

# ПРАКТИКУМ

ДЛЯ ВЫСШИХ УЧЕБНЫХ ЗАВЕДЕНИЙ

СПЕЦИАЛЬНОСТЬ



# CASE-ТЕХНОЛОГИИ

Горячая линия-Телеком



Д.Э.Федотова, Ю.Д.Семенов, К.Н.Чижик

**Д.Э.Федотова, Ю.Д.Семенов, К.Н.Чижик**

# **CASE-ТЕХНОЛОГИИ**

**Москва**  
**Горячая линия – Телеком**  
**2005**

ББК 65.050.9 (2)  
Ф 34

*Рецензенты:*

*канд. техн. наук, доцент Ю.Е. Морозовец, доцент В.А. Дихтяр*

**Федотова Д.Э., Семенов Ю.Д., Чижик К.Н.**

Ф34 CASE-технологии: Практикум. – М.: Горячая линия–Телеком,  
2005. – 160 с.: ил.

**ISBN 5-93517-121-X.**

В систематизированном виде приводятся необходимые теоретические сведения и 16 лабораторных работ, направленных на обучение технологии составления диаграмм по стандартам DFD, IDEF0, IDEF3, IDEF1X, UML. Цель книги – помочь приобрести практические навыки проектирования сложных программных систем с помощью пакетов BPWin и Rational Rose.

Для студентов, может быть полезна тем, кто начинает изучать вопросы применения CASE-средств.

**ББК 65.050.9 (2)**

*Адрес издательства в Интернет [www.techbook.ru](http://www.techbook.ru)  
e-mail: [radios\\_hl@mtu-net.ru](mailto:radios_hl@mtu-net.ru)*

Учебное издание

**Федотова Дина Эммануиловна  
Семенов Юрий Дмитриевич  
Чижик Константин Николаевич**

**CASE-технологии**

Практикум

Обложка художника В.Г. Ситникова

ЛР № 071825 от 16 марта 1999 г.

Подписано в печать 12.11.04. Печать офсетная. Формат 60x88/16. Гарнитура Arial  
Уч.-изд. л. 10 л. Тираж 1000 экз. Изд. № 121

**ISBN 5-93517-121-X**

© Д.Э. Федотова, Ю.Д. Семенов, К.Н. Чижик, 2003, 2005

© Оформление издательства  
«Горячая линия–Телеком», 2003, 2005

# Введение

Настоящий курс лабораторных работ посвящен CASE-средствам структурного и объектно-ориентированного анализа BPWin 2.5, ERWin 3.5.2 и Rational Rose 98 и направлен на обучение студентов технологии составления диаграмм по стандартам DFD, IDEF0, IDEF3, IDEF1X, UML.

Существует два основных способа проектирования программных систем – структурное проектирование, основанное на алгоритмической декомпозиции, и объектно-ориентированное проектирование, основанное на объектно-ориентированной декомпозиции. Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение агентам, которые являются либо объектами, либо субъектами действия. Однако эти способы, по сути, ортогональны, поэтому нельзя сконструировать сложную систему одновременно двумя способами. Необходимо начать разделение системы либо по алгоритмам, либо по объектам, а затем, используя полученную структуру, попытаться рассмотреть проблему с другой точки зрения.

Алгоритмическую декомпозицию можно представить как обычное разделение алгоритмов, где каждый модуль системы выполняет один из этапов общего процесса. При объектно-ориентированной декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира. С этой точки зрения объект является вполне осязаемой вещью, которая демонстрирует вполне определенное поведение. Объекты что-то делают, и мы можем, пошлав им сообщение, попросить их выполнить некоторые операции.

Объектная декомпозиция имеет несколько преимуществ перед алгоритмической.

- Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов, что приводит к существенной экономии выразительных средств.
- Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируются на устойчивых промежуточных формах. Действительно, объектная декомпозиция существенно снижает риск при создании сложной программной системы, так как она развивается из меньших систем, в которых мы уже уверены.

- Объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний.

В объектно-ориентированном анализе существует четыре основных типа моделей: динамическая, статическая, логическая и физическая. Через них можно выразить результаты анализа и проектирования, выполненные в рамках любого проекта. Эти модели в совокупности семантически достаточно богаты и универсальны, чтобы разработчик мог выразить все заслуживающие внимания стратегические и тактические решения, которые он должен принять при анализе системы и формировании ее архитектуры. Кроме того, эти модели достаточно полны, чтобы служить техническим проектом реализации практически на любом объектно-ориентированном языке программирования.

Фактически все сложные системы можно представить одной и той же канонической формой – в виде двух ортогональных иерархий одной системы: классов и объектов. Каждая иерархия является многоуровневой, причем в ней классы и объекты более высокого уровня построены из более простых. Какой класс или объект выбран в качестве элементарного, зависит от рассматриваемой задачи. Объекты одного уровня имеют четко выраженные связи, особенно это касается компонентов структуры объектов. Внутри любого рассматриваемого уровня находится следующий уровень сложности. Структуры классов и объектов не являются независимыми: каждый элемент структуры объектов представляет специфический экземпляр определенного класса. Объектов в сложной системе обычно гораздо больше, чем классов. С введением структуры классов в ней размещаются общие свойства экземпляров классов.

Структурный подход состоит в декомпозиции (разбиении) системы на элементарные функции, т. е. система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи, и т. д. Процесс разбиения продолжается вплоть до конкретных процедур. При этом создаваемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны.

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

- принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

- принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне – так называемый принцип иерархического упорядочения.

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 системы модели расширяются, уточняются и дополняются диаграммами, отражающими ее структуру.

Перечисленные модели в совокупности дают полное описание системы независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

### **Темы лабораторных работ**

1. Работа с читателями в библиотеке МИРЭА.
2. Учет информационного фонда и поиск в библиотеке периодических изданий кафедры МОВС.

### **Автоматизация работы деканата**

1. Архив учебных групп.
2. Сессия.
3. Текущая успеваемость.
4. Кадры преподавателей и сотрудников.
5. Распределение учебной нагрузки по кафедрам и учет ее выполнения.
6. Система методического обеспечения учебного процесса (учебные планы, программы).
7. Научно-исследовательские работы.

### **Автоматизация работы кафедры**

1. Архив учебных групп.
2. Кадры преподавателей и сотрудников.
3. Распределение учебной нагрузки по преподавателям и учет ее выполнения.
4. Составление и корректировка учебного плана.
5. Учебные программы.
6. Система методического обеспечения учебного процесса (задачи).
7. Расписание занятий.
8. Научно-исследовательские работы.
9. Учет материальных ресурсов на кафедре.
10. Система поддержки учебного процесса: мониторинг аппаратных и программных ресурсов.
11. Проектирование и прокладка локальной вычислительной сети кафедры.

# Лабораторная работа № 1

## Теоретическое введение в предметную область

### Цель работы:

- ознакомиться с системой «Служба занятости в рамках вуза».

Настоящий курс лабораторных работ ориентирован на изучение CASE-средств на примере диаграмм, создаваемых для проекта «Служба занятости в рамках вуза». Процесс создания диаграмм начинается с этапа изучения предметной области, которая описывается в этой лабораторной работе.

### 1. Описание системы

Система предназначена для того, чтобы помочь студенту устроиться на работу уже в процессе его обучения в вузе. Подав заявление в систему, студент становится ее клиентом и начинает обслуживаться на протяжении всего обучения в вузе. Заявление представляет собой анкету. Система предлагает профессиональные (основанные на изучаемых предметах), психологические тестирования, проводимые регулярно (раз в семестр (полгода)). Особое внимание уделяется обучению студента, по итогам успеваемости составляются экспертные оценки. На основе собранной информации составляется резюме, представляющее собой полную характеристику человека. Это резюме отсылается всем организациям, имеющим необходимые вакансии.

Основным назначением системы является автоматизация ввода и хранения отчетных данных по студентам, составления характеристик и резюме, поиска вакансий в фирмах. Система позволяет изменять, дополнять, вести поиск и просмотр информации о студентах, накладывать ограничения доступа к системе, хранить списки студентов, закончивших обучение, в виде архива, контролировать выдачу студенту заданий на курсовые работы и проекты, связывать институт с фирмами, заинтересованными в поиске сотрудников.

Также данная система может быть использована для составления отдельных списков групп, для печати зачетных ведомостей, для печати полной базы данных и для статистики.



Система состоит из четырех подсистем:

- контроля успеваемости студентов;
- профессиональных и психологических тестов;
- обработки запросов, определения категорий полномочий пользователей;
- экспертных оценок.

### 1.1. Подсистема контроля успеваемости студентов

Эта подсистема является частью системы «Служба занятости в рамках вуза», которая отвечает за статистическую отчетность по успеваемости отдельного студента, группы или целого факультета, а также за хранение и правильность ее ввода.

Входными данными подсистемы являются: оценки, даты сдачи экзаменов, имена студентов, номера групп, факультет. На выходе подсистема выдает обработанные данные: средний балл по студенту, группе или факультету, процентное соотношение оценок у студента в группе или на факультете, имена и количество стипендиатов в группе или на факультете. Подсистема «Контроль успеваемости студентов» может функционировать отдельно от всей системы, что дает возможность установить и использовать ее независимо, если это необходимо.

Подсистема «Контроль успеваемости студентов» включает следующие функции:

- ввод, вывод и редактирование информации по информационным объектам подсистемы;
- сохранение информации, поступившей от подсистемы контроля успеваемости студентов;
- расчет процентного соотношения оценок у студента в группе или на факультете и вывод его в виде таблиц, графиков и диаграмм;
- расчет среднего балла по студенту, группе или факультету;
- формирование данных по студенту, группе или факультету;
- выявление сильнейших и слабейших студентов в группе или на факультете;
- расчет количества стипендиатов в группе или на факультете;
- проверку правильности ввода данных.

1.2. Подсистема профессиональных и психологических тестов  
Модульная структура

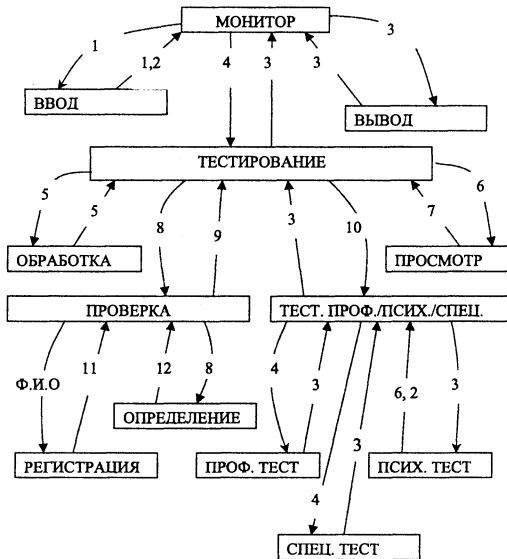


Рис. 1.1. Подсистема профессиональных и психологических тестов

1. Ф.И.О., новые тесты, название теста.
2. Ответы на тесты.
3. Вопросы, результаты теста, Ф.И.О.
4. Ответы, Ф.И.О., название теста.
5. Новый тест.
6. Ф.И.О., название теста.
7. Результаты теста, Ф.И.О.
8. Ф.И.О., специальность.
9. Ф.И.О., набор тестов.
10. Ф.И.О., профессионально-психологический тест, психологический тест, специальный тест, ответы.
11. Ф.И.О., информация о пройденных тестах.
12. Ф.И.О., набор тестов для клиента.

### Внешние сущности

Фирма	Тип производственного объединения предприятия. В данной ситуации фирма выступает как работодатель. Она отправляет запрос на специалиста.
Клиент	Человек, поступивший учиться в институт и пользующийся услугами СЛУЖБЫ ЗАНЯТОСТИ для нахождения работы. Клиент получает запрос на проведение профессионального, психологического и специального тестов и вопросы тестов. Затем КЛИЕНТ отправляет входные данные, ответы на вопросы тестов, запрос на результаты тестов.
Архив	Хранилище, где хранятся все данные о клиенте, начиная с момента пользования услугами СЛУЖБЫ ЗАНЯТОСТИ.
Дополнительные источники	Источники, из которых поступают новые, более современные тесты. Это INTERNET, журналы, какие-либо специализированные центры.

### Работа модулей подсистемы

Монитор – вызывает модуль ввода и получает от него данные для выполнения дальнейших задач; обращается к модулю тестирования, передавая ему исходные данные; после выполнения своей задачи тестирование передает ему свои результаты; вызывает модуль вывода и получает от него конечный результат.

Ввод – получает запрос от монитора, передает ему запрашиваемые данные.

Вывод – вызывается монитором и получает от него результаты, выводя их на дисплей.

Тестирование – вызывается монитором, получает от него исходные данные и в зависимости от них обращается к тому или иному модулю. Выполнив свои функции, модули посылают тестированию результаты действий, а тестирование уже передает полученные результаты монитору.

Проверка – получает исходные данные от тестирования и передает их своим подмодулям: регистрация тестируемых, определение тестов. Подмодули передают полученные результаты модулю проверки. Тестирова-

ние (профессиональное, психологическое, специальное) получает исходные данные от модуля тестирования и передает их своим подмодулям: профессиональное, психологическое, специальное тестирование. Подмодули передают полученные результаты.

Обработка – получает данные от тестирования, выполняет свои задачи и возвращает полученные результаты модулю тестирования.

Просмотр – получает данные от модуля тестирования, выполняет свои задачи и посылает полученные результаты модулю тестирования.

### **1.3. Подсистема обработки запросов, определения категорий пользователей**

Данная подсистема предназначена для определения категории, полномочий и обработки запросов пользователей службы занятости. В частности, она выполняет следующие функции:

- регистрацию новых фирм;
- регистрацию новых студентов;
- определение прав доступа зарегистрированного пользователя;
- обработку запросов;
- прием регистрационных данных от фирм;
- прием регистрационных данных от студентов;
- прием регистрационных данных от обслуживающего персонала;
- составление резюме;
- запись данных в БД студентов;
- запись данных в БД фирм;
- запись данных в БД зарегистрированных пользователей.

В соответствии с выполняемыми функциями система работает со следующими данными:

- регистрационными данными студентов;
- регистрационными данными фирм;
- личными данными студентов;
- информацией о студентах (получаемой фирмами);
- информацией о фирмах (получаемой студентами);
- идентификационными данными пользователей;
- информацией о системе;
- запросом;
- служебной информацией (для обслуживающего персонала);
- результатом психологического теста;

- результатом профессионального теста;
- экспертными оценками.

### Модульная структура



**Рис. 1.2.** Подсистема обработки запросов, определения категорий пользователей

Определение категории – модуль, определяющий категорию пользователя.

Определение полномочий – модуль, определяющий полномочия пользователя.

Обработка запроса – модуль, предназначенный для обработки запросов пользователя.

Выполнение запроса – модуль, предназначенный для выполнения запросов пользователя.

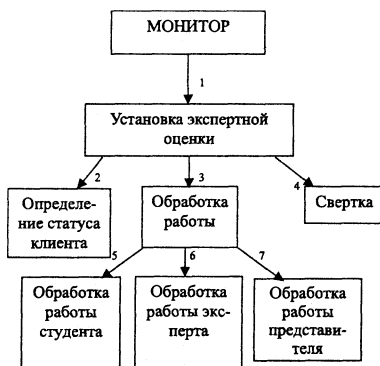
Запись в БД зарегистрированных пользователей – модуль, предназначенный для работы с базой данных зарегистрированных пользователей.

Запись в БД студентов – модуль, предназначенный для работы с БД студентов.

Запись в БД фирм – модуль, предназначенный для работы с БД фирм.

#### 1.4. Подсистема экспертных оценок

Эта подсистема предназначена для установки и просмотра экспертной оценки. Она дает краткую информацию преподавателю о студенте или группе. Студент может с помощью ее ориентироваться в учебе.



**Рис. 1.3.** Подсистема экспертных оценок

Кроме того, подсистема может дать представителю фирмы некоторое представление о студенте для рассмотрения его в качестве новых кадров. Подсистема является как информационной средой, так и средой установки экспертной оценки.

## Определение статуса клиента

- Клиент входит в подсистему с идентификационным номером. По номеру присваивается уровень доступа. Подсистема выдает меню работы, соответствующее уровню доступа.

## Обработка отчетного задания

- Эксперт: посылает запрос на предоставление информации о студенте (группе), результатах тестирования. Из общей БД поступает успеваемость студента (группы), результаты тестирования. Делает запрос на критерии. Из хранилища критериев поступают критерии для установки экспертной оценки. Вводит экспертную оценку. Делает запрос на просмотр экспертной оценки. Из общей БД поступает экспертная оценка.
- Студент: посылает запрос на предоставление информации об успеваемости, результатах тестирования. Из общей БД поступает информация об успеваемости студента, результаты тестирования. Делает запрос на просмотр экспертной оценки, которая поступает из общей БД.

- Представитель фирмы: делает запрос на просмотр экспертной оценки, которая поступает из общей БД.

Свертка: модуль, получающий из временной БД экспертные оценки, проводит их обработку и передает обработанные данные в общую БД.

Общая база данных содержит в себе:

- экспертные оценки (предоставляются подсистемой экспертных оценок),
- успеваемость студента (предоставляется подсистемой контроля студенческой успеваемости),
- успеваемость группы (предоставляется подсистемой контроля успеваемости),
- результаты профессионального теста (предоставляются подсистемой профессиональных и психологических тестов),
- результаты психологического теста (предоставляются подсистемой профессиональных и психологических тестов).

Временная база данных содержит в себе экспертные оценки (поставленные экспертом).

### Критерии экспертных оценок

- По 5-балльной системе оценить студента в начале, середине, конце семестра.
- По 5-балльной системе оценить степень запоминания курса студентом в начале, середине, конце семестра.
- По 5-балльной системе оценить степень применения знаний студентом в начале, середине, конце семестра.

## 2. Модульная структура системы

1. Запрос, имя, уровень доступа.
2. Диаграммы, графики, списки, ведомости, количество стипендиатов, отчеты об успеваемости.
3. Имя, личные данные студента, запросы на экспертные оценки.
4. Данные студента.
5. Данные студента, данные фирмы, запросы, оценки, пароль, имя, уровень доступа.



Рис. 1.4. Модульная структура системы

6. Отчеты, резюме, графики, вопросы тестов.
7. Уровень доступа.
8. Имя.
9. Пароль.
10. Полномочия.
11. Резюме.
12. Результаты тестов.
13. Запрос.
14. Найденные записи.
15. Экспертные оценки.

### 3. Информационные объекты системы

Под информационным объектом хранения (информационным элементом) понимается логически однородная единица информации, для хранения которой достаточно одной записи таблицы.

Информационные объекты хранения для БД системы:

А. Общего назначения:

1. Факультет.
2. Учебная специальность.



3. Группа студентов.
4. Студент.
5. Преподаватель.
6. Учебный предмет (дисциплина).
7. Тип отчетного задания (курсовой проект, реферат, контрольная работа, зачет, экзамен и т. д.).
8. Отчетное задание по конкретному изучаемому предмету – описание сути отчетного задания и необходимых требований для его успешного выполнения.
9. Отчетное задание, сданное конкретным студентом, – экземпляр отчетного задания.
10. Экспертная оценка успеваемости студента – кумулятивная оценка успеваемости студента по предмету (например, за семестр).
11. Экспертная оценка успеваемости группы студентов – аналогично предыдущему, но по целой группе.
12. Контрольный вопрос для контроля студентов по изучаемым дисциплинам.
13. Ответ студента на контрольный вопрос (правильный или нет).
14. Правильный ответ контрольного вопроса.
15. Вопрос профессионального теста.
16. Ответ студента на вопрос профессионального теста (правильный или нет).
17. Правильный ответ профессионального теста.
18. Вопрос психологического теста.
19. Вариант ответа на вопрос психологического теста.
20. Правильный вариант ответа.
21. Ответ студента на вопрос психологического теста.
22. Экспертная оценка по результату профессионального теста.
23. Экспертная оценка по результату психологического теста.
24. Почтовое сообщение – сообщение по электронной почте. Сообщение, например, может уведомить студента о принятии его на работу.
25. Объявление.
26. Резюме (характеристика) студента.
27. Архивная запись по группе студентов.
28. Архивная запись по студенту.
29. Архивная запись отчетного задания, сданного студентом.
30. Архивная запись экспертных оценок. Каждая экспертная оценка соответствует отчетному заданию.
31. Архивная запись результатов профессионального теста.

32. Архивная запись результатов психологического теста.
33. График успеваемости студента (зависимость средней оценки от временного промежутка).
34. График успеваемости группы студентов.
35. Архивная запись графиков успеваемости группы студентов.

**Б. Служебные:**

1. Запись об активном соединении.
2. Запись о пользовательской транзакции.
3. Уровень доступа в систему.
4. Учетная запись полномочий пользователя (для организации защиты от несанкционированного доступа).

#### **4. Функциональные характеристики системы**

1. Первоначальный ввод информации в БД.
2. Изменение содержания БД:
  - ввод новых данных,
  - изменение существующих данных,
  - архивация данных.
3. Осуществление поиска в БД по запросу пользователя.
4. Удаленный доступ к системе по протоколу ТСР/IP.
5. Обеспечение защиты и безопасности данных, в частности:
  - разграничение прав доступа к ресурсам сервера (владелец, группа и т. д.),
  - контроль вводимой информации,
  - обеспечение целостности БД.
6. Вывод найденной информации.

#### **5. Цели и задачи системы**

Система будет обеспечивать хранение, выдачу и обновление информации системы дистанционного обучения студентов и системы «Служба занятости в рамках вуза», а именно:

- обеспечивать информационную поддержку системы дистанционного обучения студентов;
- представлять и получать накопленную информацию по конкретным объектам;
- представлять и получать информацию от подсистемы обработки запросов пользователей;
- представлять и получать информацию от подсистемы контроля студентов по изучаемым на кафедре дисциплинам;
- обеспечивать разграничение доступа к информационным ресурсам системы;
- обеспечивать мониторинг активных и пассивных пользователей и системных событий;
- обеспечивать пользователей возможностью информационного обмена;
- обеспечивать связь между фирмами и службой (институтом);
- обеспечивать регулярное прохождение студентами профессиональных и психологических тестов;
- обеспечивать поиск данных по запросам.

## 6. Категории пользователей

При работе с системой на стадиях заполнения эксплуатации БД необходимо участие следующих категорий пользователей:

- администратора БД,
- группы экспертов.

Администратор системы осуществляет заполнение БД информацией, подготовленной учебной частью, деканатом или группой экспертов. Внесение изменений в БД системы осуществляется лишь администратором системы под руководством группы экспертов. Преподаватели и студенты являются внешними пользователями, работающими с системой в соответствии с ролями доступа в информационно-поисковом режиме.

Предоставляемые возможности пользователям системы:

**студенту:**

- ввод личных анкетных данных;
- просмотр экспертных оценок по отчетным заданиям и результатам тестов;

- прохождение психологических и профессиональных тестов;
- просмотр сводных таблиц и графиков;
- получение и сдача контрольных заданий;
- доступ к справочным материалам (данные из службы удаленного обучения, а именно методическое обеспечение);
- просмотр сообщений и внесение изменений в сообщения доски объявлений;
- поиск вакансии в БД по запросу;

### **эксперту (преподавателю):**

- предоставление экспертной оценки, а также изменение ее;
- просмотр других оценок;
- просмотр программы курса и внесение изменений в нее;
- ввод контрольных заданий и назначение их студенту;
- контроль ответов на задания;
- доступ к интеллектуальным ресурсам;
- составление резюме (характеристик);

### **деканату (декану, зам. декана и т. д.):**

- просмотр программы курса;
- просмотр динамики успеваемости курса, группы, отдельного студента;
- просмотр сводных таблиц и графиков;
- просмотр экспертных оценок и характеристик преподавателей;

### **администратору:**

- определение прав доступа;
- ввод и корректировка системных данных;
- контроль работы системы;
- осуществление контроля защиты системы от несанкционированного доступа;
- изменение физической модели данных системы;

### **оператору:**

- составление сводных таблиц и графиков;
- заполнение полей БД системы (ввод информации);

**фирме:**

- поиск в БД данных о студенте по запросу;
- просмотр резюме студентов.

### 7. Контрольные вопросы

1. Для чего предназначена система «Служба занятости» и из каких подсистем она состоит?
2. Укажите цели системы.
3. Изобразите модульную структуру всей системы.
4. Перечислите основные информационные объекты системы «Служба занятости».
5. Опишите порядок занесения студента в БД.
6. Перечислите функциональные характеристики системы.
7. Какие данные поступают на вход подсистемы контроля успеваемости студентов?
8. Перечислите внешние сущности подсистемы профессиональных и психологических тестов.
9. Назовите категории пользователей системы и предоставляемые им возможности.
10. Каково назначение подсистемы обработки запросов, определения категории пользователей?
11. Опишите критерии экспертных оценок.
12. Как формируется временная БД и из чего она состоит?

# Лабораторная работа № 2

## Методология IDEF0

### Цель работы:

- изучение основных принципов методологии IDEF0,
- создание нового проекта в BPWin,
- формирование контекстной диаграммы,
- проведение связей.

Описание системы с помощью IDEF0 называется функциональной моделью. Функциональная модель предназначена для описания существующих бизнес-процессов, в котором используются как естественный, так и графический языки. Для передачи информации о конкретной системе источником графического языка является сама методология IDEF0.

Методология IDEF0 предписывает построение иерархической системы диаграмм – единичных описаний фрагментов системы. Сначала проводится описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная декомпозиция – система разбивается на подсистемы и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

Каждая IDEF0-диаграмма содержит блоки и дуги. Блоки изображают функции моделируемой системы. Дуги связывают блоки вместе и отображают взаимодействия и взаимосвязи между ними.

Функциональные блоки (работы) на диаграммах изображаются прямоугольниками, означающими поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Имя работы должно быть выражено отглагольным существительным, обозначающим действие.

IDEF0 требует, чтобы в диаграмме было не менее трех и не более шести блоков. Эти ограничения поддерживают сложность диаграмм и модели на уровне, доступном для чтения, понимания и использования.

Каждая сторона блока имеет особое, вполне определенное назначение. Левая сторона блока предназначена для входов, верхняя – для управления, правая – для выходов, нижняя – для механизмов. Такое обозначение отражает определенные системные принципы: входы преобразуются в выходы,

управление ограничивает или предписывает условия выполнения преобразований, механизмы показывают, что и как выполняет функция.

Блоки в IDEF0 размещаются по степени важности, как ее понимает автор диаграммы. Этот относительный порядок называется доминированием. Доминирование понимается как влияние, которое один блок оказывает на другие блоки диаграммы. Например, самым доминирующим блоком диаграммы может быть либо первый из требуемой последовательности функций, либо планирующая или контролирующая функция, влияющая на все другие.

Наиболее доминирующий блок обычно размещается в верхнем левом углу диаграммы, а наименее доминирующий – в правом углу.

Расположение блоков на странице отражает авторское определение доминирования. Таким образом, топология диаграммы показывает, какие функции оказывают большее влияние на остальные. Чтобы подчеркнуть это, аналитик может перенумеровать блоки в соответствии с порядком их доминирования. Порядок доминирования может обозначаться цифрой, размещенной в правом нижнем углу каждого прямоугольника: 1 будет указывать на наибольшее доминирование, 2 – на следующее и т. д.

Взаимодействие работ с внешним миром и между собой описывается в виде стрелок, изображаемых одинарными линиями со стрелками на концах. Стрелки представляют собой некую информацию и именуются существенными.

В IDEF0 различают пять типов стрелок.

**Вход** – объекты, используемые и преобразуемые работой для получения результата (выхода). Допускается, что работа может не иметь ни одной стрелки входа. Стрелка входа рисуется как входящая в левую грань работы.

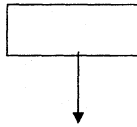
**Управление** – информация, управляющая действиями работы. Обычно управляющие стрелки несут информацию, которая указывает, что должна выполнять работа. Каждая работа должна иметь хотя бы одну стрелку управления, которая изображается как входящая в верхнюю грань работы.

**Выход** – объекты, в которые преобразуются входы. Каждая работа должна иметь хотя бы одну стрелку выхода, которая рисуется как исходящая из правой грани работы.

**Механизм** – ресурсы, выполняющие работу. Стрелка механизма рисуется как входящая в нижнюю грань работы. По усмотрению аналитика стрелки механизма могут не изображаться на модели.

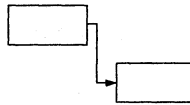
**Вызов** – специальная стрелка, указывающая на другую модель работы. Стрелка вызова рисуется как исходящая из нижней части работы и исполь-

зуется для указания того, что некоторая работа выполняется за пределами моделируемой системы.

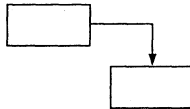


**Рис. 2.1.** *Стрелка вызова*

В методологии IDEF0 требуется только пять типов взаимодействий между блоками для описания их отношений: управление, вход, обратная связь по управлению, обратная связь по входу, выход-механизм. Связи по управлению и входу являются простейшими, поскольку они отражают прямые воздействия, которые интуитивно понятны и очень просты.



**Рис. 2.2.** *Связь по выходу*



**Рис. 2.3.** *Связь по управлению*

Отношение управления возникает тогда, когда выход одного блока непосредственно влияет на блок с меньшим доминированием.

Обратная связь по управлению и обратная связь по входу являются более сложными, поскольку представляют собой итерацию или рекурсию. А именно выходы из одной работы влияют на будущее выполнение других работ, что впоследствии повлияет на исходную работу.

Обратная связь по управлению возникает тогда, когда выход некоторого блока влияет на блок с большим доминированием.

Связи «выход-механизм» встречаются нечасто. Они отражают ситуацию, при которой выход одной функции становится средством достижения цели для другой.



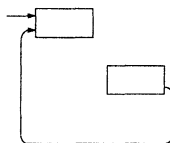


Рис. 2.4. Обратная связь по входу

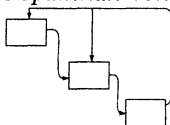


Рис. 2.5. Обратная связь по управлению

Связи «выход-механизм» характерны при распределении источников ресурсов (например, требуемые инструменты, обученный персонал, физическое пространство, оборудование, финансирование, материалы).

В IDEF0 дуга редко изображает один объект. Обычно она символизирует набор объектов. Так как дуги представляют наборы объектов, они могут иметь множество начальных точек (источников) и конечных точек (назначений). Поэтому дуги могут разветвляться и соединяться различными способами. Вся дуга или ее часть может выходить из одного или нескольких блоков и заканчиваться в одном или нескольких блоках.

Разветвление дуг, изображаемое в виде расходящихся линий, означает, что все содержимое дуг или его часть может появиться в каждом ответвлении. Дуга всегда помечается до разветвления, чтобы дать название всему набору. Кроме того, каждая ветвь дуги может быть помечена или не помечена в соответствии со следующими правилами:

- непомеченные ветви содержат все объекты, указанные в метке дуги перед разветвлением;
- ветви, помеченные после точки разветвления, содержат все объекты или их часть, указанные в метке дуги перед разветвлением.

Слияния дуг в IDEF0, изображаемое как сходящиеся вместе линии, указывает, что содержимое каждой ветви идет на формирование метки для дуги, являющейся результатом слияния исходных дуг. После слияния результирующая дуга всегда помечается для указания нового набора объектов, возникшего после объединения. Кроме того, каждая ветвь перед слиянием может помечаться или не помечаться в соответствии со следующими правилами:

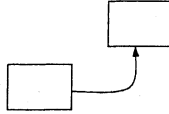


Рис. 2.6. Связь выход-механизм

- непомеченные ветви содержат все объекты, указанные в общей метке дуги после слияния;
- помеченные перед слиянием ветви содержат все или некоторые объекты из перечисленных в общей метке после слияния.

## 1. Количественный анализ диаграмм

Для проведения количественного анализа диаграмм перечислим показатели модели:

- количество блоков на диаграмме –  $N$ ;
- уровень декомпозиции диаграммы –  $L$ ;
- сбалансированность диаграммы –  $B$ ;
- число стрелок, соединяющихся с блоком, –  $A$ .

Данный набор факторов относится к каждой диаграмме модели. Далее будут перечислены рекомендации по желательным значениям факторов диаграммы.

Необходимо стремиться к тому, чтобы количество блоков на диаграммах нижних уровней было бы ниже количества блоков на родительских диаграммах, т. е. с увеличением уровня декомпозиции убывал бы коэффициент  $\frac{N}{L}$ . Таким образом, убывание этого коэффициента говорит о том, что по мере декомпозиции модели функции должны упрощаться, следовательно, количество блоков должно убывать.

Диаграммы должны быть сбалансированы. Это означает, что в рамках одной диаграммы не должно происходить ситуации, изображенной на рис. 2.7: у работы 1 входящих стрелок и стрелок управления значительно больше, чем выходящих. Следует отметить, что данная рекомендация может не выполняться в моделях, описывающих производственные процессы. Например, при описании процедуры сборки в блок может входить множество стрелок, описывающих компоненты изделия, а выходить одна стрелка – готовое изделие.

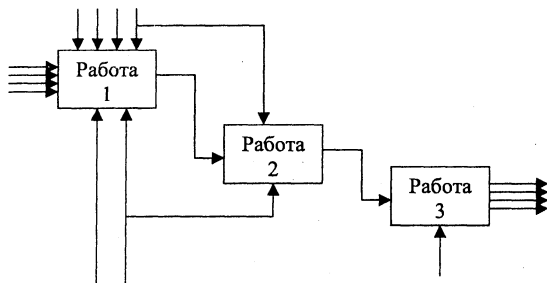


Рис. 2.7. Пример несбалансированной диаграммы

Введем коэффициент сбалансированности диаграммы:

$$K_b = \left| \frac{\sum_{i=1}^N A_i}{N} - \max_{i=1}^N (A_i) \right|.$$

Необходимо стремиться, чтобы  $K_b$  был минимален для диаграммы.

Помимо анализа графических элементов диаграммы необходимо рассматривать наименования блоков. Для оценки имен составляется словарь элементарных (тривиальных) функций моделируемой системы. Фактически в данный словарь должны попасть функции нижнего уровня декомпозиции диаграмм. Например, для модели БД элементарными могут являться функции «найти запись», «добавить запись в БД», в то время как функция «регистрация пользователя» требует дальнейшего описания.





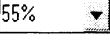




После формирования словаря и составления пакета диаграмм системы необходимо рассмотреть нижний уровень модели. Если на нем обнаружатся совпадения названий блоков диаграмм и слов из словаря, то это говорит, что достаточный уровень декомпозиции достигнут. Коэффициент, количественно отражающий данный критерий, можно записать как  $L * C$  – произведение уровня модели на число совпадений имен блоков со словами из словаря. Чем ниже уровень модели (больше  $L$ ), тем ценнее совпадения.

## 2. Инструментарий BPWin

При запуске BPWin по умолчанию появляется основная панель инструментов, палитра инструментов и Model Explorer.

Функциональность панели инструментов доступна из основного меню BPWin (табл. 2.1).

Таблица 2.1. Описание элементов управления основной панели инструментов BPWin 2.5

Элемент управления	Описание	Соответствующий пункт меню
	Создать новую модель	File->New
	Открыть модель	File->Open
	Сохранить модель	File->Save
	Напечатать модель	File->Print
	Выбор масштаба	View->Zoom
	Масштабирование	View->Zoom
	Проверка правописания	Tools->Spelling
	Включение и выключение навигатора модели Model Explorer	View->Model Explorer
	Включение и выключение дополнительной панели инструментов работы с ModelMart	ModelMart

При создании новой модели возникает диалог, в котором следует указать, будет ли создана модель заново, или она будет открыта из репозитория ModelMart, внести имя модели и выбрать методологию, в которой будет построена модель (рис. 2.8).

BPWin поддерживает три методологии – IDEF0, IDEF3 и DFD. В BPWin возможно построение смешанных моделей, т. е. модель может содержать одновременно как диаграммы IDEF0, так и IDEF3 и DFD. Состав палитры инструментов изменяется автоматически, когда происходит переключение с одной нотации на другую.

Модель в BPWin рассматривается как совокупность работ, каждая из которых оперирует с некоторым набором данных. Если щелкнуть по любому объекту модели левой кнопкой мыши, появляется всплывающее контекстное меню, каждый пункт которого соответствует редактору какого-либо свойства объекта.

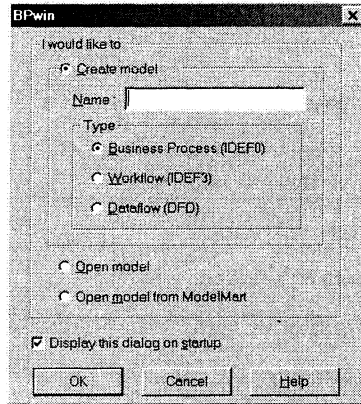











Рис. 2.8. Диалог создания модели

Для построения диаграмм в BPWin используется три панели инструментов для каждого типа диаграмм.




### Диаграммы IDEF0

-  – кнопка для добавления работы на диаграмму.
-  – проведение новой связи.
-  – инструмент редактирования объектов.
-  – декомпозиция диаграммы.

### Диаграммы DFD

-  – добавление в диаграмму внешней ссылки.
-  – добавление стрелки потока данных в диаграмму.
-  – декомпозиция диаграммы.
-  – добавление хранилища данных.
-  – ссылка на другую страницу. Этот инструмент позволяет направить стрелку на любую диаграмму.

### Диаграммы IDEF3

-  – «старшая» связь.
-  – перекресток.
-  – объект ссылки.

### 3. Пример

Построение модели системы должно начинаться с изучения всех документов, описывающих ее функциональные возможности. Одним из таких документов является техническое задание, а именно разделы «Назначение разработки», «Цели и задачи системы» и «Функциональные характеристики системы».

После изучения исходных документов и опроса заказчиков и пользователей системы необходимо сформулировать цель моделирования и определить точку зрения на модель. Рассмотрим технологию ее построения на примере системы «Служба занятости в рамках вуза», основные возможности которой были описаны в лабораторной работе № 1.

Сформулируем цель моделирования: описать функционирования системы, которое было бы понятно ее пользователю, не вдаваясь в подробности, связанные с реализацией. Модель будем строить с точки зрения пользователей (студент, преподаватель, администратор, деканат, фирма).

Начнем с построения контекстной IDEF0-диаграммы. Согласно описанию системы основной функцией является обслуживание ее клиентов посредством обработки запросов, от них поступающих. Таким образом, определим единственную работу контекстной диаграммы как «Обслужить клиента системы». Далее определим входные и выходные данные, а также механизмы и управление.

Для того чтобы обслужить клиента, необходимо зарегистрировать его в системе, открыть доступ к БД и обработать его запрос. В качестве входных данных будут использоваться «имя клиента», «пароль клиента», «исходная БД», «запрос клиента». Выполнение запроса ведет либо к получению информации от системы, либо к изменению содержимого БД (например, при составлении экспертных оценок), поэтому выходными данными будут являться «отчеты» и «измененная БД». Процесс обработки запросов будет выполняться монитором системы под контролем администратора.

Таким образом, определим контекстную диаграмму системы (рис. 2.9).

Проведем декомпозицию контекстной диаграммы, описав последовательность обслуживания клиента:

1. Определение уровня доступа в систему.
2. Выбор подсистемы.
3. Обращение к подсистеме.
4. Изменение БД (при необходимости).

Получим диаграмму, изображенную на рис. 2.10.

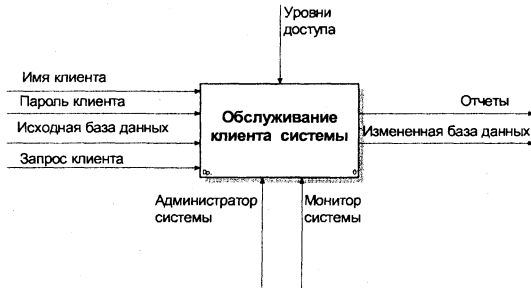


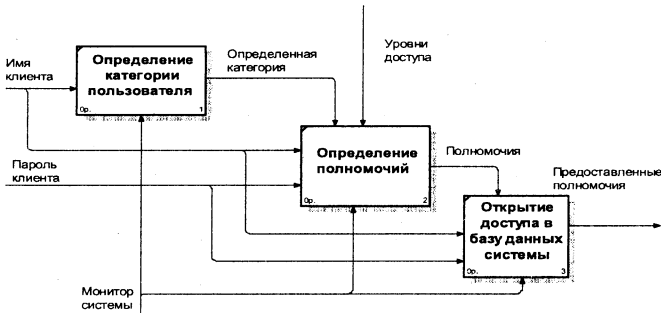
Рис. 2.9. Контекстная диаграмма системы

Закончив декомпозицию контекстной диаграммы, переходят к декомпозиции диаграммы следующего уровня. Обычно при рассмотрении третьего и более нижних уровней модели возвращаются к родительским диаграммам и корректируют их.



Рис. 2.10. Декомпозиция работы «Обслуживание клиента системы»

Декомпозируем последовательно все блоки полученной диаграммы. Первым этапом при определении уровня доступа в систему является определение категории пользователя. По имени клиента осуществляется поиск в базе пользователей, определяя его категорию. Согласно определенной категории выясняются полномочия, предоставляемые пользователю системы. Далее проводится процедура доступа в систему, проверяя имя и пароль доступа. Объединяя информацию о полномочиях и уровне доступа в систему, для пользователя формируется набор разрешенных действий. Таким образом, определение уровня доступа в систему будет выглядеть как показано на рис. 2.11.



**Рис. 2.11.** Декомпозиция работы «Определение уровня доступа в систему»

После прохождения процедуры доступа в систему монитор анализирует запрос клиента, выбирая подсистему, которая будет обрабатывать запрос. Декомпозиция работы «Обращение к подсистеме» не отвечает цели и точке зрения модели. Пользователя системы не интересуют внутренние алгоритмы ее работы. В данном случае ему важно, что выбор подсистемы будет произведен автоматически, без его вмешательства, поэтому декомпозиция обращения к подсистеме только усложнит модель.

Декомпозируем работу «Обработка запроса клиента», выполняемую подсистемой обработки запросов, определения категорий и полномочий пользователей. Перед осуществлением поиска ответа на запрос необходимо открыть БД (подключиться к ней). В общем случае БД может находиться на удаленном сервере, поэтому может потребоваться установление соединения с ней. Определим последовательность работ:

1. Открытие БД.
2. Выполнение запроса.
3. Генерация отчетов.

После открытия БД необходимо сообщить системе об установлении соединения с БД, после чего выполнить запрос и сгенерировать отчеты для пользователя (рис. 2.12).

Необходимо отметить, что в «Выполнение запроса» включается работа различных подсистем. Например, если запрос включает в себя тестирование, то его будет исполнять подсистема профессиональных и психологических тестов. На этапе выполнения запроса может потребоваться изменение





Рис. 2.12. Декомпозиция работы «Обработка запроса клиента»

содержимого БД, например при составлении экспертных оценок. Поэтому, на диаграмме необходимо предусмотреть такую возможность.

При анализе полученной диаграммы возникает вопрос, по каким правилам происходит генерация отчетов? Необходимо наличие заранее сформированных шаблонов, по которым будет производиться выборка из БД, причем эти шаблоны должны соответствовать запросам и должны быть заранее определены. Кроме того, клиенту должна быть предоставлена возможность выбора формы отчета.

Скорректируем диаграмму, добавив в нее стрелки «Шаблоны отчетов» и «Запросы на изменение БД» и туннельную стрелку «Клиент системы». Туннелирование «Клиента системы» применено для того, чтобы не выносить стрелку на диаграмму верхнего, так как функция выбора формы отчета не является достаточно важной для отображения ее на родительской диаграмме.

Изменение диаграммы потянет за собой корректировку всех родительских диаграмм (рис. 2.13 – 2.15).

Декомпозицию работы «Выполнение запроса» целесообразно провести при помощи диаграммы DFD (лабораторная работа № 3), так как методология IDEF0 рассматривает систему как совокупность взаимосвязанных работ, что плохо отражает процессы обработки информации.

Перейдем к декомпозиции последнего блока «Изменение БД». С точки зрения клиента, данные системы располагаются в одной БД. Реально в системе присутствует шесть БД:

- БД пользователей,
- БД студентов,

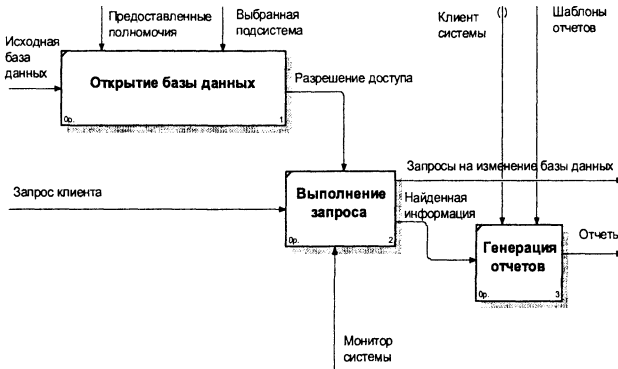


Рис. 2.13. Декомпозиция работы «Обработка запроса клиента» (вариант 2)

- БД вакансий,
- БД успеваемости,
- БД тестов,
- БД экспертных оценок,
- БД резюме.

Согласно цели моделирования клиенту важно понимать, что поступившие данные не сразу обновляются в системе, а проходят дополнительный этап обработки и контроля. Алгоритм изменения можно сформулировать следующим образом:

1. Определяется БД, в которой будет изменяться информация.
2. Оператором формируется временный набор данных и предоставляется администратору.
3. Администратор осуществляет контроль данных и вносит их в БД.

Данную модель реализовать другим способом, предоставив возможность обновления БД непосредственно по запросам, минуя процесс контроля данных. В этом случае необходимо обеспечить контроль целостности БД для избежания ее повреждения. В этом случае диаграмма будет выглядеть следующим образом (рис. 2.17).

Проведение дальнейшей декомпозиции «Изменения БД» будет усложнять модель, объясняя, как осуществляется физическое изменение БД в системе. При этом пользователь не получит никакой дополнительной ин-

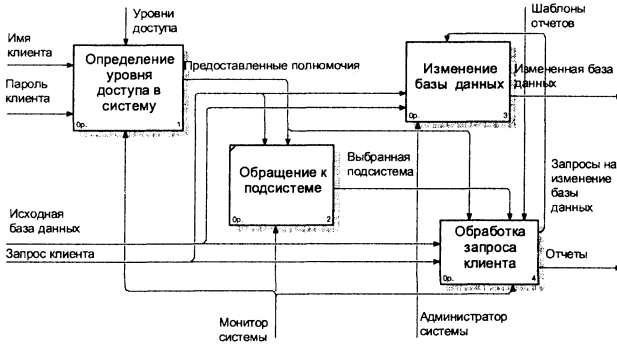


Рис. 2.14. Декомпозиция работы «Обслуживание клиента системы» (вариант 2)

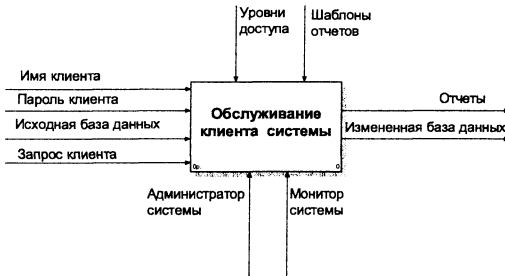


Рис. 2.15. Контекстная диаграмма системы (вариант 2)

формации о работе системы службы занятости. Декомпозицию этой работы целесообразно проводить в процессе проектирования БД системы на этапе создания логической модели БД.

Декомпозиция работы «Выполнение запроса» будет проведена в следующей лабораторной работе, иллюстрируя применение диаграмм DFD для описания процессов обработки информации.

Проведем количественный анализ моделей, изображенных на рис. 2.12 и 2.13, согласно вышеописанной методике. Рассмотрим поведение коэффициента  $\frac{N}{L}$  у этих моделей. У родительской диаграммы «Обработка запроса клиента» коэффициент равен  $4/2 = 2$ , а диаграммы декомпозиции  $3/3 = 1$ . Значение коэффициента убывает, что говорит об упрощении описания функций с понижением уровня модели.

Рассмотрим изменение коэффициента  $K_6$  у двух вариантов моделей.

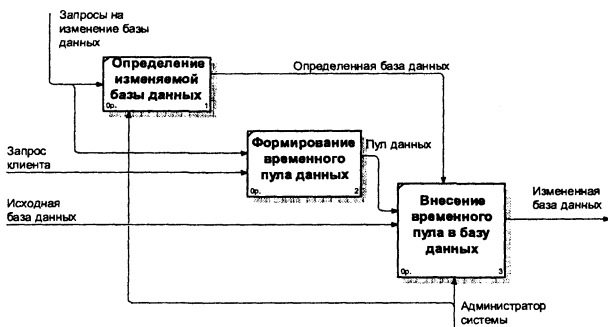


Рис. 2.16. Декомпозиция работы «Изменение БД»

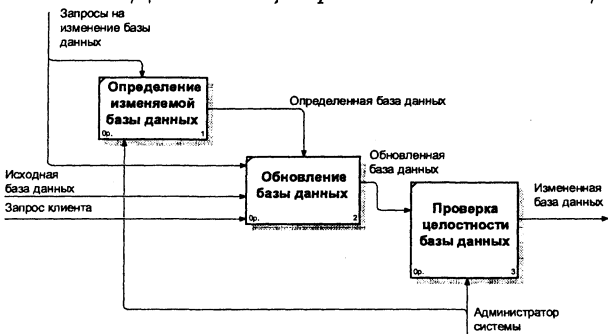


Рис. 2.17. Декомпозиция работы «Изменение БД» (вариант 2)

Для первого варианта, изображенного на рис. 2.12,

$$K_b = \left| \frac{10}{3} - 4 \right| = 0,66,$$

для второго варианта

$$K_b = \left| \frac{13}{5} - 5 \right| = 0,66.$$

Коэффициент  $K_b$  не меняет своего значения, следовательно, сбалансированность диаграммы не меняется.

Будем считать, что уровень декомпозиции рассмотренных диаграмм достаточен для отражения цели моделирования, и на диаграммах нижнего уровня в качестве наименований работ используются элементарные функции (с точки зрения пользователя системы).

Подводя итоги рассмотренного примера необходимо отметить важность рассмотрения нескольких вариантов диаграмм при моделировании системы. Такие варианты могут возникать при корректировке диаграмм, как это было сделано с «Обработкой запроса клиента» или при создании альтернативных реализаций функций системы (декомпозиция работы «Изменение БД»). Рассмотрение вариантов позволяет выбрать наилучший и включить его в пакет диаграмм для дальнейшего рассмотрения.

### 4. Задания

1. Создать новый проект в BPWin.
2. Сформировать контекстную диаграмму по системе согласно методологии IDEF0.
3. Задать входы, выходы, механизмы и управление.
4. Декомпонировать контекстную диаграмму.
5. Провести связи по выходу.
6. Провести связи по управлению.
7. Провести связи по входу.
8. Сохранить проект в отдельный файл.

### 5. Контрольные вопросы

1. Что представляет собой модель в нотации IDEF0?
2. Что обозначают работы в IDEF0?
3. Назовите порядок наименования работ?
4. Какое количество работ должно присутствовать на одной диаграмме?
5. Что называется порядком доминирования?
6. Как располагаются работы по принципу доминирования?
7. Каково назначение сторон прямоугольников работ на диаграммах?
8. Перечислите типы стрелок.
9. Назовите виды взаимосвязей.
10. Что называется граничными стрелками?
11. Объясните принцип именования разветвляющихся и сливающихся стрелок.
12. Какие методологии поддерживаются BPWin?
13. Перечислите основные элементы главного окна BPWin.
14. Опишите процесс создания новой модели в BPWin.

15. Как провести связь между работами?
16. Как задать имя работы.
17. Опишите процесс декомпозиции работы.
18. Как добавить работу на диаграмму?
19. Как разрешить туннелированные стрелки?
20. Может ли модель VPWin содержать диаграммы нескольких методологий?

# Лабораторная работа № 3

## Дополнение моделей процессов диаграммами DFD и WorkFlow (IDEF3)

### Цель работы:

- построение диаграмм потоков данных (DFD),
- описание взаимосвязей между процессами при помощи диаграмм IDEF3 (WorkFlow).

### 1. Диаграммы потоков данных (Data Flow Diagrams)

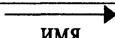
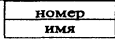
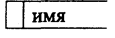
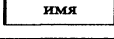
Эти диаграммы представляют сеть связанных между собой работ. Их удобно использовать для описания документооборота и обработки информации.

DFD описывает:

1. функции обработки информации (работы);
2. документы (стрелки, аггов), объекты, сотрудников или отделы, которые участвуют в обработке информации;
3. внешние ссылки (external reference), которые обеспечивают интерфейс с внешними объектами, находящимися за границами моделируемой системы;
4. таблицы для хранения документов (хранилища данных, data store).

Для построения диаграмм DFD в BPWin используется нотация Гейна – Сарсона.

Таблица 3.1. Нотация Гейна – Сарсона

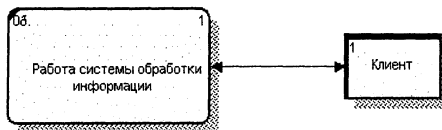
Компонент	Обозначение
Поток данных	
Процесс	
Хранилище	
Внешняя сущность	

Потоки данных являются механизмами, используемыми для моделирования передачи информации (или физических компонентов) из одной части системы в другую. Потоки изображаются на диаграмме именованными стрелками, ориентация которых указывает направление движения информации. Стрелки могут подходить к любой грани прямоугольника работы и могут быть двунаправленными для описания взаимодействия типа «команда-ответ».

Назначение процесса состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Каждый процесс должен иметь уникальный номер для ссылок на него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

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

Внешняя сущность представляет сущность вне контекста системы, являющуюся источником или приемником данных системы. Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке. Внешние сущности изображаются в виде прямоугольника с тенью и обычно располагаются по краям диаграммы. Одна внешняя сущность может быть использована многократно на одной или нескольких диаграммах.



**Рис. 3.1.** Внешняя сущность

Для дополнения модели IDEF0 диаграммой DFD нужно в процессе декомпозиции в диалоге Activity Box Count указать тип диаграммы DFD.



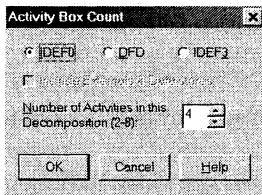


Рис. 3.2.

## 2. Диаграммы IDEF3

Диаграммы IDEF3 также называют WorkFlow diagramming – методологией моделирования, использующей графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы WorkFlow используются для анализа процедур обработки информации.

Цель IDEF3 – дать аналитикам описание последовательности выполнения процессов, а также объектов, участвующих совместно в одном процессе.

IDEF3 может быть также использован как метод создания процессов. IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые могут быть использованы для имитационного моделирования.

### Диаграммы

Диаграмма является основной единицей описания в IDEF3-модели. Организация диаграмм в IDEF3 является наиболее важной, если модель редактируется несколькими людьми. В этом случае разработчик должен определять, какая информация будет входить в ту или иную модель.

Единицы работы – Unit of Work (UOW), также называемые работами, являются центральными компонентами модели. В IDEF3 работы изображаются прямоугольниками и имеют имя, обозначающее процесс действия и номер (идентификатор). В имя обычно включается основной результат работы (например, приготовление обеда).

### Связи


Показывают взаимоотношения работ. Все связи в IDEF3 являются односторонними.

Старшая (Precedence) линия – сплошная линия (—→), связывающая единицы работ. Рисуеться слева направо или сверху вниз. Показывает, что работа-источник должна закончиться прежде, чем работа-цель начнется.

Линия отношения (Relation Link) – пунктирная линия (---), используемая для изображения связей между единицами работ, а также между единицами работ и объектами ссылок.

Потоки объектов (Object Flow) – стрелка с двумя наконечниками (——→), применяется для описания использования объекта в двух или более единицах работы, например когда объект порождается в одной работе и используется в другой.


Перекрестки (Junction) – используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы.

Различают перекрестки для слияния (Fan-in Junction) и разветвления (Fanout Junction) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. Для внесения перекрестка служит кнопка .


**Таблица 3.2. Типы перекрестков**

Обозначение	Наименование	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
	Asynchronous AND	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
	Synchronous AND	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно

Таблица 3.2. Типы перекрестков

Обозначение	Наименование	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
	XOR (Exclusive OR)	Только один процесс завершен	Только один следующий процесс запускается

Объекты-ссылки – являются специальными символами, которые ссылаются на внешние части описания процесса. Они добавляются на диаграмму для того, чтобы обратить внимание редактора на что-либо важное, что невозможно связать со стрелкой, работой или перекрестком.

Для внесения объекта-ссылки служит кнопка . Объект-ссылка отображается в виде прямоугольника. Объекты-ссылка должны быть связаны с единицами работ или перекрестками пунктирными линиями.

При внесении объектов-ссылок необходимо указать их тип.

Таблица 3.3. Типы объектов-ссылок

Тип объекта-ссылки	Цель описания
OBJECT	Описывает участие важного объекта в работе
GOTO	Инструмент циклического перехода (в повторяющейся последовательности работ), возможно на текущей диаграмме, но не обязательно. Если все работы цикла присутствуют на текущей диаграмме, цикл может также изображаться стрелкой, возвращающейся на стартовую работу. GOTO может ссылаться на перекресток
UOB (Unit of behavior)	Применяется, когда необходимо подчеркнуть множественное использование какой-либо работы, но без цикла. Например, работа «Контроль качества» может быть использована в процессе «Изготовления изделия» несколько раз, после каждой единичной операции. Обычно этот тип ссылки не используется для моделирования автоматически запускающихся работ

Таблица 3.3. Типы объектов-ссылок

Тип объекта-ссылки	Цель описания
NOTE	Используется для документирования важной информации, относящейся к каким-либо графическим объектам на диаграмме. NOTE является альтернативой внесению текстового объекта в диаграмму
ELAB (Elaboration)	Используется для усовершенствования графиков или их более детального описания. Обычно употребляется для детального описания разветвления и слияния стрелок на перекрестках

### 3. Пример

#### 3.1. Диаграммы DFD

Диаграммы DFD можно использовать как дополнение к диаграммам IDEF0 для описания документооборота и обработки информации. Рассмотрим работу «Обработка запросов клиента» из лабораторной работы № 1.

Запросы в систему поступают от пользователей, поэтому запросы от каждой категории будут обрабатываться отдельно. Выделим внешние сущности диаграммы согласно каждой категории пользователей, определяя потоки данных, которыми они обмениваются с системой. Получим диаграмму, изображенную на рис. 3.3.

Согласно описанию системы проведем декомпозицию полученных блоков (рис. 3.4 – 3.7).

Все процессы обработки запросов контролируются и выполняются монитором системы, поэтому стрелка-механизм «Монитор системы» будет повторяться на декомпозированных диаграммах. Точка зрения модели, определенная в лабораторной работе № 1, не требует рассмотрения внутренних особенностей функционирования системы, поэтому затуннелируем стрелку «Монитор системы» с тем, чтобы не переносить ее на диаграммы нижнего уровня.

Проведем анализ полученных диаграмм. Рассматривая диаграмму, изображенную на рис. 3.3, необходимо отметить наличие в ней лишнего блока «Обработать запрос администратора». При составлении первого варианта диаграммы работы были определены исходя из категорий пользователей.

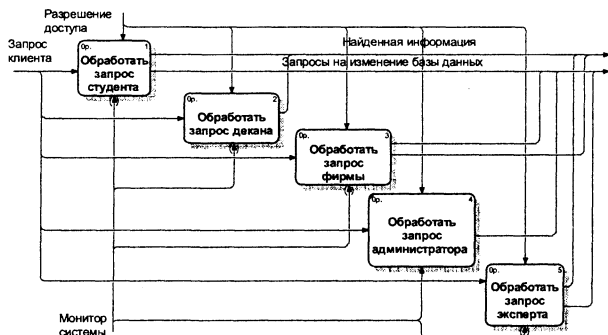


Рис. 3.3. DFD-декомпозиция работы «Выполнение запроса»



Рис. 3.4. Декомпозиция работы «Обработать запрос студента»

Это привело к возникновению конфликта с функциями системы и с точкой зрения на модель. Администратор не обслуживается системой как обычный клиент, он обеспечивает ее мониторинг. Администратор может добавлять пароли, изменять уровень доступа в систему, добавлять нового пользователя и т. д. С точки зрения клиента системы, деятельность администратора является второстепенной, поэтому на всех диаграммах отсутствует описание функций администратора, представляя его влияние как «механизм» для других функций. Поэтому принимаем решение удалить работу «Обработать запрос администратора».

Перейдем к анализу диаграммы декомпозиции работы «Обработать запрос студента». Согласно рис. 2.13, описывающему процесс обработки запроса, после выполнения запроса происходит генерация отчета выбранной пользователем формы и только затем просмотр полученных данных. На диаграмме «Обработать запрос студента» функция начинается со сло-

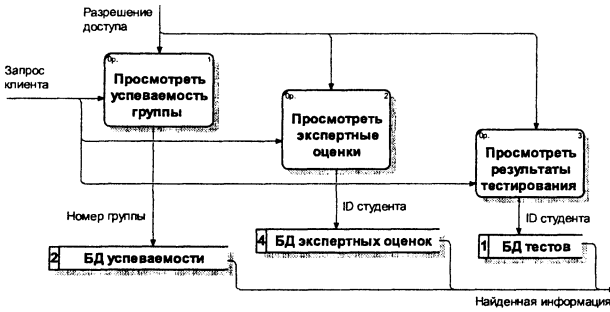


Рис. 3.5. Декомпозиция работы «Обработать запрос декана»



Рис. 3.6. Декомпозиция работы «Обработать запрос фирмы»

ва «просмотреть». Возникает противоречие. В процессе обработки запроса студента данные должны быть найдены в БД и переданы другому модулю, генерирующему отчеты. Изменим название работы на диаграмме, уточнив при этом потоки данных, составляющие дугу «Найденная информация». Получим второй вариант диаграммы (рис. 3.9).

Аналогичное противоречие наблюдается и в диаграмме на рис. 3.5. Кроме того, из хранилищ, изображающих БД выходит стрелка «Найденная информация», в то время как эта стрелка должна выходить прямо из работ, передавая данные для генерации отчета. Обращение к БД требуется в случае, когда информация необходима для выполнения функции внутри работы. Скорректируем диаграмму, получив новый вариант (рис. 3.10).

Перейдем к следующей диаграмме – «Обработать запрос эксперта» (рис. 3.7). Корректировок эта диаграмма не требует, кроме уточнения названия функций. Перед проведением экспертной оценки эксперт должен

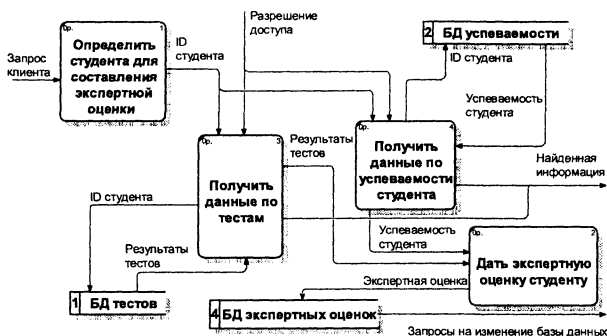


Рис. 3.7. Декомпозиция работы «Обработать запрос эксперта»

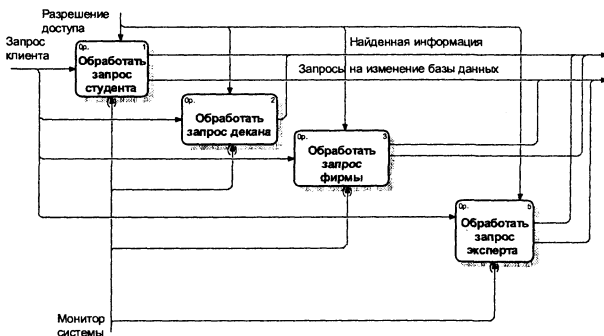


Рис. 3.8. DFD-декомпозиция работы «Выполнение запроса» (вариант 2)

выбрать студента. Термин «определить» может подразумевать вмешательство эксперта, при котором он на основании каких-либо предпочтений выбирает студента для проведения оценки. На самом деле студент просто выбирается из общего списка. Хотя здесь могут существовать внешние факторы, например требование фирмы на составление экспертной оценки для студента. Эта ситуация должна отражаться на диаграмме «Обработать запрос фирмы», т. е. необходимо внести дополнительную работу «Найти экспертные оценки студента». Причем если такие оценки не найдены, то необходимо генерировать запрос для эксперта на их составление.

Помимо указанных недостатков, диаграмма «Обработать запрос фирмы» имеет еще несколько. Можно объединить работы «Составить приглашение» и «Отослать студенту» в одну работу «Составить и отослать приглашение», упростив диаграмму.

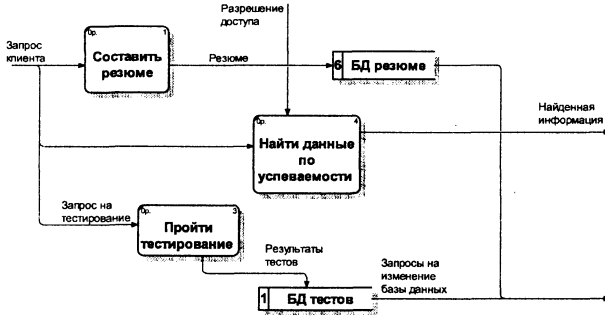


Рис. 3.9. Декомпозиция работы «Обработать запрос студента» (вариант 2)

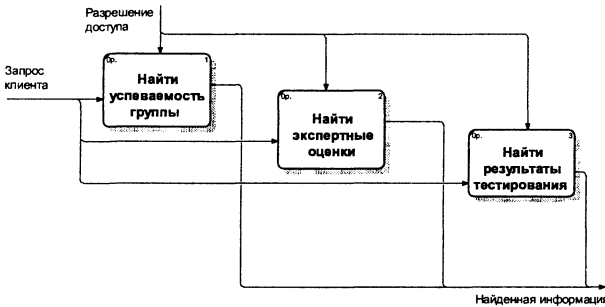


Рис. 3.10. Декомпозиция работы «Обработать запрос декана» (вариант 2)

Проведем анализ вариантов диаграммы «Выполнение запроса», изображенных на рис. 3.8 и 3.12, по методике, описанной в лабораторной работе № 2. Во втором варианте сократилось число блоков, упростив диаграмму. Таким образом, уменьшился коэффициент  $\frac{N}{L}$ .

Рассчитаем и сравним коэффициенты сбалансированности диаграмм. Для первого варианта:

$$K_b = \left| \frac{5+4+5+4+5}{5} - 5 \right| = 0,4;$$

для второго варианта

$$K_b = \left| \frac{5+4+5+5}{5} - 5 \right| = 0,25.$$



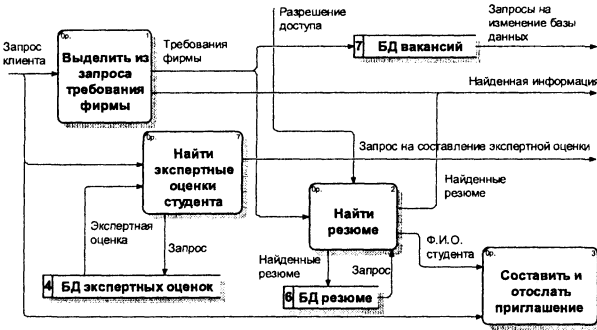


Рис. 3.11. Декомпозиция работы «Обработать запрос фирмы» (вариант 2)

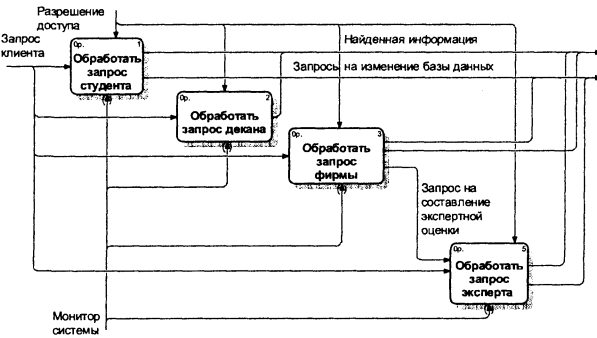


Рис. 3.12. DFD-декомпозиция работы «Выполнение запроса» (вариант 2)

Уменьшение коэффициента  $K_b$  свидетельствует о более равномерном распределении потоков данных между блоками модели.

### 3.2. Диаграммы IDEF3

С помощью диаграмм IDEF3 обычно моделируют последовательности работ, имеющие технологические и временные связи. К таким моделям можно отнести проект разработки системы службы занятости, который и будет рассмотрен в данном примере.

Перед началом моделирования необходимо создать иерархическую структуру работ, описывающую процесс разработки системы.

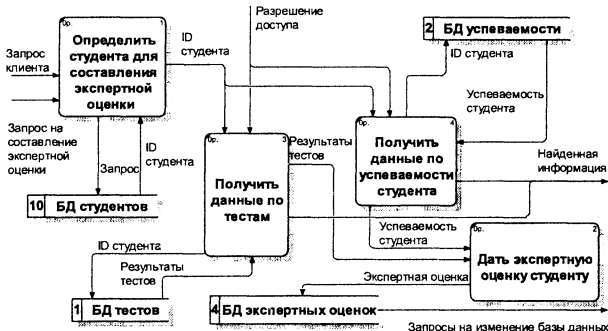


Рис. 3.13. Декомпозиция работы «Обработать запрос эксперта» (вариант 3)

1. Разработка технического задания.
  - (a) Составление технического задания.
  - (b) Утверждение технического задания.
2. Анализ.
  - (a) Определение объектов системы и их атрибутов.
  - (b) Определение категорий пользователей.
  - (c) Создание запросов к системе.
3. Разработка модульной структуры.
  - (a) Разработка модульной структуры всей системы.
  - (b) Разработка модульной структуры подсистемы обработки запросов, определения категории пользователей.
  - (c) Разработка модульной структуры подсистемы экспертных оценок.
  - (d) Разработка модульной структуры подсистемы профессиональных и психологических тестов.
  - (e) Разработка модульной структуры контроля успеваемости студентов.
4. Проектирование БД.
  - (a) Проектирование логической структуры БД.

- (b) Проектирование физической структуры БД.
- (c) Определение взаимосвязей между БД.
- (d) Выбор СУБД.

Согласно созданной структуре работ определим диаграммы, добавив на них взаимосвязи между работами.



Рис. 3.14. Диаграмма «Разработка системы службы занятости»

На стадии разработки технического задания заказчик системы играет важную роль, снабжая разработчиков необходимой информацией для создания системы. Поэтому на диаграмме показан соответствующий объект-ссылка, влияющий на работу «Разработка технического задания».

Проведем декомпозицию работ по созданию службы занятости, ориентируясь на созданную структуру работ.

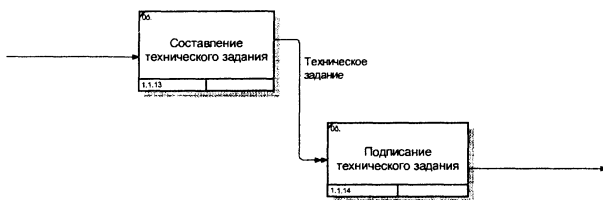


Рис. 3.15. Декомпозиция работы «Разработка технического задания»

Полученные диаграммы описывают процесс создания системы службы занятости на основе структуры работ по процессам. Обычно для более точного описания проекта создают несколько структур. В данном случае полезно создать структуру «по подсистемам», описав работы, необходимые для создания конкретных подсистем службы занятости.

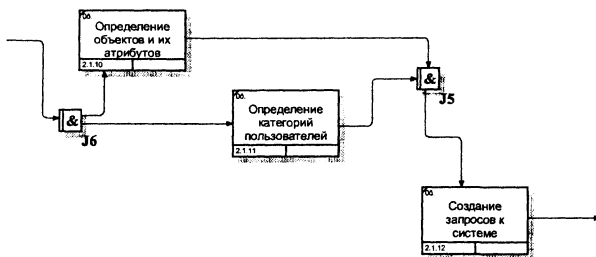


Рис. 3.16. Декомпозиция работы «Анализ»

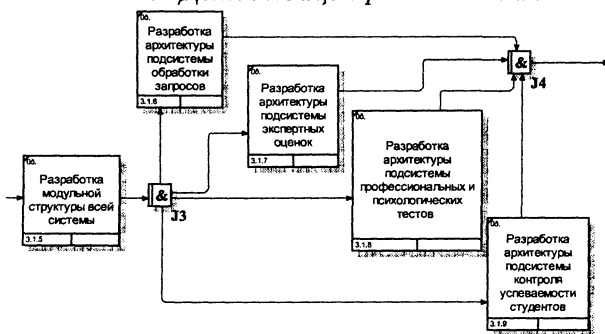


Рис. 3.17. Декомпозиция работы «Разработка модульной структуры»

Структура работ по подсистемам:

1. Разработка технического задания.
  - (а) Составление технического задания.
  - (б) Подписание технического задания.
2. Разработка подсистемы профессиональных и психологических тестов.
  - (а) Определение межсистемных соглашений.
  - (б) Определение объектов и их атрибутов.
  - (с) Определение категорий пользователей.
  - (д) Создание запросов к системе.
  - (е) Проектирование структуры БД.

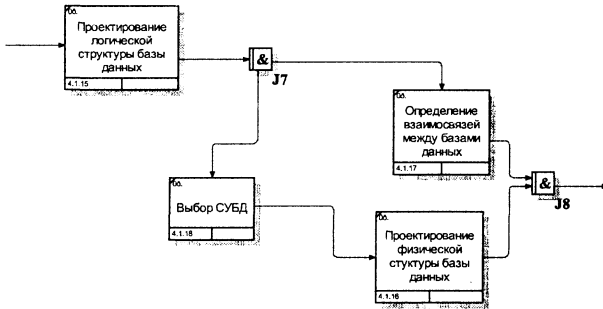


Рис. 3.18. Декомпозиция работы «Проектирование БД»

3. Разработка подсистемы обработки запросов. Определение межсистемных соглашений.
  - (а) Определение межсистемных соглашений.
  - (б) Определение объектов и их атрибутов.
  - (с) Определение категорий пользователей.
  - (д) Создание запросов к системе.
  - (е) Проектирование структуры БД.
4. Разработка подсистемы экспертных оценок.
  - (а) Определение межсистемных соглашений.
  - (б) Определение объектов и их атрибутов.
  - (с) Определение категорий пользователей.
  - (д) Создание запросов к системе.
  - (е) Проектирование структуры БД.
5. Разработка подсистемы контроля успеваемости студентов.
  - (а) Определение межсистемных соглашений.
  - (б) Определение объектов и их атрибутов.
  - (с) Определение категорий пользователей.
  - (д) Создание запросов к системе.
  - (е) Проектирование структуры БД.

6. Разработка архитектуры всей системы.
7. Объединение подсистем.
  - (а) Проверка соблюдения межсистемных соглашений.
  - (б) Определение взаимосвязей между БД.

При формировании структуры операций «по подсистемам» обнаружилась возможность создания типового фрагмента проектирования подсистемы, включающего один и тот же перечень работ. Такой подход часто упрощает описание проектов, позволяя формировать проекты любой сложности из небольших фрагментов. Выделим полученный типовой фрагмент в отдельную диаграмму (рис. 3.22).

Создадим пакет диаграмм, соответствующий структуре работ «по подсистемам».

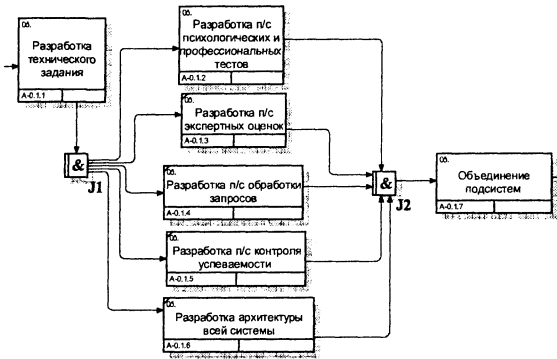


Рис. 3.19. Диаграмма «Разработка системы службы занятости» (вариант 2)

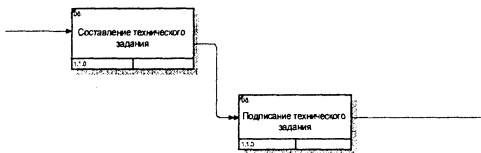


Рис. 3.20. Декомпозиция работы «Разработка технического задания» (вариант 2)

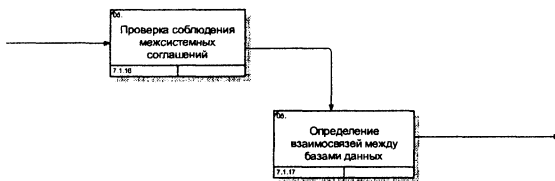


Рис. 3.21. Декомпозиция работы «Объединение подсистем»

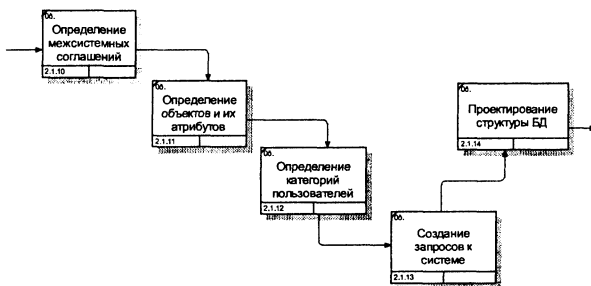


Рис. 3.22. Типовой фрагмент «Разработка подсистемы»

#### 4. Задания

1. Дополнить созданную на предыдущей работе диаграмму IDEF0 диаграммой DFD.
2. Добавить на диаграмму DFD внешнюю сущность и хранилище данных.
3. Связать диаграмму и внешнюю сущность.
4. Связать диаграмму и хранилище.
5. Определить имя связи с внешней сущностью.
6. Создать диаграмму IDEF3, определяющую последовательность выполнения БД системы.
7. Связать работы на диаграмме.
8. Добавить на диаграмму перекрестки, моделирующие параллельные события при заполнении БД.
9. Добавить объект-ссылку и связать его с диаграммой.

## 5. Контрольные вопросы

1. Что описывает диаграмма DFD?
2. Какая нотация используется в BPWin для построения диаграмм DFD?
3. Что описывает диаграмма IDEF3?
4. Перечислите составные части диаграммы DFD.
5. В чем состоит назначение процесса?
6. Что называется внешней сущностью?
7. Что описывают хранилища?
8. Объясните механизм дополнения диаграммы IDEF0 диаграммой DFD.
9. Перечислите составные элементы диаграмм IDEF3.
10. Что показывают связи в диаграммах IDEF3?
11. Перечислите типы стрелок в диаграммах IDEF3.
12. Что называется перекрестком?
13. Назовите типы перекрестков.
14. Что называется объектом-ссылкой?
15. Какие бывают типы объектов-ссылок?
16. Как добавить объект-ссылку?



# Лабораторная работа № 4

## Отчеты в BPWin

### Цель работы:

- изучить виды отчетов и способы их создания;
- освоить метод поиска ошибок в диаграммах, используя отчеты.

BPWin позволяет создавать следующие типы отчетов:

- отчет по модели (Model Report) – включает в себя всю информацию о модели, созданной в BPWin (IDEF0, IDEF3 или DFD);
- отчет о диаграмме (Diagram Report) – включает в себя информацию обо всех объектах, входящих в активную диаграмму BPWin;
- отчет об объектах диаграммы (Diagram Object Report) – содержит полный список объектов, таких, как работы, хранилища, внешние ссылки, с указанием их свойств;
- отчет о стоимостях работ (Activity Cost Report) – содержит данные о стоимостях работ и стоимостных центрах модели;
- отчет о стрелках (Arrow Report) – включает в себя информацию о стрелках и связях модели;
- отчет об использовании данных (Data Usage Report) – содержит информацию о таблицах БД, сущностях и атрибутах, сопоставленных с работами модели, а также действия, которые могут быть произведены над ними;
- отчет согласованности с методологией (Model Consistency Report) – показывает насколько активная IDEF0-модель соответствует выбранной методологии.

Вышеперечисленные отчеты вызываются выбором соответствующего подпункта из меню Reports главного окна. При этом открывается диалоговое окно для задания параметров формируемого отчета.

Каждый полученный отчет может быть открыт в режиме просмотра (кнопка Preview), распечатан (кнопка Print) или сохранен в файл (кнопка Report).

### 1. Создание отчета по модели

1. Откройте модель, по которой вы собираетесь создавать отчет.

2. Выберите Model Report из меню Report главного окна. При этом откроется диалог отчета по модели.
3. Установите в открывшемся окне опции согласно пунктам, которые будут включены в отчет. Порядок включения отображается рядом с флажком.

Model Name – название модели.

Definition – цель бизнес-процессов модели.

Score – предметная область модели.

View point – точка зрения на модель.

Time frame – временные рамки модели.

Status – степень готовности модели.

Purpose – цель создания модели.

Source – источник, на основании которого создается модель.

Author name – автор модели.

Creation date – дата создания.

System last revision date – дата последнего просмотра в системе.

User last revision date – дата последнего просмотра пользователем.

4. Выберите форму представления отчета (Preview, Print, Report).

## 2. Создание отчета по диаграмме

При создании этого типа отчета необходимо обращать внимание на методологию диаграммы, поскольку в зависимости от этого производится настройка параметров отчета.

- Для диаграмм IDEF0 параметры задаются в рамках Activity options и Link options. Параметры в других рамках не имеют смысла. Например, группа параметров для хранилищ данных (Data Store) не имеет смысла для IDEF0-диаграмм.
- Для диаграмм IDEF3 параметры задаются в рамках Activity Options, Link Options, Junction Options и Referent Options.
- Для DFD-диаграмм – в рамках Activity Options, Link Options, Data store Options и External Options.

Создание отчета состоит из следующих действий:

1. Откройте диаграмму, по которой хотите создать отчет.

2. Выберите Diagram Report из меню Report, открыв диалог создания отчета по диаграмме.
3. В открывшемся окне располагаются списки свойств объектов, сгруппированные в шесть рамок:
  - Activity Options – свойства работ.
  - Data store Options – свойства хранилищ данных.
  - External Options – свойства внешних ссылок.
  - Link Options – свойства связей (стрелок).
  - Junction Options – свойства перекрестков.
  - Referent Options – справочная информация.

Включение кнопки, расположенной рядом со свойством, помещает его в отчет.

4. Выберите форму представления отчета (Preview, Print, Report).

### 3. Создание отчета об объектах диаграммы

Аналогично предыдущему отчету устанавливаемые опции должны соответствовать методологии диаграммы.

- Для IDEF0-диаграмм выберите опцию Activities, которая включает в отчет свойства работ.
- Для IDEF3-диаграмм можно выбрать одну или несколько опций: Activities – включает в отчет свойства работ, Data stores – включает в отчет свойства хранилищ данных, External reference – включает в отчет свойства объектов внешних ссылок.
- Для диаграмм DFD можно выбрать опцию Activities, которая сформирует отчет по свойствам работ (информационным процессам).

Создание отчета производится по следующему алгоритму:

1. Откройте диаграмму, по которой хотите создать отчет.
2. Выберите пункт Diagram Object Report из меню Report. С помощью ниспадающего списка Standard Reports можно выбрать название стандартного отчета, настройки которого были сохранены ранее. В рамках Activity Options и Arrow Options задается соответственно перечень свойств работ и стрелок, включаемых в отчет. Формат отчета задается в рамке Report Format.

Отчет можно создавать по всем декомпозированным диаграммам определенной работы, которая задается в ниспадающем списке Start From Activity. Глубина декомпозиции задается в поле Number of Levels.

Способы упорядочения работ и стрелок в отчете указываются в рамках Activity Ordering и Arrow ordering.

3. Выберите способ представления отчета (Preview, Print, Report).

#### 4. Создание отчета по стрелкам

Создание отчета по стрелкам производится по следующему алгоритму:

1. Откройте диаграмму, по которой хотите создать отчет.
2. Выберите из меню Report пункт Arrow Report. При этом откроется диалоговое окно отчета по стрелкам.

Состав и функции этого окна аналогичны остальным отчетам. В рамках Arrow Report Dictionary (Основные свойства стрелок), Source/Dest (Начало и конец стрелок), Arrow Bundle (Разветвления и слияния стрелок) расположены опции, каждая из которых соответствует одному из свойств стрелок. Установка такой опции помещает соответствующее свойство стрелки в отчет.

Опция Diagram Arrow определяет состав отчета. Если установить эту опцию, то в отчет будут включены стрелки активной диаграммы. Если сбросить – то в отчет включаются все стрелки на всех диаграммах открытой модели.

При формировании отчета можно воспользоваться сохраненными ранее настройками (см. стандартные отчеты).

3. Выберите способ представления отчета (Preview, Print, Report).

#### 5. Создание отчета согласованности с методологией

Данный тип отчета фактически позволяет выявить синтаксические ошибки в моделях IDEF0, которые подразделяются на три типа:

1. Невыявляемые ошибки. К данному типу ошибок относятся неправильные наименования объектов.

2. Недопускаемые ошибки. К этому типу ошибок относится соответствие граней работ типам стрелок входящих и выходящих из них. В IDEF0 каждая грань работы предназначается только для определенного типа стрелок. Например, нельзя создать внутреннюю стрелку, выходящую из левой грани работы и входящую в правую.
3. Выявляемые, но допускаемые ошибки. К данному типу ошибок относятся такие ошибки, как наличие неименованных объектов, несвязанных концов стрелок, и т. д.

Отчет о согласованности с методологией не имеет параметров. Для его вызова необходимо воспользоваться пунктом Model Consistency Report, вызываемым из главного меню Report.

## 6. Стандартные отчеты

Для отчетов об объектах диаграммы, о стоимостях работ, о стрелках и об использовании данных можно формировать так называемые стандартные отчеты. Стандартные отчеты представляют собой совокупность настроек, сохраненных под определенным именем. Каждый из вышеперечисленных отчетов имеет свои стандартные отчеты по умолчанию. Например, отчет о стрелках имеет стандартный отчет Arrow Definition/Note.

При вызове стандартного отчета в диалоговом окне восстанавливаются сохраненные в нем опции. Например, если в диалоговом окне отчета о стрелках выбрать в списке Standard Reports стандартный отчет Arrow Definition/Note, то установятся опции Arrow Name, Definition, Note, Diagram Arrows, Fixed Columns, Header, Merge и Remove Special Char.

Помимо существующих стандартных отчетов можно создавать новые. Для этого в диалоговом окне отчета установите все необходимые опции, введите имя стандартного отчета в рамке Standard Report и нажмите New. Установленные параметры сохраняются под введенным именем.

## 7. Пример

В VPWin все отчеты, кроме отчета согласованности с методологией, носят информационный характер, позволяя получить объекты модели их свойства. Отчет согласованности с методологией позволяет находить ошибки в моделях, поэтому является наиболее важным из всех отчетов, используемых при анализе моделей.

Рассмотрим, что выдает этот отчет по диаграммам, построенным в предыдущих лабораторных работах.

Для диаграммы «Обслуживание клиента системы», содержащей IDEF0- и DFD-диаграммы, использованные для описания работы «Выполнение запроса», отчет содержит следующую запись:

Model Inconsistencies:

Diagram A1: Определение уровня доступа в систему

Activity «Определение категории пользователя» has no Control

Diagram A3: Изменение базы данных

Activity «Проверка целостности базы данных» has no Control

Отчет указал на наличие двух ошибок:

1. На диаграмме «Определение уровня доступа в систему» работа «Определение категории пользователя» не имеет стрелки управления.
2. На диаграмме «Изменение базы данных» работа «Проверка целостности базы данных» также не имеет стрелки управления.

Появление ошибок вызвано рассмотрением вышеназванных диаграмм как последовательности действий, в которой управление было не важно для поставленной точки зрения моделирования, хотя и требуется методологией.

## 8. Задания

1. Создать отчет по модели по диаграмме IDEF0, созданной в первой лабораторной работе.
2. Сохранить отчет в файл.
3. Открыть диалоговое окно отчета по стрелкам и сформировать в нем стандартный отчет, содержащий информацию о началах и концах стрелок.
4. Сохранить стандартный отчет под именем Arrows Source/Dest.
5. Создать отчет согласованности с методологией.
6. Сохранить полученный отчет в файл.
7. Проверить отчет на наличие сообщений об ошибках в модели.

## 9. Контрольные вопросы

1. Назовите типы отчетов в BPWin.
2. Опишите процедуру создания отчета по модели.
3. Что включает в себя отчет по модели?
4. Опишите процедуру создания отчета по диаграмме.
5. Что включает в себя отчет по диаграмме?
6. Опишите процедуру создания отчета об объектах диаграммы.
7. Что включает в себя отчет об объектах диаграммы?
8. Опишите процедуру создания отчета по стрелкам.
9. Что включает в себя отчет по стрелкам?
10. Опишите процедуру создания отчета согласованности с методологией.
11. Что включает в себя отчет согласованности с методологией?
12. Каким образом осуществляется поиск ошибок в диаграммах при помощи отчета согласованности с методологией?
13. В какие форматы можно экспортировать отчеты?
14. Какие виды стандартных отчетов существуют в BPWin?
15. Опишите процедуру создания пользовательского отчета.

# Лабораторная работа № 5

## Методология IDEF1X

### Цель работы:

- изучить методологию IDEF1X,
- изучить уровни методологии IDEF1X,
- освоить инструментарий ERWin.

Case-средство ERWin поддерживает методологию IDEF1X и стандарт IE (Information engineering). Методология IDEF1X подразделяется на уровни, соответствующие проектируемой модели данных системы. Каждый такой уровень соответствует определенной фазе проекта. Такой подход полезен при создании систем по принципу «сверху вниз».

Верхний уровень состоит из Entity Relation Diagram (Диаграмма сущность-связь) и Key-Based model (Модель данных, основанная на ключах). Диаграмма сущность-связь определяет сущности и их отношения. Модель данных, основанная на ключах, дает более подробное представление данных. Она включает описание всех сущностей и первичных ключей, которые соответствуют предметной области.

Нижний уровень состоит из Transformation Model (Трансформационная модель) и Fully Attributed (Полная атрибутивная модель). Трансформационная модель содержит всю информацию для реализации проекта, который может быть частью общей информационной системы и описывать предметную область. Трансформационная модель позволяет проектировщикам и администраторам БД представлять, какие объекты БД хранятся в словаре данных, и проверить, насколько физическая модель данных удовлетворяет требованиям информационной системы. Фактически из трансформационной модели автоматически можно получить модель СУБД, которая является точным отображением системного каталога СУБД.

## 1. Логические модели

Три уровня моделей, объединяющие в себе логические модели, состоят из Entity Relationship Diagram (Диаграмма сущность-связь), the Key-Based (Модель данных, основанная на ключах) Model и the Fully Attributed model (Полная атрибутивная модель).



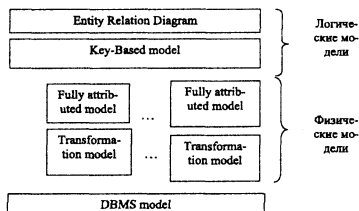


Рис. 5.1. Уровни методологии IDEF1X

## 1.1. Диаграмма сущность-связь

Диаграмма сущность-связь является самым высоким уровнем в модели данных и определяет набор сущностей и атрибутов проектируемой системы. Целью этой диаграммы является формирование общего взгляда на систему для ее дальнейшей детализации.

## 1.2. Модель данных, основанная на ключах

Этот тип модели описывает структуру данных системы, в которую включены все сущности и атрибуты, в том числе ключевые. Целью этой модели является детализация модели сущность-связь, после чего модель данных может начать реализовываться.

## 1.3. Полная атрибутивная модель

Эта модель включает в себя все сущности, атрибуты и является наиболее детальным представлением структуры данных. Полная атрибутивная модель представляет данные в третьей нормальной форме.

## 2. Физические модели

Существует два уровня физических моделей: трансформационная модель и модель СУБД. Физические модели содержат информацию, необходимую системным разработчикам для понимания механизма реализации логической модели в СУБД.

### 2.1. Трансформационная модель

Целью трансформационной модели является предоставление информации администратору БД для создания эффективной структуры хранения, включающей в себя записи, формирующие БД. Трансформационная модель должна помочь разработчикам выбрать структуру хранения данных и реализовать систему доступа к ним.

Перед началом проектирования БД необходимо убедиться в обеспечении следующих требований:

- физическая модель данных должна соответствовать требованиям, предъявляемым к проектируемой системе;
- выбор определенной физической модели должен быть аргументирован;
- должны быть определены возможности наращивания существующей структуры хранения, а также выявлены ее ограничения.

## 2.2. Модель СУБД

Модель СУБД напрямую транслируется из трансформационной модели, являясь отображением системного каталога. ERWin напрямую поддерживает эту модель через функцию генерации схемы БД. При составлении схемы БД в качестве индексов могут использоваться как ключевой атрибут, так и остальные поля БД.

## 3. Преимущества от использования CASE-средства ERWin

Первым преимуществом является использование формируемый средством документов, на основании которых производится проектирование БД и приложений, обеспечивающих доступ к БД. На основании этих документов производится формулирование системных требований к проектируемой БД.

Вторым преимуществом является возможность создания диаграмм структуры БД, позволяющих автоматически решать вопросы, связанные с сохранением ее целостности.

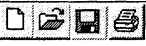






Третье преимущество заключается в независимости логической модели от используемой СУБД, что позволяет применять универсальные методы для ее экспорта в конкретные СУБД.

Кроме того, ERWin предоставляет возможность формирования большого числа отчетов, отражающих текущее состояние процесса проектирования БД.

## 4. Инструментарий ERWin

При запуске ERWin появляется основная панель инструментов и палитра инструментов (табл. 5.1).

Таблица 5.1. Основная панель инструментов ERWin

Кнопки	Назначение кнопок
	Создание, открытие и печать модели
	Вызов диалогового окна Report Browser для генерации отчетов
	Изменение уровня просмотра модели: уровень сущностей, уровень атрибутов и уровень определений
	Изменение масштаба просмотра модели
	Генерация схемы БД, выравнивание схемы с моделью и выбор сервера (доступны только на уровне физической модели)
	Вызов дополнительной панели инструментов для работы с репозитарием Model Mart
	Переключение между областями модели

Палитра инструментов выглядит различно на разных уровнях отображения модели.

На логическом уровне панель инструментов выглядит следующим образом (рис. 5.2).

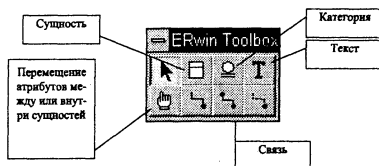


Рис. 5.2. Палитра инструментов на логическом уровне

1. Слева направо, верхний ряд:

- Кнопка указателя (режим мыши) – в этом режиме можно установить фокус на каком-либо объекте модели.
- Кнопка внесения сущности – для внесения сущности нужно щелкнуть левой кнопкой мыши по кнопке внесения сущности и один

раз по свободному пространству на модели. Повторный щелчок приведет к внесению в модель еще одной новой сущности. Для редактирования сущностей или других объектов модели необходимо перейти в режим указателя.

- Кнопка категории. Категория, или категориальная связь, – это специальный тип связи между сущностями. Для установления категориальной связи нужно щелкнуть левой кнопкой мыши по кнопке категории, затем один раз щелкнуть по сущности-родовому предку, затем по сущности-потомку.
- Кнопка внесения текстового блока. С ее помощью можно внести текстовый комментарий в любую часть графической модели.

2. Слева направо, нижний ряд:

- Кнопка перенесения атрибутов внутри сущностей и между ними. Атрибуты могут быть перемещены способом drag & drop.
- Кнопка создания связей: идентифицирующая, «многие-ко-многим» и неидентифицирующая.

На физическом уровне палитра инструментов имеет следующий вид (рис. 5.3).

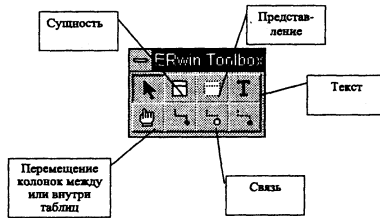


Рис. 5.3. Палитра инструментов на физическом уровне

## 5. Задания

1. Создать новый проект в ERWin.
2. Сформировать модель БД системы согласно перечню информационных объектов.
3. Включить в модель только имена сущностей, не определяя атрибуты.
4. Сохранить модель в файл.

5. Изменить масштаб модели.
6. Распечатать полученную модель.
7. Выбрать сервер БД.
8. Сгенерировать схему БД для выбранного сервера.

## 6. Контрольные вопросы

1. Назовите уровни методологии IDEF1X.
2. Из каких моделей состоит логический уровень?
3. Из каких моделей состоит физический уровень?
4. Что включает в себя диаграмма сущность-связь?
5. Что включает в себя модель данных, основанная на ключах?
6. Какую информацию содержит трансформационная модель?
7. Что включает в себя полная атрибутивная модель?
8. Сформулируйте требования, в которых необходимо убедиться перед началом проектирования БД.
9. Что называется моделью СУБД?
10. Перечислите преимущества от использования CASE-средства ERWin.
11. Как вызвать диалоговое окно Report Browser?
12. Какие кнопки панели инструментов позволяют изменить уровень просмотра модели?
13. Как сгенерировать схему БД?
14. Каким образом осуществляется выбор сервера для генерации схемы БД?
15. Как добавить сущность на диаграмму?
16. Как добавить категорию в сущность?
17. Назовите виды связей.
18. Как перемещать атрибуты внутри сущности?
19. Как добавить текст на диаграмму?
20. С помощью какой кнопки на панели инструментов переключаются области модели?

# Лабораторная работа № 6

## Создание логической модели

### Цель работы:

- ознакомиться с технологией построения логической модели в ERWin,
- изучить методы определения ключевых атрибутов сущностей,
- освоить метод проверки адекватности логической модели,
- изучить типы связей между сущностями.

Первым шагом при создании логической модели БД является построение диаграммы ERD (Entity Relationship Diagram). ERD-диаграммы состоят из трех частей: сущностей, атрибутов и взаимосвязей. Сущностями являются существительные, атрибуты – прилагательными или модификаторами, взаимосвязи – глаголами.

ERD-диаграмма позволяет рассмотреть систему целиком и выяснить требования, необходимые для ее разработки, касающиеся хранения информации.

ERD-диаграммы можно подразделить на отдельные куски, соответствующие отдельным задачам, решаемым проектируемой системой. Это позволяет рассматривать систему с точки зрения функциональных возможностей, делая процесс проектирования управляемым.

## 1. ERD-диаграммы

Как известно основным компонентом реляционных БД является таблица. Таблица используется для структуризации и хранения информации. В реляционных БД каждая ячейка таблицы содержит одно значение. Кроме того, внутри одной БД существуют взаимосвязи между таблицами, каждая из которых задает совместное пользование данными таблицы.

ERD-диаграмма графически представляет структуру данных проектируемой информационной системы. Сущности отображаются при помощи прямоугольников, содержащих имя. Имена принято выражать существительными в единственном числе, взаимосвязи – при помощи линий, соединяющих отдельные сущности. Взаимосвязь показывает, что данные одной сущности ссылаются или связаны с данными другой.

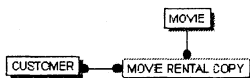


Рис. 6.1. Пример ERD-диаграммы

### 1.1. Определение сущностей и атрибутов

Сущность – это субъект, место, вещь, событие или понятие, содержащие информацию. Точнее, сущность – это набор (объединение) объектов, называемых экземплярами. В приведенном на рис. 6.1 примере сущность CUSTOMER (клиент) представляет всех возможных клиентов. Каждый экземпляр сущности обладает набором характеристик. Так, каждый клиент может иметь имя, адрес, телефон и т. д. В логической модели все эти характеристики называются атрибутами сущности.

На рис. 6.2 показана ERD-диаграмма, включающая в себя атрибуты сущностей.

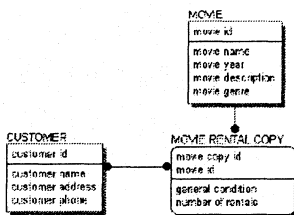


Рис. 6.2. ERD-диаграмма с атрибутами

### 1.2. Логические взаимосвязи

Логические взаимосвязи представляют собой связи между сущностями. Они определяются глаголами, показывающими, как одна сущность относится к другой.

Некоторые примеры взаимосвязей:

- команда включает много игроков,
- самолет перевозит много пассажиров,
- продавец продает много продуктов.

Во всех этих случаях взаимосвязи отражают взаимодействие между двумя сущностями, называемое «один-ко-многим». Это означает, что один экземпляр первой сущности взаимодействует с несколькими экземплярами

другой сущности. Взаимосвязи отображаются линиями, соединяющими две сущности с точкой на одном конце и глаголом, располагаемым над линией.

Кроме взаимосвязи «один-ко-многим» существует еще один тип – это «многие-ко-многим». Этот тип связи описывает ситуацию, при которой экземпляры сущностей могут взаимодействовать с несколькими экземплярами других сущностей. Связь «многие-ко-многим» используют на первоначальных стадиях проектирования. Этот тип взаимосвязи отображается сплошной линией с точками на обоих концах.

Связь «многие-ко-многим» может не учитывать определенные ограничения системы, поэтому может быть заменена на «один-ко-многим» при последующем пересмотре проекта.

### 1.3. Проверка адекватности логической модели

Если взаимосвязи между сущностями были правильно установлены, то можно составить предложения, их описывающие. Например, по модели, показанной на рис. 6.3, можно составить следующие предложения:

- Самолет перевозит пассажиров.
- Много пассажиров перевозятся одним самолетом.

Составление таких предложений позволяет проверить соответствие полученной модели требованиям и ограничениям создаваемой системы.

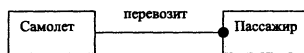


Рис. 6.3. Пример логической модели со взаимосвязью

## 2. Модель данных, основанная на ключах

Каждая сущность содержит горизонтальную линию, разделяющую атрибуты на две группы. Атрибуты, расположенные над линией, называются первичным ключом. Первичный ключ предназначен для уникальной идентификации экземпляра сущности.

### 2.1. Выбор первичного ключа

При создании сущности необходимо выделить группу атрибутов, которые потенциально могут стать первичным ключом (потенциальные ключи), затем произвести отбор атрибутов для включения в состав первичного ключа, следуя следующим рекомендациям:



- Первичный ключ должен быть подобран таким образом, чтобы по значениям атрибутов, в него включенных, можно было точно идентифицировать экземпляр сущности.
- Никакой из атрибутов первичного ключа не должен иметь нулевое значение.
- Значения атрибутов первичного ключа не должны меняться. Если значение изменилось, значит, это уже другой экземпляр сущности.

При выборе первичного ключа можно внести в сущность дополнительный атрибут и сделать его ключом. Так, для определения первичного ключа часто используют уникальные номера, которые могут автоматически генерироваться системой при добавлении экземпляра сущности в БД. Применение уникальных номеров облегчает процесс индексации и поиска в БД.

Первичный ключ, выбранный при создании логической модели, может быть неудачным для осуществления эффективного доступа к БД и должен быть изменен при проектировании физической модели.

Потенциальный ключ, не ставший первичным, называется альтернативным ключом (Alternate Key). ERWin позволяет выделить атрибуты альтернативных ключей, и по умолчанию в дальнейшем при генерации схемы БД по этим атрибутам будет генерироваться уникальный индекс. При создании альтернативного ключа на диаграмме рядом с атрибутом появляются символы (АК).

Атрибуты, участвующие в неуникальных индексах, называются инверсионными входами (Inversion Entries). Инверсионные входы – это атрибут или группа атрибутов, которые не определяют экземпляр уникальным образом, но часто используются для обращения к экземплярам сущности. ERWin генерирует неуникальный индекс для каждого инверсионного входа.

При проведении связи между двумя сущностями в дочерней сущности автоматически образуются внешние ключи (foreign key). Связь образует ссылку на атрибуты первичного ключа в дочерней сущности, и эти атрибуты образуют внешний ключ в дочерней сущности. Атрибуты внешнего ключа обозначаются символами (FK) после своего имени.

### 3. Пример

Рассмотрим процесс построения логической модели на примере БД студентов системы «Служба занятости в рамках вуза». Первым этапом является определение сущностей и атрибутов. В БД будут храниться записи о студентах, следовательно, сущностью будет студент.

Таблица 6.1. Атрибуты сущности «Студент»

Атрибут	Описание
Номер	Уникальный номер для идентификации пользователя
Ф.И.О.	Фамилия, имя и отчество пользователя
Пароль	Пароль для доступа в систему
Возраст	Возраст студента
Пол	Пол студента
Характеристика	Мето-поле с общей характеристикой пользователя
E-mail	Адреса электронной почты
Телефон	Номера телефонов студента (домашний, рабочий)
Опыт работы	Специальности и опыт работы студента по каждой из них
Специальность	Специальность, получаемая студентом при окончании учебного заведения
Специализация	Направление специальности, по которому обучается студент
Иностранный язык	Список иностранных языков и уровень владения ими
Тестирование	Список тестов и отметки о их прохождении
Экспертная оценка	Список предметов с экспертными оценками по каждому из них
Оценки по экзаменам	Список сданных предметов с оценками

В полученном списке существуют атрибуты, которые нельзя определить в виде одного поля ВД. Такие атрибуты требуют дополнительных определений и должны рассматриваться как сущности, состоящие, в свою очередь, из атрибутов. К таковым относятся: опыт работы, иностранный язык, тестирование, экспертная оценка, оценки по экзаменам. Определим их атрибуты.

Таблица 6.2. Атрибуты сущности «Опыт работы»

Атрибут	Описание
Специальность	Название специальности, по которой у студента есть опыт работы

Таблица 6.2. Атрибуты сущности «Опыт работы»

Атрибут	Описание
Опыт	Опыт работы по данной специальности в годах
Место работы	Наименование предприятия, где приобретался опыт

Таблица 6.3. Атрибуты сущности «Иностранный язык»

Атрибут	Описание
Язык	Название иностранного языка, которым владеет студент
Уровень владения	Численная оценка уровня владения иностранным языком

Таблица 6.4. Атрибуты сущности «Тестирование»

Атрибут	Описание
Название	Название теста, который прошел студент
Описание	Содержит краткое описание теста
Оценка	Оценка, которую получил студент в результате прохождения теста

Таблица 6.5. Атрибуты сущности «Экспертная оценка»

Атрибут	Описание
Дисциплина	Наименование дисциплины, по которой оценивался студент
Ф.И.О. преподавателя	Ф.И.О. преподавателя, который оценивал студента
Оценка	Экспертная оценку преподавателя

Таблица 6.6. Атрибуты сущности «Оценки по экзаменам»

Атрибут	Описание
Предмет	Название предмета, по которому сдавался экзамен
Оценка	Полученная оценка

Составим ERD-диаграмму, определяя типы атрибутов и проставляя связи между сущностями (рис. 6.4). Все сущности будут зависимыми от сущности «Студент». Связи будут типа «один-ко-многим».

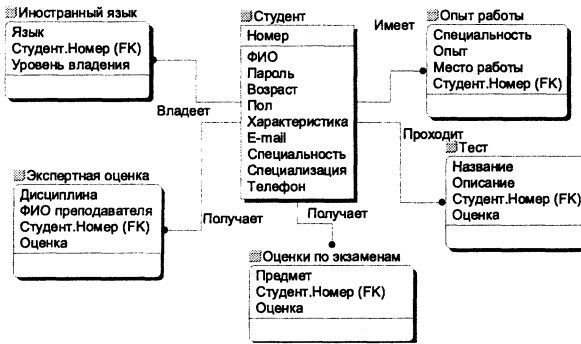


Рис. 6.4. ERD-диаграмма БД студентов

На полученной диаграмме рядом со связью отражается ее имя, показывающее соотношение между сущностями. При проведении связи между сущностями первичный ключ мигрирует в дочернюю сущность.

Следующим этапом при построении логической модели является определение ключевых атрибутов и типов атрибутов.

Таблица 6.7. Типы атрибутов

Атрибут	Тип
Номер	Number
Ф.И.О.	String
Пароль	String
Возраст	Number

Таблица 6.7. Типы атрибутов

Атрибут	Тип
Пол	String
Характеристика	String
E-mail	String
Специальность	String
Специализация	String
Опыт	Number
Место работы	String
Язык	String
Уровень владения	Number
Название	String
Описание	String
Оценка	Number
Дисциплина	String
Ф.И.О. преподавателя	String
Предмет	String

Выберем для каждой сущности ключевые атрибуты, однозначно определяющие сущность. Для сущности «Студент» это будет уникальный номер, для сущности «Опыт работы» все поля являются ключевыми, так как по разным специальностям студент может иметь разный опыт работы в разных фирмах. Сущность «Тест» определяется названием, так как студент по одному тесту может иметь только одну оценку. Оценка по экзамену определяется только названием предмета, экспертная оценка зависит от преподавателя, который ее составил, поэтому в качестве ключевых атрибутов выберем «Дисциплину» и «Ф.И.О. преподавателя». У сущности «Иностранный язык» уровень владения зависит только от наименования языка, следовательно, это и будет являться ключевым атрибутом.

Получим новую диаграмму, изображенную на рис. 6.5, где все ключевые атрибуты будут находиться над горизонтальной чертой внутри рамки, изображающей сущность.

#### 4. Задания

1. Задать атрибуты сущностей, созданных в предыдущей лабораторной работе.



Рис. 6.5. ERD-диаграмма БД студентов с ключевыми атрибутами

2. Определить первичные ключи в сущностях.
3. Определить состав альтернативных ключей.
4. Связать сущности между собой, используя описанные типы связей.
5. После проведения связей определить состав внешних ключей.
6. Проверить модель на соответствие предметной области.
7. Сохранить полученную диаграмму.

## 5. Контрольные вопросы

1. Назовите основные части ERD-диаграммы.
2. Цель ERD-диаграммы.
3. Что является основным компонентом реляционных БД?
4. Что называется сущностью?
5. Сформулируйте принцип именования сущностей.
6. Что показывает взаимосвязь между сущностями?
7. Назовите типы логических взаимосвязей.
8. Каким образом отображаются логические взаимосвязи?
9. Опишите механизм проверки адекватности логической модели.
10. Что называется первичным ключом?
11. Назовите принципы, согласно которым формируется первичный ключ.
12. Что называется альтернативным ключом?
13. Что называется инверсионным входом?
14. В каком случае образуются внешние ключи?

# Лабораторная работа № 7

## 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  $B$  сущности  $E$  функционально зависит от атрибута  $A$  сущности  $E$ , если и только если каждое значение  $A$  в  $E$  связало с ним точно одно значение  $B$  в  $E$ . Другими словами,  $A$  однозначно определяет  $B$ .

### **Полная функциональная зависимость**

Атрибут  $E$  сущности  $B$  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 поддерживает корректность имен следующим образом:

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

## 2. Создание физической 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 трансформируется в физическую по следующему принципу: сущности становятся таблицами, атрибуты становятся столбцами, а ключи становятся индексами.



**Таблица 7.1.** *Сопоставление компонентов логической и физической модели*

<b>Логическая модель</b>	<b>Физическая модель</b>
Сущность	Таблица
Атрибут	Столбец
Логический тип (текст, число, дата, blob)	Физический тип (корректный тип, зависящий от выбранной СУБД)
Первичный ключ	Первичный ключ, индекс РК
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, чтобы 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).

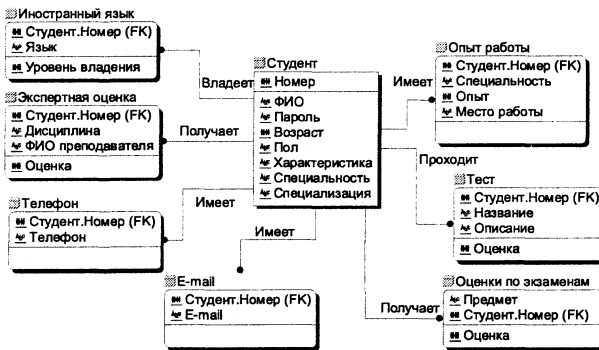


Рис. 7.1. ERD-диаграмма БД студентов в первой нормальной форме

Проверим соответствие БД второй нормальной форме. Все неключевые атрибуты полностью должны зависеть от первичного ключа. Нетрудно 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).

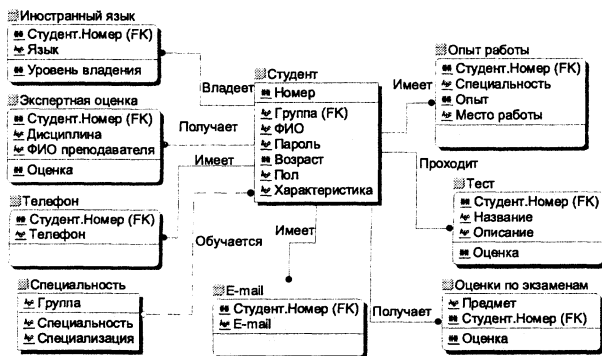


Рис. 7.2. ERD-диаграмма БД студентов в третьей нормальной форме

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. 7.4).

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.

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	
Возраст	Number		>10 и <100
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	

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

Validation Name	Validation Rule	Sort By Type
Проверка возраста	BETWEEN 10 AND 100	Client/Server
Проверка опыта	> 0	Client/Server
Проверка оценки	BETWEEN 2 AND 5	Client/Server
Проверка пола	IN ('М', 'Ж')	Client/Server
Проверка уровня языка	BETWEEN 2 AND 5	Client/Server

**Рис. 7.3.** Правила валидации

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

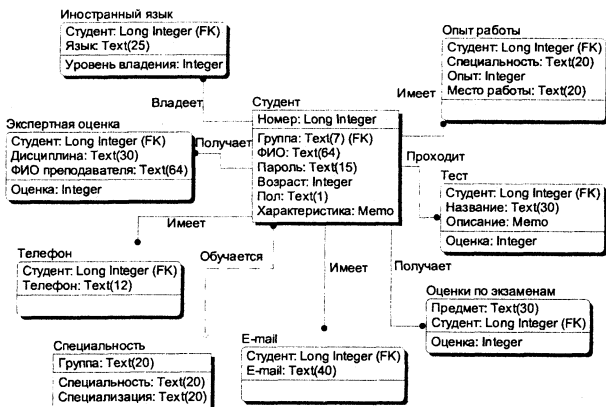


Рис. 7.4. Физическая модель БД студентов

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.

## 6. Контрольные вопросы

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»?


## Лабораторная работа № 8

### 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 predetermined 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 необходимо 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. 8.1, 8.2).

**Таблица 8.1.** 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 колонок и сортировка полученного отчета
	Связь отчета с иконкой
	Сохранение отчета в виде представления

**Таблица 8.2.** Кнопки нижней панели инструментов






Кнопка	Назначение кнопки
	Редактировать выделенный отчет
	Удалить отчет
	Показать только верхний уровень дерева




Таблица 8.2. Кнопки нижней панели инструментов

Кнопка	Назначение кнопки
	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, непредопределенного отчета необходимо:

1. Выбрать в меню пункт File->New или щелкнуть на кнопке  панели инструментов.
2. В появившемся диалоговом окне ERWin Report Editor (рис. 8.1) 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.).

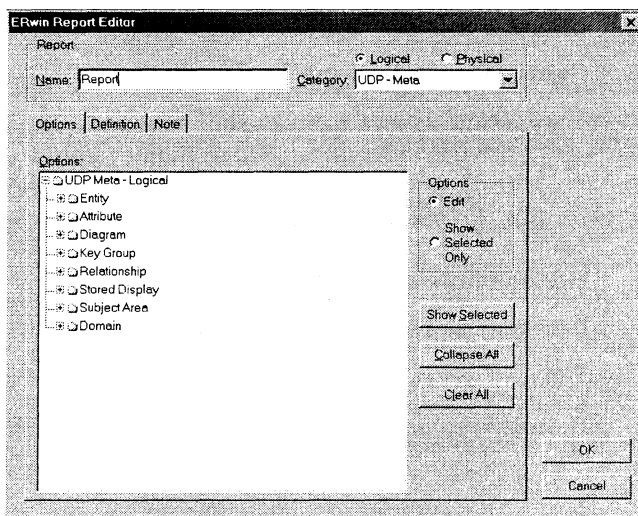





Рис. 8.1. Диалоговое окно Report Editor


3. Указать категории, которые будут включены в отчет, при помощи 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 сохранить 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 8.3. 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. все 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. 8.2).

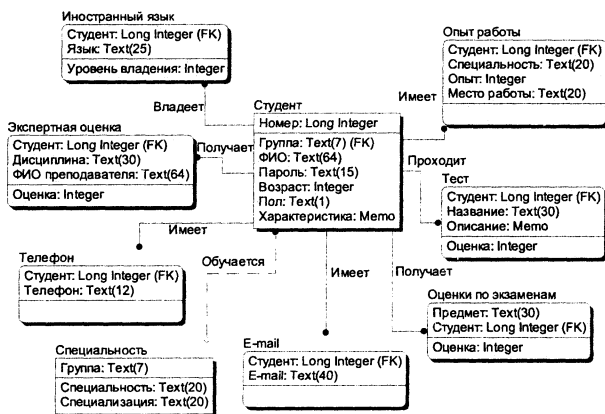


Рис. 8.2. Скорректированная физическая модель БД студентов

### 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.

### 4. Контрольные вопросы

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.

# Лабораторная работа № 9

## 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 на систему: 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, и 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.). На рис. 9.1 приведен внешний вид Rational Rose.

Создание нового проекта в Rational Rose производится выбором меню File/New. При этом создается несколько пустых диаграмм верхнего 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.

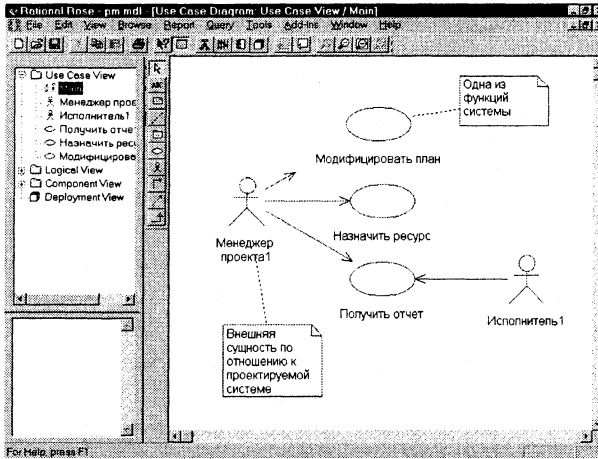


Рис. 9.1. Главное окно Rational Rose 98

Таблица 9.1. 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 сравнения 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:

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}},$$

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

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}}.$$

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

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

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. 9.2 и 9.3 приводятся оценки для различных типов элементов и связей.



Таблица 9.2. Основные элементы языка UML

Тип элемента	Оценка для 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 (note)	2

Таблица 9.3. 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. 9.4 приведены диапазоны оптимальных оценок для основных типов диаграмм.

Таблица 9.4. 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

Таблица 9.4. Диапазоны оценок для диаграмм UML

Тип диаграммы	Диапазон оценок
Развертывания (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.

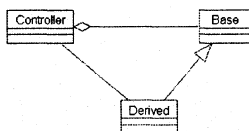


Рис. 9.2.

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

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

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 подсистемы «Служба 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».

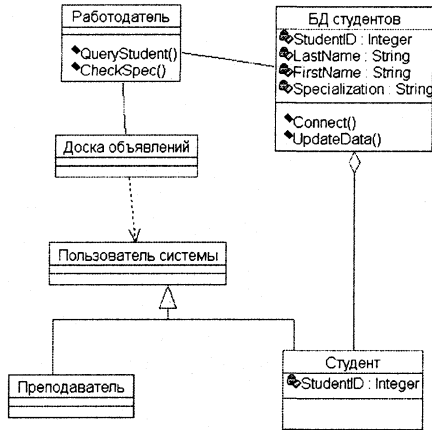


Рис. 9.3. Диаграмма 1

«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:

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

### 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:

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

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

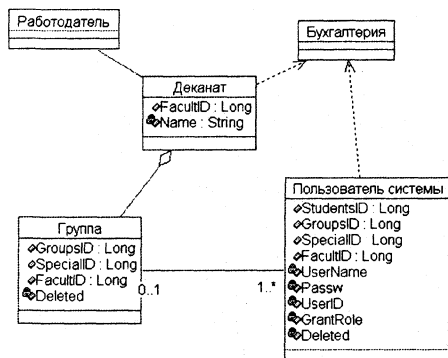


Рис. 9.4. Диаграмма 2

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.

#### 4. 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.

## 5. Контрольные вопросы

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?

# Лабораторная работа № 10

## Диаграммы вариантов использования

### Цель работы:

- изучение диаграмм вариантов использования,
- изучение их применения в процессе постановки задачи.

### 1. Диаграммы вариантов использования (use-case diagrams)

Одна из моделей формализации процесса постановки целей и задач проекта была предложена фирмой Rational и вошла в стандарт языка UML. Для этого применяются диаграммы вариантов использования (use-case), иногда называемые диаграммами прецедентов. Вариант использования представляет собой типичное взаимодействие пользователя и проектируемой системы. Варианты использования характеризуются рядом свойств:

- вариант использования охватывает некоторую очевидную для пользователей функцию;
- вариант использования может быть как небольшим, так и достаточно крупным;
- вариант использования решает некоторую дискретную задачу пользователя.

В простейшем случае вариант использования создается в процессе обсуждения с пользователями тех вещей, которые они хотели бы получить от системы. При этом каждой отдельной функции, которую они хотели бы реализовать, присваивается некоторое имя и записывается ее краткое текстовое описание.

Это все, что необходимо в фазе анализа. Знание некоторых деталей может потребоваться, если предполагается, что данный вариант использования содержит важные архитектурные ответвления. Большинство вариантов использования может быть детализировано во время конкретной итерации в процессе проектирования.

На рис. 10.1 приводится вариант использования, описывающий одну из функций системы управления проектами – обратную связь между менеджером проекта и исполнителем.

Основными элементами диаграммы вариантов использования являются действующие лица, варианты использования и отношения между ними. Действующее лицо – это роль, которую пользователь играет по отношению к системе.

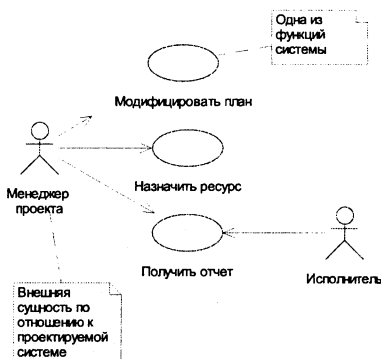


Рис. 10.1.

На рис. 10.1 присутствуют два действующих лица: «Менеджер проекта» и «Исполнитель». Менеджеров и исполнителей может быть много, но с точки зрения системы они выполняют одну и ту же роль. Говоря о действующих лицах, важно видеть в них роли, а не конкретных людей или наименования работ. Действующие лица вовсе не обязаны быть людьми, несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок. Действующее лицо может также быть внешней системой, которой необходима некоторая информация от нашей системы.











На рис. 10.1 присутствуют также три варианта использования: «Модифицировать план», «Назначить ресурс» и «Получить отчет». Все варианты использования так или иначе связаны с внешними требованиями к функциональности системы. Варианты использования всегда следует анализировать вместе с действующими лицами системы, определяя при этом реальные задачи пользователей и рассматривая альтернативные способы решения этих задач.

Действующие лица могут играть различные роли по отношению к варианту использования. Они могут применять его результаты или сами непосредственно в нем участвовать.

Хорошим источником для идентификации вариантов использования служат внешние события. Для этого необходимо перечислить все происходя-

шие во внешнем мире события, на которые система должна реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать чисто пользовательскую реакцию. Идентификация событий, на которые необходимо реагировать, поможет идентифицировать варианты использования.

**Таблица 10.1.** *Описание кнопок панели инструментов диаграмм вариантов использования Rational Rose*

<b>Кнопка</b>	<b>Описание</b>	<b>Название</b>
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item
	Добавление пакета	Package
	Добавление варианта использования	Use Case
	Добавление действующего лица	Actor
	Однонаправленная связь	Unidirectional Association
	Зависимость	Dependency
	Наследование	Generalization

## 2. Пример

На рис. 10.2 и 10.3 приведены две диаграммы вариантов использования, описывающие одну из функций подсистемы «Служба занятости в рамках вуза» системы «Дистанционное обучение». Отличие этих диаграмм в том, что первая из них более подробно описывает процесс взаимодействия пользователя и системы.

Найдем численную оценку для каждой из диаграмм.



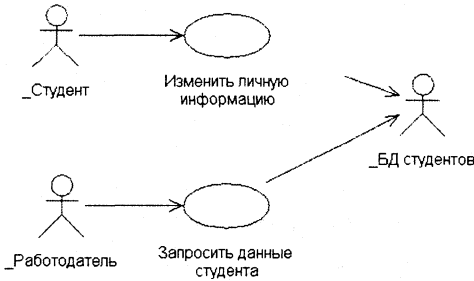


Рис. 10.2. Диаграмма 1

Диаграмма 1

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{17 + 4}{1 + 5 + \sqrt{3}} = 2,72.$$

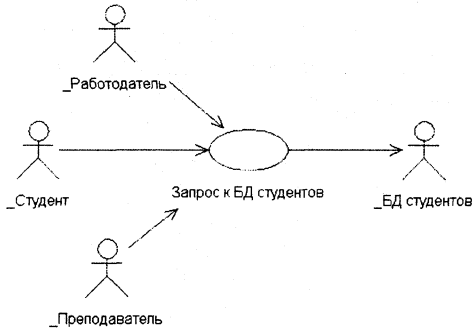


Рис. 10.3. Диаграмма 2

Диаграмма 2

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{21 + 4}{1 + 5 + \sqrt{3}} = 3,23.$$

Оценка для диаграммы 1 попадает в оптимальный для диаграмм вариантов использования диапазон, оценка для диаграммы 2 превышает этот диапазон.

Полученный результат можно объяснить тем, что диаграмма 2 описывает взаимодействие системы с пользователем на слишком высоком уровне. Фактически диаграмма 1 является детализированным до уровня простых операций вариантом диаграммы 2.

### 3. Задания

1. Определить основные функции системы.
2. Для нескольких функций выделить действующие лица, участвующие в них или использующие их результаты.
3. Для каждой выбранной функции построить диаграмму вариантов использования.

### 4. Контрольные вопросы

1. В чем смысл варианта использования?
2. Каково назначение диаграмм вариантов использования?
3. Назовите основные свойства вариантов использования.
4. Назовите основные компоненты диаграмм вариантов использования.
5. Что такое «действующее лицо»?
6. Какую роль могут играть действующие лица по отношению к варианту использования?
7. Каким образом анализ внешних событий позволяет определить варианты использования системы?

# Лабораторная работа № 11

## Диаграммы классов

### Цель работы:

- изучение диаграмм классов,
- изучение их применения в процессе проектирования.

### 1. Диаграммы классов (class diagrams)

Диаграммы классов являются центральным звеном методологии объектно-ориентированного анализа и проектирования.

Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов. На стадии анализа диаграммы классов используются, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграммы классов используются, чтобы передать структуру классов, формирующих архитектуру системы.

Каждый класс должен иметь имя; если имя слишком длинно, его можно сократить или увеличить сам значок на диаграмме. Имя каждого класса должно быть уникально в содержащем его проекте.

Диаграмма классов определяет типы объектов системы и различного рода статические связи, которые существуют между ними. Имеется два основных вида статических связей:

- ассоциации (например, менеджер может вести несколько проектов),
- подтипы (работник является разновидностью личности).

На диаграммах классов изображаются также атрибуты классов, операции и ограничения, которые накладываются на связи между объектами.

На рис. 11.1 изображена типичная диаграмма классов.

Далее будут рассмотрены различные фрагменты диаграммы.

#### Ассоциации

Ассоциации представляют собой связи между экземплярами классов (личность работает в компании, компания имеет ряд офисов).

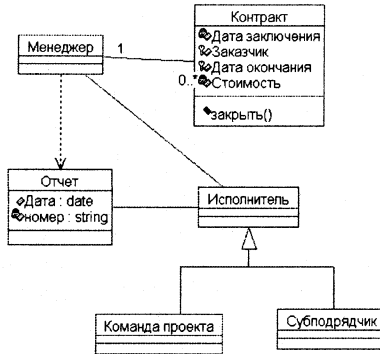


Рис. 11.1.

Любая ассоциация обладает двумя ролями; каждая роль представляет собой направление ассоциации. Таким образом, ассоциация между «Исполнителем» и «Отчетом» содержит две роли: одна от «Исполнителя» к «Отчету»; другая – от «Отчета» к «Исполнителю». Роль может быть явно поименована с помощью метки. Если такая метка отсутствует, роли присваивается имя класса-цели; таким образом, роль ассоциации от «Исполнителя» к «Отчету» может быть названа «Отчетом».

Роль также обладает множественностью, которая показывает, сколько объектов может участвовать в данной связи. На рис. 11.1 символ «0..\*» над ассоциацией между «Менеджером» и «Контрактом» показывает, что с одним «Менеджером» может быть связано много «Контрактов»; символ «1» показывает, что любой «Контракт» управляется одним «Менеджером».

В общем случае множественность показывает нижнюю и верхнюю границы количества объектов, которые могут участвовать в связи. Для этого могут использоваться единственное число, диапазон или дискретная комбинация из чисел и диапазонов.

Для ассоциации может быть указано направление навигации. Если навигация указана только в одном направлении, то такая ассоциация называется однонаправленной (ассоциация между «Менеджером» и «Отчетом» на рис. 11.1). У двунаправленной ассоциации навигация указана в обоих направлениях. В языке UML отсутствие стрелок у ассоциации трактуется следующим образом: направление навигации неизвестно или ассоциация является двунаправленной.

### Атрибуты

Атрибуты во многом подобны ассоциациям. Разница между ними заключается в том, что атрибуты предполагают единственное направление навигации – от типа к атрибуту.

На рис. 11.1 атрибуты указаны для классов «Контракт» и «Отчет». В зависимости от степени детализации диаграммы, обозначение атрибута может включать имя атрибута, тип и значение, присваиваемое по умолчанию. В синтаксисе UML это выглядит следующим образом: <признак видимости> <имя> : <тип> = <значение по умолчанию>, где признак видимости может принимать одно из следующих четырех значений:

- общий (public) – атрибут доступен для всех клиентов класса,
- защищенный (protected) – атрибут доступен только для подклассов и друзей класса,
- секретный (private) – атрибут доступен только для друзей класса,
- реализация (implementation) – атрибут доступен только внутри обрамляющего пакета.

### Операции

Операции представляют собой процессы, реализуемые классом. Наиболее очевидное соответствие существует между операциями и методами над классом.

Полный синтаксис UML для операций выглядит следующим образом: <признак видимости> <имя> (<список-параметров>) : <тип-выражения-возвращающего-значение> = <строка-свойств>, где

- признак видимости может принимать те же значения, что и для атрибутов;
- имя представляет собой символьную строку;
- список-параметров содержит необязательные аргументы, синтаксис которых совпадает с синтаксисом атрибутов;
- тип-выражения-возвращающего-значение является необязательной спецификацией и зависит от конкретного языка программирования;
- строка-свойств показывает значения свойств, которые применяются к данной операции. Примером операции на рис. 11.1 является операция закрыть() класса «Контракт».

### Обобщение

Типичный пример обобщения включает «Команду проекта» и «Субподрядчика» (см. рис. 11.1). Они обладают некоторыми различиями, однако

у них также много общего. Одинаковые характеристики можно поместить в обобщенный класс «Исполнитель» (супертип), при этом «Команда проекта» и «Субподрядчик» будут выступать в качестве подтипов.

Смысл обобщения заключается в том, что интерфейс подтипа должен включать все элементы интерфейса супертипа. Другая сторона обобщения связана с принципом подстановочности. Субподрядчика можно подставить в любой код, где требуется «Исполнитель», и при этом все должно нормально работать. Это означает, что, разработав код, предполагающий использование «Исполнителя», можно свободно употреблять экземпляр любого подтипа «Исполнителя». Субподрядчик может реагировать на некоторые команды отличным от другого «Исполнителя образом» (в соответствии с принципом полиморфизма), но это отличие не должно беспокоить вызывающий объект.

Обобщение с точки зрения реализации связано с понятием наследования в языках программирования. Подкласс наследует все методы и поля суперкласса и может переопределять наследуемые методы. Подтип можно также реализовать, используя механизм делегирования.

### Ограничения

При построении диаграмм классов основным занятием является отображение различных ограничений. На рис. 11.1 показано, что «Контракт» может управляться только одним «Менеджером».

С помощью конструкций ассоциации, атрибута и обобщения можно специфицировать наиболее важные ограничения, но невозможно выразить их все.

В UML отсутствует строгий синтаксис описания ограничений, за исключением помещения их в фигурные скобки {}.

**Таблица 11.1.** Описание кнопок панели инструментов диаграмм классов *Rational Rose*





Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item

Таблица 11.1. Описание кнопок панели инструментов диаграмм классов Rational Rose

Кнопка	Описание	Название
	Класс	Class
	Интерфейс	Interface
	Ассоциация	Association
	Агрегация	Aggregation
	Привязка атрибута	Link Attribute
	Добавление пакета	Package
	Зависимость	Dependency
	Наследование	Generalization
	Реализация	Realize

## 2. Пример

На рис. 11.2 и 11.3 приведены две диаграммы классов, реализующие один и тот же фрагмент подсистемы «Служба занятости в рамках вуза»: взаимодействие пользователя с БД, содержащей персональные сведения.

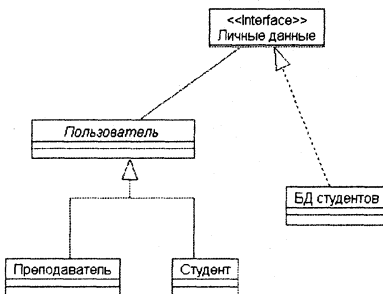


Рис. 11.2. Диаграмма 1

Проведем расчет оценки для каждой из диаграмм.

**Диаграмма 1**

Атрибуты и операции на диаграмме не указаны, поэтому сразу рассчитаем полное значение для диаграммы:

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{25 + 6}{1 + 5 + \sqrt{1 + 3}} = 3,875$$

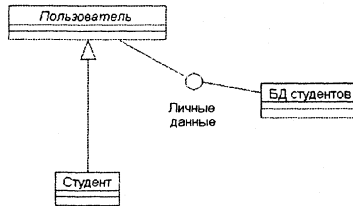


Рис. 11.3. Диаграмма 2

**Диаграмма 2**

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{19 + 5}{1 + 4 + \sqrt{1 + 3}} = 3,43$$

Соответственно значение для диаграммы 2 находится в допустимых пределах, значение для диаграммы 1 превышает допустимую величину.

Такой результат можно объяснить двумя причинами.

1. На диаграмме 1 отображен класс «Преподаватель», который хотя и является потомком класса «Пользователь системы», но не участвует во взаимодействии с БД. Соответственно он усложняет модель, не внося при этом полезной информации.
2. На диаграмме 1 интерфейс «Личные данные» изображен в развернутой форме (как класс, помеченный стереотипом), а на диаграмме 2 он показан значком интерфейса. Последняя форма отображения более предпочтительна, когда не указываются операции интерфейса.

**3. Задания**

1. Выделить основные классы объектов в проектируемой системе.



2. Построить диаграмму классов, в общем виде демонстрирующую архитектуру системы.
3. Построить одну-две диаграммы классов, детализирующие отдельные подсистемы. Указать для классов основные атрибуты и операции, указать вид и направление ассоциаций.

### 4. Контрольные вопросы

1. Каково назначение диаграмм классов?
2. Для чего используется диаграмма классов на стадии анализа?
3. Для чего используется диаграмма классов на стадии проектирования?
4. Назовите основные компоненты диаграмм классов.
5. Назовите основные типы статических связей между классами.
6. Что представляет собой ассоциация?
7. В чем смысл множественности ассоциаций?
8. В чем отличие атрибутов от ассоциаций?
9. Что такое признак видимости?
10. Что представляет собой операция класса?
11. В чем смысл обобщения?
12. Каково назначение ограничений на диаграммах классов?

# Лабораторная работа № 12

## Диаграммы взаимодействия

### Цель работы:

- изучение диаграмм взаимодействия,
- изучение их применения в процессе проектирования.

## 1. Диаграммы взаимодействия (interaction diagrams)

Диаграммы взаимодействия являются моделями, описывающими поведение взаимодействующих групп объектов.

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

Данный подход будет проиллюстрирован на примере простого варианта использования, который описывает следующее поведение:

- «Менеджер» запрашивает текущий «Отчет» «Исполнителя»;
- если «Отчет» устарел, «Менеджер» посылает запрос «Исполнителю» на обновление «Отчета»;
- «Исполнитель» создает новый «Отчет»;
- «Менеджер» делает повторный запрос «Отчета».

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

## 2. Диаграммы последовательности

На диаграмме последовательности объект изображается в виде прямоугольника на вершине пунктирной вертикальной линии (рис. 12.1).

Эта вертикальная линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны

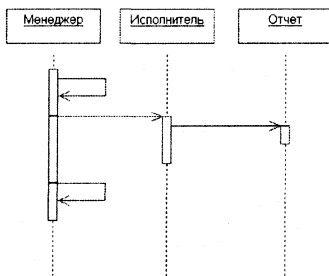


Рис. 12.1.

на диаграмме (сверху вниз). Каждое сообщение может быть помечено именем, при желании можно показать также аргументы и некоторую управляющую информацию. Также можно показать самоделегирование – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.






Изю всей возможной управляющей информации два ее вида имеют существенное значение. Во-первых, это условие, показывающее, в каком случае посылается сообщение (например, `[ОтчетУстарел() == true]`). Сообщение посылается только при выполнении данного условия. Другой полезный управляющий маркер – это маркер итерации, показывающий, что сообщение посылается много раз для множества объектов-адресатов (например, \*обновить).

Активизации – прямоугольники на линиях жизни – показывают, когда метод становится активным (во время его выполнения либо при ожидании результата выполнения какой-либо процедуры). Используя механизм активизаций, можно более четко показать смысл самоделегирования. Без них довольно трудно определить, где же выполняются следующие после самоделегирования вызовы – в вызывающем методе или в вызываемом. Активизации вносят ясность в этот вопрос.

Таблица 12.1. Описание кнопок панели инструментов диаграмм взаимодействия Rational Rose

Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box

Таблица 12.1. Описание кнопок панели инструментов диаграмм взаимодействия Rational Rose

Кнопка	Описание	Название
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item
	Объект	Object
	Сообщение	Object Message
	Самоделегирование	Message to Self

### 3. Кооперативные диаграммы

Вторым видом диаграммы взаимодействия является кооперативная диаграмма (рис. 12.2).

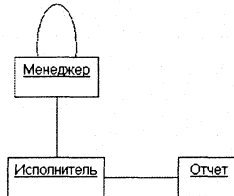


Рис. 12.2.

На кооперативной диаграмме экземпляры объектов показаны в виде пиктограмм. Линии между ними обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования.

Каждый вид диаграмм взаимодействия имеет свои преимущества, выбор обычно осуществляется исходя из предпочтений разработчика. На диаграммах последовательности делается акцент именно на последовательности сообщений, при этом легче наблюдать порядок, в котором происходят различные события. В случае кооперативных диаграмм можно использовать пространственное расположение объектов для того, чтобы показать их статическое взаимодействие.

Одним из главных свойств любой диаграммы взаимодействия является ее простота. Посмотрев на диаграмму, можно легко увидеть все сообщения.

Однако при попытке изобразить нечто более сложное, чем единственный последовательный процесс без множества условных переходов или циклов, данный подход может не сработать.

Для отображения условного поведения на диаграммах взаимодействия существует два подхода. Один из них состоит в использовании отдельных диаграмм для каждого сценария. Второй заключается в том, что сообщения сопровождаются условиями, показывающими поведение объектов.

**Таблица 12.2.** Описание кнопок панели инструментов кооперативных диаграмм Rational Rose

Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item
	Объект	Object
	Представитель класса	Class Instance
	Связь	Object Linkf
	Самоделегирование	Link to Self
	Сообщение	Link Message
	Ответ	Reverse Link Message
	Поток данных	Data Flow
	Обратный поток данных	Reverse Data Flow

## 4. Пример

На рис. 12.3 и 12.4 приведены диаграммы последовательности модели подсистемы «Служба занятости», показывающие взаимодействие двух классов модели: Студент и БД студентов. На рис. 12.5 и 12.6 то же взаимодействие показано с помощью кооперативных диаграмм.

Найдем численную оценку для каждой из диаграмм.

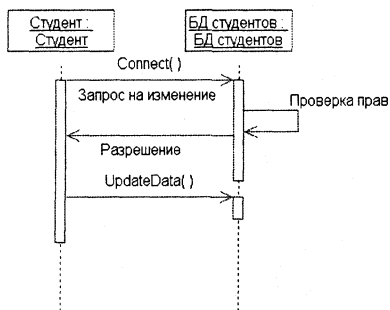


Рис. 12.3. Диаграмма 1

**Диаграмма 1**

Так как на диаграмме последовательности связи отсутствуют, проведем расчет по сокращенной формуле:

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{34}{1 + 6 + \sqrt{2}} = 4,04.$$

**Диаграмма 2**

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{22}{1 + 4 + \sqrt{2}} = 3,43.$$

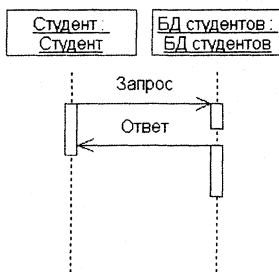


Рис. 12.4. Диаграмма 2

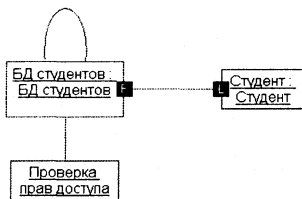


Рис. 12.5. Диаграмма 3

Теперь рассчитаем оценку для кооперативных диаграмм.  
**Диаграмма 3**

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{15 + 3}{1 + 3 + 1} = 3,6.$$

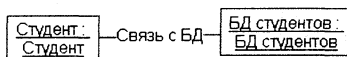


Рис. 12.6. Диаграмма 4

**Диаграмма 4**

$$S = \frac{\sum S_{Obj} + \sum S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{10 + 1}{1 + 2 + 1} = 2,75.$$

В результате значения для диаграмм 1 и 3 соответствуют оптимальным, для диаграмм 2 и 4 – ниже оптимальных. Это можно объяснить низкой информативностью диаграмм 2 и 4, так как взаимодействие классов показано на них на слишком высоком уровне.

## 5. Задания

1. Выбрать в моделируемой системе вариант использования, для которого будут строиться диаграммы взаимодействия.
2. Построить для выбранного варианта использования диаграмму последовательности.
3. Построить для того же варианта использования кооперативную диаграмму.

4. Сформулировать достоинства и недостатки каждого вида диаграмм при моделировании данного варианта использования.

### 6. Контрольные вопросы

1. Каково назначение диаграмм взаимодействия?
2. Как относятся между собой диаграммы вариантов использования и диаграммы взаимодействия?
3. Назовите два вида диаграмм взаимодействия.
4. Что такое «жизненная линия» на диаграмме последовательности?
5. Как на диаграмме последовательности представляются сообщения?
6. Что такое самоделегирование?
7. Что показывает активизация объекта?
8. В чем отличие кооперативных диаграмм от диаграмм взаимодействия?
9. Каковы преимущества и недостатки каждого вида взаимодействия?
10. Как отображается условное поведение на диаграммах взаимодействия?



# Лабораторная работа № 13

## Диаграммы состояний

### Цель работы:

- изучение диаграмм состояний,
- изучение их применения в процессе проектирования.

## 1. Диаграммы состояний (state diagrams)

Диаграммы состояний являются хорошо известным средством описания поведения систем. Они определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате влияния некоторых событий.

На рис. 13.1 показана диаграмма состояний UML, отражающая поведение отчета в системе управления проектами. На диаграмме изображены различные состояния, в которых может находиться отчет.

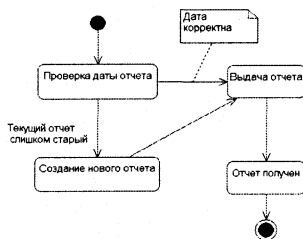


Рис. 13.1.

Процесс начинается с начальной точки, затем следует самый первый переход в состояние «Проверка даты отчета». В поведении объекта в системе можно выделить действия, отображаемые переходами, и деятельности, отображаемые состояниями. Хотя и то и другое – это процессы, реализуемые, как правило, некоторым методом класса «Отчет», они трактуются различным образом. Действия связаны с переходами и рассматриваются, как мгновенные и непрерываемые. Деятельности связаны с состояниями и могут длиться достаточно долго. Деятельность может быть прервана в результате наступления некоторого события.

Переход может содержать метку. Синтаксически метка перехода состоит из трех частей, каждая из которых является необязательной: <Событие> [<Условие>]/<Действие>. Если метка перехода не содержит никакого события, это означает, что переход происходит, как только завершается какая-либо деятельность, связанная с данным состоянием.

Из состояния «Проверка даты отчета» возможны два перехода. Метка одного из них включает условие. Условие – это логическое условие, которое может принимать два значения: «истина» или «ложь». Условный переход выполняется только в том случае, если условие принимает значение «истина», в противном случае выполняется переход, не помеченный условием.

Из конкретного состояния в данный момент времени может быть осуществлен только один переход; таким образом, условия являются взаимно исключающими для любого события.

Существует два особых состояния: вход и выход. Любое действие, связанное с событием входа, выполняется, когда объект входит в данное состояние. Событие выхода выполняется в том случае, когда объект выходит из данного состояния.

Диаграммы состояний хорошо использовать для описания поведения некоторого объекта в нескольких различных вариантах использования. Они не слишком пригодны для описания поведения ряда взаимодействующих объектов.

Рекомендуется строить диаграммы состояний только для тех классов, поведение которых влияет на общее поведение системы, например для классов пользовательского интерфейса и управляющих объектов.

**Таблица 13.1.** Описание кнопок панели инструментов диаграмм состояний Rational Rose









Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item
	Состояние	State
	Вход	Start State

Таблица 13.1. Описание кнопок панели инструментов диаграмм состояний Rational Rose

Кнопка	Описание	Название
	Выход	End State
	Переход в состояние	State Transition
	Возвращение	Transition to Self

## 2. Пример

На рис. 13.2 и 13.3 приведены диаграммы состояний экземпляра класса «Студент». Эти диаграммы показывают состояния экземпляра в ходе взаимодействия объекта класса «Студент» с БД студентов. Первая диаграмма расписывает состояния объекта подробно, а вторая показывает только общее состояние взаимодействия с БД.

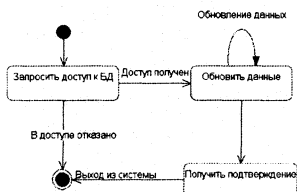


Рис. 13.2. Диаграмма 1

Найдем численную оценку для каждой из диаграмм.

### Диаграмма 1

Так как на диаграмме состояний связи отсутствуют, проведем расчет по сокращенной формуле:

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{20}{1 + 5 + 1} = 2,86.$$

### Диаграмма 2

$$S = \frac{\sum S_{Obj}}{1 + Obj + \sqrt{T_{Obj}}} = \frac{12}{1 + 3 + 1} = 2,4.$$



**Рис. 13.3.** *Диаграмма 2*

Полученный результат объясняется наличием недостаточно детализированного состояния на диаграмме 2.

### 3. Задания

1. Выбрать в моделируемой системе классы, для объектов которых будут строиться диаграммы состояний.
2. Построить для каждого выбранного класса диаграмму состояний, характеризующую поведение его объектов в нескольких вариантах использования.

### 4. Контрольные вопросы

1. Каково назначение диаграмм состояния?
2. Как отображаются действия и деятельности на диаграммах состояния?
3. Что такое условный переход и как он описывается на диаграмме?
4. Какие особые состояния объекта отображаются на диаграмме?
5. Каковы преимущества и недостатки диаграмм состояния?

# Лабораторная работа № 14

## Диаграммы пакетов, компонентов и размещения

### Цель работы:

- изучение диаграмм пакетов, диаграммы компонентов и диаграммы размещения,
- изучение их применения в процессе проектирования.

### 1. Диаграммы пакетов (package diagrams)

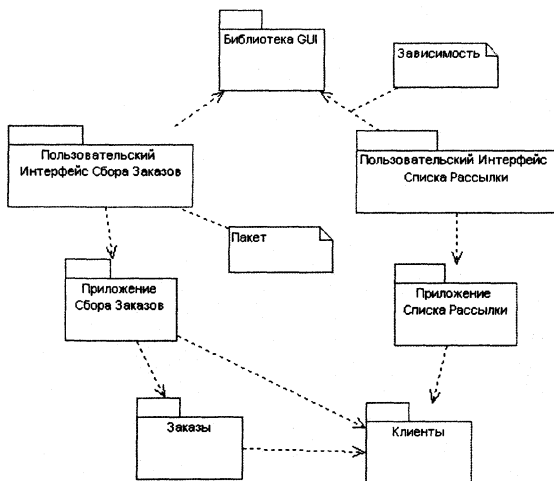
Один из важнейших вопросов методологии создания программного обеспечения – как разбить большую систему на небольшие подсистемы? Именно с этой точки зрения изменения, связанные с переходом от структурного подхода к объектно-ориентированному, являются наиболее заметными. Одна из идей заключается в группировке классов в компоненты более высокого уровня. В UML такой механизм группировки носит название пакетов (package).

Диаграммой пакетов является диаграмма, содержащая пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, т. е. диаграмма пакетов – это всего лишь форма диаграммы классов. Однако на практике причины построения таких диаграмм различны.

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс ссылается на другой как на параметр операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может стать неправильным.

В идеальном случае только изменения в интерфейсе класса должны воздействовать на другие классы. Искусство проектирования больших систем включает в себя минимизацию зависимостей, которая снижает воздействие изменений и требует меньше усилий на их внесение.

На рис. 14.1 мы имеем дело с классами предметной области, моделирующими деятельность организации и сгруппированными в два пакета: «Клиенты» и «Заказы».



**Рис. 14.1.**

«Приложение сбора заказов» имеет зависимости с обоими пакетами предметной области. «Пользовательский интерфейс сбора заказов» имеет зависимости с «Приложением сбора заказов» и «Библиотекой GUI».

Зависимость между двумя пакетами существует в том случае, если имеется какая-либо зависимость между любыми двумя классами в пакетах. Например, если любой класс в пакете «Список рассылки» зависит от какого-либо класса в пакете «Клиенты», то между соответствующими пакетами существует зависимость.

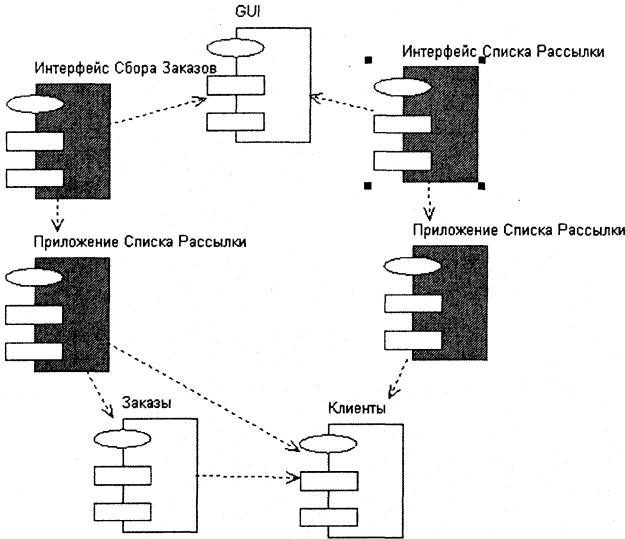
Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится трудночитаемой.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в разрабатываемой системе, однако они помогают выделить эти зависимости. Сведение к минимуму количества зависимостей позволяет снизить связанность компонентов системы. Но эвристический подход к этому процессу далек от идеала.

Пакеты особенно полезны при тестировании. Каждый пакет при тестировании может содержать один или несколько тестовых классов, с помощью которых проверяется поведение пакета.

## 2. Диаграммы компонентов (component diagrams)

Компоненты на диаграмме компонентов представляют собой физические модули программного кода (рис. 14.2). Обычно они в точности соответствуют пакетам на диаграмме пакетов (см. рис. 14.1); таким образом, диаграмма компонентов отражает выполнение каждого пакета в системе.














**Рис. 14.2.**

Зависимости между компонентами должны совпадать с зависимостями между пакетами. Эти зависимости показывают, каким образом одни компоненты взаимодействуют с другими. Направление данной зависимости показывает уровень осведомленности о коммуникации.

**Таблица 14.1.** Описание кнопок панели инструментов диаграмм компонентов Rational Rose

Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note

*Таблица 14.1. Описание кнопок панели инструментов диаграмм компонентов Rational Rose*

<b>Кнопка</b>	<b>Описание</b>	<b>Название</b>
	Связь комментария с элементом	Anchor Note to Item
	Компонент	Component
	Пакет	Package
	Зависимость	Dependency
	Спецификация подпрограммы	Subprogram Specification
	Тело подпрограммы	Subprogram Body
	Главная программа	Main Program
	Спецификация пакета	Package Specification
	Тело пакета	Package Body
	Спецификация задания	Task Specification
	Тело задания	Task Body

### **3. Диаграммы размещения (deployment diagrams)**

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе.

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

На рис. 14.3 изображен персональный компьютер (ПК), связанный с UNIX-сервером посредством протокола TCP/IP. Соединения между узлами показывают коммуникационные каналы, с помощью которых осуществляются системные взаимодействия.

На практике данные диаграммы применяются не слишком часто. В целом эти диаграммы полезно применять, чтобы выделить особенные физи-



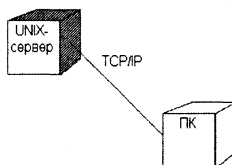


Рис. 14.3.

ческие характеристики данной системы. По мере распространения распределенных систем важность данных диаграмм возрастает.

Таблица 14.2. Описание кнопок панели инструментов диаграмм размещения Rational Rosee

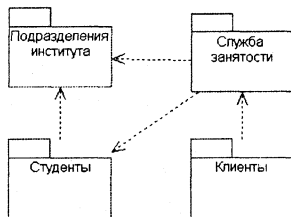
Кнопка	Описание	Название
	Выбор элемента модели	Selection Tool
	Ввод текста	Text Box
	Комментарий	Note
	Связь комментария с элементом	Anchor Note to Item
	Процессор	Processor
	Соединение	Connection
	Устройство	Device

## 4. Примеры

Проводить сравнение диаграмм пакетов, компонентов и размещения в общем случае бессмысленно, так как эти диаграммы не существуют сами по себе, а являются интерпретацией некоторой диаграммы классов, для которой и уместно проводить сравнение с другими диаграммами классов.

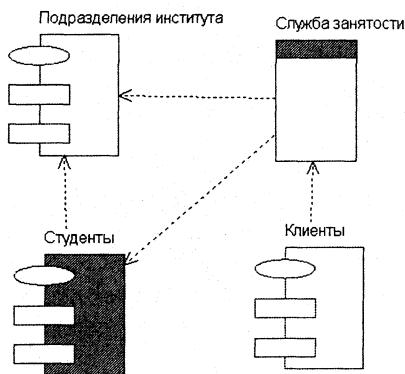
Диаграммы пакетов содержат один тип элементов – пакет и один тип связей – зависимость, поэтому численная оценка для диаграммы пакетов не столь важна, как для диаграммы классов.

На рис. 14.4 изображена диаграмма пакетов подсистемы «Служба занятости в рамках вуза» системы «Дистанционное обучение». Численная оценка для нее равна:



**Рис. 14.4.** *Диаграмма пакетов*

Диаграммы компонентов и размещения строятся и используются на этапе реализации и сопровождения, когда базовая архитектура системы уже обычно определена; поэтому они однозначно получаются из диаграммы классов и для них достаточно привести по одному примеру.



**Рис. 14.5.** *Диаграмма компонентов*

На рис. 14.5 изображена диаграмма компонентов, построенная на основе диаграммы пакетов, изображенной на рис. 14.4. На рис. 14.6 изображена диаграмма размещения подсистемы «Служба занятости в рамках вуза». Оценка для данной диаграммы компонентов равна:

$$S = \frac{\sum S_{Obj} + S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{16 + 8}{1 + 4 + \sqrt{2}} = 3,74,$$

т. е. равна оценке для соответствующей диаграммы пакетов.

Оценка для диаграммы размещения равна:

$$S = \frac{\sum S_{Obj} + S_{Lnk}}{1 + Obj + \sqrt{T_{Obj} + T_{Lnk}}} = \frac{12 + 4}{1 + 5 + 2} = 2.$$

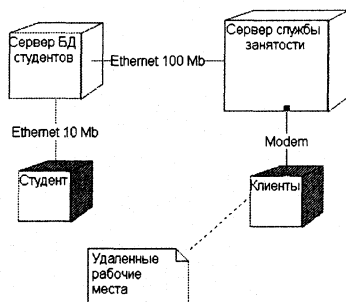


Рис. 14.6. Диаграмма размещения

### 5. Задания

1. Построить для моделируемой системы общую диаграмму пакетов, отметить на ней пакеты с необходимыми системными библиотеками, отобразить зависимости между пакетами.
2. Построить для данной системы диаграмму компонентов, соответствующую построенной диаграмме пакетов, системные пакеты изобразить в виде спецификаций пакетов.
3. Построить для проектируемой системы несколько вариантов диаграммы размещения (для архитектуры «клиент-сервер», трехуровневой архитектуры и т. д.), обосновать каждый вариант, предложить наиболее оптимальный.

### 6. Контрольные вопросы

1. Какую проблему проектирования призваны решить диаграммы пакетов?
2. В чем отличие диаграмм пакетов от диаграмм классов?

3. В чем смысл зависимости между элементами диаграммы пакетов?
4. Что такое интерфейс класса?
5. По каким признакам классы группируются в пакеты?
6. Какие виды элементов модели представлены на диаграмме компонентов?
7. Как связаны между собой диаграммы пакетов и диаграммы компонентов?
8. Что показывает диаграмма размещения?
9. Какие сущности отображаются на диаграммах размещения?
10. В каких случаях необходимо применение диаграмм размещения?

# Лабораторная работа № 15

## Генерация исходных текстов программ

### Цель работы:

- изучение возможностей кодогенерации Rational Rose 98.

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. Система кодогенерации Rational Rose позволяет, наряду с другими средствами проектирования, построить процесс разработки программного обеспечения как производственный процесс со строгим распределением ролей, полномочий и т. д.

Для демонстрационных целей достаточно спроектировать только один класс. Назовем его `String`. В его обязанности должны входить основные операции над массивами (печать, копирование, сравнение, получение размера). В качестве примера опишем сначала данный класс на C++:

```
Class String
Protected:
    Char *TmpString;
Public:
    Int Counter;
    Int Stat;
    Int GetStringSize(Char *);
    Int PrintString(Char *);
    Int CmpString(Char *, Char *);
    Int CpyString(Char *, Char *);
};
```

Теперь средствами Rose все спроектируем в графическом виде.

Каждый атрибут задается отдельно, с комментарием, и расписывается тип (`public`, `protected`, `private`). На рис. 15.1 показан разворот спецификации для `TmpString`.

Подобным образом расписываются все переменные.

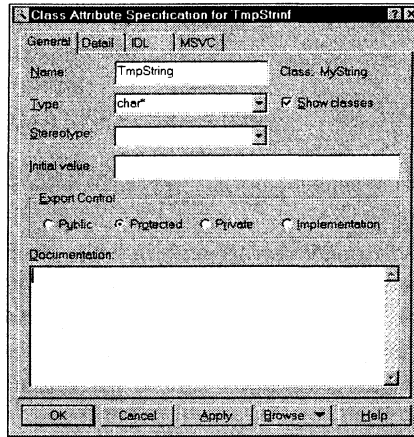


Рис. 15.1.

В плане описания функций все аналогично, только помимо описания самой функции (тип возвращаемого значения) необходимо расписать специфику каждого входного параметра, снабдив все это подробнейшими комментариями. Во всех продуктах компании Rational принято давать комментарии для любой малозаметной операции, поскольку впоследствии, при генерации отчетов не нужно будет еще раз вручную доводить документ, чтобы показать его руководству или передать разработчику в качестве технического задания.

Результатом выполнения вышеописанных действий будет появление класса с расписанными спецификациями. Сам класс показан на рис. 15.2. Можно отметить, что в графическом виде можно оценить основные свойства каждого элемента.

Следующий шаг в работе – получение кода на C++. Здесь хочется развеять существующий миф о 100 %-ной генерации кода. Rational Rose 98 в принципе не может дать готового кода, она способна лишь спроектировать класс и расписать спецификацию каждого элемента, подставить шаблоны членов класса для дальнейшего заполнения кодом. А вот для 100 %-ной генерации рабочего кода на C++ используется Rational Rose RealTime, которая в данном случае не рассматривается.

Итак, вернемся к кодогенерации (точнее сказать, к классогенерации). Через систему меню (Tools) выбираем поддерживаемый язык для описания спроектированного класса (в данном случае это C++), вызываем Code

MyString
oTmpStrinf char
oStat integer
oCount integer
*GetStringSize()
*PrintString()
*CmpString()
*CpyString()

Рис. 15.2.

Generational. Результатом работы будет появление двух файлов: MyString.h и MyString.cpp. В первом расписывается сам класс, а второй является шаблоном для дальнейшего заполнения соответствующим кодом.

Ниже приводятся распечатки обоих файлов.

Весь приведенный материал получен без изменения настроек и без дополнительной правки.

Имея подобный шаблон, становится не важно, какой именно разработчик начал создавать кодирование логики класса.

Для получения же подробного отчета по классу или технического задания можно воспользоваться инструментом Rational SoDA.

Следующая задача, с которой поможет справиться Rational Rose, – анализ существующей системы. Зачем переписывать и документировать крупные системы заново, если можно воспользоваться функцией обратного проектирования, что позволит из имеющегося кода построить визуальную модель и уже визуально дописать необходимые свойства и атрибуты, дописать новые классы. А под конец сгенерировать весь спектр файлов, необходимых для дальнейшей работы программистов. Данный подход называется Round-Trip и полностью поддерживается в RationalRose.

## 1. Задания

- Сгенерировать код для нескольких классов системы, сравнить модель и полученный исходный код.

## 2. Пример кодогенерации

Файл MyString.h

```
//## begin module%1.3%.codegen_version preserve=yes
// Read the documentation to learn more about C++ code generator versioning.
//## end module%1.3%.codegen_version
```

```
//## begin module%395AF70D0321.cm preserve=no
// %X% %Q% %Z% %W%
//## end module%395AF70D0321.cm

//## begin module%395AF70D0321.cp preserve=no
//## end module%395AF70D0321.cp

//## Module: MyString%395AF70D0321; Pseudo Package specification
//## Source file: C:\Program Files\Rational\Rose\C++\source\MyString.h

#ifndef MyString_h
#define MyString_h 1

//## begin module%395AF70D0321.additionalIncludes preserve=no
//## end module%395AF70D0321.additionalIncludes

//## begin module%395AF70D0321.includes preserve=yes
//## end module%395AF70D0321.includes

//## begin module%395AF70D0321.additionalDeclarations preserve=yes
//## end module%395AF70D0321.additionalDeclarations

//## begin MyString%395AF70D0321.preface preserve=yes
//## end MyString%395AF70D0321.preface

//## Class: MyString%395AF70D0321
// Данный класс позволяет проводить различные операции
// над массивами символов.
//## Category: <Top Level>
//## Persistence: Transient
//## Cardinality/Multiplicity: n

class MyString
{
//## begin MyString%395AF70D0321.initialDeclarations preserve=yes
//## end MyString%395AF70D0321.initialDeclarations
```



## Лабораторная работа № 15

---

```
public:
    /// Constructors (generated)
    MyString();

    /// Destructor (generated)
    ~MyString();

    /// Assignment Operation (generated)
    MyString & operator=(const MyString &right);

    /// Equality Operations (generated)
    int operator==(const MyString &right) const;

    int operator!=(const MyString &right) const;

    /// Other Operations (specified)
    /// Operation: GetStringSize%395AF87900E9
    // Подсчитывает количество символов в переданном массиве
    Int GetStringSize (Char *massiv // Указатель на массив
    );

    /// Operation: PrintString%395AF88800B9
    // Печатает на экране переданный массив
    Int PrintString (Char *Massiv // Указатель на массив
    );

    /// Operation: CmpString%395AF892013F
    // Сравнивает два массива.
    Int CmpString (Char *Str1, // Указатель на первый массив
    Char *Str2 // Указатель на второй массив
    );

    /// Operation: CpyString%395AF89C00D5
    // Копирует один массив в другой
    Int CpyString (Char *Dest, // Назначение
    Char *Source // Источник
    );

    /// Get and Set Operations for Class Attributes (generated)
```

```
//### Attribute: Stat%395AF8BB0289
// Общедоступная переменная числа обращений к PrintString
const Int get_Stat () const;
void set_Stat (Int value);

//### Attribute: Count%395AF8C20148
// Определяет статус определенного объекта
const Int get_Count () const;
void set_Count (Int value);

// Additional Public Declarations
//### begin MyString%395AF70D0321.public preserve=yes
//### end MyString%395AF70D0321.public

protected:
// Additional Protected Declarations
//### begin MyString%395AF70D0321.protected preserve=yes
//### end MyString%395AF70D0321.protected

private:
//### Get and Set Operations for Class Attributes (generated)

//### Attribute: TmpString%395AF8B201E5
// Временный указатель на строковый массив. Можно использовать
// в качестве буфера
const Char * get_TmpString () const;
void set_TmpString (Char * value);

// Additional Private Declarations
//### begin MyString%395AF70D0321.private preserve=yes
//### end MyString%395AF70D0321.private

private:
//### implementation
// Data Members for Class Attributes

//### begin MyString::TmpString%395AF8B201E5.attr preserve=no
// private: Char * U Char *TmpString;
//### end MyString::TmpString%395AF8B201E5.attr
```

## Лабораторная работа № 15

---

```
///  
/// begin MyString::Stat%395AF8BB0289.attr preserve=no public: Int U  
Int Stat;  
///  
/// end MyString::Stat%395AF8BB0289.attr  
  
///  
/// begin MyString::Count%395AF8C20148.attr preserve=no public: Int U  
Int Count;  
///  
/// end MyString::Count%395AF8C20148.attr  
  
// Additional Implementation Declarations  
///  
/// begin MyString%395AF70D0321.implementation preserve=yes  
///  
/// end MyString%395AF70D0321.implementation  
};  
  
///  
/// begin MyString%395AF70D0321.postscript preserve=yes  
///  
/// end MyString%395AF70D0321.postscript  
  
// Class MyString  
  
///  
/// Get and Set Operations for Class Attributes (inline)  
  
inline const Char * MyString::get_TmpString () const  
{  
///  
/// begin MyString::get_TmpString%395AF8B201E5.get preserve=no  
  
return TmpString;  
///  
/// end MyString::get_TmpString%395AF8B201E5.get  
}  
  
inline void MyString::set_TmpString (Char * value)  
{  
///  
/// begin MyString::set_TmpString%395AF8B201E5.set preserve=no  
TmpString = value;  
///  
/// end MyString::set_TmpString%395AF8B201E5.set  
}  
  
inline const Int MyString::get_Stat () const  
{  
///  
/// begin MyString::get_Stat%395AF8BB0289.get preserve=no  
return Stat;  
}
```

```

### end MyString::get_Stat%395AF8BB0289.get
}

inline void MyString::set_Stat (Int value)
{
### begin MyString::set_Stat%395AF8BB0289.set preserve=no
Stat = value;
### end MyString::set_Stat%395AF8BB0289.set }

inline const Int MyString::get_Count () const
{
### begin MyString::get_Count%395AF8C20148.get preserve=no
return Count;
### end MyString::get_Count%395AF8C20148.get
}

inline void MyString::set_Count (Int value)
{
### begin MyString::set_Count%395AF8C20148.set preserve=no
Count = value;
### end MyString::set_Count%395AF8C20148.set
}

### begin module%395AF70D0321.epilog preserve=yes
### end module%395AF70D0321.epilog

#endif

```

### Файл MyString.cpp

```

### begin module%1.3%.codegen_version preserve=yes
// Read the documentation to learn more about C++ code generator versioning.
### end module%1.3%.codegen_version
### begin module%395AF70D0321.cm preserve=no
// %X% %Q% %Z% %W%
### end module%395AF70D0321.cm

### begin module%395AF70D0321.cp preserve=no
### end module%395AF70D0321.cp

```



```
int MyString::operator==(const MyString &right) const
{
  /// begin MyString::operator==%395AF70D0321_eq.body preserve=yes
  /// end MyString::operator==%395AF70D0321_eq.body
}
```

```
int MyString::operator!=(const MyString &right) const
{
  /// begin MyString::operator!=%395AF70D0321_neq.body preserve=yes
  /// end MyString::operator!=%395AF70D0321_neq.body
}
```

```
/// Other Operations (implementation)
Int MyString::GetStringSize (Char *massiv)
{
  /// begin MyString::GetStringSize%395AF87900E9.body preserve=yes
  /// end MyString::GetStringSize%395AF87900E9.body
}
```

```
Int MyString::PrintString (Char *Massiv)
{
  /// begin MyString::PrintString%395AF88800B9.body preserve=yes
  /// end MyString::PrintString%395AF88800B9.body
}
```

```
Int MyString::CmpString (Char *Str1, Char *Str2)
{
  /// begin MyString::CmpString%395AF892013F.body preserve=yes
  /// end MyString::CmpString%395AF892013F.body
}
```

```
Int MyString::CpyString (Char *Dest, Char *Source)
{
  /// begin MyString::CpyString%395AF89C00D5.body preserve=yes
  /// end MyString::CpyString%395AF89C00D5.body
}
```



# Лабораторная работа № 16

## Обратное проектирование (Reverse engineering)

### Цель работы:

- изучение средств обратного проектирования Rational Rose.

Обратным проектированием называется процесс преобразования в модель кода, записанного на каком-либо языке программирования. В результате этого процесса получается огромный объем информации, часть которой находится на более низком уровне детализации, чем необходимо для построения полезных моделей. В то же время обратное проектирование никогда не бывает полным. Поскольку прямое проектирование ведет к потере информации, полностью восстановить модель на основе кода не удастся, если только инструментальные средства не включали в комментариях к исходному тексту информацию, выходящую за пределы языка реализации.

Одно из неоспоримых преимуществ Rational Rose – обратное проектирование, поскольку разработчику и проектировщику важно увидеть перед изменениями уже работающую систему в нормальном графическом представлении. Как правило, визуально-графический ряд оказывает куда большее воздействие, нежели пролистывание технических заданий и программных текстов. Тем более что проект, подвергшийся обратному проектированию, может быть доработан и вновь сгенерирован (а впоследствии и скомпилирован). Rational Rose предоставляет для этого все необходимые средства.

Для осуществления обратного проектирования в Rational Rose предусмотрен мощный модуль Analyzer, чье основное предназначение, вытекающее из названия, – анализ программ, написанных на C и C++. Данный модуль способен проанализировать имеющийся файл на одном из вышеупомянутых языков 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. В поставки Data Modeler данный модуль не входит, поскольку специфика поставки не предусматривает генерации кода и обратного проектирования.



Для правильного преобразования кода в модель необходимо провести несколько настроек.

На рис. 16.1 показан внешний вид программы в стандартных настройках и с незагроможденным экраном.

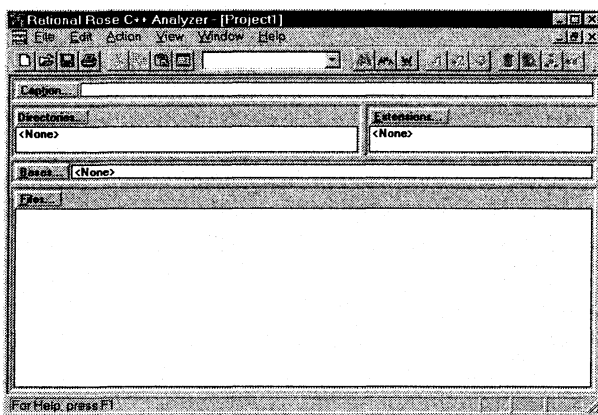


Рис. 16.1.

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

- **Caption** – имя проекта. Впоследствии имя модели будет определено по имени проекта.
- **Directories** – путь к исходящей директории. По умолчанию Rose использует для хранения исходящих модельных файлов директорию 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 обратного проектирования необходимо заполнить вышеописанные поля. Все файлы, подлежащие обратному проектированию, указываются в поле Files. Следует учитывать, что при этом вы получаете визуальную модель взаимодействия классов и структур; стало

быть, речь не идет о том, чтобы на визуальной модели отразился существующий код системы. Далее, все нестандартные конструкции не будут выведены в модель (анализатор 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 есть инструменты тонкой настройки, позволяющие настроить все таким образом, чтобы специфика конкретного проекта была полностью учтена.

Процесс обратного проектирования делится на два этапа: анализ и генерацию 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 обратного проектирования, 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 способа обратного проектирования.

Поговорим подробнее о следующих трех стандартных способах:

- 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 по усмотрению пользователя. 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 Analyser: 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, т. е. 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 проектный файл, содержащий в себе и директивы `#INCLUDE` для определений, и комментарии, а также прочие сопроводительные инструкции. И естественно, разработчику хочется иметь такой инструмент, который адекватно будет реагировать на все составляющие. Для этого 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 к соответствующему элементу (переменной, массиву... и т. д.).
- Способен закачать в проект все заголовочные файлы (по цепочке один за другим).

Теперь от общих фраз перейдем к практике. Нашей целью будет получение графической модели из класса на языке программирования.

Особо хочется еще раз обратить внимание на комментарии. Каждая строка снабжена комментарием. Смысл обратного проектирования состоит не только в том, чтобы корректно нарисовать модель, но и для правильного описания спецификации каждой составляющей класса.

За основу программы возьмем следующий класс:

```
//It's main class
class string
public:
    char *string; //Structure's pointer
    int buffer[100]; //Temporary buffer
    char name[10]={"Massiv"}; //Name of data
    int a; //Integer
    int b; //Integer
    void string(void); //constructor
```

```

void ~string(void); //destructor
char StringCopy(char *, //Buffer
char *, //source1
char *); //source2
private:
    int tmp_a;
    int tmp_b;
;

```

Результат обратного проектирования представлен на рис. 16.2.

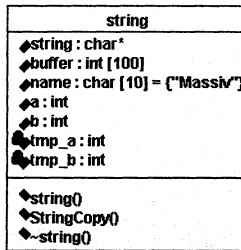


Рис. 16.2.

Рис. 16.2 показывает модель класса string. А на рис. 16.3 отображается вкладка, описывающая функции класса. Как и в случае с переменными имена функций отображаются на экране. Также доступен вход в спецификации конкретной функции. Если еще раз вернуться к листингу, то можно обратить внимание на декларацию функции StringCopy, в которой входные параметры подробно документированы. Так вот: если был применен подобный подход к документированию, то комментарий каждого параметра перенесется в качестве описательного комментария в соответствующую часть атрибута модели класса. То есть, получается, очень выгодно подвергать обработке исходные тексты, написанные по всем правилам программирования.

Выразительные средства визуального проектирования и анализа делают Rational Rose незаменимым инструментом при создании крупных информационных систем. Особенно полно Rose раскрывает свои возможности при анализе эффективности не новой системы, а уже существующей. Вышеуказанные примеры показывают, что даст инструмент при анализе проекта на предмет повышения его эффективности. Данная проблема

не является надуманной, поскольку подобный анализ нужен компаниям переводящим, например, старое программное обеспечение на новые платформы и новые технологии.

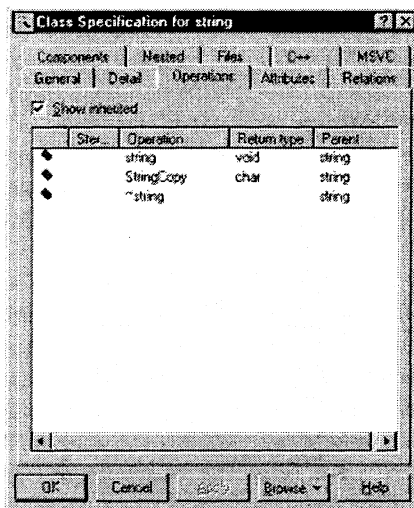


Рис. 16.3.

Как видим, модель ничего не теряет в результате обратного проектирования, в то же время само обратное проектирование представляет мощный механизм анализа эффективности существующих программных наработок нажатием на одну-две кнопки.

Rational Rose имеет в своем арсенале возможность прямого и обратного проектирования на ADA, Java, C++, COM, DDL, Basic, XML; схемы Oracle и Sql srv. Rose имеет открытое, хорошо документированное API, позволяющее любому человеку создать дополнительный модуль (мост) для любого языка. На сегодняшний день Rose – это уникальный продукт в плане открытости архитектуры.

## 1. Задания

1. Построить модель Rational Rose на основе произвольной программы на C++.
2. Сравнить исходные тексты программы и полученную модель.

## 2. Контрольные вопросы

1. Для чего предназначено обратное проектирование?
2. Какой модуль используется для анализа исходного кода?
3. Какая часть кода не используется при построении модели?
4. Какие основные способы обратного проектирования присутствуют в Rational Rose?
5. Как при обратном проектировании описываются внешние связи системы?

# Заключение

## Сравнение объектно-ориентированного и структурного методов проектирования

### Введение

При проектировании сложной программной системы необходимо разделять ее на все меньшие и меньшие подсистемы, каждую из которых можно совершенствовать независимо. В этом случае мы не превысим пропускной способности человеческого мозга: для понимания любого уровня системы нам необходимо одновременно держать в уме информацию лишь о немногих ее частях (отнюдь не о всех). Декомпозиция вызвана сложностью программирования системы, поскольку именно эта сложность вынуждает делить пространство состояний системы.

### 1. Алгоритмическая декомпозиция

Большинство проектировщиков формально обучено структурному проектированию «сверху вниз», и мы воспринимаем декомпозицию как обычное разделение алгоритмов, где каждый модуль системы выполняет один из этапов общего процесса. На рис. 17.1 приведен в качестве примера один из продуктов структурного проектирования: структурная схема, которая показывает связи между различными функциональными элементами системы. Данная структурная схема иллюстрирует часть программной схемы, изменяющей содержание управляющего файла. Она была автоматически получена из диаграммы потока данных специальной экспертной системой, которой известны правила структурного проектирования.

#### 1.1. Объектно-ориентированная декомпозиция

Предположим, что у этой задачи существует альтернативный способ декомпозиции. На рис. 17.2 мы разделили систему, выбрав в качестве критерия декомпозиции принадлежность ее элементов к различным абстракциям данной проблемной области. Прежде чем разделять задачу на шаги типа `Get formatted update` (Получить изменения в отформатированном виде) и `Add check sum` (Прибавить к контрольной сумме), мы должны определить такие объекты, как `Master File` (Основной файл) и `Check Sum`

(Контрольная сумма), которые заимствуются из словаря предметной области.

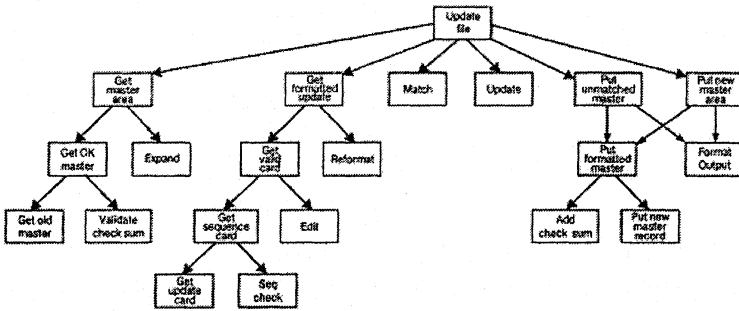
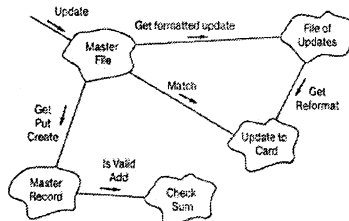


Рис. 17.1. Алгоритмическая декомпозиция

Хотя обе схемы решают одну и ту же задачу, но они делают это разными способами. Во второй декомпозиции мир представлен совокупностью автономных действующих лиц, которые взаимодействуют друг с другом, чтобы обеспечить поведение системы, соответствующее более высокому уровню. Get formatted update (Получить изменения в отформатированном виде) больше не присутствует в качестве независимого алгоритма; это действие существует теперь как операция над объектом File of Updates (Файл изменений). Эта операция создает другой объект – Update to Card (Изменения в карте). Таким образом, каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира. С этой точки зрения объект является вполне осязаемой вещью, которая демонстрирует вполне определенное поведение. Объекты что-то делают, и мы можем, пошлав им сообщение, попросить их выполнить то-то и то-то. Так как наша декомпозиция основана на объектах, а не на алгоритмах, мы называем ее объектно-ориентированной декомпозицией.

При проектировании сложной системы важны оба способа декомпозиции – по алгоритмам и по объектам. Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение агентам, которые являются либо объектами, либо субъектами действия. Однако нельзя сконструировать сложную систему одновременно двумя способами, тем более что эти способы, по сути, ортогональны. Необходимо начать разделение системы либо по алгоритмам, либо по объектам, а затем, используя полученную структуру, попытаться рассмотреть проблему с другой точки зрения.





**Рис. 17.2.** Объектно-ориентированная декомпозиция

Опыт показывает, что полезнее начинать с объектной декомпозиции. Такое начало помогает лучше организовать сложные программные системы. Объектная декомпозиция имеет несколько чрезвычайно важных преимуществ перед алгоритмической. Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов, что приводит к существенной экономии выразительных средств. Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируются на устойчивых промежуточных формах. Действительно, объектная декомпозиция существенно снижает риск при создании сложной программной системы, так как она развивается из меньших систем, в которых мы уже уверены. Более того, объектная декомпозиция помогает разобраться в сложной программной системе, предлагая разумные решения относительно выбора подпространства большого пространства состояний.

### 1.2. Методы проектирования программных систем

Необходимо разграничить понятия «метод» и «методология». Метод – это последовательный процесс создания моделей, которые описывают вполне определенными средствами различные стороны разрабатываемой программной системы. Методология – это совокупность методов, применяемых в жизненном цикле разработки программного обеспечения и объединенных одним общим философским подходом. Методы важны по нескольким причинам. Во-первых, они упорядочивают процесс создания сложных программных систем, как общие средства, доступные для всей группы разработчиков. Во-вторых, они позволяют менеджерам в процессе разработки оценить степень продвижения и риска.

Методы появились как ответ на растущую сложность программных систем. В 60 – 70-х годах было разработано много методов, помогающих справиться с растущей сложностью программ. Наибольшее распространение

ние получило структурное проектирование по методу «сверху вниз». Метод был непосредственно основан на топологии традиционных языков высокого уровня типа FORTRAN или COBOL. В этих языках основной базовой единицей является подпрограмма и программа в целом принимает форму дерева, в котором одни подпрограммы в процессе работы вызывают другие подпрограммы. Структурное проектирование использует именно такой подход: алгоритмическая декомпозиция применяется для разбиения большой задачи на более мелкие. Однако с ростом вычислительной мощности компьютеров оказалось, что структурный подход не работает, если объем программы превышает приблизительно 100 000 строк. В последнее время появились десятки методов, в большинстве которых устранены очевидные недостатки структурного проектирования. Большинство этих методов представляют собой вариации на одни и те же темы. Их можно разделить на три основные группы:

- метод структурного проектирования «сверху вниз»,
- метод потоков данных,
- объектно-ориентированное проектирование.

В каждом из этих подходов присутствует алгоритмическая декомпозиция. Следует отметить, что большинство существующих программ написано, по-видимому, в соответствии с одним из этих методов. Тем не менее структурный подход не позволяет выделить абстракции и обеспечить ограничение доступа к данным; он также не предоставляет достаточных средств для организации параллелизма. Структурный метод не может обеспечить создание предельно сложных систем, и он, как правило, неэффективен в объектных и объектно-ориентированных языках программирования.

В методе потоков данных программная система рассматривается как преобразователь входных потоков в выходные. Метод потоков данных, как и структурный метод, с успехом применялся при решении ряда сложных задач, в частности в системах информационного обеспечения, где существуют прямые связи между входными и выходными потоками системы и где не требуется уделять особого внимания быстродействию. В основе объектно-ориентированного проектирования лежит представление о том, что программную систему необходимо проектировать как совокупность взаимодействующих друг с другом объектов, рассматривая каждый объект как экземпляр определенного класса, причем классы образуют иерархию. Объектно-ориентированный подход отражает топологию новейших языков высокого уровня, таких, как Smalltalk, Object Pascal, C++, CLOS и Ada.

Объектно-ориентированный анализ и проектирование – метод, использующий объектную декомпозицию; объектно-ориентированный подход имеет свою систему условных обозначений и предлагает богатый набор логических и физических моделей, с помощью которых можно получить представление о различных аспектах рассматриваемой системы.

## 2. Сравнение диаграмм, используемых в структурном и объектно-ориентированном методах

Основными видами диаграмм, применяемых в структурном проектировании, являются функциональные диаграммы (IDEF0) и диаграммы потоков данных (DFD). Как было указано выше, функциональный и объектно-ориентированный методы проектирования ортогональны. Соответственно прямое сравнение диаграмм этих методов невозможно, так как на них отображаются принципиально разные сущности, выделенные на стадии анализа.

Однако можно сопоставить аналогичные по назначению диаграммы методов. На рис. 17.4 приведена диаграмма IDEF0, демонстрирующая обслуживание клиента в системе «Служба занятости в рамках вуза». На рис. 17.5 приведена аналогичная по назначению диаграмма последовательности. Диаграмма потоков данных представлена на рис. 17.6.



Рис. 17.3.

Из приведенных диаграмм видно, что при декомпозиции системы с точки зрения функций системы или потоков данных диаграммы IDEF0 и DFD содержат больше выразительных средств для демонстрации особенностей системы. Следовательно, можно сделать вывод о важности правильной декомпозиции системы на этапе анализа с точки зрения выбранного метода проектирования.

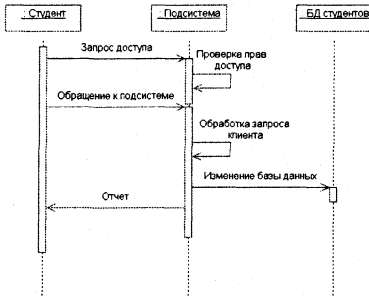


Рис. 17.4.

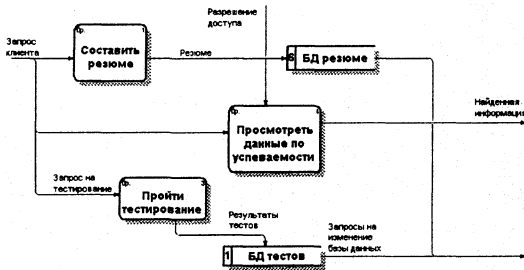


Рис. 17.5.

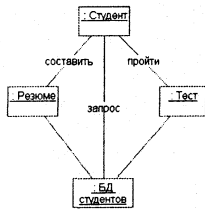


Рис. 17.6.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>Лабораторная работа №1.</b> <i>Теоретическое введение в предметную область</i>	7
<b>Лабораторная работа №2.</b> <i>Методология IDEF0</i>	21
<b>Лабораторная работа №3.</b> <i>Дополнение моделей процессов диаграммами</i>	38
<b>Лабораторная работа №4.</b> <i>Отчеты в BPWin</i>	56
<b>Лабораторная работа №5.</b> <i>Методология IDEF1X</i>	63
<b>Лабораторная работа №6.</b> <i>Создание логической модели</i>	69
<b>Лабораторная работа №7.</b> <i>Нормализация. Создание физической модели</i>	78
<b>Лабораторная работа №8.</b> <i>Отчеты в ERWin</i>	86
<b>Лабораторная работа №9.</b> <i>Введение в CASE-пакет Rational Rose 98</i>	92
<b>Лабораторная работа №10.</b> <i>Диаграммы вариантов использования</i>	101
<b>Лабораторная работа №11.</b> <i>Диаграммы классов</i>	106
<b>Лабораторная работа №12.</b> <i>Диаграммы взаимодействия</i>	113

<b>Лабораторная работа №13.</b>	
<i>Диаграммы состояний</i>	120
<b>Лабораторная работа №14.</b>	
<i>Диаграммы пакетов, компонентов и размещения</i>	124
<b>Лабораторная работа №15.</b>	
<i>Генерация исходных текстов программ</i>	132
<b>Лабораторная работа №16.</b>	
<i>Обратное проектирование (Reverse engineering)</i>	143
<b>Заключение</b>	150