идут следующие команды:

- Show Visibility – позволяет убрать маленькие иконки, находящиеся внутри каждой диаграммы вначале перед каждым атрибутом или операцией. Эта иконка показывает, к какой из четырех представленных областей видимости относится данный атрибут или класс.
- Show Operation Signature – показывает все параметры функции (внутри скобок) и тип операции.
- Show All Attributes – позволяет показывать\скрывать все атрибуты в классах (причем, когда скрыть все атрибута, сама строка с ними остается пустой).

- Show All Operations – выполняет те же действия, что и предыдущая команда, только по отношению к операциям.
- Suppress Attributes – полностью убирает строку с атрибутами из диаграммы класса.
- Suppress Operations – такая же команда, что и предыдущая, только по отношению к операциям.
- Layout Diagram – автоматически выстраивает диаграммы.
- Autosize All – автоматическая размерность.

Таким образом, можно получить, например сокращенный автоматически выстроенный вид диаграммы, (рис. 9):

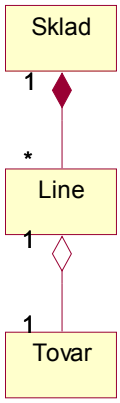


Рис.9. Сокращенный автоматически выстроенный вид диаграммы.

Так же в меню есть еще некоторые команды, с которыми можно разобраться самостоятельно.

А теперь отметим главное достоинство Rational Rose. Это конечно перевод диаграмм в код программного языка. Причем Rational Rose позволяет проводить эту операция со множеством типов языков программирования. Мы же начнем с языка C++, потому что он ближе нам больше всего. Итак, для начала выделим часть диаграммы, то есть один из классов, чтобы для простоты рассмотреть все на одном примере. После этого переходим в меню: Menu=>Tools=>Java (C++) (или любой другой язык) =>Code Generation. После этого происходит компиляция и выдается окно статуса (Code Generation Status). Здесь можно увидеть информацию о том, какой класс был закодирован и количество ошибок и предупреждений. Если у вас произошла, какая либо ошибка или же предупреждение, то их можно увидеть на рабочем поле в Rational Rose, для этого и существует самое нижнее окно, в нем передаются все ваши действия и ошибки, произошедшие в ходе кодогенерации. В результате кодогенерации Rational Rose создает файлы с расширением “.java”, названия у них те же, что и название класса. Итак, выполнив эти действия, нажимаем правой клавишей на класс, появляется окошко, в нем ищем “Java”. Эти файлы открываются с помощью блокнота и теперь легко можно увидеть скелет класса, с различными комментариями, которые писали вы на диаграммах, и комментарии которые вставляет сама Rose. Ниже мы приведем содержание этих файлов, просто для того, чтобы иметь представление о том, как

это выглядит. Теперь можно открыть один из файлов в Java и доработать класс, описать работу функций, добавить различные нововведения.

Можно выделить всю диаграмму и сделать кодогенерацию, тогда у каждого класса появиться свои файлы.

Одно из неоспоримых преимуществ Rational Rose – *обратное проектирование*, поскольку разработчику и проектировщику важно увидеть перед изменениями уже работающую систему в нормальном графическом представлении. Проект, подвергшийся обратному проектированию, может быть доработан и вновь сгенерирован (а впоследствии и скомпилирован).

Для осуществления обратного проектирования в Rational Rose предусмотрен мощный модуль – Analyzer, чье основное предназначение, вытекающее из названия, анализ программ, написанных на Java, C и C++. Данный модуль способен проанализировать имеющийся файл на одном из языков программирования и преобразовать его в визуальную модель, присвоив выходному файлу расширение .mdl. Далее файл можно спокойно открыть для модификации из Rational Rose уже в визуальном режиме.

Analyzer представляет собой отдельный программный файл, вызываемый как из самой Rose, так и обычным способом. Модуль входит не во все поставки Rational Rose, а только в Enterprise, Professional и RealTime. Для правильного преобразования кода в модель необходимо провести несколько настроек, о которых сейчас пойдет речь.

На рисунке 10 показан внешний вид программы в стандартных настройках и с не загроможденным экраном.

Основные поля, подлежащие обязательному заполнению (на первом этапе), это:

- *Caption* - Имя проекта. В последствии имя модели будет определено по имени проекта.
- *Directories* - Путь к исходящей директории. По умолчанию Rose использует для хранения исходящих модельных файлов директорию Java\Source(C++\Source) из домашней директории, что в некоторых случаях может приносить некоторые неудобства.
- *Extensions* - Типы используемых расширений. Здесь можно настроить систему так, чтобы она распознавала только определенные виды расширений.
- *Bases* - Место сохранения текущего проекта.
- *Files* - Список из файлов, подлежащих генерации.

Для проведения правильного реинженирингa (Реинжениринг модели это процесс исследования программного кода для извлечения информации о его структуре) Analyzer извлекает информацию из Java, C++ кодов и использует ее для построения модели программного кода. необходимо заполнить вышеописанные поля. Все файлы, подлежащие реинженирингу, указываются в поле «Files». Следует учитывать, что при этом вы получаете визуальную модель взаимодействия классов и структур, стало быть не идет ни какой речи о том, чтобы на визуальной модели отразился существующий код системы. Далее: все нестандартные конструкции не будут выведены в модель (анализа-

тор их просто проигнорирует), это значит, что любое отклонение от заранее известных конструкций приводит к тому, что в изначальном варианте Rose не сможет правильно проанализировать код. Этот факт не является недостатком, поскольку в арсенале Analyser'а есть инструменты тонкой настройки, позволяющие настроить все таким образом, чтобы специфика конкретного проекта была бы полностью учтена.

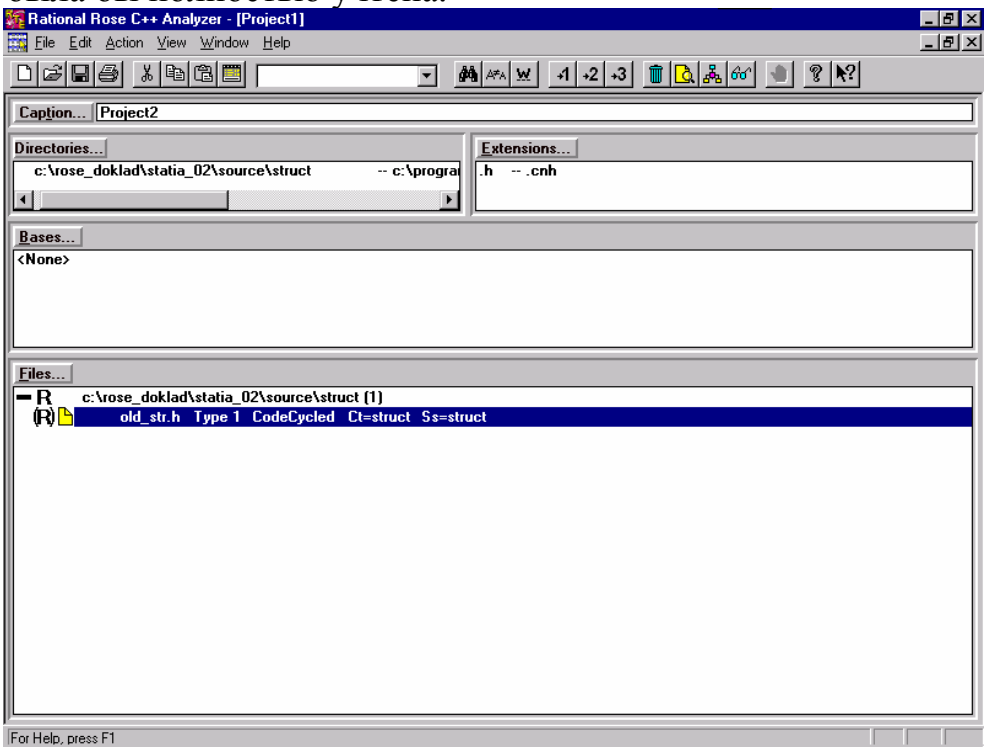


Рис. 10. Рабочее окно Analyzer-а.

Процесс реинжиниринга делится на два этапа: анализ и генерация модели.

На первом этапе производятся все подготовительные операции по анализу текста программы на отсутствие синтаксических ошибок. Второй этап – это преобразование кода в модель.

Все операции выполняются независимо, что дает большой маневр для разработчика, который, например, хочет провести только синтаксический разбор теста, без генерации модели.

Соответственно, при отсутствии ошибок в файле, можно приступить к генерации модели. В целях оптимизации времени генерации в Rose предусмотрено три способа проведения реинжиниринга, каждый из которых может охватить и превосходно выполнить определенный сегмент работ. Если пользователю по каким либо причинам не подходит ни один из трех предустановленных способов, то Rose допускает создание собственного реверсинжиниринга.

Три стандартных способа:

- *FirstLook* - приближенная пробежка по «телу» программы.
- *DetailedAnalysis* - детальный анализ проекта.

- *RoundTrip* - комбинация двух вышеперечисленных способов. Позволяет безболезненно строить и перестраивать разрабатываемые приложения по принципу круговой разработки.

Все настройки могут быть изменены пользователем по усмотрению. При сохранении изменений, возможно, указать новое имя шаблона или перезаписать уже существующее, что позволит при частом использовании обратного проектирования не терять времени на установку нужного пункта. Выбор соответствующего пункта обязательно сказывается на скорости анализа, чем больше, тем дольше. Еще хочется отметить такую особенность модуля Analyzer: после анализа создается не только модель, но и лог-файл с сообщениями, возникшими в результате сканирования программы. Лог может содержать как предупреждения, так и ошибки. А особенность генерации модели состоит в том, что она состоится несмотря ни на что, то есть, невзирая на ошибки в тексте программы. Естественно, никакой речи нет о какой-либо правильной модели! Эту особенность следует учитывать, и внимательно анализировать файл отчета после генерации модели.

Еще одна немаловажная ремарка. Как правило, обратному проектированию подвергается полноценный проект. И, естественно, разработчику хочется иметь такой инструмент, который адекватно будет реагировать на все составляющие. Модуль Analyzer в режиме (DetailedAnalysis) обеспечивает следующее:

- Анализ и преобразование в визуальную модель классов и структур
- Генерацию связей в модели (между классами или структурами)
- Нахождение в исходном тексте комментариев и перенос их в качестве атрибутов компонентов модели. То есть, если исходный текст снабжен комментариями, то они все перейдут в виде атрибутов соответствующему элементу (переменной, массиву… и т.д.).
- Способен закачать в проект все заголовочные файлы (по цепочке один за другим)

Итак, теперь мы можем дописать, например нами сгенерированный файл и перевести его с помощью Analayzer’а в диаграмму. Это облегчает работу и намного сокращает время ее выполнения.

Создание кода класса Visual C++

Возможности создания кода класса.

Класс в Rational Rose — это описание общей структуры (данных и связей) для дальнейшего создания объектов. Для того чтобы Rational Rose имел возможность создавать на основе описанной модели программный код, для каждого класса необходимо указать язык, для которого будет создаваться код. Если в качестве языка для создания кода указан VC++, то мы получаем доступ ко всей иерархии классов библиотеки MFC при помощи визуальных средств Model Assistant.

При создании класса необходимо указать стереотип, который влияет на получаемый исходный код класса. Так, при Изменении стереотипа на struct или union, будут созданы указанные типы данных.

Как было описано выше, Rational Rose поддерживает обычные для классов C++ обозначения области видимости, такие как public, private, protected. Таким образом, каждый атрибут или операция в спецификации классов при создании заголовочного файла класса будут определены в одну из секций public, private, или protected. Также имеется возможность не создавать программный код для определенных классов.

Структура создаваемого кода класса.

Для каждого создаваемого класса Rational Rose создает следующую структуру кода:

- директивы #include, которые создаются исходя из необходимости включения атрибутов и связей классов;
- декларация класса, имя класса, тип, наследование;
- переменные Data members, которые создаются по описанию атрибутов класса и его связей;
- декларация методов класса и скелет этих методов для дальнейшего наполнения каждой операции, заданной в описании класса;
- документация для каждого создаваемого класса, переменных, методов, заданная в описании модели;
- идентификатор ID модели, который включается в код как комментарий для каждого создаваемого класса, атрибута или метода, заданных в текущей модели. Например, //##ModelID=3237F8CE0053.

Ассоциация класса с языком VC++

Для того чтобы использовать класс в программном проекте, необходимо его ассоциировать с выбранным языком, в нашем случае с VC++. Для этого проделаем следующее Menu:Tools=>Visual C++=>Component Assigned Tools. Получаем окно, показанное на рис.11.

В появившемся окне выбираемы класс и перетаскиваем его на значок VC++. На вопрос, желаем ли мы создать VC++ компонент и ассоциировать его с классом, отвечаем Yes и попадаем в окно выбора проекта VC++ (рис.12). Здесь можно создать проект или выбрать из уже имеющихся для помещения в него нового класса. Нажмите Add и OK. У нас уже создан проект **sklad**.

Вы у себя можете создать проект с таким же именем при помощи MFC App Wizard exe (мастер создания исполняемого приложения) с типом Single document (однооконный документ).

По большому счету, класс может быть ассоциирован с любым языком, поддерживаемым генератором кода Rational Rose, и в результате ассоциации будет создан код программы на том языке, с которым ассоциирован класс, причем от конкретного языка зависят некоторые свойства класса.

Поэтому неплохой идеей будет изначальная ассоциация классов с определенным языком программирования.

Для того чтобы вновь создаваемые классы сразу ассоциировались с VC++, нужно проделать Menu:Tool=>Options=>Notation=>Default language=VC++

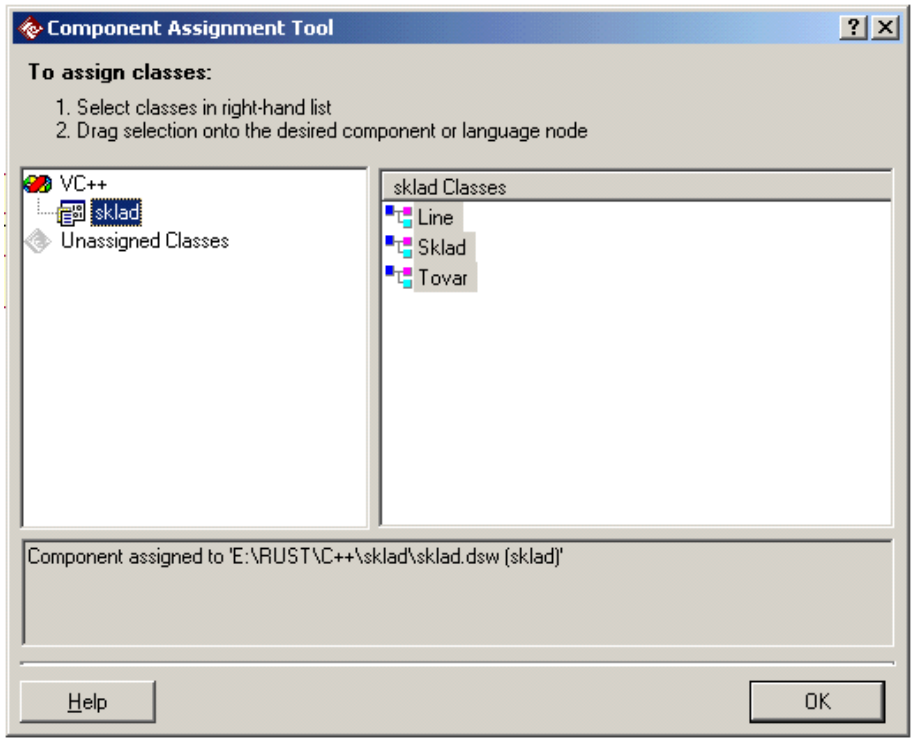


Рис.11. Назначение класса в VC++ проект.

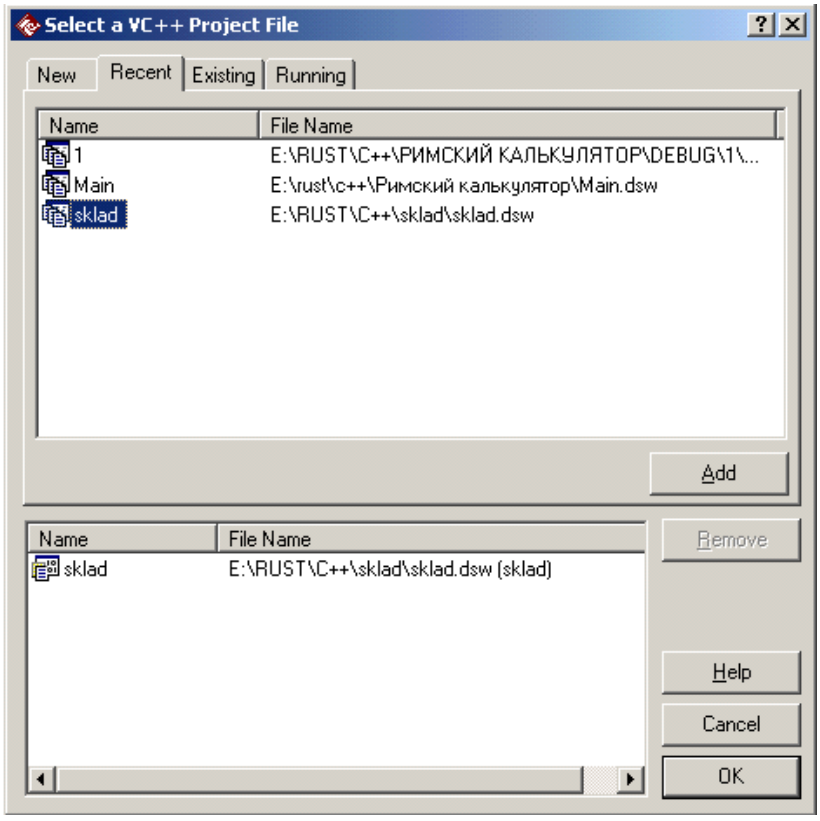


Рис.12. Выбор проекта.

Меню инструментов Visual C++

После ассоциации класса с языком программирования можно воспользоваться пунктом главного меню Tools для Visual C++, показанном на рис.13.

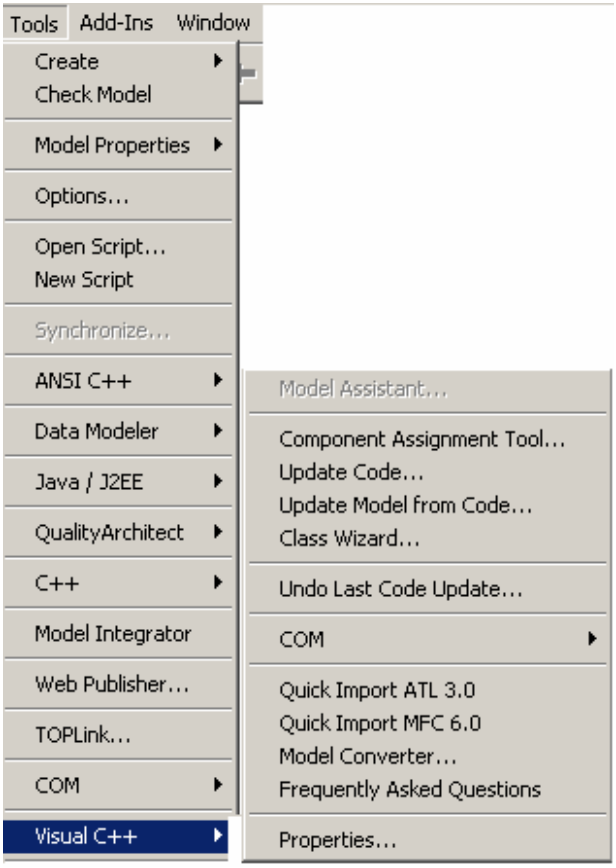


Рис.13. Меню инструментов Visual C++.

Model Assistant.

Model Assistant позволяет обновлять и конкретизировать классы в модели, используя дополнительные ключевые слова C++ для необходимой генерации кода. Model Assistant представляет собой окно, позволяющее создавать атрибуты и операции и изменять их свойства (рис.14), причем значительно проще и нагляднее, чем это происходит для C++.

Те свойства, которые нельзя изменить, подсвечены серым цветом

В этом окне следующие поля:

• Preview (предварительный просмотр) показывает описание класса так, как оно определено в текущий момент;

• Code Name (имя программы) показывает имя программного файла для данного класса;

• Generate Code (создавать исходный текст) — ключ, определяющий, необходимо ли создавать для данного класса исходный текст на языке VC++. Если ключ снят, то генерация кода не будет происходить, и этот класс не будет показываться в списке классов для обновления кода в окне Code Update (обновление исходного текста);

• Class Type (тип класса) позволяет установить тип класса, такой как «class», «struct» или «union»;

• Documentation (документация) позволяет задавать произвольные комментарии для класса. Если это поле заполнено, то при генерации исходного кода текст из него будет включен в программу как комментарии. Это очень

удобно при просмотре программистом кода, созданного при помощи генератора Rational Rose. По крайней мере, данная возможность позволяет создавать документацию к программе непосредственно в момент создания класса.

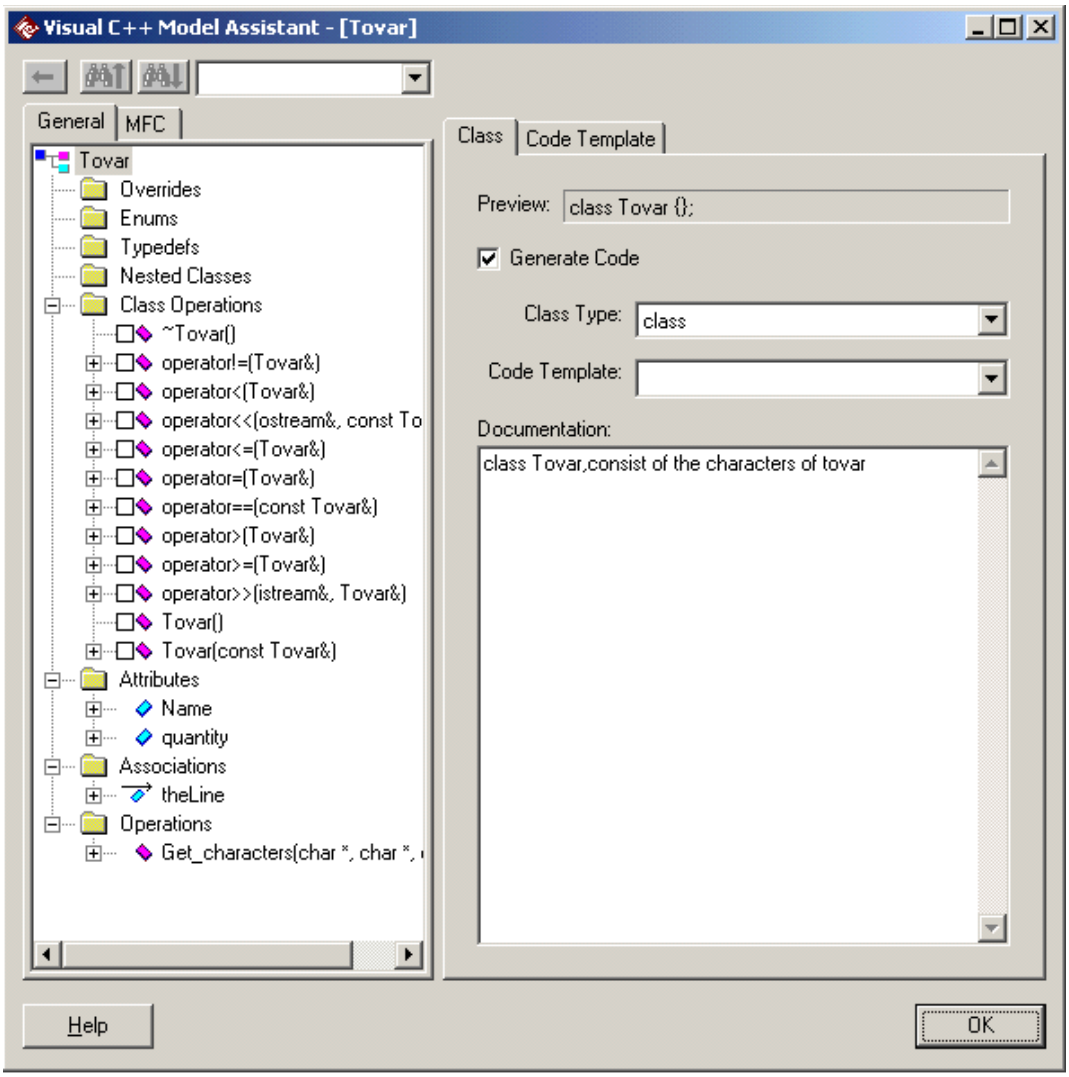


Рис.14. Model Assistant для класса Tovar.

Rational Rose позволяет создавать комментарии еще на этапе проектирования, пока не написано ни строки кода, что довольно удобно, потому что именно в момент проектирования класса еще не забыты те цели и ограничения, с которыми создается проектируемый класс или метод.

Значительные возможности предоставляет Rational Rose по интерактивной установке свойств методов класса. Рассмотрим свойства операции Get_characters, для чего активизируем строку Get_characters в окне Model Assistant. При этом программа активизирует диалоговое окно, показанное на рис.15. Это окно позволяет устанавливать и изменять атрибуты для операции, а также перегружать определенные в классе операции.

Для того чтобы добавить атрибут или операцию, не выходя из Model Assistant воспользуйтесь пунктом Insert контекстного меню.

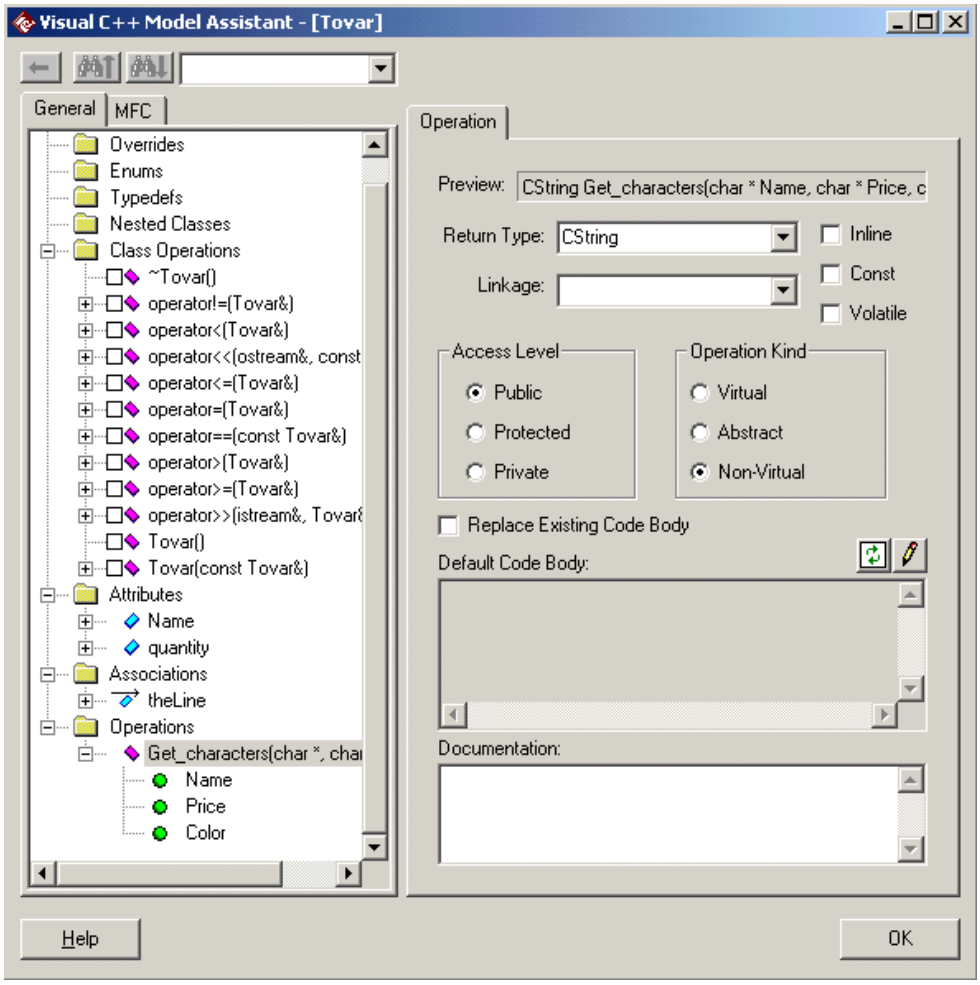


Рис.15. Свойства операции Get_characters.

Назначение полей в окне следующее:

• Preview (предварительный просмотр) показывает описание операции, таким образом, как оно было определено в текущий момент;

• Code Name (наименование кода) показывает имя кода для операции. Может быть скрыто, если такое имя не задано;

• Return Type (тип) позволяет выбрать из списка тип, возвращаемый операцией;

• Linkage (присоединение) позволяет установить тип операции, который может быть Friend или Static:

1. Static обозначает, что к данной операции можно обращаться еще до создания объекта класса;

2. Friend определяет, что данная функция хоть и не является членом класса, но имеет доступ к его защищенным и собственным компонентам. Таким образом, определяя операцию как дружественную, мы тем самым удаляем ее из методов класса и подразумеваем, что данная функция будет описана вне класса.

• Inline позволяет указать в операции ключевое слово inline, то есть операция будет создана как inline подстановка. В этом случае компилятор при создании объектного кода будет стараться подставить в текст программы

код операторов ее тела. Таким образом, при многократных вызовах подставляемой функции размеры программы могут увеличиваться, однако исключатся затраты на передачу управления вызываемой функции и возвраты из нее.

- Const определяет тип операции как Const.
- Access Level (уровень доступа) указывает доступ к операции и может быть Public, Protected или Private;
- Operation Kind — тип-операции Virtual, Abstract, или Non-Virtual. К механизму виртуальных функций обращаются в тех случаях, когда необходимо в базовый класс поместить функцию, которая должна по-разному выполняться в производных классах. Например, базовый класс может описывать фигуру на экране без конкретизации ее вида, а производные классы уже описывать реализацию конкретных треугольников, эллипсов, квадратов и т.д. При этом класс, который содержит хотя бы одну виртуальную функцию, называется абстрактным. В данном случае нет разницы между установкой пункта Virtual или Abstract. И в том, и в другом случае будет создана функция с ключевым словом virtual, которая потребует переопределения в производных классах или как минимум создания дочерних классов для класса, имеющего виртуальную функцию.

Если щелкнуть по атрибуту класса, то открывается окно для редактирования свойств этих атрибутов, показанное на рис.16, в котором можно установить основные атрибуты класса, не выходя из Model Assistant.

В данном окне имеются следующие поля:

- Preview (предварительный просмотр) показывает описание атрибута, таким образом, как оно было определено в текущий момент;
- Type (тип) позволяет выбрать из списка тип атрибута;
- Initial Value (инициализация) позволяет устанавливать значение для инициализации атрибута;
- Access Level (уровень доступа) указывает доступ к атрибуту и определяет секцию, в которой будет создан атрибут: Public, Protected или Private;
- Static обозначает, что атрибут является статичным (общим для всех объектов класса);
- Derived обозначает, что данный атрибут является производным;
- Documentation позволяет редактировать описание атрибута.

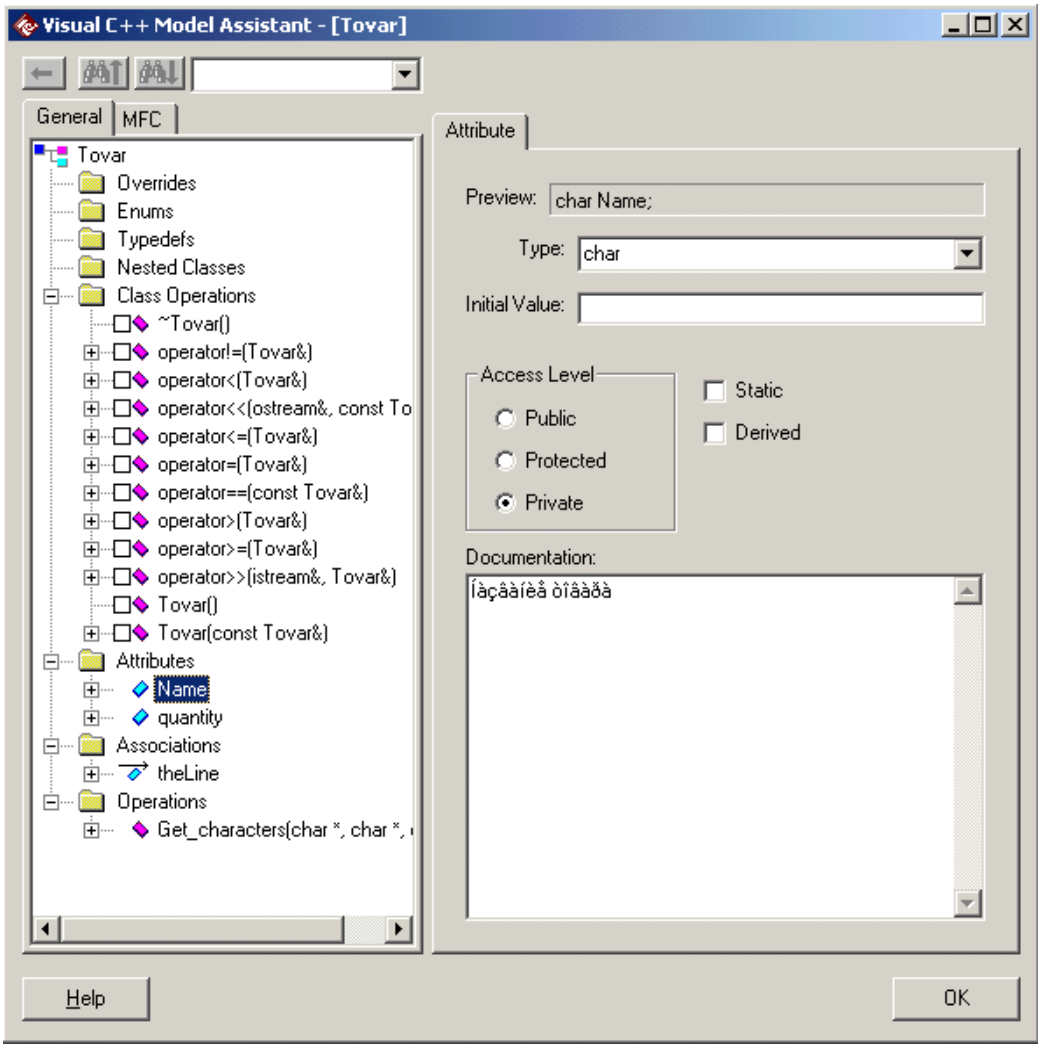


Рис.16. Редактирование атрибутов класса при помощи Model Assistant.

В этом окне есть вкладка MFC, которая предназначена для классов, наследуемых из базовых классов MFC.

Model Assistant — основное окно для работы со свойствами класса и для создания необходимого кода, поэтому к нему также есть доступ из контекстного меню класса.

Component Assignment Tool

Активизирует диалоговое окно назначения классов в компоненты и назначения языка для класса (рис. 14). Это окно предоставляет возможность создания новых компонентов в модели, ассоциации компонентов с проектами на конкретных языках программирования и назначения классов в компоненты. Для того чтобы получить преимущества использования данного инструмента, необходимо создавать компоненты здесь, а не через окно Browser или в диаграмме компонентов. При этом созданные компоненты будут содержать всю необходимую информацию для генерации кода на выбранном языке программирования. Данное средство позволяет просмотреть классы, которые еще не назначены в компоненты, что уменьшает вероятность ошибки.

Component Assignment Tool может быть открыт как посредством меню Tools, так и из контекстного меню компонента в окне Code Update Tool.

Update Code/Update Model (обновить код/модель)

Это возможность создать проект Visual C++ по разработанной модели и обновить модель по уже готовому проекту, созданному при помощи MFC. Есть возможность не просто закачать уже готовый код в программу Rational Rose, как это предусмотрено пунктом меню C++ Reverse Engineering, а поддерживать обмен постоянно, то есть работать именно в том средстве, которое позволяет наиболее быстро получить необходимый результат.

Например, необходимо быстро подправить что-либо в уже готовой, работающей программе, причем, как всегда бывает, это необходимо было сделать «еще вчера». Программист быстро дописывает что-то в исходном коде, добавляет или изменяет методы и атрибуты и быстро сдает работающую программу. Затем просто выбирает пункт Update Model from code, и эти изменения тут же попадают на рабочий стол Rational Rose. Теперь вся программа составляет одну обозримую модель.

Допустим, что у нас уже создан проект sklad и мы знаем, что после того как последний раз изменялась модель, исходный код класса Tovar был исправлен, и его необходимо обновить.

Выберем пункт Updave Model from code. Причем в контекстном меню класса, ассоциированного с VC++, также есть этот пункт, только он называется Update Model, что, впрочем, не имеет никакого значения, так как действия, выполняемые этими пунктами, одинаковы. После выбора появится окно с описанием дальнейших действий, его можно погасить, выбрав кнопку Next (дальше), и, к тому же, еще предотвратить его назойливое появление в дальнейшем путем установки флажка в поле Don't show this page in the future (больше не показывать эту страницу). После этого появится окно, показанное на рис.17.

Здесь можно обновить как все классы модели, так и отдельные классы, при помощи установки и снятия отметок с определенных классов.

Если классы модели еще не ассоциированы ни с одним проектом VC++, то при помощи кнопки Add Component (добавить компонент) это можно сделать прямо из данного окна.

Некоторые классы проекта могут иметь ошибки или недостаток данных при создании их в Rational Rose с последующей генерацией кода или же при переносе данных из готового кода в модель Rational Rose. Такие классы отмечаются знаком вопроса в ярко желтом кружочке. Если выбрать такой знак вопроса, то появится сообщение Rational Rose, которое указывает на возникшую проблему или ошибку в классе, и предлагает методы ее устранения.

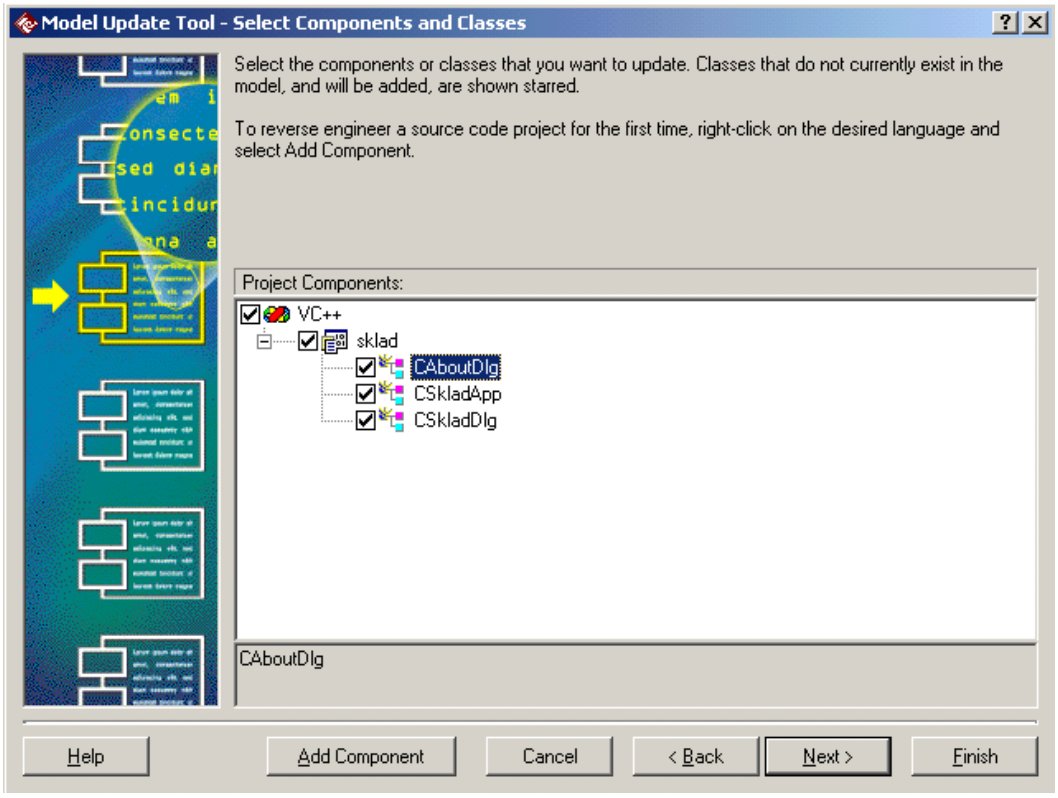


Рис.17. Окно Update Model Tool.

Затем Rational Rose получает информацию из проекта Visual C++, для этого загружается Microsoft Visual Studio и активизируется нужный проект Visual C++. После того как обмен произошел, может быть активизировано окно удаления компонентов. Если, например, мы проводили эксперименты с полученным кодом, а затем удалили некоторые классы, операции или атрибуты, то программа отследит, что элементы существуют в модели Rational Rose, но никак не отражены в исходном коде. Программа считает, что эти компоненты были удалены из исходного кода и предлагает их удалить и из модели.

Внимательно отнеситесь к этому вопросу. Возможно, в проект Visual C++ были добавлены классы, например, при помощи Class Wizard, и они еще не отражены в модели Rational Rose. В этом случает необходимо провести обновление кода при помощи функции Update Code (обновить код). Можно ничего не удалять, если не будем устанавливать флажки на предложенных компонентах.

После завершения обмена программой будет представлен отчет о том, как прошло обновление. В окне имеется две вкладки:
Summary – краткая общая информация и *Log* – полный отчет об обновленных классах (рис.18). Если все прошло нормально, то ошибок и предупреждений быть не должно, как показано на рисунке. Фатальная ошибка будет возникать, если на компьютере установлен пакет Visual Studio 5, а не 6, с которым работает Rational Rose 98i-2000.

При этом будет возникать ошибка обновления, и процесс завершится неудачно.

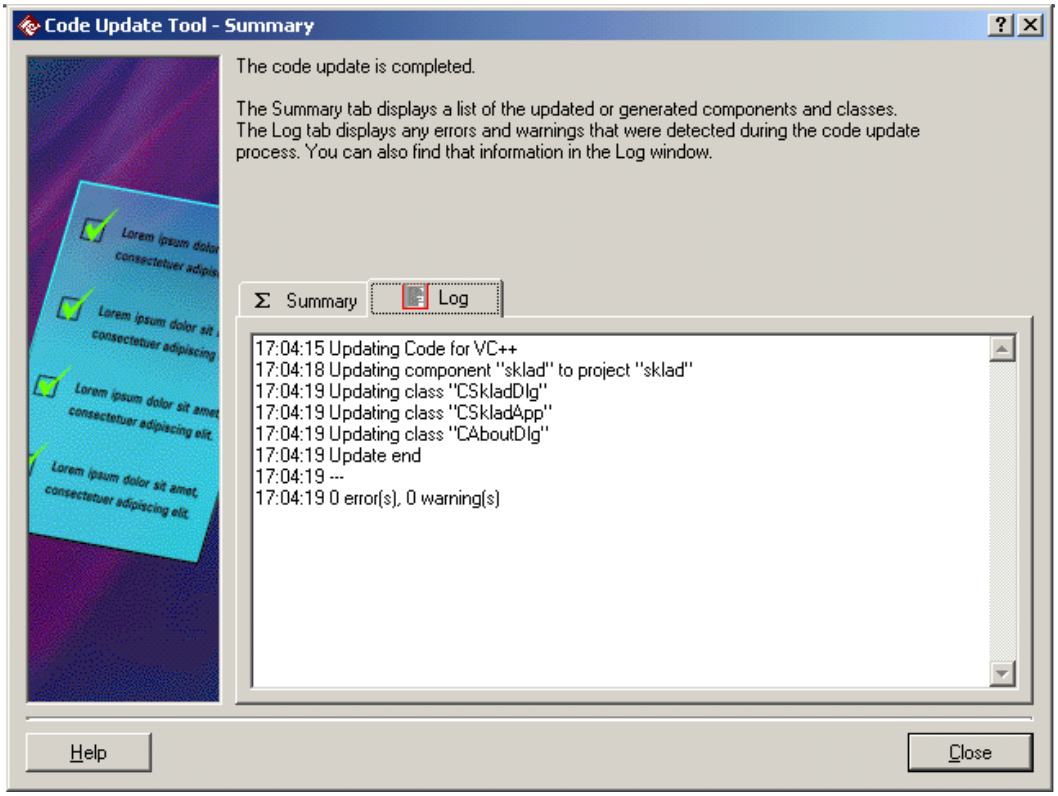


Рис.18. Отчет о проведенных обновлениях компонентов.

При этом будет возникать ошибка обновления, и процесс завершится неудачно.

Процесс обновления кода по изменениям в модели происходит аналогично. Причем Rational Rose позволяет выбрать конкретные классы, которые необходимо обновить.

Class Wizard (мастер создания класса)

Class Wizard помогает создавать новые классы. Как и большинство мастеров, он посредством последовательно активизируемых окон ведет пользователя к созданию класса с необходимой информацией. Мастер предоставляет возможность устанавливать три различных варианта создания классов:

1. Создание нового, пустого класса.
2. Создание подкласса уже созданного класса.
3. Создание нового класса по шаблону уже существующего.

Undo Last Code Update (отмена последнего обновления)

Пункт Undo Last Code Update позволяет активизировать диалоговое окно, где вы можете заменить полученный исходный код на его предыдущую версию.

Причем, мы можем отменить только самое последнее обновление кода.

COM

Активизирует меню установок объектов COM. Мы не будем рассматривать это меню по причине того, что в нашей системе таких объектов нет.

Quick Import MFC 6.0

Позволяет импортировать классы библиотеки классов MFC в текущую модель, для того чтобы ими можно было воспользоваться при создании иерархии приложения.

Properties.

Позволяет устанавливать свойства генератора Visual C++, которые влияют на создаваемый код. Мы не будем на них подробно останавливаться по причине того, что для нашей задачи достаточно установок по умолчанию.

Еще одна немаловажная возможность в Rational Rose, это моделирование генерация шаблонов класса. Ниже представлен пример иллюстрирующий данную возможность.

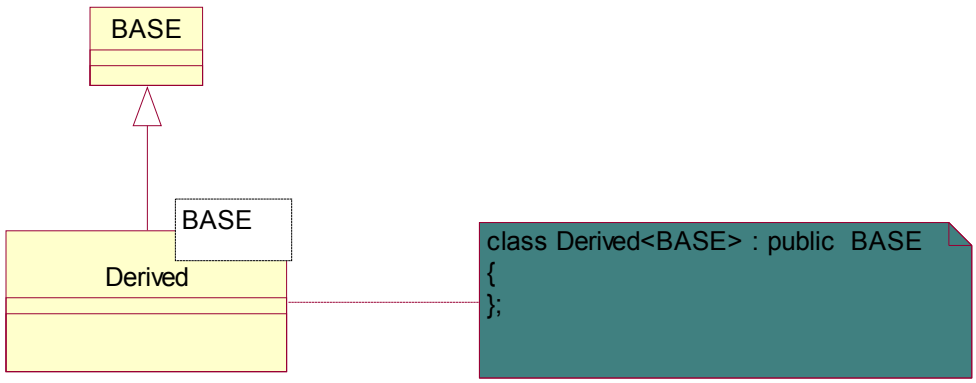


Рис.19. Определение класса Шаблона.

Ограничения в Rose Visual C++

Rose Visual C++ не поддерживает namespaces, вложенные классы (или structs – структуры), и typedefs. Нет никакой поддержки для производства их от модели, и не будет выведено никакого предупреждения, если они встречаются в коде. Эти конструкции могут присутствовать в коде и при этом не будут нарушены генерацией объектного кода. Они будут обработаны подобно дополнительным пользовательским комментариям – в конце концов, просто игнорируются. Однако код, который содержит namespaces, может все еще причинять проблемы, поскольку Rose, Visual C++ не может различать тождественно названные классы, которые присутствуют в отдельном namespaces. Это свойство перепроектирует или модифицирует первый, с которым сталкивается.

Rose Visual C++ не использует полный C ++ синтаксический анализатор, чтобы анализировать и извлекать семантику от кода. Вместо этого, Rose использует многие из тех же самых интерфейсов, которые использует Visual C ++ ClassView. В настоящее время, C ++ директивы препроцессора полно-

стью не поддержаны. Таким образом, условные инструкции трансляции и #defines игнорируются. Условно исключенный код, который иначе игнорировался бы транслятором, будет казаться видимым для Rose Visual C++. Таким образом, следующее макроопределение и использование:

```
#define CLASSDECL(X, Y) \
class X : public Y  \
{                           \
        Y* parent;          \
};

CLASSDECL(Cfoo, Cbase)
```

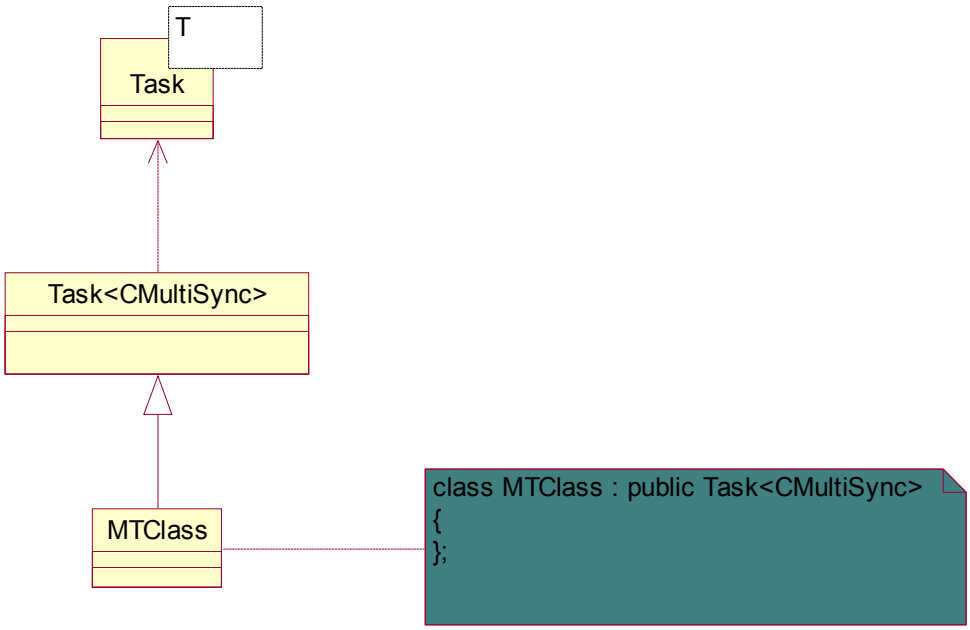


Рис.20. Иллюстрация Шаблона.

Появятся следующие предупреждения при генерации:
10:46:53 PM Warning: Unsupported declaration encountered:
CLASSDECL(Cfoo, Cbase)
(See line 32 in “D:\Projects\tests\unsupported\Unsupported.h”.) This declaration will be ignored.

Rose Visual C ++ - не способен развернуть CLASSDECL макрокоманду, чтобы получить в основном имеющем силу C ++ декларацию и таким образом интерпретирует инструкцию как недействительная декларация и игнорирует это.

Члены данных Класса, объявленные с совокупными типами типа emus и анонимного structs или союзов не поддержаны. Таким образом:

```
class Cinvalid {
  enum {Red, Green, Blue } color;
  struct {int x; int y; } posn;
};
```

Результат следующих из этого предупреждений:

10:46:55 PM Warning: Data members with aggregate types are not supported. (See line 71 in “D:\Projects\tests\unsupported\Unsupported.h”.) The member Cinvalid::posn will not be updated.

10:46:55 PM Warning: Data members with enum types are not supported. (See line 70 in “D:\Projects\tests\unsupported\Unsupported.h”.) The member Cinvalid::color will not be updated.

Для работы с этим, используйте typedefs, чтобы представить названный тип для членов данных. Например,

```
class Cinvalid {
  typedef enum {Red, Green, Blue } COLOR;
  typedef struct {int x; int y; } POSN;
COLOR color;
  POSN posn;
};
```

И последнее, декларации мультилинии не поддержаны и будут игнорироваться. Например, инструкция кода:

```
struct Coord { int x; int y; int z; };
```

Предупреждения:

11:06:54 PM Warning: Unsupported declaration encountered in class Coord:
struct Coord { int x; int y; int z; };
(See line 40 in “D:\Projects\tests\unsupported\Unsupported.h”.) This declaration will be ignored.

11:06:54 PM Warning: Unsupported declaration encountered in class Coord:
struct Coord { int x; int y; int z; };
(See line 40 in “D:\Projects\tests\unsupported\Unsupported.h”.) This declaration will be ignored.

11:06:54 PM Warning: Unsupported declaration encountered in class Coord:
struct Coord { int x; int y; int z; };
(See line 40 in “D:\Projects\tests\unsupported\Unsupported.h”.) This declaration will be ignored.

Есть три отдельных предупреждения, одно для каждого члена данных, перечисленного на той же самой линии. Вообще, Rose, Visual C++ требует каждый член данных или декларации функции членства класса или структуры, чтобы появиться на ее собственной линии.

Генерация ассоциаций на Rose Visual C++

Ассоциации генерируются и полностью изменяются по- другому чем, Rose C ++. Rose Visual C ++ оптимизирован к уменьшению энтропии, которая обычно происходит при обратной ассоциации. Отображение от модели, чтобы закодировать просто и прямо и не требует никаких дополнительных свойств генерации объектного кода. Соединение частей, независимое от сдерживания – для примера, by-val или by-ref-always отображает к объектному случаю. Ассоциация всегда отображает ссылку (рекомендации) указателя.

Контейнерные классы определены, непосредственно используя синтаксис спецификатора интерфейса роли.

Выполнение Контейнерных классов смоделировано, используя новый ситаксис спецификатора интерфейса UML – для примера, rolename: classname – Чтобы непосредственно определить желательный класс. Образцовый Помощник может использоваться также, чтобы выбирать и определить контейнерный класс, но эффективно это только модифицирует rolename с отобранным контейнерным классом. В отличие от Rose C++, разнообразие не используется, чтобы управлять выбором контейнерного класса.

Есть три возможных выбора: 1) определяет, что содержимый объект (цель) класса поставщика, 2) определяет указатель на класс поставщика, или 3) ни один из вышеупомянутых – не определяет никакой желательный класс, чтобы использовать, осуществить роль. Следующие диаграммы суммируют эти выборы:

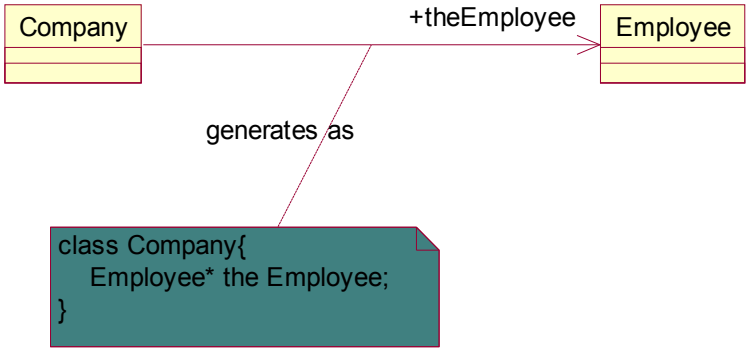


Рис.21. Простая ассоциация.

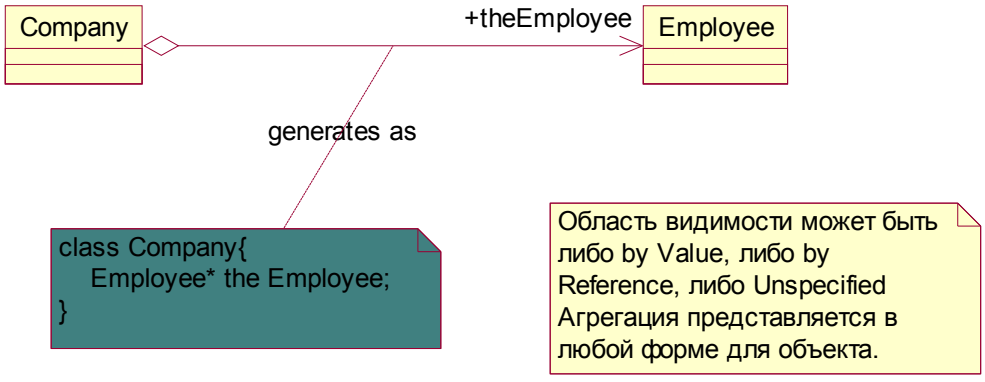


Рис.22. Простая агрегация.

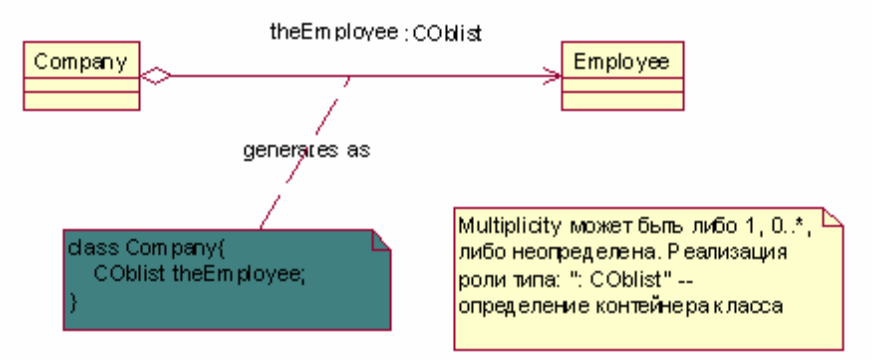


Рис.23. Контейнер класса.

Отображение фактически единственное: только выполнение роли (спецификатор интерфейса) используется, чтобы управлять генерацией объектного кода. Для не раскрашенного rolename-one без указанного выполнения роли – следующие значения по умолчанию приняты:

Aggregation implies rolename : SupplierClass
Association implies rolename : SupplierClass*
As you can see, the defaults get us back to the original definitions:
aggregation = object instance,
association = pointer reference.

Технология обратного проектирования следует тем же самым правилам, но наоборот. Например, перемена, проектирующая указатель кончается созданными отношениями ассоциации; обратная разработка объектный случай кончается созданными отношениями соединения частей. Изменение (замена) кода, чтобы добавлять или удалить * от типа члена данных фактически изменит (заменит) ассоциацию на соединение частей и наоборот в модели когда проектируемая перемена. (Если класс поставщика не обозначенный тип атрибута – для примера, Cstring или Crect – в этом случае, проектируется как атрибут переменой.)
Так как выполнение роли используется исключительно, чтобы управлять 126енерацией объектного кода, с ассоциацией и соединением частей, обеспечивающим заданное по умолчанию выполнение роли, количество элементов и сдерживание не требуются. Вместо этого, используется выполнение роли как единственное (отдельное) примечание отображения, количество элементов и сдерживание освобождены до использования как чистый семантический проект/модель, который сохраняется, пересекая генерацию объектного кода и обратной разработки. Следующая диаграмма иллюстрирует, как первоначальная семантика проекта сохраняется, даже при использовании контейнерных классов.

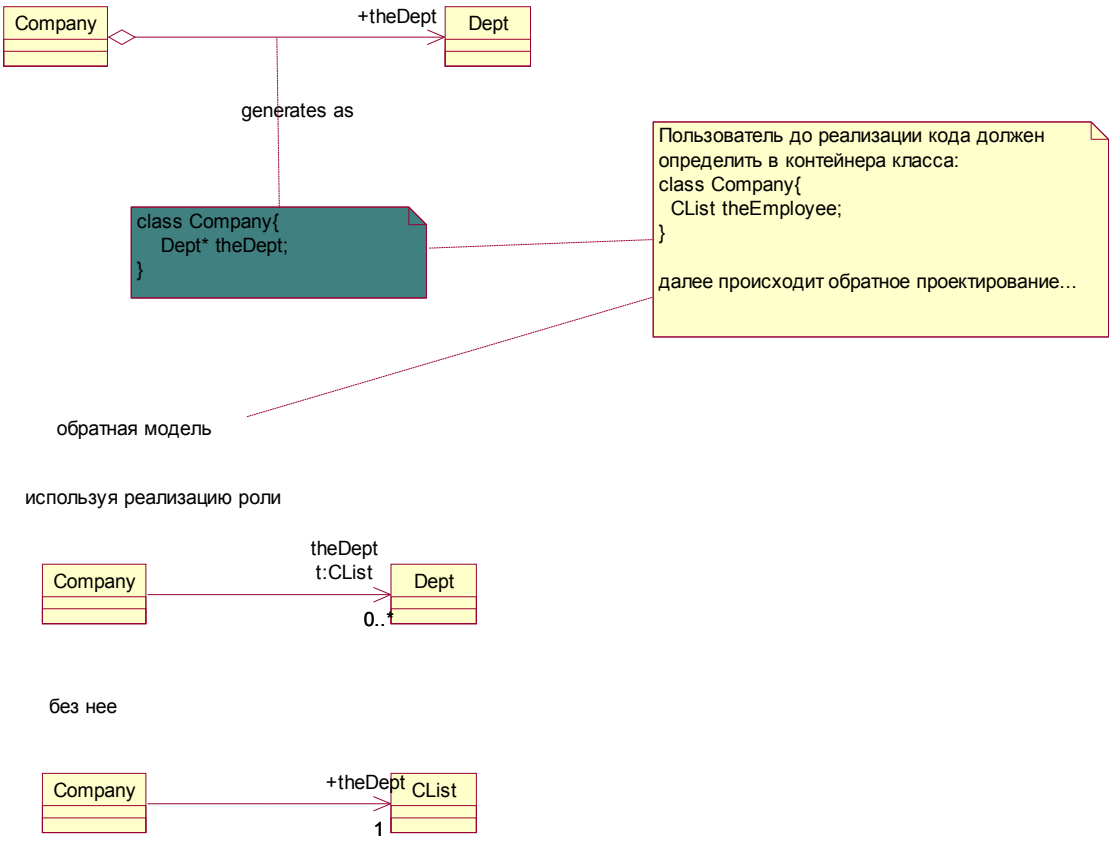


Рис.24. Сохранение первоначальной семантики проекта.

Реализация семантики

Генерация конструкторов и деструкторов без Model Assistant

Rose Visual C ++ поддерживает генерацию объектного кода «образец», который позволяет Вам определять, которые конструкторы операций, деструкторы, и операторы – каждого нового класса должны быть по умолчанию. Кроме того, возможно, определить, что каждый новый член данных класса должен получать или устанавливать добавочный метод по умолчанию. Эти параметры настройки применяются всякий раз, когда класс первоначально сгенерирован, и Применяющийся Образец на опции Code generation включен. Эта опция и желательный образец определены в диалоговых свойствах Visual C++. «Образец» определен, проверяя желательные операции класса и добавочные методы во вкладках Operation и Accessors Класса.

Что такое "диаграмма краткого обзора"

Диаграмма краткого обзора создается для компонента, каждый раз когда он обратно проектирован. Если диаграмма краткого обзора уже существует, резервная копия создается перед перезаписью данной. Таким образом резервная диаграмма краткого обзора представляет результаты предыдущих изменений, проектирующей попытку для того компонента. Это очень полез-

но для сравнения, что изменилось с последнего раза, когда компонент был обратно спроектирован.

Создание диаграмм краткого обзора управляется установкой (настройкой) Create Overview Diagrams на вкладке диалоговых свойств: Model Update Visual C ++.

11.МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО И ЭКЗАМЕНАЦИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ.

1. Межсессионная аттестация студентов проводится дважды в семестр на 7 и 13 неделях 8-го и 9-го семестров.

2. Аттестационная оценка складывается из оценок, полученных на аттестационных занятиях по лабораторным работам и собеседованиях:

8-й семестр:

2.1. Первое аттестационное занятие. Проверка знаний студентов по основным направлениям развития проектирования Информационных систем.

2.2. Второе аттестационное занятие. Проверка знаний студентов по представлению этапов проектирования ИС, состав работ и проектной документации.

2.3. Первое собеседование. Функциональные и обеспечивающие подсистемы ИС;

2.4. Второе собеседование. Методологические основы проектирования ИС.

9-й семестр:

2.51. Первое аттестационное занятие. Проверка знаний и навыков студентов по проектированию клиент-серверных корпоративных ИС.

2.6. Второе аттестационное занятие. Проверка знаний и навыков студентов по возможностям функционально-ориентированного проектирования ИС.

2.7. Первое собеседование. Объектно-ориентированное проектирование ИС. Прототипное проектирование ИС (RAD-технология);

2.8. Второе собеседование. Типовое проектирование ИС.

3. Организация аттестации студентов, проводится в соответствии с положением АмГУ о курсовых экзаменах и зачетах*.

* 3.1. Организация аттестации студентов в университете по специальностям и направлениям высшего профессионального образования регламентируется рабочим учебным планом, расписанием учебных занятий и программами учебных дисциплин, утверждаемыми в установленном в университете порядке.

Контроль за качеством освоения образовательных программ осуществляется путем текущей внутрисеместровой аттестации, ректорской контрольной аттестации, промежуточной аттестации студентов в форме курсовых экзаменов и зачетов и итоговой аттестации выпускников.

3.2. Курсовые экзамены и зачеты проводятся по дисциплинам утвержденного учебного плана по соответствующим специальностям и направлениям высшего профессионального образования. Знания, умения и навыки обучающегося определяются оценками "отлично", " хорошо", " удовлетворительно", " неудовлетворительно", "зачтено" и "незачтено".

3.3. Студенты, обучающиеся по основным программам высшего профессионального образования, сдают в течение учебного года не более 10 экзаменов и 12 зачетов. В это число не входит аттестация по физической культуре и факультативным дисциплинам.

Студенты, обучающиеся в сокращенные сроки (по индивидуальным планам), в течение учебного года сдают не более 20 экзаменов и 24 зачетов.

3.4. Сроки проведения курсовых зачетов и экзаменов (экзаменационная сессия) и начало очередного учебного семестра устанавливаются графиком учебного процесса, утвержденным проректором по учебной работе.

Расписание экзаменов составляется в соответствии с графиком учебного процесса, утверждается проректором по учебно-научной работе и доводится до сведения преподавателей и студентов не позднее, чем за две недели до начала сессии. Расписание составляется таким образом, чтобы на подготовку к экзаменам по каждой дисциплине было отведено не менее 3 дней, исключая день предыдущего экзамена. По согласованию с деканами и заведующими соответствующих кафедр отдельные экзамены (зачеты) могут проводиться в течение семестра по завершении преподавания дисциплины.

12. КОМПЛЕКТЫ ЗАДАНИЙ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Контрольные вопросы к лабораторной работе № 1

1. Что называется процессом нормализации?
2. Что называется функциональной зависимостью?
3. Что называется полной функциональной зависимостью?
4. Первая нормальная форма.
5. Вторая нормальная форма.
6. Третья нормальная форма.
7. Нормальная форма Бойсса-Кодда.
8. Что называется процессом денормализации?
9. В чем смысл денормализации?
10. Какова цель создания физической модели?
11. Назовите функции ERWin по поддержке денормализации.
12. Как осуществляется разрешение связей «многие-ко-многим»?

Контрольные вопросы к лабораторной работе № 2

1. Каково назначение инструмента Report Browser?.
2. Назовите основные элементы окна Report Browser.
3. Как создать новый отчет?
4. Как связать отчет с иконкой?
5. Как выполнить существующий отчет?

6. Что такое представление отчета и для чего оно предназначено?
7. Как сохранить отчет в виде представления?
8. Какие категории отчетов присутствуют в Report Browser по умолчанию?
9. Как выбрать условия фильтрации данных отчета?
10. В какие форматы можно экспортировать отчет?
11. Как отредактировать отчет?
12. Что называется результирующим набором?
13. Какой тип отчета позволяет проверить отсутствие ошибок в модели?
14. Опишите механизм поиска ошибок в модели при помощи отчетов.

Контрольные вопросы к лабораторной работе № 3

1. Какие три типа моделей используются при проектировании?
2. Каково назначение концептуальной модели?
3. Назовите основной вид диаграмм в концептуальной модели.
4. Каково назначение логической модели?
5. Назовите основной вид диаграмм в логической модели.
6. Назовите два взгляда на моделируемую систему в логической модели.
7. Какова роль диаграмм взаимодействия объектов в логической модели?
8. Какова роль диаграмм последовательности взаимодействий в логической модели?
9. Каково назначение физической модели?
10. Назовите основной вид диаграмм в физической модели.
11. В чем смысл процедуры итерационного моделирования?

13. ФОНД ТЕСТОВЫХ И КОНТРОЛЬНЫХ ЗАДАНИЙ ДЛЯ ОЦЕНКИ КАЧЕСТВА ЗНАНИЙ ПО ДИСЦИПЛИНЕ.

ТЕСТЫ

по дисциплине «Проектирование информационных систем»

1. Какое утверждение ***неверно*** для каскадного способа разработки информационных систем (ИС): (d)

a) Его основной характеристикой является разбиение всей разработки на этапы
b) Переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем.
c) Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.
d) Последовательность шагов разработки следующая: Анализ – Проектирование – Сопряжение – Реализация – Внедрение

2. Какое утверждение ***неверно*** для спиральной модели жизненного цикла ИС: (b)

a) Делает упор на начальные этапы жизненного цикла: анализ и проектирование.
b) Переход на следующий уровень не может быть осуществлен до полного завершения предыдущего.
c) Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения (ПО), на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.
d) Основная проблема спирального цикла - определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла.

3. Объект в ООА представляет собой: (b)

a) Описывает реально не существующий элемент,
b) Один типичный, но неопределенный экземпляр в реальном мире,
c) Конкретный экземпляр в реальном мире,
d) Аналогичен понятию объекта в программировании (Object)

4. Абстракции цели или назначения человека, части оборудования или организации: (b)

a) реальные объекты;
b) роли;
c) прецедент;
d) взаимодействия;

5. Абстракции фактического существования некоторых предметов в физическом мире, это: (a)

a) реальные объекты;
b) роли;
c) прецедент;
d) взаимодействия;

6. Объекты, получаемые из отношений между другими объектами: (d)

a) реальные объекты;
b) роли;
c) прецедент;
d) взаимодействия;

7. Абстракция чего-то постоянно происходящего: (c)

a) реальные объекты;
b) роли;
c) прецедент;
d) взаимодействия;

8. Абстракция сигнала в реальном мире, который сообщает нам о перемещении чего-либо в новое состояние (b)

a) Сущность,

b) Событие,
c) Действие,
d) Состояние.

9. Положение объекта, в котором применяется определенный набор правил, линий поведения, предписаний и физических законов (d)

a) Сущность,
b) Событие,
c) Действие,
d) Состояние.

10. Деятельность или операция, которая должна быть выполнена экземпляром, когда он достигает состояния (c)

a) Сущность,
b) Событие,
c) Действие,
d) Состояние.

11. Связь в ООА это: (c)

a) Абстракция фактического существования некоторых предметов в физическом мире
b) Абстракция прецедента или сигнала в реальном мире, который сообщает нам о перемещении чего-либо в новое состояние
c) Абстракция набора отношений, которые систематически возникают между различными видами предметов в реальном мире
d) Абстракция чего-то произошедшего или случившегося

12. На диаграммах “Сущность-связь” связи изображаются: (b)

a) Не изображаются
b) Линиями
c) Прямоугольниками
d) Овалами

13. Функциональные диаграммы могут изображаться в нотации: (b)

a) DFD
b) IDEF0
c) IDEF1X
d) IDEF2

14. Диаграммы потоков данных могут изображаться в нотации: (a)

a) DFD
b) IDEF0
c) IDEF1X
d) IDEF2

15. Диаграммы сущность-связь могут изображаться в нотации: (c)

a) DFD
b) IDEF0
c) IDEF1X
d) IDEF2

16. Какое из следующих высказываний ***неверно*** для моделей состояний в ООА: (c)

a) Модель состояний связана с поведением объектов и связей во времени.
b) Модели состояний используются для формализации жизненных циклов объектов и связей.
c) Модели состояний изображаются в виде диаграмм потоков данных
d) Модели состояний выражаются в переходных диаграммах и таблицах

17. По какому из приведенных типов атрибуты (в ООА) ***не могут*** классифицироваться: (b)
a) описательные;
b) связующие;
c) указывающие;
d) вспомогательные.

18. Отдельный реальный, гипотетический или абстрактный мир, населенный отчетливым набором объектов, которые ведут себя в соответствии с характерными для него правилами и линиями поведения, это (c)
a) Множество;
b) Сущность;
c) Домен;
d) Класс.

19. Домен, который обеспечивает общие механизмы и сервисные функции, необходимые для поддержки прикладного домена, это (b)
a) Домен механизмов;
b) Сервисный домен;
c) Архитектурный домен;
d) Домены реализации

20. Предметная область системы с точки зрения конечного пользователя системы (в ООА), это: (a)
a) Прикладной домен;
b) Сервисный домен;
c) Архитектурный домен;
d) Домены реализации

21. Домен, включающий в себя языки программирования, сети, операционные системы и общие библиотеки классов и обеспечивающий концептуальные сущности, в которых будет реализована вся система, это (d)
a) Домен механизмов;
b) Сервисный домен;
c) Архитектурный домен;
d) Домены реализации.

22. Домен, который обеспечивает общие механизмы и структуры для управления данными и управления системой как единым целым, это: (c)
a) Домен механизмов;
b) Сервисный домен;
c) Архитектурный домен;
d) Домены реализации

23. В ООА справедлива следующая цепочка декомпозиции задачи: (d)
    a) Задача – объект – процесс – действие;
    b) Задача – процесс – объект – действие;
    c) Задача – процесс – действие – объект;
    d) Задача – объект – действие – процесс;

24. В ООА при формализации связи один-к-одному вспомогательные атрибуты могут быть добавлены: (d)
    a) к первому объекту
    b) ко второму объекту
    c) к обоим объектам вместе
    d) к любому объекту (но не к обоим)

25. В ООА при формализации связи один-ко-многим вспомогательные атрибуты должны быть: (b)
    a) добавлены к объекту на стороне "один"
    b) добавлены к объекту на стороне "много"
    c) добавлены к обоим объектам
    d) не должны добавляться

26. В диаграмме переходов в состояние переход обозначается: (c)
    a) прямоугольником
    b) овалом
    c) стрелкой
    d) надписью

27. Что из ниже перечисленного не может включаться в диаграммы потоков данных: (a)
    a) таймер,
    b) внешняя сущность,
    c) процессы,
    d) накопители данных

28. Определяет информацию, передаваемую через некоторое соединение от источника к приемнику (в ДПД): (d)
    a) внешняя сущность
    b) процесс
    c) накопитель данных
    d) поток данных

29. Преобразование входных потоков в выходные в соответствии с определенным алгоритмом (в ДПД): (b)
    a) внешняя сущность
    b) процесс
    c) накопитель данных
    d) поток данных

30. Абстрактное устройство для хранения информации (в ДПД): (c)
    a) внешняя сущность
    b) процесс
    c) накопитель данных
    d) поток данных

31. Материальный предмет или физическое лицо, представляющие собой источник и приемник информации (в ДПД): (a)

a) внешняя сущность
b) процесс
c) накопитель данных
d) поток данных

32. Чем характеризуется информационная переменная: (a)

a) наименованием, значением и обозначением
b) множеством допустимых значений
c) наименованием переменной
d) перечнем ее основных характеристик

№ пп в билете	Вариант 1		Вариант 2		Вариант3		Вариант4	
	№ из списка	Правильный ответ	№ из списка	Правильный ответ	№ из списка	Правильный ответ	№ из списка	Правильный ответ
1	1	d	2	b	3	b	4	b
2	5	a	6	d	7	c	8	b
3	9	d	10	c	11	c	12	b
4	13	b	14	a	15	c	16	c
5	17	b	18	c	19	b	20	a
6	21	d	22	c	23	d	24	d
7	25	b	26	c	27	a	28	d
8	29	b	30	c	31	a	32	a

На выполнение заданий теста дается 40 минут.

В каждом вопросе за каждый полный ответ – 2 балла.

За неполный ответ – 1 балл.

Максимальное количество набранных баллов – 16.

Критерии оценок:

14 – 16 баллов – «отлично»
12 – 13 баллов – «хорошо»
9 – 11 баллов – «удовлетворительно»
0 – 8 баллов – «неудовлетворительно»

14. КОМПЛЕКТ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 1

1. Специфика информационных программных систем. Задачи информационных систем.
2. Серверы баз данных как базовая системная поддержка ИС в архитектуре клиент-сервер.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 2

1. Транзакции в СУБД.
2. Журнализация изменений БД.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 3

1. Проблемы построения ИС.
2. Intranet – приложения.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 4

1. Требования к техническим средствам, поддерживающим ИС.
2. Склады данных и системы оперативной аналитической обработки данных.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 5

1. Жизненный цикл ПО ИС.
2. Интегрированные распределенные приложения.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г.
Заведующий кафедрой
Утверждаю: _____

Факультет *М и И*
Курс *5*
Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 6

1. Модели жизненного цикла ПО.
2. Основные производители серверов БД и характеристика их продуктов.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 7

1. Методологии и технологии проектирования ИС.
2. CASE – системы для проектирования ИС.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 8

1. Сущность структурного подхода при проектировании ИС.
2. Диаграммы потоков данных. Модели доступа к данным.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 9

1. Методология функционального моделирования SADT.
2. Операционные системы, характеристика.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 10

1. Моделирование потоков данных.
2. СУБД, характеристика.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 11

1. Моделирование данных.
2. Техническое задание на ИС.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 12

1. Методология IDEF1.

2. Выбор средств разработки приложений.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 13

1. Файл-серверные приложения.
2. Схема взаимодействия.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 14

1. Традиционные и новые средства разработки файл-серверных приложений.
2. LAN – сети.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 15

1. Перенос файл-серверных приложений в среду клиент-сервер.

2. MAN – сети.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 16

1. Клиент-серверные приложения.
2. Wan – сети.

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра *И и УС*

Утверждено на заседании кафедры

“ __ “ _____ 200_ г. Факультет *М и И*

Заведующий кафедрой Курс *5*

Утверждаю: _____ Дисциплина *Проектирование ИС*

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 17

1. Базовые средства построения ИС в архитектуре клиент-сервер.
2. Специфика информационных программных систем.

13.КАРТА КАДРОВОЙ ОБЕСПЕЧЕНОСТИ ДИСЦИПЛИНЫ

Лектор – доцент, канд.техн.наук, Бушманов А.В.
Руководитель лабораторных работ – ассистент Жилиндина О.В.

Бушманов Александр Вениаминович,
к.т.н., доцент.

ПРОЕКТИРОВАНИЕ ИС
Учебно-методический комплекс дисциплины

Изд-во АмГУ. Подписано к печати ???. Формат ???. Усл. печ. л. ???, уч. - изд. л. ???. Тираж 100. Заказ ???.