Т.И. Сергеева М.Ю. Сергеев

ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Воронеж 2017

ФГБОУ ВО «Воронежский государственный
технический университет»

Т.И. Сергеева М.Ю. Сергеев

ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Утверждено учебно-методическим советом
университета в качестве учебного пособия

Воронеж 2017

УДК 681.32 (07)
ББК 32.81 я7
С32

Сергеева Т.И. Проектирование распределенных информационных систем: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017. - 152 с.

В учебном пособии рассматриваются методологии системного анализа и проектирования распределенных информационных систем.

Издание соответствует требованиям федерального государственного образовательного стандарта высшего образования для магистров направления 090401 «Информатика и вычислительная техника» по магистерской программе «Распределенные автоматизированные системы», дисциплине «Проектирование распределенных информационных систем».

Учебное пособие предназначено для студентов первого курса магистратуры.

Табл. 3. Ил. 77. Библиогр.: 10 назв.

Научный редактор д-р техн. наук, проф. С.Л. Подвальный

Рецензенты: кафедра вычислительной математики
и прикладных информационных технологий
Воронежского государственного
университета (зав. кафедрой д-р техн. наук,
проф. Т.М. Леденева);
д-р техн. наук, проф. В.Ф. Барабанов

ВВЕДЕНИЕ

Информационные системы предназначены для сбора, хранения и обработки информации, поэтому в основе любой из них лежит среда хранения и доступа к данным. В зависимости от предметной области информационные системы (ИС) могут весьма значительно различаться по своим функциям, архитектуре, реализации. Современные информационные системы ориентированы, прежде всего, на конечного пользователя, от которого не требуется высокая квалификация в области вычислительной техники.

Важнейшим этапом жизненного цикла ИС является проектирование. Решения, принятые на этапе проектирования, влияют на эффективность функционирования информационной системы.

В пособии рассматриваются две методологии проектирования информационных систем:

- методология системного анализа и функционального моделирования;
- объектное проектирование с применением языка UML.

В первой главе рассматриваются общие вопросы построения информационных систем, классификация ИС, состав ИС, этапы жизненного цикла ИС, классификация CASE-средств проектирования ИС.

Вторая глава посвящена изложению основ структурного подхода к проектированию ИС и технологии построения моделей SADT, DFD, ERD, STD.

Третья глава содержит описание технологии проектирования ИС с применением языка UML. Рассматриваются диаграммы, входящие в состав проекта ИС, и технологии их создания.

Четвертая глава описывает функциональные возможности инструментальной среды AllFusion Component Modeler, ко-

торую используют для объектно-ориентированного проектирования ИС.

Пятая глава посвящена описанию методологии и инструментальных средств BPwin для моделирования процессов.

Шестая глава описывает функциональные возможности ERwin по созданию моделей данных.

Пособие соответствует типовой программе по дисциплине «Проектирование распределенных информационных систем» и предназначено для магистров, обучающихся по программе «Распределенные автоматизированные системы».

1. ИНФОРМАЦИОННЫЕ СИСТЕМЫ

1.1. Определение информационной системы

Информационная система – организационно-техническая система, которая предназначена для выполнения информационно-вычислительных работ или предоставления информационно-вычислительных услуг.

ИС обеспечивают информационные потребности системы управления и ее пользователей путем использования и/или создания информационных продуктов. В качестве пользователей могут выступать: управленческий персонал, внешние пользователи (инвесторы, поставщики, покупатели).

Информационно-вычислительная работа – деятельность, связанная с созданием и использованием информационных продуктов. Типичный пример информационной работы – поддержка информационных технологий управления.

Информационный продукт – вещественный или нематериальный результат интеллектуального человеческого труда, связанный со сбором, хранением, передачей и обработкой информации. Результат может быть материализован на определенном носителе (программный продукт (приложение)), или является выходной информацией в виде документов, баз данных, хранилищ данных, баз знаний, проектов ИС.

Информационно-вычислительная услуга – это разовая информационно-вычислительная работа.

Методологическую основу изучения любой системы, в том числе ИС составляет системный подход. В соответствии с системным подходом любая система представляет собой совокупность взаимосвязанных объектов (элементов), функционирующих совместно для достижения общей цели.

С точки зрения системного подхода **ИС** представляет собой совокупность функциональной структуры, информационного, математического, технического, организационного и кадрового обеспечения, которые объединены в единую систему в целях сбора, хранения, обработки и выдачи необходимой информации для выполнения функций управления.

1.2. Классификация информационных систем

Информационные системы могут классифицироваться по следующим признакам:

- параметры объекта управления (сфера деятельности, масштаб, состав функций управления);
- организационная структура ИС;
- степень интеграции ИС;
- информационно-технологическая архитектура ИС;
- технологические процессы обработки данных;
- методология разработки ИС и др.

Классификация по сфере деятельности объекта управления может быть следующей:

- промышленное предприятие;
- сфера обращения (торговля, банки и кредитные организации);
- образование;
- социальная сфера и др.

Классификация по функциональной структуре ИС может быть следующая:

- автоматизация технической подготовки производства;
- маркетинг и стратегия развития предприятий;
- технико-экономической планирование;
- финансы (бухгалтерский учет, финансовый анализ);
- материально-техническое обеспечение;
- оперативно-календарное управление производством;
- управление сбытом готовой продукции;
- управление персоналом и др.

Классификация по организационной структуре ИС и границам ИС может быть следующей:

- автоматизированное рабочее место (АРМ) управленческого персонала;
- комплекс взаимосвязанных АРМ;
- ИС предприятия (организации);
- ИС отрасли;
- государственная ИС;
- международная ИС.

Классификация по степени интеграции ИС может быть следующей:

- локальная ИС (изолированное информационное пространство);
- частично интегрированная ИС (общее информационное пространство);
- полностью интегрированная корпоративная ИС.

Классификация по информационно-технологической архитектуре ИС может быть следующей:

- ИС с централизованной архитектурой построения (один центр хранения и обработки данных);
- ИС с распределенной архитектурой (компьютерные сети, наличие множества центров обработки и хранения информации).

Классификация по специализации ИС может быть следующей:

- информационные системы менеджмента или организационно-экономического управления (Information Management System – IMS);

- информационно-поисковые системы (Information Retrieval System – IRS);

- системы автоматизированного обучения (Education Information System – EIS) и др.

Наибольшее распространение получили ИС менеджмента. **ИС менеджмента** можно разделить на следующие группы:

- АСУП – автоматизированные системы управления ресурсами предприятия и организаций;

- АСУТП – автоматизированные системы управления технологическими процессами производства продукции;

- САПР – системы автоматизированного проектирования конструкций и технологий производства продукции и др.

Информационные системы менеджмента в качестве компонентов включают в себя другие специализированные ИС, предназначенные для следующих целей:

- автоматизация делопроизводства (Office Automation System – OAS);

- поддержка принятия решений (Design Support System – DSS);

- системы искусственного интеллекта (Artificial Intelligence System – AIS) и др.

Специализированные системы поддержки принятия решений создают для выработки стратегии развития предприятия (перспективные направления, планирование, инвестиционное проектирование и др.). Данные системы используют методы статистического анализа и прогнозирования, моделирования данных и бизнес-процессов, имитационного моделирования, так называемые корпоративные стратегические системы (Enterprise Strategic System – ESS). В ИС поддержки принятия решений нашли применение технологии оперативного анализа и обработки данных, полученных из хранилищ данных (Data

Warehouse), технологии извлечения информации из данных (Data Mining), моделирования бизнес-процессов.

В современных ИС менеджмента значительна роль и **ИС искусственного интеллекта** (AIS). Эти ИС:

- поддерживают естественно-языковый интерфейс для пользователей (специалистов по формализации знаний);
- предоставляют методы искусственного интеллекта для решения слабоструктурированных и плохо формализованных задач.

Ядром AIS является база знаний (Knowledge Base – KB). Она используется для формирования новой информации путем логического вывода. Для представления экономического объекта и его окружения, исследования его поведения и реакций на внешние события применяется математическое моделирование, средства дедуктивных и правдоподобных выводов, полученных на основе неполной или неточной информации.

Примеры информационных систем искусственного интеллекта:

- экспертные системы, с помощью которых на основе реальных данных выдвигается и дается оценка некоторой гипотезы;
- ИС полнотекстового поиска;
- нейронные сети;
- ИС аналитических вычислений на основе методов исследования операций, математического моделирования, статистического анализа и прогнозирования и др.

1.3. Состав информационных систем

В соответствии с системным подходом ИС как система разделяется на подсистемы.

Прежде всего, информационная система может быть разделена на функциональные подсистемы.

Функциональная подсистема ИС представляет собой комплекс задач с высокой степенью информационных обменов (связей) между задачами. При этом под задачей понимают некоторый процесс обработки информации с четко определенным множеством входной и выходной информации. Например, вычисление сдельной зарплаты, учет прихода материалов, оформление заказов на закупку и т.д. Таким образом, **функциональные подсистемы ИС** информационно обслуживают определенные виды деятельности системы (предприятия, фирмы, отдела), характерные для его структурных подразделений и (или) функций управления.

Состав функциональных подсистем во многом определяется особенностями системы, ее отраслевой принадлежностью, формой собственности, размером, характером деятельности предприятия или фирмы.

Функциональные подсистемы могут строиться на различных принципах:

- предметный;
- функциональный;
- проблемный;
- смешанный (предметно-функциональный).

Предметный принцип позволяет разделить ИС по следующим направлениям использования, например, применительно к предприятию:

- подсистема управления производственными и финансовыми ресурсами;
- подсистема материально-технического снабжения;
- подсистема управления персоналом;
- подсистема сбыта готовой продукции и т.д.

При этом в подсистемах рассматривается решение задач на всех уровнях управления, информационные потоки интегрируются по вертикали. Для реализации функций управления выделяют функциональные подсистемы: прогнозирование, нормирование, планирование (технико-экономическое и опера-

тивное), учет, анализ и регулирование. Эти функции реализуются на различных уровнях управления и объединены в следующие контуры управления: маркетинг, производство, логистика, финансы.

Примеры возможных задач функциональных подсистем, реализуемых на разных уровнях управления предприятием, представлены в табл.1.

Таблица 1

Задачи функциональных подсистем

Уровень управления	Функциональные подсистемы			
	Маркетинг	Производство	Логистика	Финансы
Стратегический	Поиск новых продуктов и услуг	Анализ производственных мощностей. Выбор технологий	Анализ материальных источников. Товарный прогноз	Анализ финансовых источников. Выбор модели уплаты налогов
Тактический	Анализ и планирование объемов сбыта	Анализ и планирование производственных программ	Анализ и планирование объемов закупок	Анализ и планирование денежных потоков
Оперативный	Обработка заказов клиентов. Выписка счетов и накладных	Обработка производственных заказов	Складские операции. Заказы на закупку	Ведение бухгалтерских книг

Функциональный принцип означает, что ИС разбивается на подсистемы в соответствии с функциями управления. Примером применения функционального подхода может служить многопользовательский сетевой комплекс полной автоматизации корпорации «Галактика» (АО «Новый атлант»), который включает в себя четыре контура автоматизации в соответствии с функциями управления: контуры планирования, оперативного управления, учета и контроля, анализа.

Проблемный принцип означает, что функциональные подсистемы ИС создаются для решения отдельных задач в рамках систем поддержки принятия решений. Например, решение задач бизнес-планирования, управления проектами. Такие подсистемы могут реализовываться в виде локальных информационных систем, импортирующих данные из корпоративной информационной системы (например, система бизнес-планирования на основе Project-Expert), или в виде специальных подсистем в рамках корпоративной информационной системы (например, информационной системы руководителя).

На практике чаще всего применяется смешанный (предметно-функциональный) подход. Согласно **предметно-функциональному** подходу построение функциональной структуры ИС – это разделение ее на подсистемы по характеру хозяйственной деятельности, которое должно соответствовать структуре объекта и системе управления, а также выполняемым функциям управления.

Используя этот подход, можно выделить следующий типовой набор функциональных подсистем в общей структуре ИС предприятия.

Функциональный принцип:

- стратегическое развитие;
- технико-экономическое планирование;
- бухгалтерский учет и анализ хозяйственной деятельности.

Предметный принцип (подсистемы управления ресурсами): техническая подготовка производства; основное и вспомогательное производство; качество продукции; логистика; маркетинг; кадры.

Подсистемы, построенные по функциональному принципу, охватывают все виды хозяйственной деятельности предприятия (производство, снабжение, сбыт, персонал, финансы). Подсистемы, построенные по предметному принципу, относятся в основном к оперативному уровню управления ресурсами.

Пример структуры подсистем ИС предприятия, выделенных по функционально-предметному принципу, приведен на рис. 1.

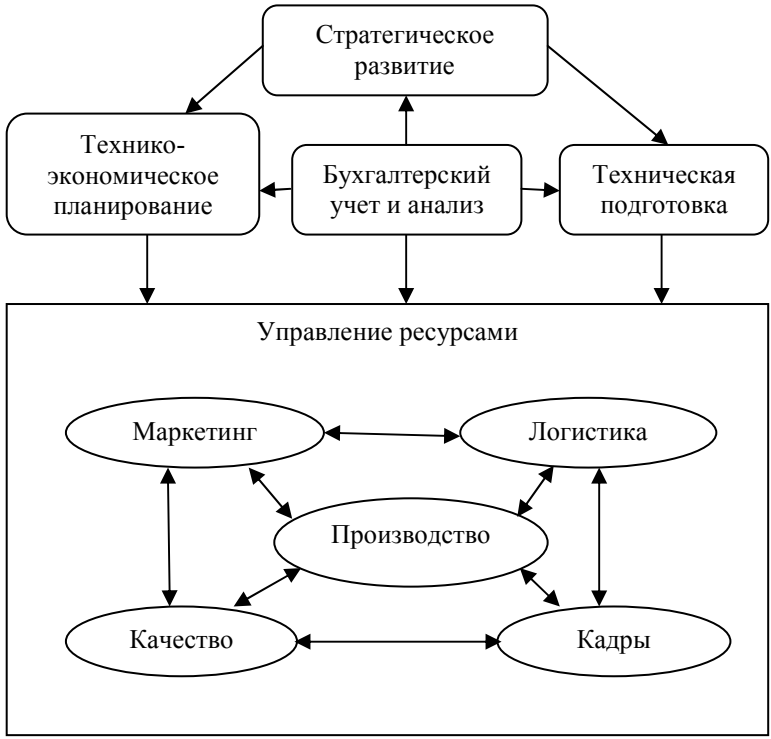


Рис. 1. Структура подсистем ИС

Обеспечивающие подсистемы информационных систем. Обеспечивающие подсистемы являются общими для всей ИС независимо от конкретных функциональных подсистем.

Состав обеспечивающих подсистем не зависит от выбранной предметной области и может включать:

- функциональную структуру;
- информационное обеспечение;
- математическое (алгоритмической и программное) обеспечение;
- техническое обеспечение;
- организационное обеспечение;
- кадровое обеспечение.

Функциональная структура представляет собой перечень реализуемых функций (задач) и отражает их соподчиненность. Функция ИС – это набор действий ИС, направленных на достижение частной цели управления. Состав функций, реализуемых в ИС, подразделяется на информационные и управляющие функции.

Информационные функции – это централизованный контроль (например, контроль значений параметров) и вычислительные и логические операции (например, тестирование работоспособности ИС, подготовка и обмен информацией с другими системами).

Управляющие функции должны осуществлять поиск и расчет рациональных режимов управления, реализацию заданных режимов управления.

Информационное обеспечение – это совокупность средств и методов построения информационной базы. Оно определяет способы и формы отображения состояния объекта управления в виде данных внутри ИС, документов, графиков и сигналов вне ИС. Информационное обеспечение подразделяют на внешнее и внутреннее. Состав информационного обеспечения представлен на рис. 2.

Рис. 2. Информационное обеспечение ИС

Математическое обеспечение состоит из алгоритмического и программного обеспечений (рис. 3).

Алгоритмическое обеспечение представляет собой совокупность математических методов, моделей и алгоритмов, используемых в системе для решения задач и обработки информации.

Программное обеспечение состоит из общего и специального обеспечения. В общее программное обеспечение входят, например, ОС, трансляторы, тесты и диагностика и т.д. В специальное программное обеспечение входят, например, прикладное программное обеспечение, обеспечивающее автоматизацию процессов управления в заданной предметной области.

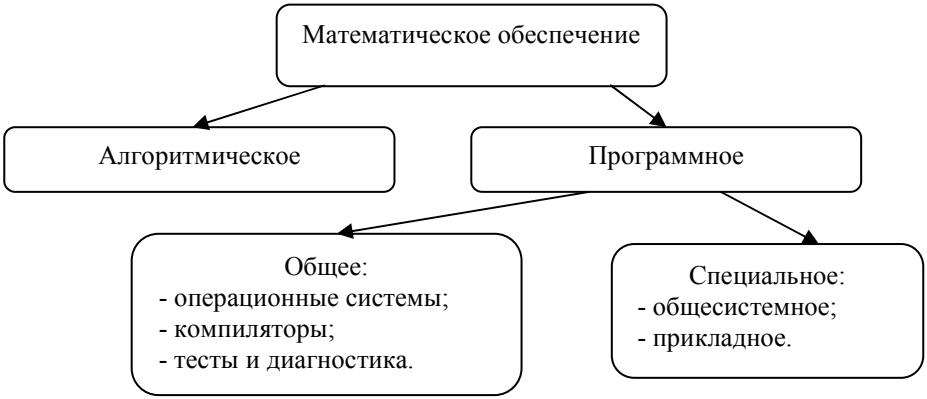


Рис. 3. Математическое обеспечение ИС

Техническое обеспечение состоит из устройств измерения, преобразования, передачи, хранения, обработки, отображения, регистрации, ввода/вывода информации и исполнительных устройств (рис. 4).

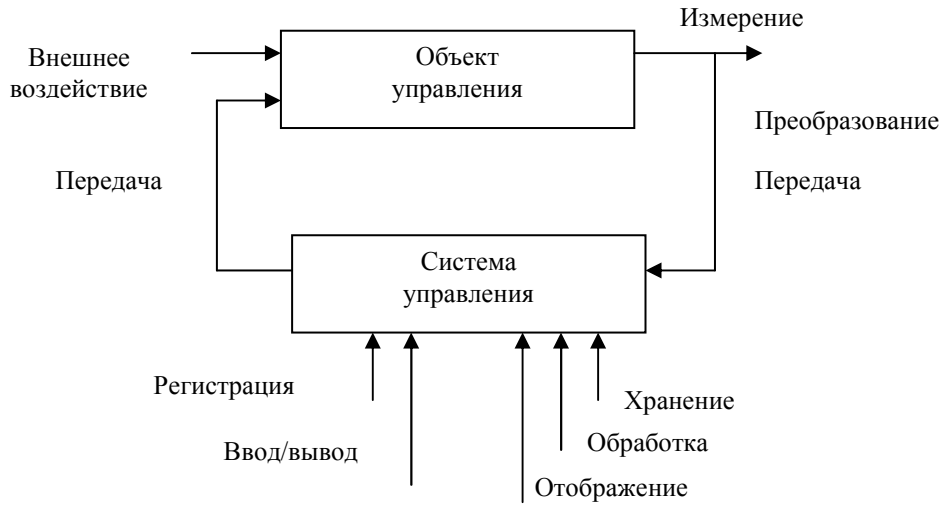


Рис. 4. Техническое обеспечение ИС

Кадровое обеспечение – это совокупность методов и средств по организации и проведению обучения персонала приемам работы с ИС. Его целью является поддержание работоспособности ИС и возможности дальнейшего ее развития. Кадровое обеспечение включает в себя методики обучения, программы курсов и практических занятий, технические средства обучения и правила работы с ними и т.д.

Организационное обеспечение – это совокупность средств и методов организации производства и управления им в условиях внедрения ИС. Целью организационного обеспечения является:

- выбор и постановка задач управления;
- анализ системы управления и путей ее совершенствования;
- разработка решений по организации взаимодействия ИС и персонала;
- внедрение задач управления.

Организационное обеспечение включает в себя методики проведения работ, требования к оформлению документов, должностные инструкции и т.д.

Это обеспечение является одной из важнейших подсистем ИС, от которой зависит успешная реализация целей и функций системы. В состав организационного обеспечения входят четыре группы компонентов.

Первая группа компонентов включает методические материалы, регламентирующие создание и функционирование системы. Это могут быть общеотраслевые руководящие методические материалы по созданию ИС, типовые проектные решения, методические материалы по организации и проведению предпроектного обследования на предприятиях, методические материалы по вопросам создания проектной документации.

Вторая группа компонентов включает средства, необходимые для эффективного проектирования и функционирования ИС. Это могут быть типовые пакеты прикладных программ,

типовые структуры управления предприятием, унифицированные системы документов, общесистемные и отраслевые классификаторы.

В третью группу входит техническая документация, получаемая в процессе обследования, проектирования, разработки и внедрения системы. В эту группу могут включить технико-экономическое обоснование; техническое задание; технический и рабочий проекты; документы, оформляющие поэтапную сдачу системы в эксплуатацию.

К четвертой группе относится подсистема, в которой представлено организационно-штатное расписание, определяющее, например, состав специалистов по функциональным подсистемам управления.

В состав обеспечивающих подсистем ИС дополнительно могут входить правовое обеспечение, лингвистическое обеспечение, технологическое обеспечение.

Правое обеспечение регламентирует процесс создания и эксплуатации ИС. Оно включает в себя совокупность юридических документов, которые регламентируют формирование, хранение, обработку промежуточной и результирующей информации системы.

Лингвистическое обеспечение представляет собой совокупность научно-технических терминов и других языковых средств, используемых в ИС, а также правил формализации естественного языка, методов сжатия и раскрытия текстовой информации.

Средства, входящие в лингвистическое обеспечение, делятся на две группы:

- традиционные языки (естественные, математические, алгоритмические, языки моделирования);

- языки для диалога с ЭВМ (информационно-поисковые, языки СУБД, операционных средств, входные языки пакетов прикладных программ).

Технологическое обеспечение ИС соответствует разделению ИС на подсистемы по технологическим этапам обработки различных видов информации:

- обеспечение обработки первичной документации (этап технологического процесса сбора, передачи, накопления, хранения, обработки первичной информации, получения и выдачи результирующей информации);

- распространение организационно-распорядительной документации (этап получения входящей документации, передачи на исполнение, формирования и хранения дел, составления и размножения внутренних документов и отчетов);

- хранение и распространение технологической документации и чертежей (этап ввода в систему и актуализации шаблонов изделий, ввода исходных данных и формирования проектной документации для новых видов изделий, выдачи на плоттер чертежей, обновление банка ГОСТов, ОСТов, технических условий, нормативных данных, подготовки и выдачи технологической документации по новым видам изделий);

- ведение баз данных и знаний (этап формирования баз данных и знаний, ввода и обработки запросов на поиск решения, выдачи вариантов решения);

- хранение и обработка научно-технической информации, ГОСТов и технических условий, правовых документов и дел (этап формирования поисковых образов документов, информационного фонда, ведение тезауруса справочника ключевых слов и их кодов, кодирования запросов на поиск, выполнение поиска и выдачи документов или адресов хранения документов).

Информационная система поддерживает работу следующих **категорий пользователей** (User).

Конечные пользователи – это управленческий персонал, специалисты, технический персонал, которые по роду своей деятельности используют информационные технологии управления.

Администрация ИС включает несколько категорий работников. Это системный аналитик, который обеспечивает управление эффективностью ИС и определяет перспективы развития ИС. Это администратор приложений, который отвечает за формализацию информационных потребностей бизнес-приложений, управление эффективностью и развитием бизнес-приложений. Администратор баз данных, который осуществляет эксплуатацию и поддержание в адекватном состоянии базы данных. Сетевой администратор, который обеспечивает надежную работу сети, управляет санкционированным доступом пользователей, устанавливает защиту сетевых ресурсов.

Системные и прикладные программисты – осуществляют создание, сопровождение и модернизацию программного обеспечения ИС.

Технический персонал – обеспечивает обслуживание технических средств обработки данных.

Внешние пользователи – потребители выходной информации ИС.

Техническое обеспечение можно классифицировать в соответствии с той ролью, которую она играет в технологическом процессе обработки информации:

- вычислительные машины (рабочие станции, серверы, персональные компьютеры), являющиеся центральным звеном системы обработки данных;
- периферийные технические средства, обеспечивающие ввод и вывод информации;
- сетевые коммуникации для передачи данных (компьютерные сети и телекоммуникационное оборудование);
- средства оргтехники и связи.

Технические средства обработки данных, программное обеспечение и организация БД в совокупности определяют **информационно-технологическую архитектуру ИС** (ИТА). Различают следующие **типы ИТА**.

Централизованная архитектура. Она означает, что хранение и обработка данных осуществляют на центральном компьютере. Это обеспечивает удобство в администрировании ИС. Недостатки такой организации следующие: ограничение на рост объемов хранимых данных, высокий уровень неработоспособности ИС.

Система телеобработки данных. Обеспечивает наиболее дешевый способ организации одновременной работы большого числа пользователей при использовании мощного центрального компьютера. Высокопроизводительные каналы телекоммуникации позволяют не зависеть от места обработки и хранения данных.

Многомашинный комплекс. Характеризуется интеграцией вычислительных ресурсов (внешней памяти, процессоров) нескольких компьютеров, расположенных недалеко друг от друга, в один «объединенный» компьютер. Такая организация характеризуется возможностью эффективного выполнения сложных вычислений, повышением надежности ИС, ростом объемов хранимых данных. При этом сохраняется централизованный характер хранения и обработки данных, существует зависимость пользователей от места обработки данных.

Телекоммуникационные ИТА. Это наиболее распространенный вариант построения системы обработки данных для крупномасштабных ИС на базе компьютерных сетей (КС).

Основное назначение КС – поддержка взаимодействия пользователей сети за счет сетевых ресурсов – вычислительных и информационных ресурсов, создания сетевых сервисов (услуг), обеспечивающих рост производительности ИС и повышение надежности и качества работы ИС.

Серверные сети имеют различную архитектуру построения:

- файл-серверная;
- клиент-серверная;
- сервис-ориентированная.

В первом варианте единицей обмена данных между сервером и рабочей станцией является файл, в других – сообщение.

Файл-серверные сети при увеличении числа пользователей имеют большой сетевой трафик. Общие данные, хранимые на сервере и поступающие на рабочие станции для обработки, недоступны для одновременного использования в процессе редактирования. Это ограничивает пропускную способность и доступность ИС.

Клиент-серверные сети используют более сложное программное обеспечение, серверная и клиентская части программного кода различаются между собой, устранены основные недостатки файл-серверных сетей, когда единицей обмена между сервером и рабочей станцией является запрос и релевантная запросу выборка, а не целый файл. При редактировании данные доступны для коллективного доступа, уменьшена нагрузка на сетевой трафик.

Разновидности клиент-серверной архитектуры:

- **двухуровневый толстый клиент**; на рабочей станции находится программное обеспечение в виде пользовательского интерфейса, программ бизнес-приложений; обработка данных для функциональных задач выполняется на рабочей станции; сервер обеспечивает хранение файлов и БД, управление сетевыми ресурсами (доступ к файлам и БД, сетевые принтеры);

- **двухуровневый тонкий клиент**; на рабочей станции находится только программное обеспечение в виде пользовательского интерфейса; на сервере находятся общесетевые ресурсы (БД, бизнес-приложения, принтеры); обработка запросов к БД с использованием общесетевых бизнес-приложений выполняется на сервере;

- **трехуровневый клиент-сервер**; на рабочей станции находится только программное обеспечение в виде пользовательского интерфейса, сетевые ресурсы (бизнес-приложения, БД, принтеры) находятся на разных серверах. При этом воз-

можны трехзвенные конструкции: клиент – сервер приложений – сервер ресурсов. Сервер приложений использует специальное программное обеспечение (монитор обработки транзакций, программный интерфейс взаимодействия серверов приложений с серверами БД).

Сервис-ориентированная архитектура поддерживает различные Интранет/Интернет технологии:

- браузер – сервер приложений – сервер ресурсов;
- сервер динамических страниц – веб-сервер.

Взаимодействие подсистем ИС

Все обеспечивающие подсистемы связаны между собой и с функциональными подсистемами. Так, например, подсистема «Организационное обеспечение» определяет порядок разработки и внедрения ИС, ее организационную структуру и состав работников. В то же время правовые инструкции для работников ИС содержатся в подсистеме «Правовое обеспечение».

Функциональные подсистемы определяют состав и постановку задач, их математические модели и алгоритмы. Решения этих задач разрабатываются в подсистеме «Математическое обеспечение» и служат базой для создания прикладных программ, входящих в подсистему «Программное обеспечение».

Функциональные подсистемы, компоненты математического и программного обеспечения определяют принципы организации, состав классификаторов документов и информационной базы. Разработка структуры и состава информационной базы позволяет интегрировать все задачи функциональных подсистем в единую экономическую информационную систему, функционирующую по принципам, сформулированным в документах организационного и правого обеспечения.

Объем данных, степень сложности расчетов, разрабатываемых алгоритмов и программ влияют на выбор компонентов технического обеспечения. Выбранный комплекс технических

средств дает возможность определить тип операционной системы. Разработанное программное и информационное обеспечение влияют на технологию обработки информации при решении задач, входящих в соответствующие функциональные подсистемы.

1.4. Жизненный цикл информационных систем

Жизненный цикл – это период времени от момента возникновения идеи о создании информационной системы до момента окончания ее эксплуатации.

В литературе по информационным системам рассматривают различные этапы жизненного цикла информационных систем, которые несколько различаются между собой. Если обобщить имеющиеся варианты, то можно выделить следующие основные этапы жизненного цикла ИС: анализ требований, разработка и внедрение, эксплуатация.

1 этап. Анализ требований. Основная цель данного этапа – определение основных функций ИС и её технических характеристик. Основными выходными документами этой стадии являются: отчеты и технико-экономическое обоснование целесообразности создания ИС с выбранными функциями и их характеристиками; заявка на создание ИС и исходные технические требования к ИС.

2 этап. Разработка и внедрение. Цель этапа – создание ИС и передача ее в эксплуатацию.

Данный этап распадается на несколько стадий.

Стадия «Техническое задание». Основные цели стадии: подтверждение целесообразности создания ИС; детальное обследование предметной области; формирование исходных технических требований к системе; планирование всех работ и сроков их выполнения. Выходными документами данной стадии являются: техническое задание на создание ИС; уточненное технико-экономическое обоснование; эскизный проект ИС.

Стадия «Технический проект». Цель данной стадии – разработка основных технических решений по создаваемой системе и определение ее сметной стоимости. Работы этой стадии завершаются разработкой технического проекта ИС. Технический проект содержит: перечень общесистемных решений, проектно-сметную документацию, заявки на разработку новых технических средств; документация на специальное математическое и техническое обеспечение, включая техническое задание на программирование.

Стадия «Рабочая документация». Цель стадии – выпуск рабочей документации на создаваемую систему. Стадия завершается выпуском рабочего проекта ИС, состоящего из проектной документации, необходимой и достаточной для приобретения, монтажа и наладки комплекса технических средств системы; документации программного и организационного обеспечений, необходимых и достаточных для наладки и эксплуатации системы, а также изготовлением программ специального программного обеспечения на машинных носителях.

Стадия «Внедрение». Цель стадии и главный результат – передача созданной системы в промышленную эксплуатацию.

3 этап. Эксплуатация ИС. Цель этапа – сопровождение и развитие созданной ИС. Выполняется анализ функционирования, получение объективных и систематизированных данных о качестве созданной системы, текущем состоянии и реальном эффекте функционирования системы на основе опыта ее промышленной эксплуатации.

Методология создания ИС отражена также в нормативных документах международных стандартов. Существует международный стандарт, регламентирующий жизненный цикл информационных систем – ISO/IEC 12207 (ISO – International Organization of Standardization (международная организация по стандартизации), IEC – International Electrotechnical Commission (международная комиссия по электротехнике)).

Согласно данному стандарту структура жизненного цикла основывается на трех группах процессов:

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

1.5. Классические модели жизненного цикла ИС

Существуют три классические модели жизненного цикла ИС – каскадная, итерационная и спиральная модели.

Модели ЖЦ определяют последовательности прохождения этапов создания ИС, возможности их объединения, повтора и возврата к предыдущим этапам, выбор стратегии разработки, планирования и распределения работ в коллективе разработчиков.

Первой по времени появления и самой распространенной является **каскадная модель**.

Каскадная (или водопадная) модель подразумевает переход на следующий этап только после окончания работ на предыдущем этапе (рис. 5).

Каскадная модель имеет преимущества перед другими моделями, которые заключаются в том, что возможно четкое планирование и определение времени разработки ПС на всех этапах ее создания. Основными недостатками такой модели являются невозможность заказчика получить результаты на ранних сроках разработки, трудности в самом начале разработки в полной и точной формулировке требований. Здесь применяется стратегия **однократного прохода**, когда в самом начале из-

вестны все требования, отсутствуют циклы этапов, промежуточные варианты ПС не распространяются.

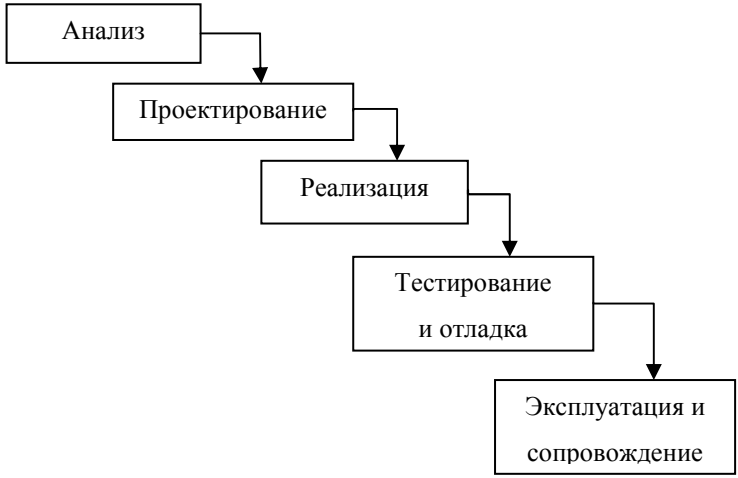


Рис.5. Каскадная модель разработки ИС

Таким образом, каскадная модель характеризуется следующими основными особенностями:

- последовательное выполнение входящих в ее состав этапов;
- окончание каждого предыдущего этапа до начала следующего;
- отсутствие временного перекрытия этапов (последующий этап не начнется, пока не завершится предыдущий);
- отсутствие (или определенное ограничение) возврата к предыдущим этапам;
- наличие результата только в конце разработки.

Выявление и устранение ошибок в каскадной модели производится только на стадии тестирования, которая может растянуться во времени или вообще никогда не завершится.

Следующей моделью ЖЦ ИС является **итерационная (поэтапная) модель** ЖК.

Данную модель называют еще итерационной моделью с циклами обратной связи (рис. 6).

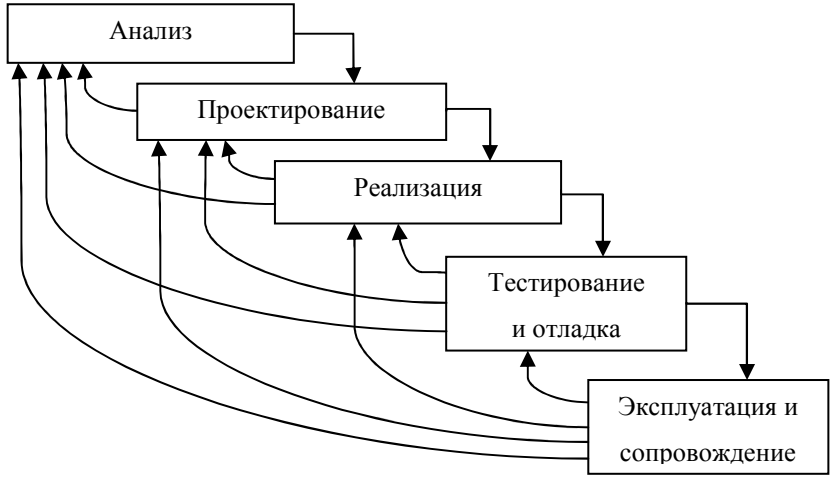


Рис. 6. Поэтапная модель разработки ПС

Модель имеет следующие особенности.

1. Наличие обратных связей между этапами. Это обеспечивает возможность проведения проверок и корректировок проектируемой ИС на каждой стадии разработки и возможность вернуться назад.

2. Более низкая трудоемкость отладки по сравнению с каскадной моделью. Если на каком-либо этапе в ходе промежуточной проверки обнаружена ошибка, допущенная на более ранней стадии разработки, то происходит возврат назад и повтор работ на предыдущих стадиях. При этом анализируются причины ошибки и корректируются в случае необходимости исходные данные этапа или его содержание (последовательность действий).

Если в процессе разработки системы меняются начальные требования, то итерационная модель становится неэффективной.

Спиральная модель является третьей моделью ЖЦ ИС (рис. 7). Данная модель поддерживает итерации поэтапной модели.

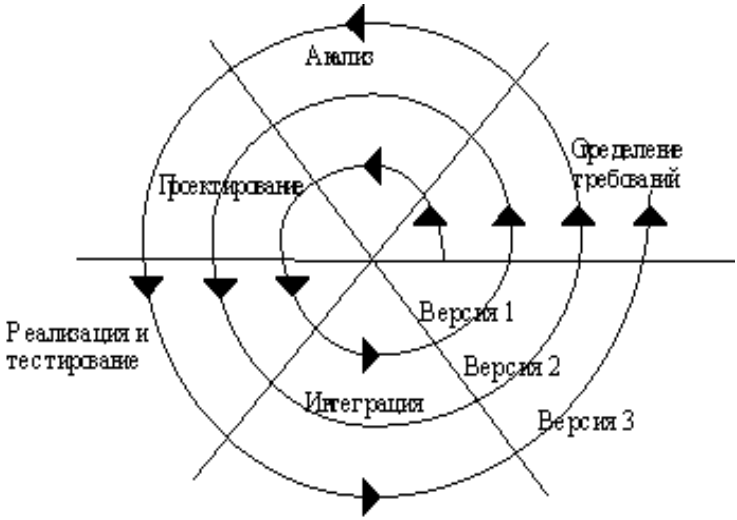


Рис. 7. Спиральная модель ЖЦ ИС

Особенности, присущие спиральной модели, следующие.

1. Особое внимание в этой модели уделяется начальным этапам проектирования: анализу требований, проектированию спецификаций, предварительному проектированию и детальному проектированию.

2. Данная модель является типичным примером *эволюционной стратегии*, когда каждый виток соответствует созданию новой версии или фрагмента ИС.

3. Требования к ИС не определены в начале разработки, а уточняются в ее процессе. На каждом витке планируются новые изменения и анализируется связанный с ними риск для

создания нового витка. Планирование изменений строится на основе оценок прототипа заказчиком и внесенных заказчиком предложений. Создание прототипа может быть выполнено по каскадной модели ЖЦ или заменено **макетом ИС**. Анализ риска дает ответ на вопрос о целесообразности продолжении проекта: проект может быть остановлен или продвинут к более универсальной модели системы. На первых 2-х этапах создаются прототипы, проверяются и обосновываются технические решения.

Преимущества спиральной модели состоят в следующем:

- накопление и повторное использование моделей, прототипов, программных средств;
- ориентация на модификацию ИС при проектировании;
- анализ риска и затрат при проектировании;
- наиболее адекватное отражение реального процесса разработки ИС.

Но и спиральная модель не лишена недостатков, заключающихся в следующем:

- трудности планирования работ и определения сроков окончательной разработки;
- необходимость постоянного присутствия при разработке квалифицированного заказчика.

1.6. Методология и технология разработки информационных систем

Методология создания информационных систем заключается в организации процесса построения информационной системы и в управлении этим процессом для того, чтобы гарантировать выполнение требований, как к самой системе, так и к характеристикам процесса разработки.

Методология создания информационных систем должна обеспечить решение следующих основных задач:

- соответствие создаваемой информационной системы целям и задачам предприятия и требованиям по автоматизации бизнес-процессов;

- гарантирование создания системы с заданными параметрами в течение заданного времени и в рамках оговоренного бюджета;

- простота сопровождения, модификации и расширения системы с целью обеспечения ее соответствия изменяющимся условиям работы предприятия;

- соответствие информационной системы требованиям открытости, переносимости и масштабируемости;

- возможность использования в создаваемой системе разработанных ранее и применяемых на предприятии средств информационных технологий (программного обеспечения, баз данных, средств вычислительной техники, телекоммуникаций).

Методологии, технологии и инструментальные средства проектирования составляют основу проекта любой информационной системы.

Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства.

Технология проектирования включает технологические инструкции, которые описывают последовательность технологических операций по проектированию ИС, условия выполнения операций и сами операции.

Таким образом, технология проектирования может быть представлена как совокупность трех составляющих:

- заданной последовательности выполнения технологических операций проектирования;

- критериев и правил, используемых для оценки результатов выполнения технологических операций;

- графических и текстовых средств (нотаций), используемых для описания проектируемой системы.

Результаты выполнения операции должны представляться в некотором стандартном виде, обеспечивающем их адекватное восприятие при выполнении следующей технологической операции.

Существует ряд общих требований, которым должна удовлетворять технология проектирования, разработки и сопровождения информационных систем:

- поддержка полного жизненного цикла информационных систем;
- гарантированное достижение целей разработки системы с заданным качеством и в установленное время;
- возможность разделения крупных проектов на ряд подсистем, разрабатываемых отдельными группами исполнителей;
- обеспечение минимального времени получения работоспособной системы;
- возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, выпуска проектной документации;
- независимость выполняемых проектных решений от средств реализации системы – систем управления базами данных, операционной системы, системы программирования.

Существует множество методологий и технологий разработки информационных систем. Их можно разделить на структурные и объектно-ориентированные методологии.

Структурными называют методы проектирования, которые первоначально описывают объект исследования как общую систему, а затем реализуют ее последовательную декомпозицию (детализацию).

Объектно-ориентированный подход к построению информационных систем отличается от структурного подхода большим уровнем абстракции и основывается на представлении системы в виде совокупности объектов, взаимодействующих между собой путем передачи определенных сообщений. В качестве объектов предметной области могут служить кон-

кретные предметы или абстрагированные сущности. Объектно-ориентированный подход не противопоставляется структурному подходу, а может служить его дополнением.

1.7. CASE-средства проектирования ИС

Для автоматизации проектирования и разработки информационных систем были созданы программно-технологические средства, получившие название CASE-средств и реализующие CASE-технологии создания и сопровождения информационных систем.

Термин CASE (Computer Aided Software Engineering) дословно переводится как разработка программного обеспечения с помощью компьютера. В настоящее время этот термин получил более широкий смысл, означающий автоматизацию разработки информационных систем.

CASE-средства представляют собой программные средства, поддерживающие процессы создания и/или сопровождения информационных систем. Основные операции, выполняемые данными средствами, - это анализ и формулировка требований, проектирование баз данных и приложений, генерация кода, тестирование, обеспечение качества, управление конфигурацией и проектом.

CASE-систему можно определить как набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого программного продукта.

CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения для пользователей.

CASE-индустрия объединяет сотни фирм и компаний различной ориентации. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов превышает 500 наименований.

Основная цель CASE-систем и средств состоит в том, чтобы отделить проектирование программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и пр.), а также автоматизировать весь процесс создания программных систем, или инжиниринг (от англ. engineering – разработка).

В последнее время все чаще разработка программ начинается с некоторого предварительного варианта системы. В качестве такого варианта может выступать специально для этого разработанный прототип либо устаревшая и не удовлетворяющая новым требованиям система. В последнем случае для восстановления знаний о программной системе с целью последующего их использования применяют повторную разработку – реинжиниринг.

Повторная разработка сводится к построению исходной модели программной системы путем исследования ее программных кодов. Имея модель, можно ее усовершенствовать, а затем вновь перейти к разработке. Так часто и поступают при проектировании. Одним из наиболее известных принципов такого типа является принцип «возвратного проектирования» – Round Trip Engineering (RTE).

Современные CASE-системы не ограничиваются только разработкой, а чаще всего обеспечивают и повторную разработку. Это существенно ускоряет создание приложений и повышает их качество.

При **классификации CASE-средств** используют следующие признаки:

- ориентацию на этапы жизненного цикла;
- функциональную полноту;

- тип используемой модели;
- степень независимости от СУБД;
- допустимые платформы.

Рассмотрим классификацию CASE-средств по наиболее часто используемым признакам.

*По **ориентации на этапы жизненного цикла*** выделяют следующие основные типы CASE-средств:

- средства анализа, предназначенные для построения и анализа моделей предметной области, например: Design/IDEF (Meta Software), BPwin (Logic Works);
- средства анализа и проектирования, обеспечивающие создание проектных спецификаций, например: Vantage Team Builder (Cayenne), Silverrun (Silverrun Technologies), PRO-IV (MeDonnell Douglas), CASE Аналитик (МакроПроджект);
- средства проектирования баз данных, обеспечивающие моделирование данных и разработку схем баз данных для основных СУБД, например: ERwin (Logic Works), S-Designor (SPD), DataBase Designer (ORACLE);
- средства разработки приложений, например: Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer (ORACLE), New Era (Informix), SQL Windows (Centura), Delphi (Borland), Visual Studio (Microsoft).

По ***функциональной полноте*** CASE-системы и средства можно условно разделить на следующие типы:

- системы, предназначенные для решения частных задач на одном или нескольких этапах жизненного цикла, например, ERwin, S-Designor, CASE.Аналитик, Silerrun;
- интегрированные системы, поддерживающие весь жизненный цикл ИС и связанные с общим репозиторием, например, система Vantage Team Builder, система Designer с системой разработки приложений Developer.

По **типу используемых моделей** CASE-системы условно можно разделить на три основные разновидности: структурные, объектно-ориентированные и комбинированные.

Исторически первые *структурные* CASE-системы основаны на методах структурного и модульного программирования, структурного анализа и синтеза, например, система Vantage Team Builder.

Объектно-ориентированные методы и CASE-системы получили массовое использование с начала 90-х годов. Они позволяют сократить сроки разработки, а также повысить надежность и эффективность функционирования ИС. Примерами объектно-ориентированных CASE-систем являются Rational Rose и Object Team.

Комбинированные инструментальные средства поддерживают одновременно структурные и объектно-ориентированные методы, например: Designer.

По **степени независимости от СУБД** CASE-системы можно разделить на две группы: независимые системы, встроенные в СУБД.

Независимые CASE-системы поставляются в виде автономных систем, не входящих в состав конкретной СУБД. Обычно они поддерживают несколько форматов баз данных через интерфейс ODBC. К числу независимых CASE-систем относятся S-Designor, ERwin, Silverrun.

Встроенные CASE-системы обычно поддерживают главным образом формат баз данных СУБД, в состав которой они входят. Примером встроенной системы является Designer, входящая в состав СУБД ORACLE.

Компания Computer Associates разработала ряд программных продуктов, поддерживающих CASE-технологии разработки ИС и их компонентов [10].

Интегрированный пакет инструментальных средств Computer Associates включает следующие программные продукты:

- AllFusion Process Modeler (ранее BPwin) - моделирование бизнес-процессов;

- AllFusion ERwin Data Modeler (ERwin) –инструмент создания моделей данных и генерации схем баз данных;

- AllFusion Model Manager – система организации коллективной работы, хранилище моделей BPwin и ERwin;

- AllFusion Component Modeler – инструмент создания объектных моделей.

2. СТРУКТУРНЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ ИНФОРМАЦИОННЫХ СИСТЕМ

2.1. Общая характеристика структурного подхода

Структурный подход (структурный анализ) к проектированию ИС означает, что сначала объект исследования рассматривается как единый целый, а затем осуществляется его декомпозиция на составляющие элементы.

Таким образом, **сущность структурного подхода** к разработке ИС заключается в декомпозиции (разбиении) системы на автоматизируемые функции. Система разбивается на функциональные подсистемы, которые, в свою очередь, делятся на подфункции. Процесс разбиения продолжается вплоть до конкретных процедур обработки данных. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны.

Методологии структурного подхода базируются на следующих базовых принципах:

- принцип «разделяй и властвуй», означающий разделение общей задачи на множество меньших задач, выполняющих определенную функцию и легких для понимания и решения;

- принцип иерархического упорядочения, означающий организацию составных частей задачи в иерархические древовидные структуры с определением взаимосвязей между ними;

- использование графического представления взаимосвязей элементов системы.

Кроме базовых принципов можно выделить другие принципы, не менее важные.

Абстрагирование – выделение существенных свойств системы и отвлеченность от несущественных аспектов системы для представления данного уровня проблемы в общем виде.

Формализация – строгий методический подход к решению проблемы.

Упрятывание – каждый элемент системы на конкретном этапе и уровне «знает только необходимую ему информацию».

Концептуальная общность – единая философия на всем ЖЦ (структурный анализ → структурное проектирование → структурное программирование → структурное тестирование).

Полнота – контроль на присутствие лишних компонентов.

Непротиворечивость – обоснованность и согласованность элементов.

Логическая независимость – концентрация на логическом, а не физическом проектировании.

Доступ конечного пользователя – пользователь должен иметь доступ к данным, которые он может использовать без программирования.

Модель ИС, построенная с применением структурных методов, представляет собой иерархический набор диаграмм. На диаграммах графически изображаются выполняемые системой функции и взаимосвязи между ними. В диаграммы также включается текстовая информация для более точного описания выполняемых функций и имеющихся взаимосвязей между элементами системы. Использование графического представления системы и ее элементов существенно повышает наглядность модели ИС и облегчает ее восприятие.

В структурном анализе используются следующие основные методологии:

- SADT (Structured Analysis and Design Technique) - технология структурного анализа и проектирования, ее подмножество – стандарт IDEF; предназначена для разработки функциональных диаграмм;

- DFD (Data Flow Diagrams) – диаграммы потоков данных; описывают взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;

- ERD(Entity Relationship Diagrams) – диаграммы сущность-связь, получившие развитие в методологии IDEFI; описывают структуру базы данных разрабатываемой системы;

- STD (State Transition Diagrams) – диаграммы переходов состояний; характеризуют поведение системы во времени.

2.2. Методология SADT

Методология SADT разработана Дугласом Россом и представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели информационной системы из какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, осуществляемые им действия и связи между этими действиями.

На основе методологии SADT разработана, в частности, известная методология IDEF0.

Функциональная модель SADT – это набор функциональных диаграмм. Функциональными называют диаграммы, отражающие функции и взаимосвязи функций разрабатываемой информационной системы.

Данные диаграммы создаются на ранних этапах проектирования систем, они помогают проектировщику выявить основные функции и, значит, определить составные части проектируемой ПС. Современные методы структурного анализа и проектирования предоставляют разработчику графические

средства проектирования функциональных диаграмм информационных систем.

Методология SADT может использоваться для моделирования и разработки широкого круга систем, удовлетворяющих определенным требованиям и реализующих требуемые функции.

В уже разработанных системах методология SADT может быть использована для анализа выполняемых ими функций, а также для указания механизмов, посредством которых они осуществляются.

На SADT-диаграмме функции представляют в виде блока (прямоугольника), а интерфейсы входа-выхода – в виде дуг (стрелок), соответственно входящих в блок и выходящих из него. Интерфейсные дуги отображают взаимодействие функций друг с другом.

Место соединения дуги с блоком определяет тип интерфейса.

Дуга, обозначающая управление, входит в блок сверху.

Информация, которая подвергается обработке, представляется дугой с левой стороны блока, а результаты обработки – дугами с правой стороны.

Механизм (человек или автоматизированная система), который осуществляет операцию, представляется в виде дуги, входящей в блок снизу (рис. 8).

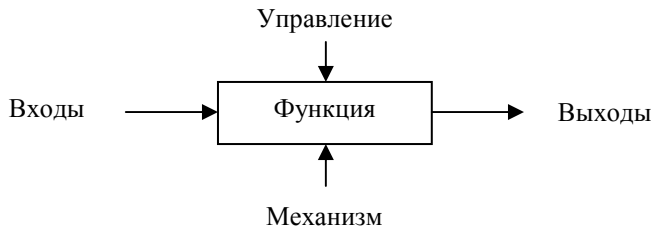


Рис. 8. Функциональный блок и интерфейсные дуги

Общая постановка задачи проектирования ИС в методологии SADT представлена на рис. 9.

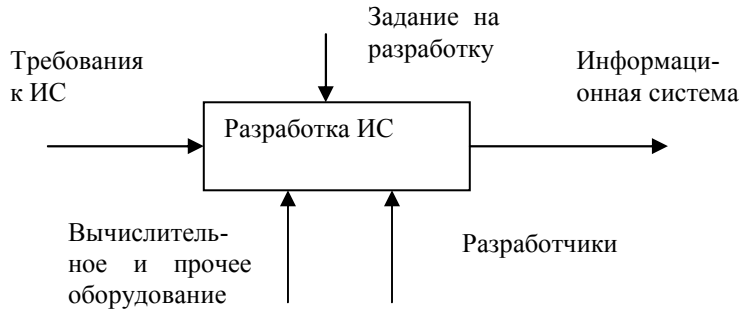


Рис. 9. Функциональный блок общей задачи на разработку ИС

В функциональных диаграммах SADT различают пять типов влияния блоков друг на друга:

вход, выход блока подается на вход следующего блока (на блок с меньшим доминированием) (рис. 10, а);

управление, выход блока используется как управления для следующего блока (блока с меньшим доминированием) (рис. 10, б);

обратная связь по входу, выход блока подается на вход блока с большим доминированием (рис. 10, в);

обратная связь по управлению, выход блока используется как управляющая информация для блока с большим доминированием (рис. 10, г);

выход-исполнитель, выход блока используется как механизм для другого блока (рис. 10, д).

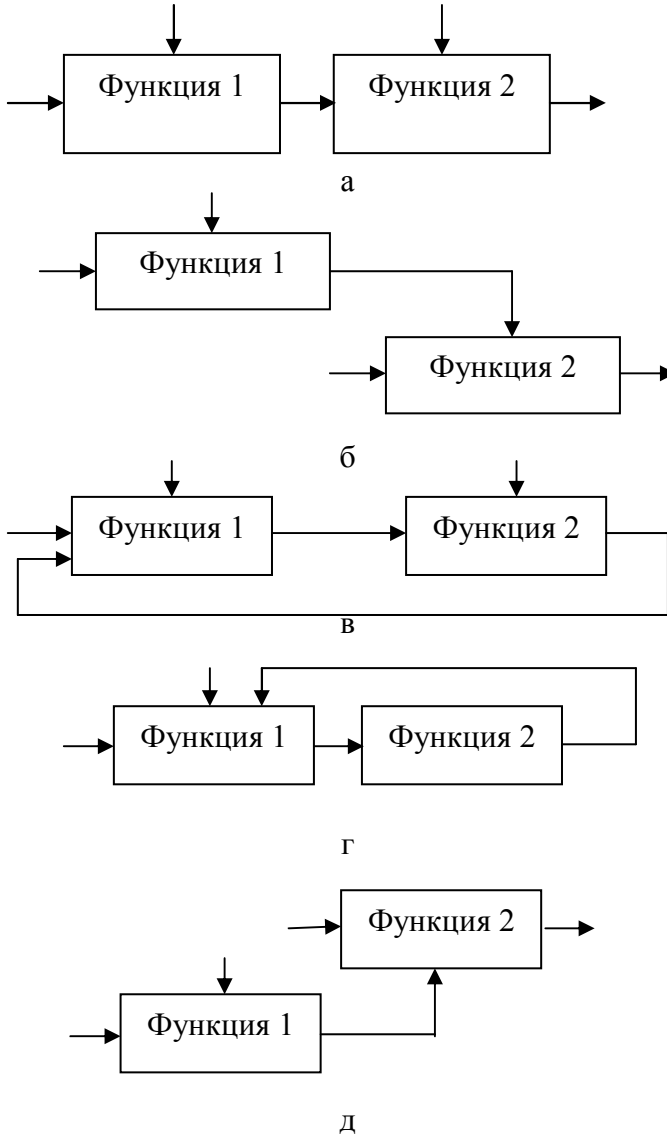


Рис. 10. Типы влияний блоков

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

На рис. 11 приведены диаграммы и их взаимосвязи, показывающие структуру SADT-модели. Каждый компонент может быть декомпозирован на другой диаграмме. Детальная диаграмма иллюстрирует внутреннее строение блока «родительской» диаграммы.

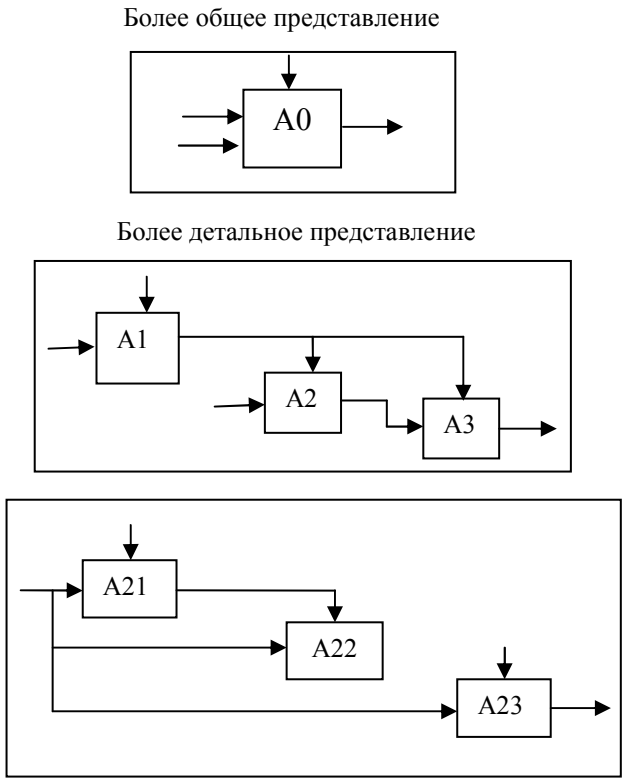


Рис. 11. Структура SADT-модели. Декомпозиция диаграмм

Иерархия диаграмм. Первоначально вся ИС представляется в виде простейшей компоненты – одного блока и дуг, представляющих собой интерфейсы с внешними по отношению к данной системе функциями. Имя блока является общим для всей системы. Затем блок, который представляет систему в целом, детализируется на следующей диаграмме. Он представляется в виде нескольких блоков, соединенных интерфейсными дугами. Каждый блок детальной диаграммы представляет собой подфункцию, границы которой определены интерфейсными дугами. Каждый из блоков детальной диаграммы может быть также детализирован на следующей в иерархии диаграмме. Все диаграммы связывают друг с другом иерархической нумерацией блоков: первый уровень – А0, второй – А1, А2, А3 и т.д., третий – А11, А12, А13 и т.д., где первые цифры – номер родительского блока, а последняя – номер конкретного блока детальной диаграммы.

Пример 1. Разработать функциональные диаграммы для ИС «Ремонт и модернизация вычислительной техники». Диаграмма верхнего уровня представлена на рис. 12.

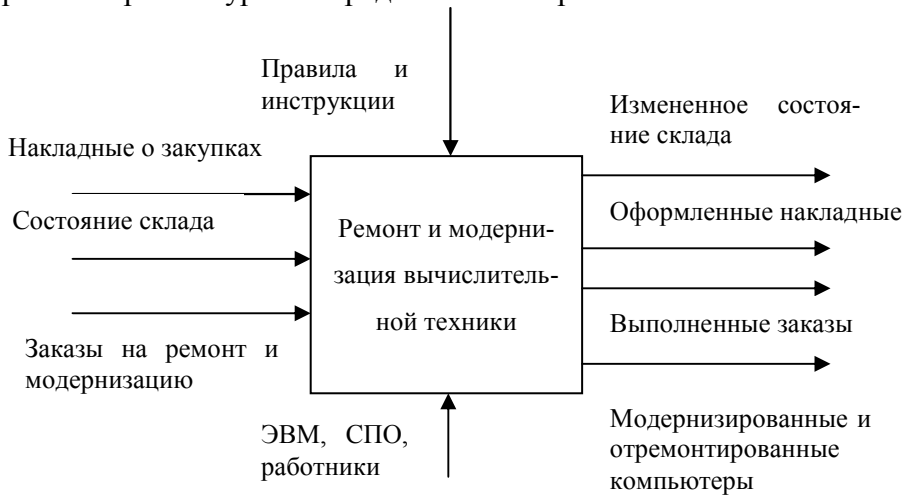


Рис. 12. Диаграмма верхнего уровня

Диаграмма, представленная на рис. 13, является декомпозицией диаграммы верхнего уровня.

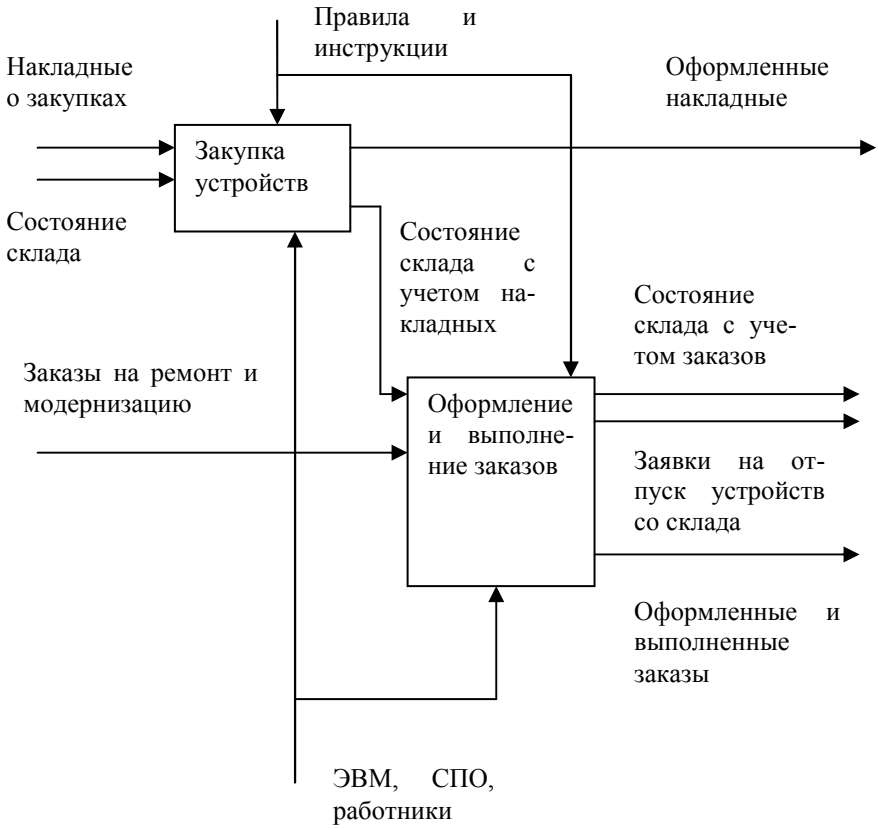


Рис. 13. Декомпозиция диаграммы верхнего уровня

Методология IDEF0 построена на основе методологии SADT.

Методология IDEF0 использует четыре основных понятия: функциональный блок, интерфейсная дуга, декомпозиция и глоссарий.

Функциональный блок обозначает определенную функцию в рамках рассматриваемой системы и в графическом виде обозначается прямоугольником. Каждая из четырех сторон прямоугольника имеет свое значение: левая сторона – вход, верхняя сторона – управление, нижняя сторона – механизм и правая сторона – выход.

Интерфейсная дуга обозначает элемент системы, который обрабатывается функциональным блоком, является результатом обработки или оказывает некоторое влияние на выполнение блоком своей функции. Графически интерфейсная дуга изображается в виде однонаправленной стрелки. В зависимости от того, к какой из сторон блока примыкает интерфейсная дуга, она носит название входящей, исходящей, управляющей или дуги механизма. Входящими и исходящими дугами могут обозначаться финансовые потоки, материальные потоки (сырье, инструменты, изделия и т.д.), потоки информации (документы, распоряжения и т.д.) и ресурсы (персонал, оборудование и др.). Управляющими дугами обозначаются только объекты, относящиеся к потокам информации, а дугами механизмов – только ресурсы.

Декомпозиция предполагает разбиение сложного процесса (системы) на составные части. В итоге общая модель процесса формируется в виде иерархической структуры диаграмм. Модель в нотации IDEF0 всегда начинается с представления процесса как единого функционального блока с интерфейсными дугами, исходящими за пределы рассматриваемой области. Полученная диаграмма называется контекстной. В процессе декомпозиции функциональные блоки диаграммы верхнего уровня детализируются на диаграмме следующего уровня.

Глоссарием является набор определений и ключевых слов, характеризующий объекты, отображенные на диаграмме. Глоссарий обеспечивает включение в диаграммы IDEF0 необходимой дополнительной информации.

2.3. Методология DFD

В соответствии с методологией DFD (моделирование потоков данных (процессов)) модель системы определяется как иерархия диаграмм потоков данных. Диаграммы описывают процесс преобразования информации от ее ввода в систему до выдачи пользователю.

Таким образом, диаграммы DFD (Data Flow Diagrams) описывают взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе. В данной методологии исследуемый процесс разбивается на подпроцессы и представляется в виде сети, связанной потоками данных.

Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процессы становятся элементарными и детализировать их далее невозможно.

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

Таким образом, основными компонентами диаграмм потоков данных являются:

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

Внешняя сущность – это материальный предмет или физическое лицо, представляющее собой источник или приемник информации. Например, заказчики, персонал, поставщики, клиенты, склад. Определение некоторого объекта в качестве внешней сущности указывает на то, что он находится за пределами ИС. Имя внешней сущности – имя существительное («Склад товаров», «Поставщик»). Внешняя сущность не участвует в обработке данных. Внешняя сущность может обозначаться прямоугольником с тенью.

Системы и подсистемы (процессы). На диаграммах изображаются прямоугольниками с закругленными углами. Уникальный номер подсистемы служит для ее идентификации. В имени подсистемы вводится наименование в виде предложения.

Накопитель данных (хранилище) представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь. Накопитель данных может обозначаться прямоугольником с вертикальной линией, в котором записана буква «D» и номер.

Поток данных определяет информацию, передаваемую от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по каналу связи, взятой из базы данных, переносимой с компьютера на компьютер на носителе информации. Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока. Каждый поток данных имеет имя, отражающее его содержание.

Формально построение модели можно представить следующими шагами.

1. Общие требования к системе разделяются и организуются в функциональные группы (в дальнейшем это будут процессы).

2. Идентифицируются внешние объекты, с которыми общается система (таким образом, определяются внешние сущности).

3. Идентифицируется информация между системой и внешними объектами (т.е. задаются потоки данных).

4. Разрабатывается предварительная контекстная диаграмма на основе процессов (шаг 1), внешних сущностей (шаг 2) и потоков данных (шаг 3).

5. Изучается и корректируется предварительная контекстная диаграмма на предмет соответствия общим требованиям на разработку системы.

6. Строится контекстная диаграмма с объединением всех процессов предварительной контекстной диаграммы в один (возможно, абстрактный) процесс, с группировкой потоков данных.

7. Из контекстной диаграммы через детализацию процесса строится DFD первого уровня. Одновременно производится необходимая для данного уровня декомпозиция потоков данных.

8. Проверяются основные требования к системе по DFD первого уровня. При необходимости диаграмма корректируется.

9. Проводится декомпозиция построенной DFD с помощью детализирующих DFD следующего уровня или спецификаций процесса.

10. Проверяются требования к системе по DFD соответствующего уровня. Если необходимо, вносятся исправления.

11. Пополняется словарь данных для определения характеристик потоков данных при каждом первом их появлении.

12. Параллельно с декомпозицией диаграмм потоков данных требования разбиваются на элементарные, проверяются соответствующие им процессы или спецификации.

13. Каждый раз проводится ревизия после построения двух-трех уровней диаграмм с целью улучшения и проверки корректности модели.

14. Рассматривается поведение модели системы в реальном времени.

Пример 1. Разработать контекстную диаграмму потоков данных для ИС обслуживания работы банкомата.

Общая схема работы следующая. Клиент для получения денег в банкомате вводит кредитную карточку со своими данными, пароль и необходимую сумму. При правильном вводе данных клиент получает деньги, кредитную карточку и выписку по проведенной операции. Компьютер банка реализует контроль за снятием денег со счета.

При построении диаграмм потоков данных учитывают следующее.

1. Система должна обеспечить сервис клиенту по снятию денег с его банковского счета. Общее назначение системы – обслуживание, соответственно и название самого общего процесса – «Обслужить».

2. Внешними по отношению к системе являются «Клиент» (только поставляет данные) и «Компьютер банка» (также не участвует в обработке данных внутри системы, а лишь дает необходимые данные для проверки и записывает информацию по проведенной операции).

3. Определяются потоки данных системы:

«Сообщение» - информация о готовности принятия данных от «Клиента»;

«Кредитная карта» получает банкомат от клиента и возвращает клиенту;

«Деньги», «Выписка» получает «Клиент»;

«Ключевые данные» вводятся «Клиентом»;

«Данные по счету» - информация о расчетном счете «Клиента»;

«Протокол обслуживания» - информация для изменения суммы после снятия денег со счета «Клиента».

4. Строится контекстная диаграмма (рис. 14), на которой представлен один процесс - «Обслужить».

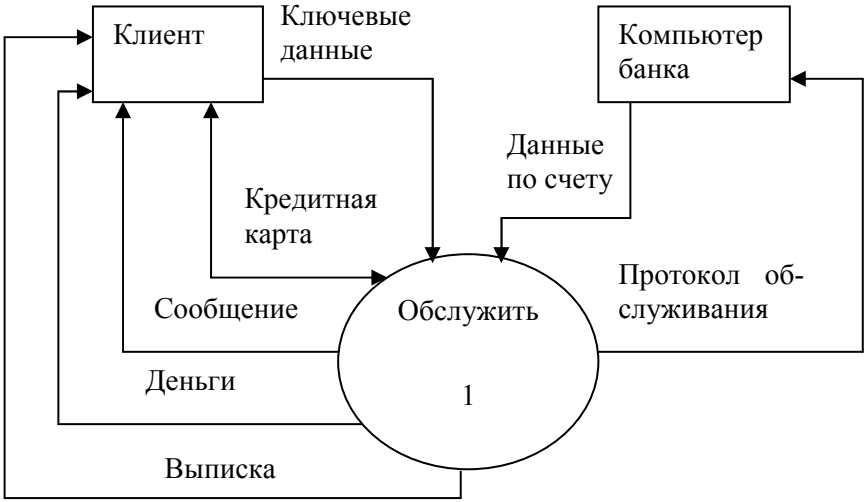


Рис. 14. Контекстная диаграмма

2.4. Методология ERD

Методология ERD (Entity Relationship Diagrams) используется для построения моделей данных и обеспечивает стандартизованный способ описания данных и определения связей между ними.

Основными элементами диаграммы являются сущности (объекты предметной области), их свойства (атрибуты) и отношения (связи) сущностей.

Методология ERD используется для проектирования реляционных баз данных.

Первый шаг моделирования – это выделение сущностей. Сущность (entity) – реальный либо воображаемый объект,

имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению. Графическое изображение сущности представлено на рис. 15.

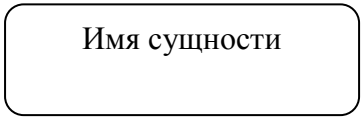


Рис. 15. Графическое изображение сущности

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данной сущности. Каждая сущность должна обладать следующим набором свойств:

- иметь уникальное имя;
- иметь один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются;
- иметь один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр сущности;
- может иметь любое количество связей с другими сущностями модели.

Например, для фирмы, занимающейся продажей автомобилей, можно выделить следующие сущности: автомобиль, сотрудник, покупатель, заказ.

Следующим шагом моделирования является определение связей. Связь (Relationship) – поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Типичная связь между сущностями следующая: каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком. Обычно обязательная связь обозначается сплошной линией, необязательная связь – пунк-

тирной линией. Обозначения степени связи и обязательности приведены на рис. 16.

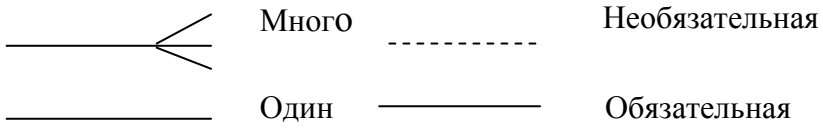


Рис. 16. Обозначение степени связи и обязательности

Связи может даваться имя, помещаемое возле линии связи. Имя каждой связи между двумя сущностями должно быть уникальным. Однако имена связей в модели не должны быть уникальными. Имя связи всегда формируется с точки зрения родителя. Пример связи между сущностями «Сотрудник» и «Заказ» представлен на рис. 17.

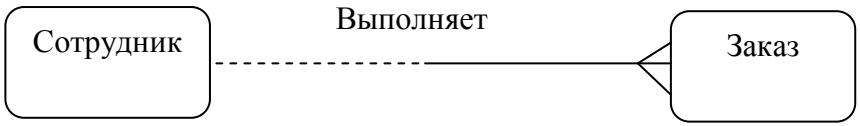


Рис. 17. Схема связи сотрудника с заказом

Последним шагом моделирования является идентификация атрибутов. Атрибут – это любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для обозначения какой-то характеристики сущности или ее состояния. Таким образом, атрибут представляет собой характеристики или свойства, ассоциированные с множеством реальных или абстрактных объектов. Для сущности «Сотрудник» могут быть выделены следующие атрибуты: номер сотрудника, фамилия, имя, отчество, должность, дата поступления на работу. Для сущности «Заказ» могут быть вы-

делены следующие атрибуты: номер заказа, дата заказа, номер сотрудника, выполнен заказ или нет.

Атрибут может быть обязательным или необязательным. Обязательный атрибут перед именем имеет символ «*», необязательный атрибут помечен символом «о».

Атрибут может быть ключевым или описательным. Ключевой атрибут используют для обозначения идентификатора (первичного ключа). С помощью первичного ключа отличают (идентифицируют) один экземпляр сущности от другого. В первичный ключ могут входить один или несколько атрибутов.

Каждый атрибут имеет уникальное имя, которое является существительным. Атрибуты изображаются в виде списка внутри блока сущности после имени сущности. Атрибуты, входящие в состав первичного ключа, выделяют знаком «#». Пример сущности с атрибутами представлен на рис. 18.

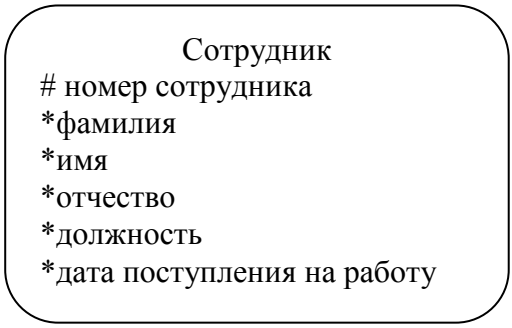


Рис. 18. Сущность с атрибутами

Методология IDEFI построена на основе методологии ERD и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме.

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой «/». Они располагаются над блоком.

Связь может дополнительно определяться с помощью указания степени и мощности.

Возможны следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;

- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;

- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;

- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае – неидентифицирующей.

Связь изображается линией, проводимой между сущностью-родителем и сущностью потомком с точкой на конце линии у сущности-потомка.

Идентифицирующая связь изображается сплошной линией. Пунктирная линия изображает неидентифицирующую связь.

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой.

Пример взаимодействия двух сущностей «Группы», «Студенты» представлена на рис. 19.

Сущность «Группы» является сущностью-родителем, имеет ключевой атрибут «Номер группы».

Сущность «Студенты» является сущностью-потомком, имеет ключевой атрибут «Номер студента».

Сущности связаны между собой по атрибуту «Номер группы».

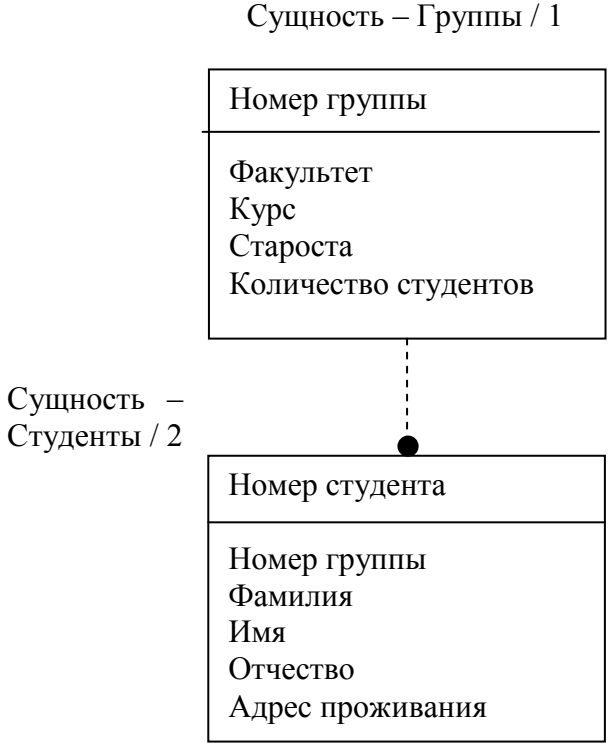


Рис. 19. Сущность-родитель и сущность потомок

2.5. Методология STD

Методология STD (State Transition Diagrams) описывает поведение разрабатываемой информационной системы, зависящее от времени и при получении управляющих воздействий (извне).

В диаграммах такого вида узлы соответствуют состояниям динамической системы, а дуги – переходу системы из одного состояния в другое.

Узел, из которого выходит дуга, является начальным состоянием.

Узел, в который входит дуга, является следующим состоянием.

Дуга помечается именем входного сигнала или события, вызывающего или сопровождающего переход.

Условные обозначения, используемые при построении диаграмм переходов состояний, показаны на рис. 20.

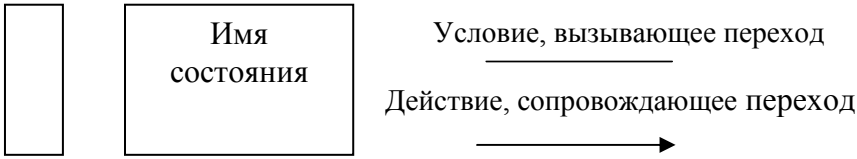


Рис. 20. Условные обозначения диаграмм переходов состояний:
а – терминальное состояние; б – промежуточное состояние;
в – переход

На рис. 21 представлена диаграмма переходов торгового автомата, активно взаимодействующего с покупателем.

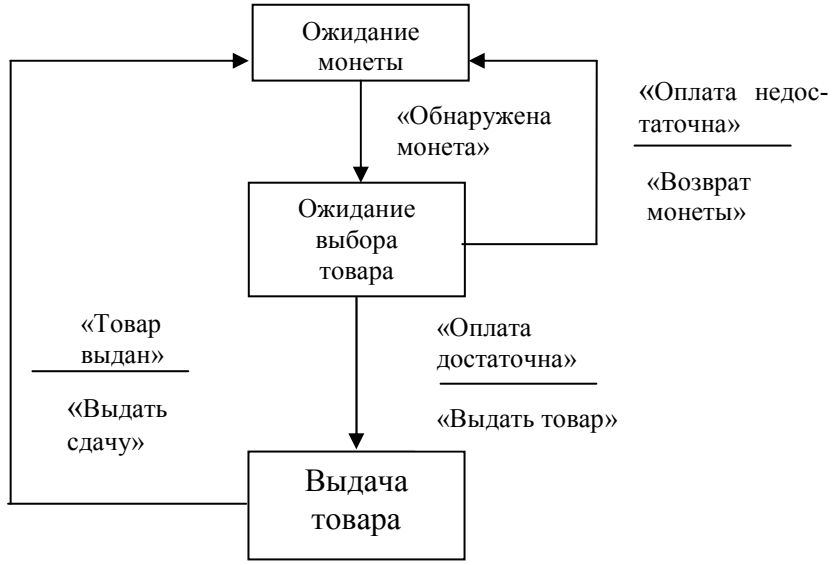


Рис. 21. Диаграмма переходов состояний торгового автомата

3. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ ИС С ПРИМЕНЕНИЕМ ЯЗЫКА UML

3.1. Объектно-ориентированное проектирование

Объектно-ориентированное проектирование – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логических, физических, а также статических и динамических моделей проектируемой системы [1].

Объектно-ориентированное программирование - методология программирования, основанная на представлении программ в виде связанной совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию по наследованию [1].

Объектно-ориентированное проектирование предполагает деление (декомпозицию) базы данных на составные части, а также определение алгоритма работы (обработки данных) и формирование интерфейса пользователя с разработкой функций, которые будет выполнять интерфейс и его отдельные компоненты.

При объектно-ориентированном подходе данные называют свойствами, а алгоритмы методами.

Доступ к классу в статическом режиме осуществляется через свойства. В процессе выполнения (работы) программы доступ к экземплярам класса осуществляется через методы.

Начало работы экземпляров класса задается с помощью специальных внутренних (например, нажатие кнопки) или внешних (вызов из другой подпрограммы) сигналов, называемых событиями.

Программную реализацию класса называют компонентой. Реализация компонента в некоторой прикладной программе получила название объекта.

3.2. Унифицированный язык моделирования

В основе объектного подхода к разработке программного и информационного обеспечения лежит объектная декомпозиция, т.е. представление разрабатываемой системы в виде совокупности объектов, в процессе взаимодействия которых через передачу сообщений происходит выполнение требуемых функций.

Для создания моделей анализа и проектирования объектно-ориентированных программных и информационных систем используют языки визуального моделирования, самым популярным из которых на сегодняшний день является UML (Unified Modeling Language) – унифицированный язык моделирования.

UML является совместной разработкой известных специалистов Г.Буч, Д.Рамбо, И. Джекобсон, реализованной при поддержке фирмы Rational Software [2].

UML представляет собой единый язык моделирования, предназначенный для спецификации, визуализации, конструирования и документирования описания программных систем, а также для моделирования бизнес-процессов и других непрограммных систем. В основу создания UML положены три наиболее распространенные модели:

Booch, получившая название по фамилии автора Гради Буча (Grady Booch);

OMT (Object Modeling Technique – метод моделирования объектов);

OOSE (Object-Oriented Software Engineering – объектно-ориентированное проектирование программного обеспечения).

UML можно определить так же, как промышленный объектно-ориентированный стандарт моделирования. Он включает в себя в унифицированном виде лучшие методы визуального (графического) моделирования. В настоящее время имеется целый ряд инструментальных средств, производители которых заявляют о поддержке UML, среди них можно выделить: Rational Rose, All Fusion Modeling Suite, Select Enterprise, Platinum, Visual Modeler.

Спецификация разрабатываемого информационного и программного обеспечения при использовании UML объединяет несколько моделей: логическую, использования, реализации, процессов, развертывания.

Модель использования содержит описание функций программного обеспечения с точки зрения пользователя.

Логическая модель описывает ключевые понятия моделируемого информационного и программного обеспечения (классы, интерфейсы и т.д.), т.е. средства, обеспечивающие выполнение функций разрабатываемого средства.

Модель реализации определяет реальную организацию программных модулей в среде разработки.

Модель процессов отображает организацию вычислений и позволяет оценить производительность, масштабируемость и надежность программного обеспечения.

Модель развертывания показывает, каким образом программные компоненты размещаются на конкретном оборудовании.

Все вместе указанные модели, каждая из которых характеризует определенную сторону проектируемого продукта, составляют относительно полную модель разрабатываемого программного обеспечения.

Типы диаграмм UML. Создаваемый с помощью UML проект информационной или программной системы может включать в себя несколько видов диаграмм, которые дополняют друг друга и могут входить в различные модели. Это диаграммы вариантов (прецедентов) использования; классов; последовательностей действий; деятельностей; состояний; кооперации; компонентов; размещения.

Диаграммы вариантов (прецедентов) использования (use case diagram) позволяют определить функции, выполняемые ИС или ПС и видимые пользователями.

Диаграммы классов (class diagram) описывают концептуальную логическую модель проектируемой ИС или ПС и отражают отдельные сущности предметной области и взаимосвязи между ними. Диаграммы классов представляют статическую структурную модель проектируемой системы.

Для описания особенностей поведения ИС или ПС применяют **диаграммы последовательностей действий (sequence diagram), деятельностей (interaction diagram) и состояний (statechart diagram)**.

Диаграммы активности (activity diagram) позволяют показать движения потоков данных в проектируемой системе.

Диаграммы компонентов (component diagram) и размещения (deployment diagram) описывают физическую реализацию программной системы.

3.3. Определение прецедентов (вариантов использования)

Диаграмма прецедентов служит для выявления и формального представления требований заказчика к проектируемой системе [10]. Диаграмма описывает, какие возможности будет предоставлять система конечному пользователю, какую будет предоставлять информацию, на какие запросы отвечать, какие будет генерировать отчеты. При этом механизм функционирования системы от пользователя скрыт и на диаграмме прецедентов не показывается.

В роли конечного пользователя может выступать человек (например, студент, покупатель или оператор) или техническое устройство (например, мобильный телефон, банкомат).

Разработку информационного или программного обеспечения начинают с анализа функциональных требований, указанных в техническом задании. В процессе анализа выявляют внешних пользователей разрабатываемого ИС или ПО и особенности поведения ИС или ПО в процессе взаимодействия с конкретными пользователями.

Прецеденты (варианты использования – Use Cases) – это подробные процедурные описания вариантов использования системы всеми заинтересованными лицами, а также внешними системами. Заинтересованные лица и внешние системы рассматриваются как актеры (actors) – действующие лица (в переводной литературе могут называться акторами). Действующие лица могут называть сущностями системы. Термин «сущность» объединяет понятия субъект (сущность, производящая действия) и объект (сущность, над которой производятся действия). По сути, варианты использования - это алгоритмы работы с системой с точки зрения внешнего мира.

Прецеденты отражают функциональные требования к системе, описывают границы проектируемой системы, ее интерфейс, а затем выступают как основа для тестирования системы заказчиком с помощью приемочных тестов.

В зависимости от цели выполнения конкретной задачи различают следующие варианты использования:

- основные, обеспечивают выполнение функций проектируемой системы;

- вспомогательные, обеспечивают выполнение настроек системы и ее обслуживание;

- дополнительные, служат для удобства пользователя (реализуются в том случае, если не требуют серьезных затрат каких-либо ресурсов ни при разработке, ни при эксплуатации).

Пример. Определить варианты использования и построить диаграммы вариантов использования для системы тестирования.

Система тестирования работает со следующими заинтересованными лицами: обучаемый и тестируемый (студент); составитель тестов и экзаменатор (преподаватель).

Основные прецеденты (варианты использования) будут следующие.

Прецедент для студента: П1 – пройти тестирование.

Прецеденты для преподавателя: П2 – создать/изменить тест; П3 – просмотреть результаты тестирования.

Вариант использования можно описать кратко или подробно. Краткое описание первого прецедента приведено в табл. 2.

Таблица 2

Краткое описание прохождения теста

Название варианта	Прохождение теста
Цель	Получение оценки
Действующие лица (актеры)	Студент

Продолжение табл. 2

Краткое описание	Регистрация студента, запуск теста, выбор ответа из нескольких предложенных или ввод ответа, завершение теста, получение оценки
Тип варианта	Основной

Подробное описание первого прецедента приведено в табл. 3.

Таблица 3

Подробное описание прохождения теста

Действия исполнителя	Отклик системы
1. Студент вводит свои данные (Номер зачетки, Шифр группы, ФИО), т.е. регистрируется в системе	2. Система запоминает сведения о студенте в соответствующей таблице базы данных, присваивает ему электронный номер и предлагает выбрать тест
3. Студент выбирает тест	4. Система запускает тест
5. Студент последовательно отвечает на вопросы	6. Система регистрирует правильные и неправильные ответы студента и запоминает эти данные в соответствующей таблице базы данных
7. Студент завершает тестирование	8. Система подсчитывает процент правильных ответов
9. Студент ожидает результат	10. Система демонстрирует результат и сохраняет его в соответствующей таблице базы данных
11. Студент завершает работу	12. Система завершает работу

Для большей наглядности используют разработку диаграмм вариантов использования.

На рис. 22 приведены условные обозначения, которые применяют при изображении диаграмм прецедентов.

Прецедент (вариант использования) изображается как овал, внутри которого пишется наименование варианта использования. Конечный пользователь, называемый «актером» (actors), изображается в виде стилизованной фигурки человека. Актером является любая сущность, взаимодействующая с системой извне, например человек, оборудование, другая система.

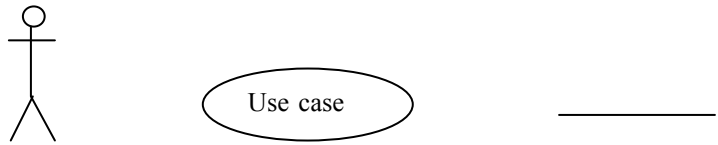


Рис. 22. Условные обозначения на диаграмме прецедентов:
а – актер; б – вариант использования; в – связь

Прецедент описывает, что система предоставляет актеру определяет набор транзакций, выполняемый актером при диалоге с системой. На диаграмме изображается один актер, но пользователей, выступающих в роли актера, может быть много. Диаграмма прецедентов использования имеет высокий уровень абстракции и позволяет определить функциональные требования к ПС.

Диаграмма прецедентов для вышеописанного примера тестирования приведена на рис. 23.

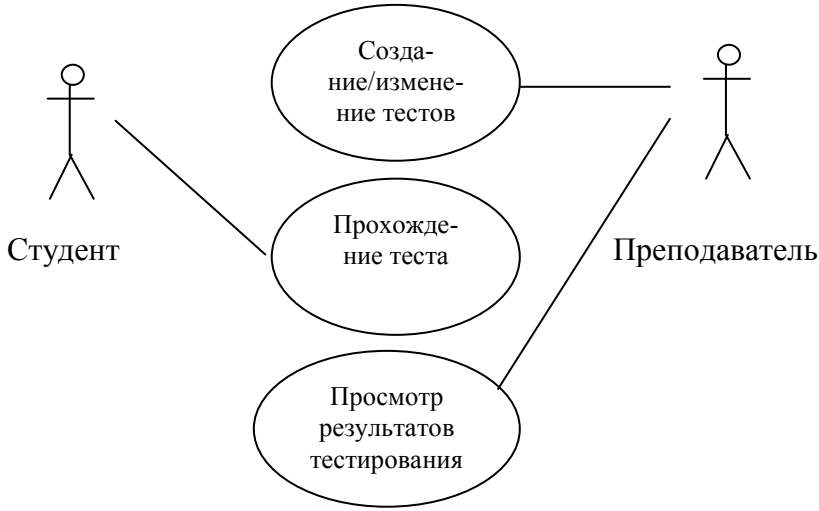


Рис. 23. Диаграмма вариантов использования

На диаграммах прецедентов может быть дополнительный элемент - интерфейс.

Интерфейс – это совокупность операций, предоставляемых классом или компонентом. Интерфейс описывает поведение класса или компонента, видимое извне. Интерфейс определяет только описание операций класса или компонента, но не определяет физическую реализацию операций.

Интерфейс представляет собой сущность, которая дает пользователю возможность совершить определенное действие, получить информацию. Для пользователя интерфейс может быть доступен как датчик, обращение к базе данных, кнопки, бланк заявления и т.д. Графически интерфейс обозначается небольшим кружком, рядом с которым указывается его наименование (рис. 24).

Рис. 24. Графическое изображение интерфейса

В нотации UML английские имена интерфейсов принято начинать с буквы I.

Между компонентами диаграммы прецедентов могут существовать различные отношения. Отношения могут быть между пользователями и прецедентами, между несколькими пользователями. Пользователь может взаимодействовать с несколькими прецедентами.

Ниже перечислены определенные в нотации UML виды отношений между компонентами на диаграммах прецедентов [10].

Отношение ассоциации (association relationship) устанавливает участие пользователя в работе прецедента. Обозначается сплошной линией между пользователем и прецедентом (рис. 25).

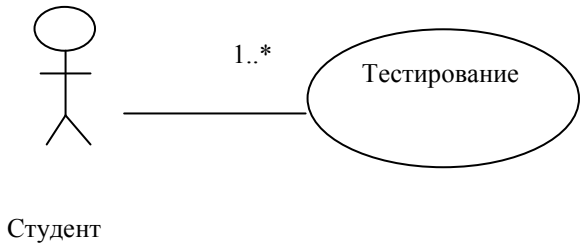


Рис. 25. Графическое изображение отношения ассоциации

Цифры над линией обозначают кратность отношения и показывают количество возможных компонентов данного отношения. Случай на рисунке означает, что один и тот же пользователь может применять прецедент много раз (обозначается *). В данном случае студент может проходить тестирование несколько раз.

Отношение расширения (extend relationship) определяет взаимосвязь прецедента с прецедентом, возможности которого он может использовать. Графически изображается пунктирной стрелкой с пометкой «extend» от дополняющего прецедента к расширяемому (рис. 26). Прецеденты «Биометрическая идентификация», Идентификация с помощью имени и пароля» являются дополняющими прецедентами по отношению к расширяемому прецеденту «Идентификация пользователя». Оба дополняющих прецедента могут выполняться в рамках расширяемого прецедента.

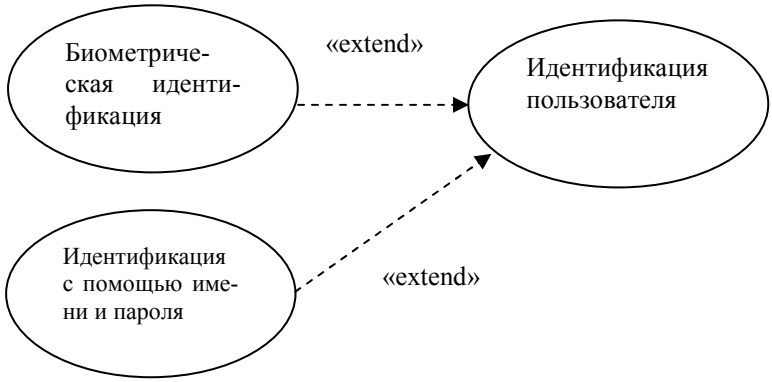


Рис. 26. Графическое изображение отношения расширения

Отношение обобщения (generalization relationship) показывает, что компонент (пользователь или прецедент) являет-

ся частным случаем другого компонента. Графически обозначается непрерывной стрелкой от общего к частному (рис. 27).

В примере прецедент «Оформление заявки» может быть реализован в трех вариантах:

- оформление заявки на подключение новой услуги;
- оформление заявки на выполнение ремонта;
- оформление заявки на отказ от услуги.

Рис. 27. Графическое изображение отношения обобщения

Отношение включения (include relationship) указывает на включение прецедента в другой прецедент в качестве его составной части. Один и тот же прецедент может быть включен в несколько более крупных прецедентов. Графически данное

отношение обозначается пунктирной линией со стрелкой, направленной от базового прецедента к включаемому с пометкой «include» (рис. 28).

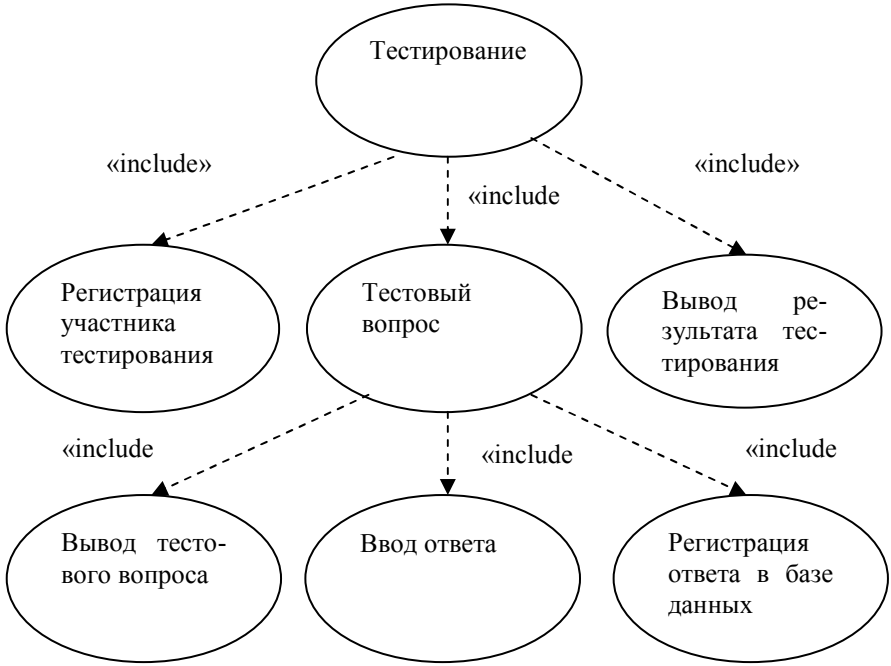


Рис. 28. Графическое изображение отношения включения

Пример различных отношений на диаграмме прецедентов представлен ниже. Предположим, что создается подсистема регистрации поступления больного в больницу. Подсистема содержит следующие прецеденты:

- проведение анализов;
- первичный осмотр;
- оформление приема пациента;
- направление в отделение.

Диаграмма прецедентов представлена на рис. 29.

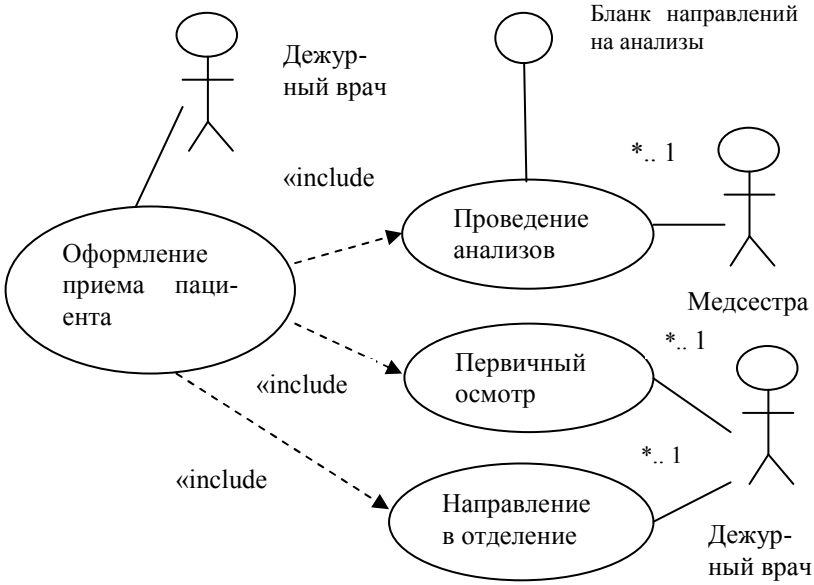


Рис. 29. Диаграмма прецедентов «Прием пациента в больницу»

3.4. Диаграммы классов

Центральное место в объектно-ориентированном подходе к проектированию программного обеспечения занимает разработка логической модели системы в виде диаграммы классов (class diagram).

UML предлагает использовать три уровня диаграмм классов в зависимости от степени их детализации:

- концептуальный уровень, на котором диаграммы классов отображают связи между основными понятиями предметной области;

- уровень спецификаций, на котором диаграммы классов отображают связи объектов этих классов;

- уровень реализации, на котором диаграммы классов непосредственно показывают поля и операции конкретных классов.

Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. Диаграммы классов обычно содержат следующие элементы:

- классы;
- интерфейсы;
- отношения между классами.

Диаграммы классов отображают статическую структурную модель проектируемой системы. Диаграммы классов принято считать графическим представлением таких взаимосвязей логической модели системы, которые не зависят от времени.

Класс (class) в языке UML служит для обозначения множества объектов, которые имеют одинаковую структуру, поведение и отношения с объектами других классов. На диаграмме класс изображают в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на две или три секции (рис. 30). В этих секциях могут указываться имя класса, атрибуты (переменные) и операции (методы). Иногда в графическом изображении класса добавляется четвертая секция, содержащая описание исключительных ситуаций.

Обязательным элементом обозначения класса является его имя. На начальных этапах разработки диаграммы класс может обозначаться простым прямоугольником с указанием только его имени (рис. 30, а). В дальнейшем диаграммы описания классов дополняются атрибутами (рис. 30, б) и операциями (рис. 30, в).

Чтобы отличить класс от других элементов языка UML, секции атрибутов и операций выделяют горизонтальными ли-

ниями, даже если они пустые. На рис. 31 приведены примеры графического изображения классов на диаграмме классов.

Рис. 30. Графическое изображение класса на диаграмме классов

В первом случае для класса Прямоугольник (рис. 31, а) указаны только его атрибуты – точки на координатной плоскости, определяющие его местоположение.

Для класса Окно (рис. 31, б) указаны только его операции (показать(), скрыть()), секция атрибутов оставлена пустой.

Для класса Счет (рис. 31, в), кроме операции проверки кредитной карточки, дополнительно изображена четвертая секция, в которой указано исключение - отказ от обслуживания просроченной кредитной карточки.

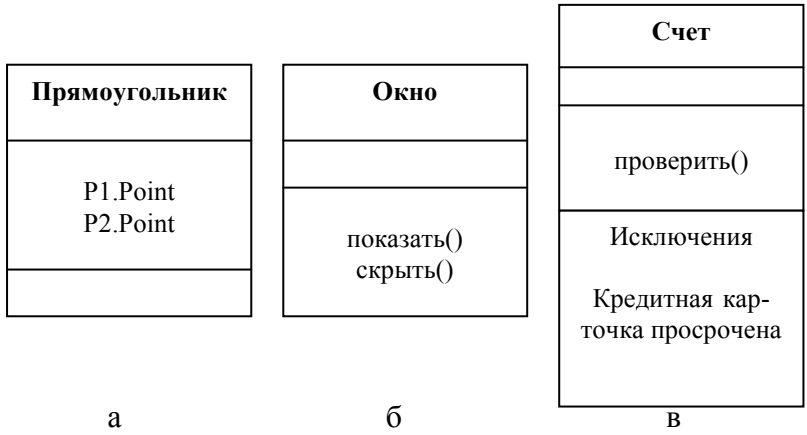


Рис. 31. Примеры графического изображения классов

Имя класса. Имя класса должно быть уникальным в пределах диаграммы или совокупности диаграмм классов пакета. Оно указывается в первой верхней секции прямоугольника, записывается по центру секции имени полужирным шрифтом и начинается с заглавной буквы. В качестве имен классов рекомендуется использовать одно или несколько существительных без пробелов между ними. Кроме того в секции имени могут находиться ссылки на классы, о которых образован данный класс и, соответственно, от которых он наследует свойства и методы.

Примерами имен классов могут быть такие существительные, как «Сотрудник», «Фирма», «Покупатель», «Студент», «Дисциплина», «Факультет» и многие другие, имеющие отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Если класс не имеет экземпляров (объектов), то он называется абстрактным классом, его имя записывается курсивом.

Атрибуты класса. Во второй секции прямоугольника – графического изображения класса – записывают его атрибуты

(attributes) или свойства. Стандартная запись атрибута в языке UML имеет следующий вид:

<квантор видимости><имя атрибута>[кратность]: <тип атрибута> = <исходное значение>{строка - свойство}

Квантор видимости может быть опущен. Это значит, что видимость атрибута не указывается либо должна принимать одно из трех возможных значений:

общедоступный (public) – обозначается «+»;

защищенный (protected) – обозначается «#»;

закрытый (private) – обозначается «-».

Имя атрибута – единственный обязательный элемент обозначения атрибута. Это строка текста, которая используется в качестве идентификатора соответствующего атрибута и является уникальной в пределах данного класса.

Кратность атрибута показывает количество конкретных атрибутов данного типа, входящих в состав класса, и обозначается следующим образом:

[нижняя_граница1 .. верхняя_граница1, нижняя_граница2 .. верхняя_граница2, …, нижняя_границаk .. верхняя_границаk],

где нижняя_граница и верхняя_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел. В качестве верхней границы может использоваться специальный символ «*», который означает произвольное положительное число, т.е. неограниченное сверху значение кратности соответствующего атрибута.

Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. Нижние и верхние границы интервалов включаются в значение кратности. Если

же указывается единственный знак «*», то это означает, что кратность атрибута может быть произвольным положительным числом или нулем.

Могут использоваться следующие варианты задания кратности атрибутов:

[0..1] означает, что кратность атрибута может принимать значения 0 или 1; при этом 0 означает отсутствие значения для данного атрибута;

[0..*] или просто [*] означает, что кратность атрибута может принимать любое положительное целое значение, большее или равное 0;

[1..*] означает, что кратность атрибута может принимать любое положительное целое значение, большое или равное 1;

[1..5] означает, что кратность атрибута может принимать любое значение из чисел 1, 2, 3, 4, 5;

[1..3, 7..10] означает, что кратность атрибута может принимать любое значение из чисел 1, 2, 3, 7, 8, 9, 10;

[1..3, 7..*] означает, что кратность атрибута может принимать любое значение из чисел 1, 2, 3, а также любое положительное целое значение, большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение, равное 1.

Тип атрибута указывается строкой текста, имеющей некоторое осмысленное значение. Тип атрибута можно определять в зависимости от языка программирования, который будет использоваться для реализации данной модели. Например:

Имя_студента [1..100] : String

В данном примере имя_студента – это имя атрибута; String – тип атрибута (строка).

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Например:

Имя_студента [1..100] : String = Андрей

Строка-свойство служит для указания фиксированных значений атрибута. Эти значения не могут быть изменены в программе при работе с данным типом объектов. При отсутствии строки-свойства значение соответствующего атрибута может быть изменено в программе. Например, строка-свойство в записи атрибута

Стипендия : Integer = {900}

означает фиксированную стипендию для всех объектов класса «Студент» независимо от результатов сессии.

Запись данного атрибута в виде

Стипендия : Integer = 900

означает, что при создании нового экземпляра Студент для него устанавливается по умолчанию стипендия в 900 рублей. Но по результатам очередной сессии эта стипендия может измениться.

Операция. Операцией класса (методом класса) называется именованный сервис, который предоставляется по требованию любым объектом данного класса. Другими словами, операция – это то, что может делать объект, или то, что можно сделать с объектом. Класс может содержать любое число операций (в частности, не содержать ни одной операции). Набор операций класса является общим для всех объектов данного класса.

Операции класса определяются в секции, расположенной ниже секции атрибутов. При этом можно ограничиться только указанием имен операций, оставив детальную спецификацию выполнения операций на более поздние этапы моделирования. Для именования операций используют глаголы. Описание операции имеет следующий общий вид:

<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения> {строка-свойство}.

Квантор видимости принимает такие же значения, как и в случае атрибутов класса, и может быть опущен. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private.

Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции, и поэтому должна быть уникальной в пределах данного класса. Имя является единственным обязательным элементом для обозначения операции.

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых может быть представлен в следующем виде:

<вид параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>.

Вид параметра – это одно из ключевых слов in, out, inout со значением in по умолчанию.

Имя параметра – идентификатор соответствующего формального параметра.

Выражение типа зависит от конкретного языка программирования и описывает тип возвращаемого значения для соответствующего формального параметра.

Значение параметра по умолчанию в общем случае представляет собой выражение для значения формального параметра.

Выражение типа возвращаемого значения также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Операция может не возвращать никакого значения. Для указания кратности возвращаемого значения данная спецификация может быть записана в виде списка отдельных выражений.

Строка-свойство служит для определения значений свойств данного элемента. Строка-свойство может отсутствовать, если никакие свойства не специфицированы.

Операция может изменять или не изменять состояние системы.

Описание операции на самом верхнем уровне объявляет эту операцию на весь класс, при этом данная операция наследуется всеми потомками данного класса.

Интерфейсы. Интерфейс (interface) – именованное множество операций, характеризующих поведение отдельного элемента модели извне без указания их внутренней структуры.

В языке UML интерфейс является специальным случаем класса, у которого имеются операции, но отсутствуют атрибуты.

Для обозначения интерфейса на диаграмме классов используется специальный графический символ – окружность, рядом с которой указывается имя интерфейса, или стандартный способ – прямоугольник класса с обозначением «Interface» (рис. 32).

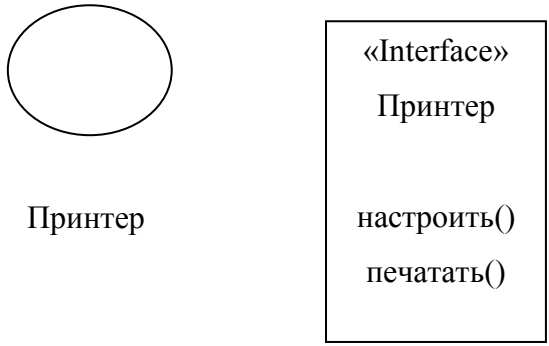


Рис. 32. Обозначения интерфейсов

Отношения между классами. На диаграмме классов необходимо отобразить отношения между классами. Основными отношениями или связями в языке UML являются:

отношение зависимости;
отношение ассоциации;
отношение агрегации;
отношение композиции;
отношение обобщения.

Все эти отношения обозначаются по-своему на диаграмме и отражают различные типы взаимосвязей между классами и их объектами.

Отношение зависимости используется в ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов («→» или «←»). На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от подчиненного класса к главному классу (рис. 33).

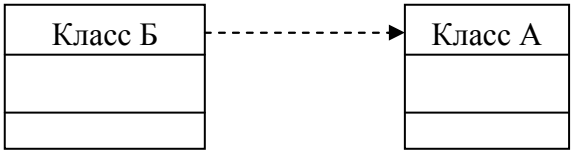


Рис. 33. Графическое изображение отношения зависимости на диаграмме классов

Отношение ассоциации (смысловая зависимость между классами) обозначается сплошной линией с дополнительными специальными символами, которые характеризуют от-

дельные свойства конкретной ассоциации. Это могут быть имя ассоциации, а также имена ролей и кратность классов.

Имя ассоциации является необязательным, но если оно задано, то записывается с прописной (заглавной) буквы рядом с линией соответствующей ассоциации.

Ассоциация, связывающая два класса (или класс с самим собой), называется бинарной. Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны основания треугольника), а другой – вторым (со стороны вершины треугольника). Отсутствие данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

На рис. 34 показано отношение бинарной ассоциации между классом «Группа» и классом «Студент». Они связаны между собой бинарной ассоциацией «Учеба», имя которой указано на рисунке над линией ассоциации. Порядок следования классов в данном отношении таков: первым является класс «Группа», вторым – класс «Студент».

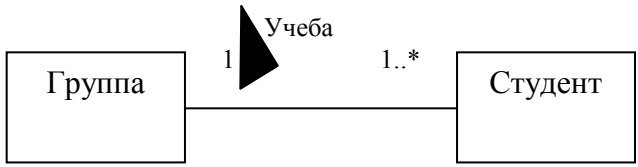


Рис. 34. Графическое изображение отношения бинарной ассоциации между классами

Кроме имени ассоциации на диаграмме может быть указана кратность отдельных классов. Кратность показывает число экземпляров в классе. На рис. 8 кратность «1» для класса

«Группа» означает, что каждый студент может учиться только в одной группе. Кратность «1..*» для класса «Студент» означает, что в каждой группе могут учиться несколько студентов, общее число которых заранее неизвестно и ничем не ограничено, но всегда больше нуля.

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение применяется для представления системных взаимосвязей типа «часть-целое». Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они взаимосвязаны между собой. При этом части системы могут не наследовать ее свойств и поведения, поскольку являются вполне самостоятельными сущностями. Более того, части целого обладают своими собственными атрибутами и операциями, которые могут существенно отличаться от атрибутов и операций целого.

Агрегация является частным случаем ассоциации и изображается в виде пустой ассоциации с незакрашенным ромбом со стороны «целого». Примером отношения агрегации может служить деление компьютера на составные части: системный блок, монитор, клавиатура и мышь (рис. 35).

Отношение композиции. Отношение композиции является частным случаем отношения агрегации. Это отношение служит для описания специальной формы отношения «часть-целое», при которой составляющие части в некотором смысле находятся внутри целого. Причем части не могут выступать в отрыве от целого, то есть с уничтожением целого уничтожаются и все его составные части.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на «целое».

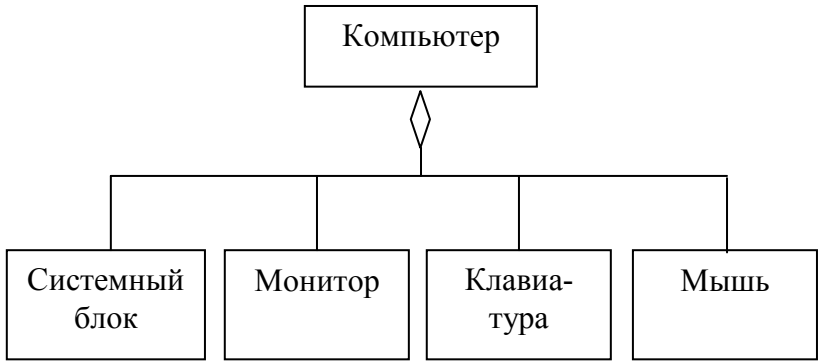


Рис. 35. Диаграмма классов для иллюстрации отношения агрегации на примере структуры компьютера

Пример отношения композиции – окно интерфейса программы, которое может состоять из строки заголовка, главного меню, рабочей области, кнопок управления размером, полос прокрутки и строки состояния.

В качестве дополнительных обозначений для отношения композиции и агрегации могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, указание кратности ассоциации и имени данной ассоциации. Дополнительные обозначения не являются обязательными.

Диаграмма классов для класса «Окно программы» может иметь следующий вид (рис. 36).

Отношение обобщения. Отношение обобщения является отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства

и поведение, которые отсутствуют у класса-предка. Графически отношение обобщения изображается в виде линии с большой незакрашенной стрелкой, направленной на родителя.

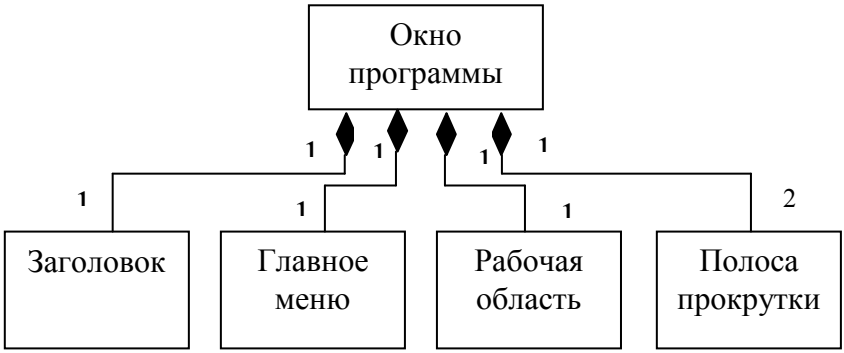


Рис. 36. Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

Пример отношения обобщения показан на рис. 37. Здесь абстрактный класс «Геометрическая фигура» выступает в качестве суперкласса (класса-предка) для подклассов (классов-потомков), соответствующих конкретным геометрическим фигурам «Прямоугольник», «Окружность», «Эллипс» и др.

С целью упрощения обозначений на диаграмме классов совокупность линий, обозначающих одно и то же отношение обобщения, может быть объединена в одну линию. В этом случае отдельные линии изображаются сходящимися к единственной стрелке, имеющей с ними общую точку пересечения (рис. 37).

Многоточие вместо прямоугольника на диаграмме означает возможность наличия других классов-потомков, не включенных в обозначения представленных на диаграмме классов.

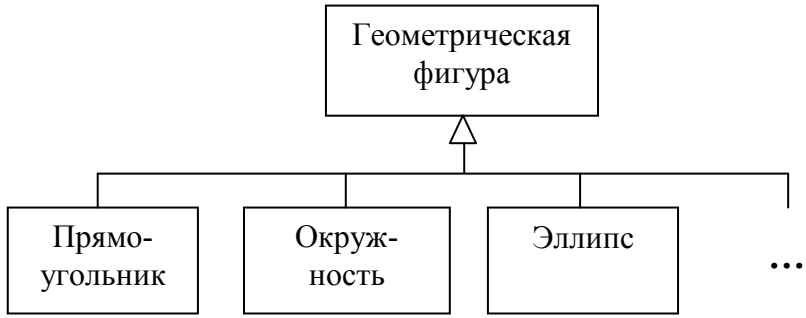


Рис. 37. Пример графического изображения обобщения классов

Пример. Разработать диаграмму классов для некоей компании.

Класс «Компания состоит:

- из нескольких отделов – класс «Отдел»;
- каждый отдел располагается в своем офисе – класс «Офис»;
- каждый отдел содержит штат сотрудников – класс «Сотрудник», сведения о которых содержатся в системе кадрового учета.

Каждый из вышеперечисленных классов обладает своими атрибутами и операциями и связан с другими классами определенным типом отношений.

Полученная диаграмма приведена на рис. 38.

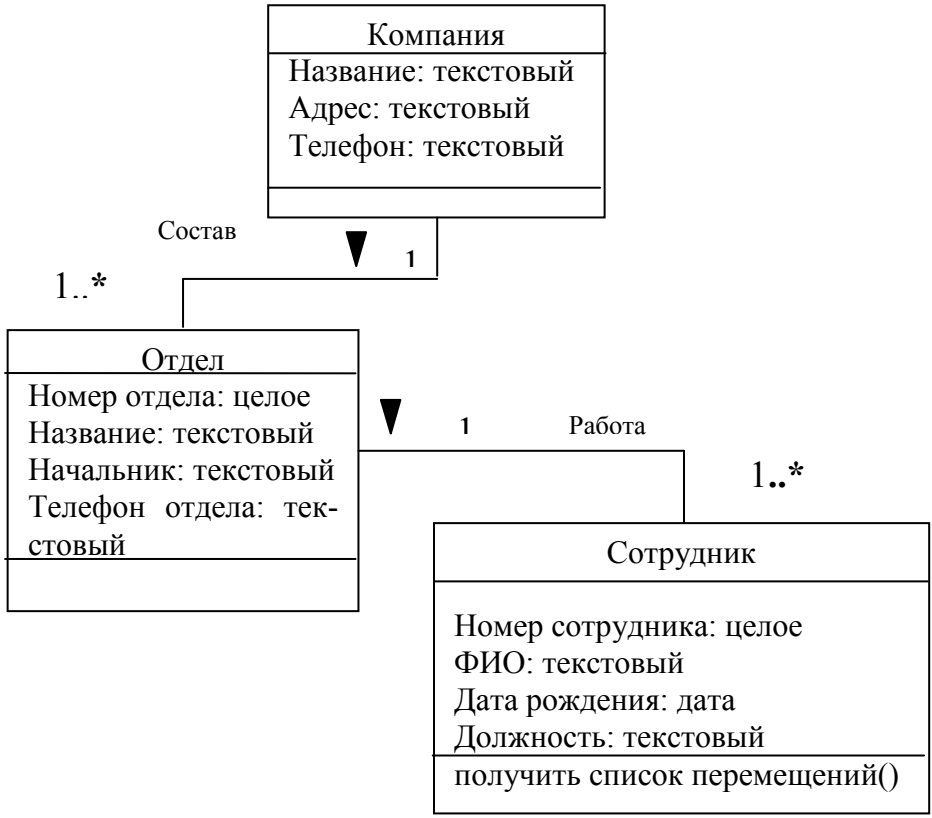


Рис. 38. Диаграммы классов

3.5. Диаграммы последовательностей, деятельности и состояний

Для описания особенностей поведения программных систем используют диаграммы последовательностей, деятельности и состояний.

Диаграмма последовательностей системы – графическая модель, которая для определенного сценария варианта ис-

пользования показывает динамику взаимодействия объектов во времени.

Для построения диаграммы последовательностей системы необходимо:

идентифицировать каждое действующее лицо (объект, актер) и изобразить для него линию жизни. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым;

из описания варианта использования определить множество системных событий и их последовательность;

изобразить системные события в виде линий со стрелкой на конце между линиями жизни действующих лиц и системы, а также указать имена событий и списки передаваемых значений.

Графически каждый **объект** изображается прямоугольником и располагается в верхней части своей линии жизни (рис. 2.18). Внутри прямоугольника записывают имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который представляет собой экземпляр класса.

Линия жизни объекта служит для обозначения периода времени, в течение которого объект существует в системе. На диаграмме линия жизни изображается пунктирной вертикальной линией, связанной с объектом. Если объект существует в системе постоянно, то его линия жизни должна продолжаться по всей плоскости диаграммы от самой верхней ее части до самой нижней.

Если объекты разрушаются в какой-то момент времени, то их линия жизни обрывается. Для обозначения такого момента в языке UML используется специальный символ «X». Ниже этого символа пунктирная линия не изображается.

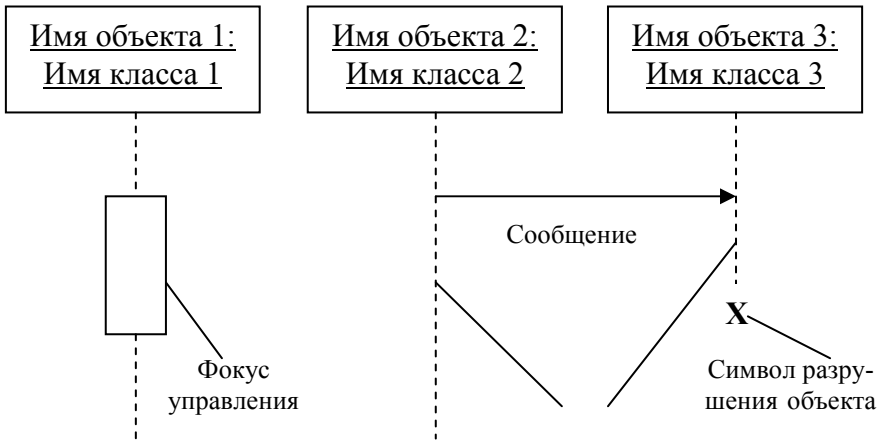


Рис. 39. Различные графические примитивы диаграммы последовательности

Фокус управления. Объекты на диаграмме последовательности могут находиться в двух состояниях, активном - непосредственно выполняя какие-либо действия, и пассивном, ожидая сообщения от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления. Фокус управления изображается в форме вытянутого узкого прямоугольника. Верхняя сторона прямоугольника обозначает начало получения фокуса управления объектом (начало активности), а ее нижняя сторона – окончание фокуса управления (окончание активности).

Сообщения. Диаграмма последовательности описывает динамику взаимодействий между множеством объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми обмениваются между собой участвующие в нем объекты.

Сообщение представляет собой законченный фрагмент информации, который отправляется одним объектом другому.

Объект, принявший сообщение, должен отреагировать на него какой-либо последовательностью действий, направленных на решение поставленной задачи.

Графически сообщения изображаются горизонтальными стрелками, соединяющими линии жизни или фокусы управления двух объектов на диаграмме последовательности.

Пример построения диаграммы последовательности для моделирования процесса работы банкомата. Объектами в этом примере являются: a: клиент, b: банк, c: служба безопасности банка, d: банкомат (рис. 40).

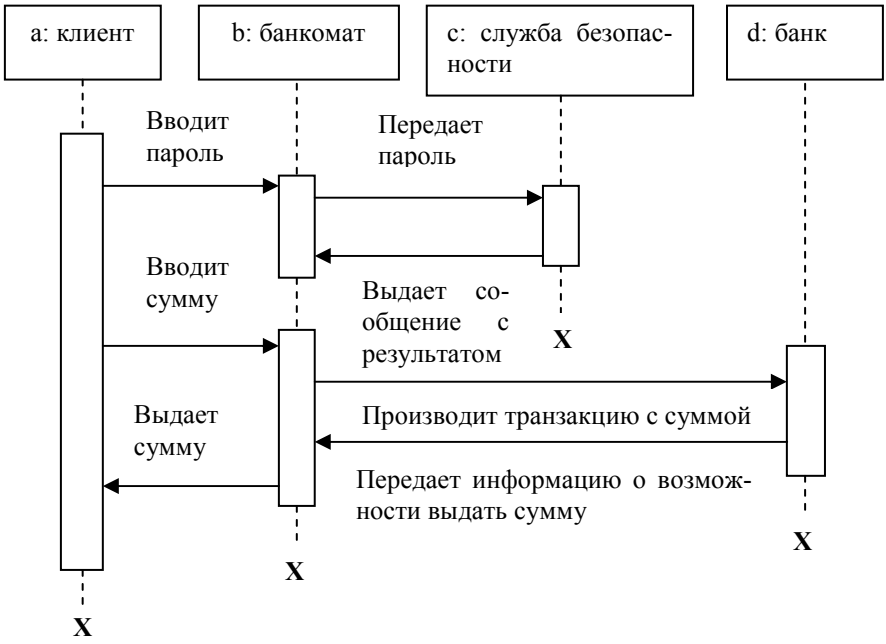


Рис. 40. Диаграмма последовательности для моделирования работы банкомата

Пример 2. Построение диаграммы последовательности для моделирования поиска информации о сотруднике. Объектами в этом примере являются: a – клиент, b – программа, c – база данных (рис. 41).

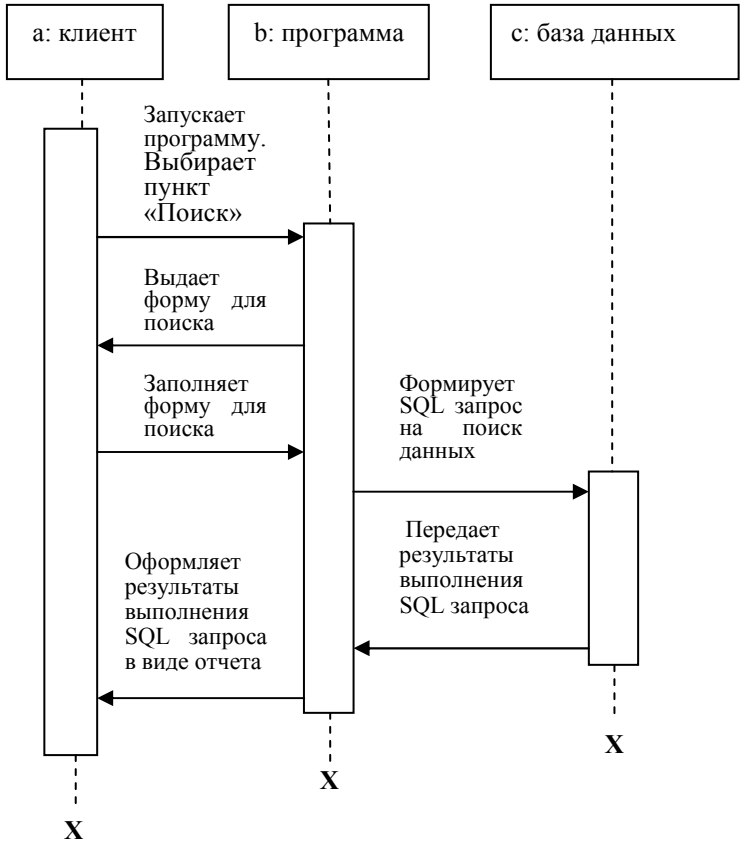


Рис. 41. Диаграмма последовательности для моделирования поиска информации о сотруднике

Диаграммы деятельностей. Диаграммы деятельностей позволяют конкретизировать основные функции разрабатываемого программного обеспечения.

Под деятельностью понимают задачу (операцию), которую необходимо выполнить вручную или с помощью средств автоматизации.

Каждому варианту использования соответствует своя последовательность задач. В теоретическом плане диаграммы деятельности являются обобщенным представлением алгоритма, реализующего анализируемый вариант использования.

Графически состояние действия представляется прямоугольником со скругленными углами (рис. 42). Внутри этой фигуры записывается выражение действия, которое должно быть уникальным в пределах одной диаграммы деятельности.

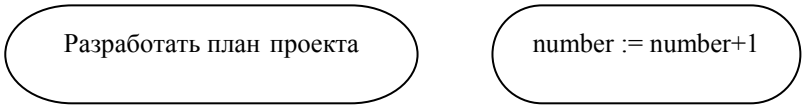


Рис. 42. Графическое изображение состояния действия:
а - простое действие; б - выражение

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Рекомендуется в качестве имени действия на естественном языке использовать глагол с пояснительными словами (рис. 43, а). Если это возможно, то допускается запись действия на том языке программирования, на котором предполагается реализовывать конкретный проект (рис. 43, б).

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния (рис. 43).

Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз.

На диаграмме деятельности также показывают переходы. Если переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии, то такой переход изображают сплошной линией со стрелкой.

Рис. 43. Графическое изображение:
а – начальное состояние; б – конечное состояние

Если последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата, то указывается символ ветвления (ромб), а на альтернативных ветках в квадратных скобках указывают условие выполнения данной ветки (рис. 44).

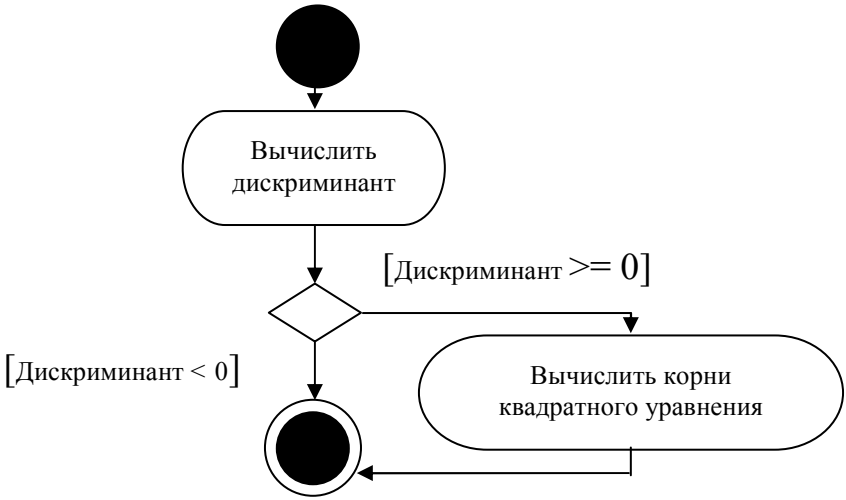


Рис. 44. Фрагмент диаграммы деятельности

Для представления параллельных процессов в языке UML используется специальный символ разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая горизонтальная линия большей толщины, чем все остальные. Пример разделения показан на рис. 45, а. Пример слияния изображен на рис.45, б.

Рис. 45. Графическое изображение:
а – разделения; б – слияния параллельных потоков управления

На рис. 46 показан пример диаграммы деятельности.

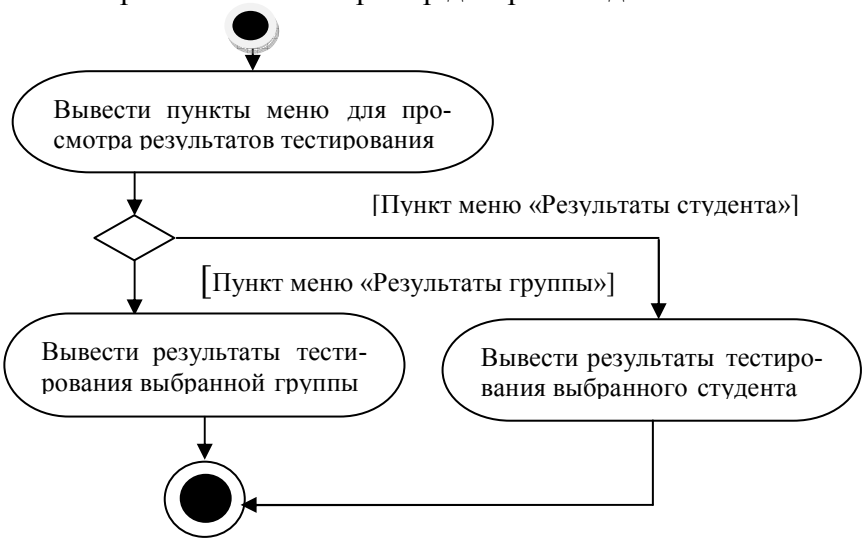


Рис. 46. Пример диаграммы деятельности

На рис. 47 показан пример диаграммы деятельности для работы с главным меню.

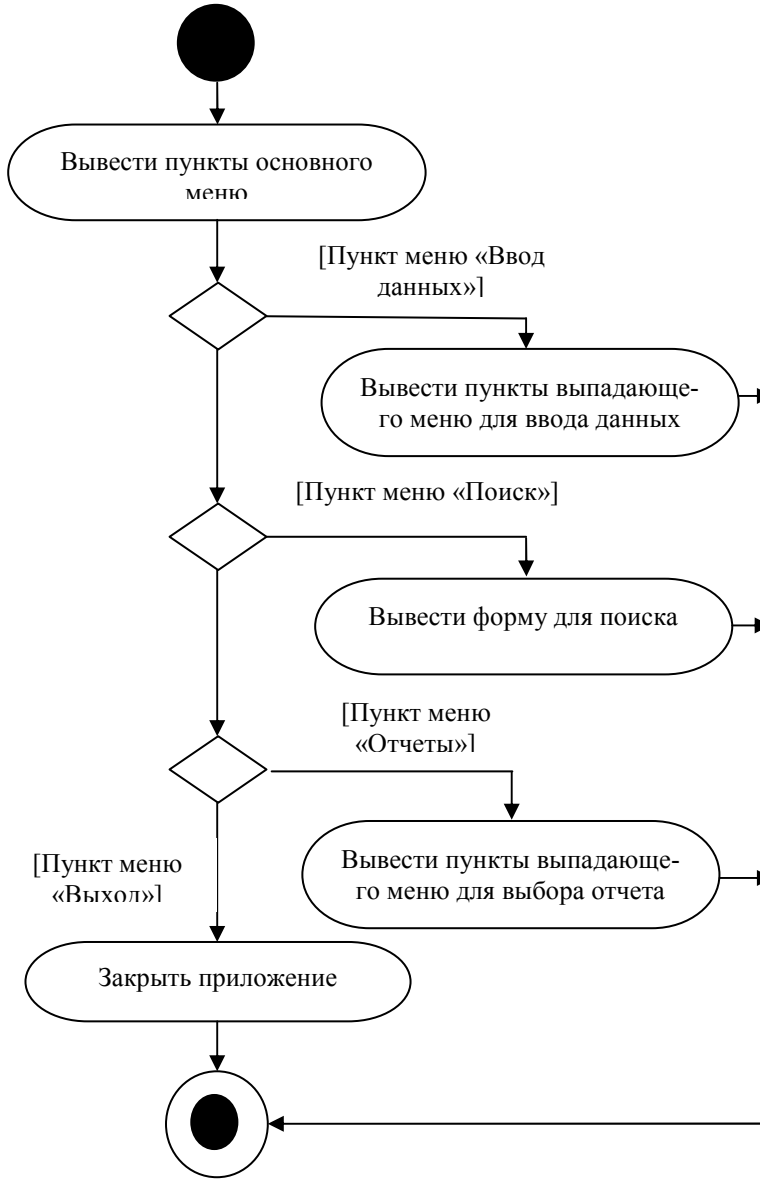


Рис. 47. Пример диаграммы деятельности

Диаграммы состояний. Диаграмма состояний описывает возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Данные диаграммы моделируют поведение реактивных объектов. Реактивным называется объект, поведение которого лучше всего характеризуется его реакцией на события, произошедшие вне его собственного контекста. У реактивного объекта есть четко выраженный жизненный цикл, когда текущее поведение обусловлено прошлым. Если внешние действия, изменяющие состояние системы, инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

В качестве примера можно рассмотреть моделирование поведения встроенной системы безопасности в доме. Она работает непрерывно, реагируя на внешние события, например, на разбитое окно. Это событие изменяет поведение системы. Обнаружение разбитого окна вызовет срабатывание сигнализации. Поведение такой системы лучше всего описывается путем моделирования ее устойчивых состояний (например, Ожидание, Активна, Проверка и т.д.), событий, инициирующих смену состояний, и действий, выполняемых при каждой такой смене.

Автомат. Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния, которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние.

Простейшим пример автомата с двумя состояниями демонстрирует компьютер. Он имеет два самых общих состояния: исправен, неисправен и два перехода: выход из строя, ремонт. Графически эта информация представлена на рис. 48.

Основными понятиями, описывающими автомат, являются состояние и переход. Предполагается, что система находится в каком-либо состоянии в течение некоторого времени,

тогда как переход объекта из состояния в состояние происходит мгновенно.

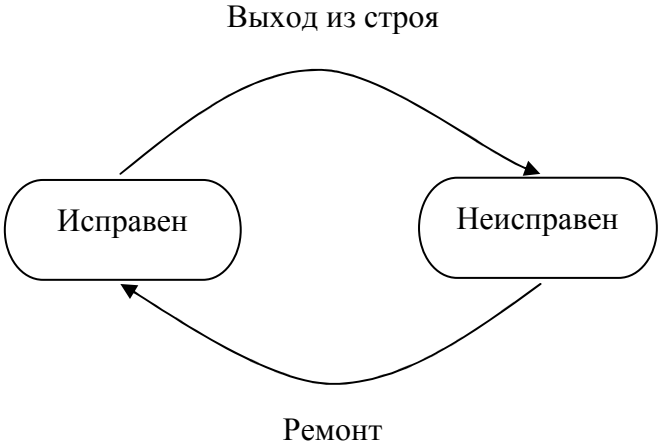


Рис. 48. Пример диаграммы состояний

Состояние. Состояние – это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 49), который может быть разделен горизонтальной линией на две секции.

а б

Рис. 49. Графическое изображение состояний

В прямоугольнике может располагаться имя состояния (первая секция) и список внутренних действий в данном состоянии (вторая секция). При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения.

Имя состояния. Имя состояния – это строка текста, начинающаяся с заглавной буквы и раскрывающая содержательный смысл данного состояния. Имя является необязательным элементом. Рекомендуется в качестве имени использовать глаголы в настоящем времени (печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено).

Список внутренних действий. Эта секция содержит перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения моделируемого объекта в данном состоянии. Каждое из действий записывается в отдельной строке и имеет следующий формат:

<метка действия / выражение действия>

Метка действия указывает на условия, при которых будет выполняться деятельность, определенная выражением действия. При этом выражение действия может использовать любые атрибуты и связи, которые принадлежат области имен или контексту моделируемого объекта. Если список выражений пуст, то разделитель в виде наклонной черты может не указываться.

На рис. 50 показан пример состояния «Считывает запись» после открытия файла, содержащего несколько записей.

Начальные и конечные состояния описываются так же, как и на диаграмме деятельности.

Переход. Простой переход представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим состоянием.

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в следующее состояние. Рядом с линией может находиться строка текста, описывающая событие, вызывающее переход, и сторожевое условие, по которому осуществляется переход.

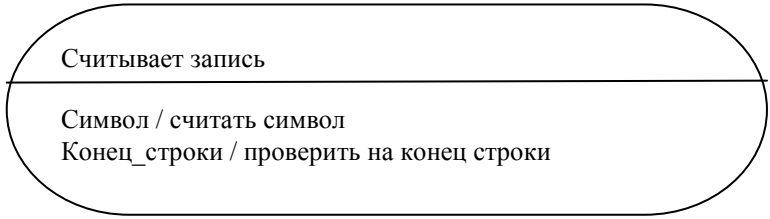


Рис. 50. Пример состояния с непустой секцией внутренних действий

Событие. Событие – это спецификация существенного факта, который происходит во времени и пространстве. В контексте автоматов событие – это стимул, способный вызвать срабатывание перехода.

Сторожевое условие. Сторожевое условие, если оно есть, всегда записывается в прямых скобках после события и представляет собой некоторое булевское выражение (выражение, результатом которого является «истина» или «ложь»).

Пример диаграммы состояний почтовой программы-клиента показан на рис. 51.

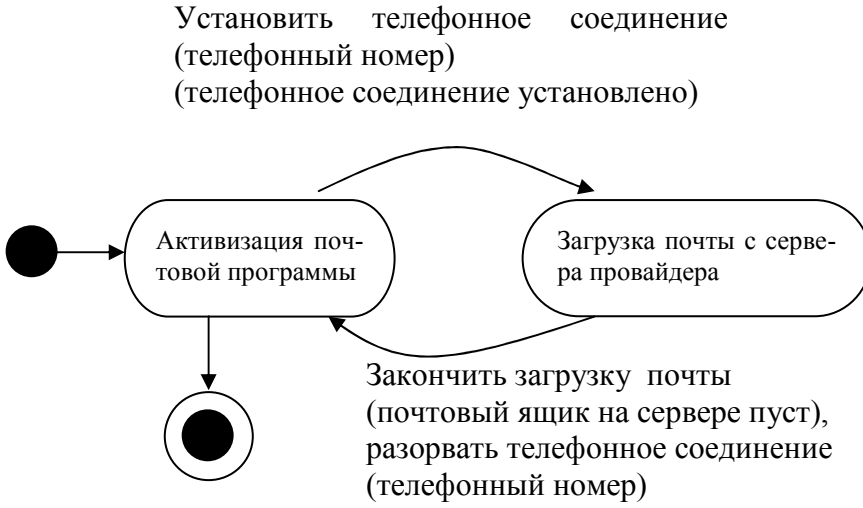


Рис. 51. Диаграмма состояний для моделирования почтовой программы-клиента

Пример диаграммы состояний для программы заказа книг в электронном книжном магазине представлен на рис. 52.

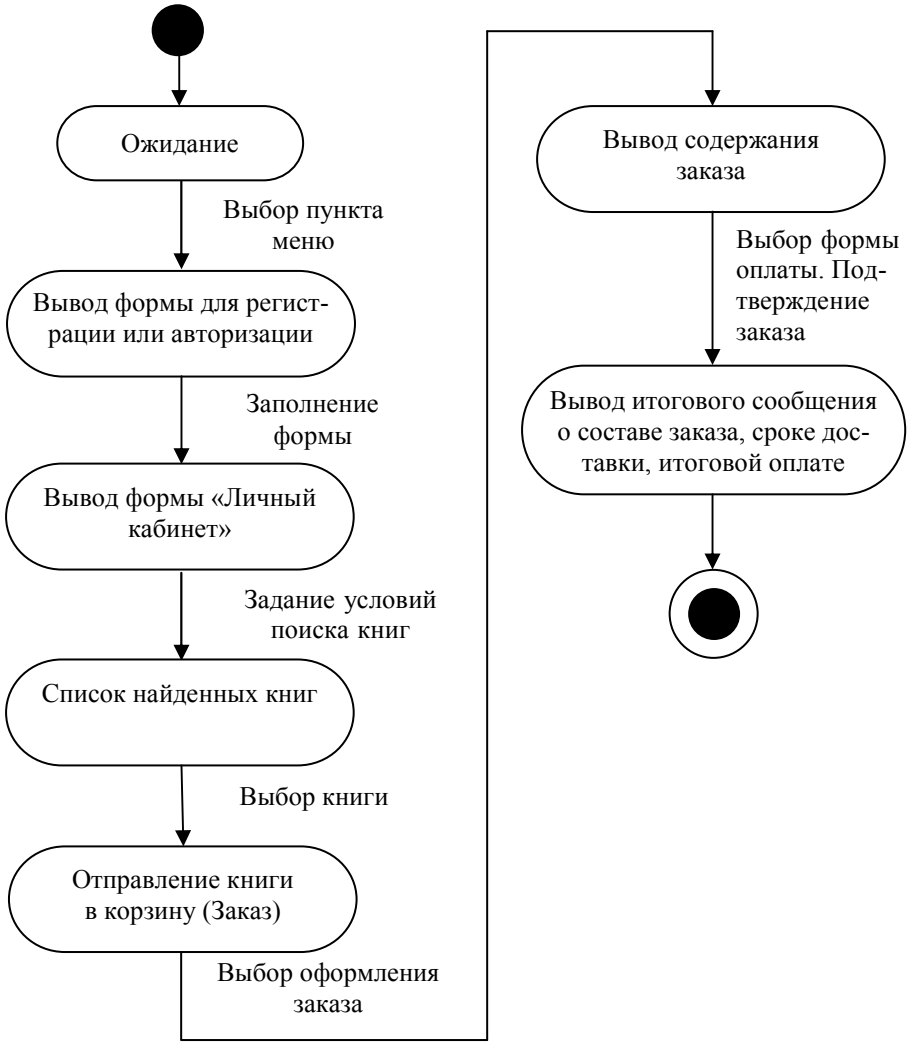


Рис. 52. Диаграмма состояний для программы заказа книг в электронном магазине

3.6. Диаграммы активности

Диаграммы активности показывают движение потоков данных в проектируемой системе.

Начало и конец диаграммы изображаются так же, как и на диаграмме состояний.

Прямоугольник с округлыми боковыми сторонами изображает действие над данными. Внутри такого прямоугольника указывают выполняемое действие (например, записывают текст или математическое выражение). Действие может состоять из нескольких частей. Если на диаграмме не показывают эти части, то в прямоугольнике указывают значок составного действия (рис. 53).

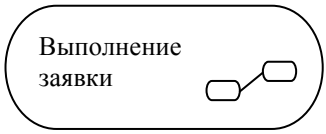


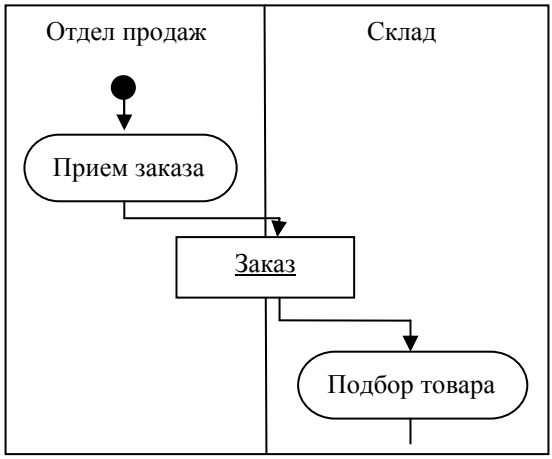
Рис. 53. Составное действие

Движение данных (переходы) изображается сплошными стрелками. Стрелки могут сопровождаться надписями. Если при движении данных возможно ветвление, то ромбом указывают проверку некоторого условия.

Разделение потока данных и слияние параллельных потоков данных изображается сплошной жирной линией, связывающей стрелки движения потоков данных.

Субъекты, участвующие в обработке данных, на диаграмме изображаются отдельно, каждый в своей полосе или дорожке (swimlanes).

Передаваемые данные могут быть указаны в явном виде, тогда они изображаются на границе дорожек. Например, на рис. 54 показана передача заказа из отдела продаж на склад для реализации процедуры комплектации заказа товарами.



54. Движение заказа между отделами

На рис. 55 показана диаграмма активности, которая демонстрирует процесс рассмотрения заявки на получение кредита.

На диаграмме выделено четыре субъекта (соответственно четыре дорожки): менеджер по кредитам, специалист по финансовым рискам, начальник кредитного отдела, служба безопасности.

Действие «Удовлетворение заявки» является составным. Оно может включать следующие действия: оценка достоверности данных о клиенте, оценка правильности расчетов, непосредственно принятие решения.

На диаграмме предусмотрено ветвление данных. В ходе первичной проверки представленных документов возможны два действия: отказ от выдачи кредита или дальнейшее рассмотрение документов на получение кредита.

На диаграмме также показано разделение потока данных, которые проверяются параллельно: это расчет финансовых рисков и сбор информации о клиенте.

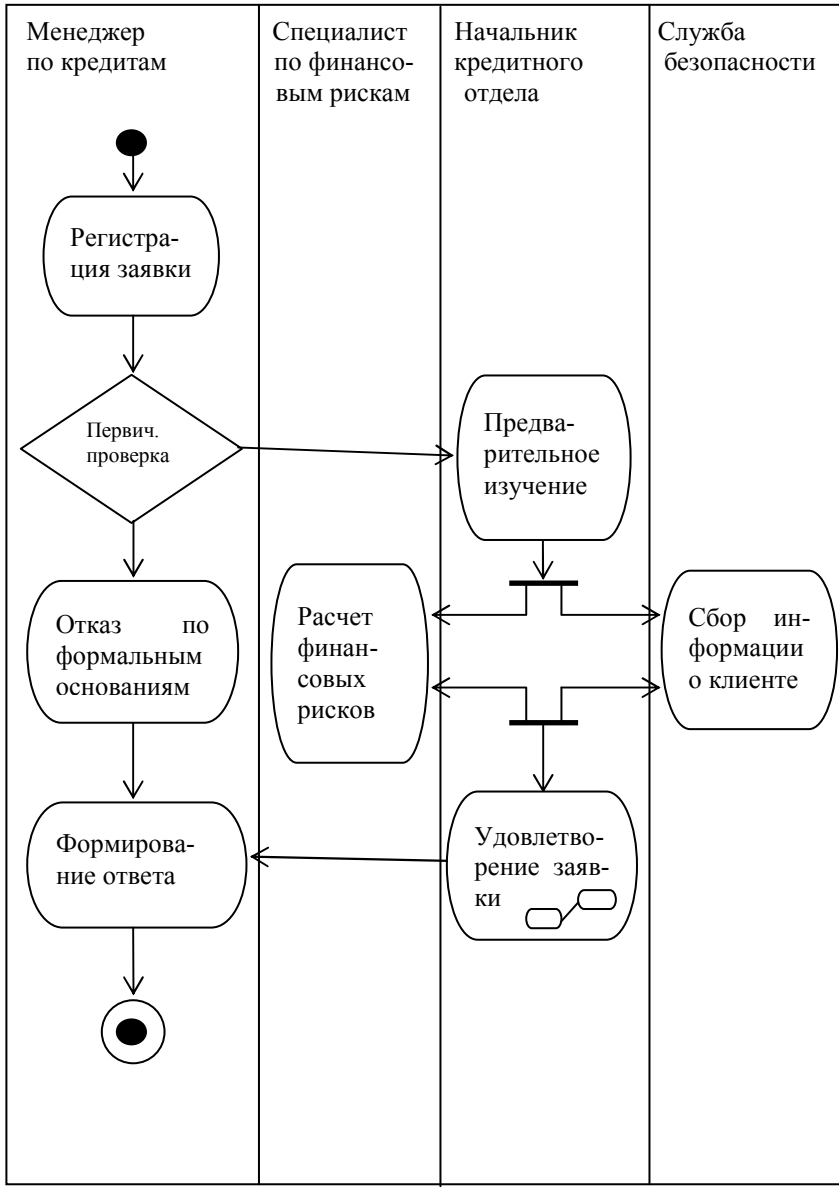


Рис. 55. Пример диаграммы активности

3.7. Диаграммы сотрудничества (кооперации)

Диаграммы сотрудничества являются разновидностью диаграмм взаимодействия.

Данные диаграммы показывают взаимодействие между объектами и передачу данных во время этого взаимодействия.

Диаграммы сотрудничества бывают двух видов: уровня спецификаций и уровня примеров.

Диаграммы сотрудничества уровня спецификаций оперируют классами, пользователями, кооперациями и ролями, которые играют пользователи и классы. На диаграмме используют следующие обозначения:

- окружность – это кооперация (объединенное выполнение работы или функции);
- пунктирная линия – роль пользователя в кооперации;
- стрелка – отношение обобщения.

Кооперация определяет взаимодействие классов. Участвуя в кооперации, классы совместно производят некоторый кооперативный результат. Пример диаграммы сотрудничества уровня спецификаций представлен на рис. 56.

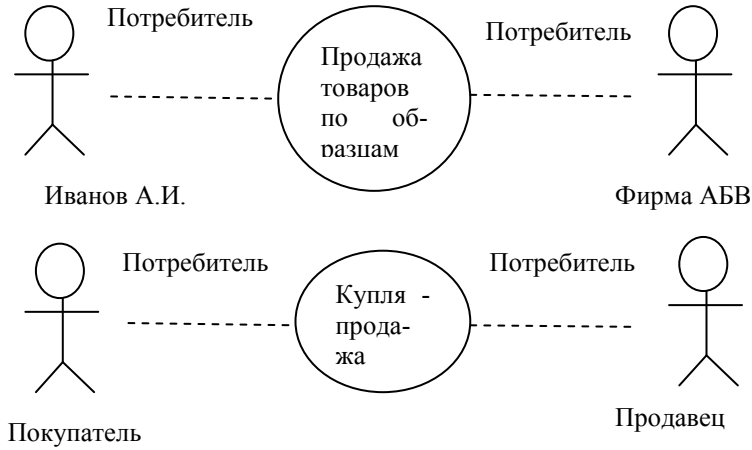


Рис. 56. Диаграмма сотрудничества уровня спецификаций

Объекты на диаграммах сотрудничества обозначаются прямоугольниками. Однако на диаграммах данного вида имя объекта может дополняться его ролью в сотрудничестве, а также указывается класс, к которому относится объект. Любая из трех частей обозначения объекта может отсутствовать. Общее обозначение объекта следующее: объект / роль объекта: класс.

На рис. 57 приведен пример обозначения имени объекта.

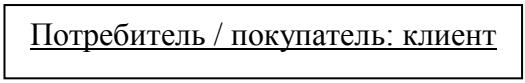


Рис. 57. Графическое изображение объектов на диаграмме сотрудничества

Объекты, которые могут управлять другими объектами, называются активными (active object) и помечаются словом (active).

Для обозначения группы объектов, которым адресован один и тот же сигнал, вводится понятие мультиобъекта. Мультиобъект изображается графически двумя прямоугольниками, наложенными друг на друга (рис. 58).

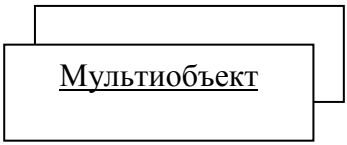


Рис. 58. Графическое изображение мультиобъекта

Объект может быть также составным, его изображение представлено на рис. 59.

Рис. 59. Графическое изображение составного объекта

Между объектами диаграммы сотрудничества существуют связи (links), по которым объекты посылают друг другу сообщения. Связи не имеют названий, но могут быть обозначены ключевыми словами, обозначающими тип связи:

- association – связь означает некую зависимость объектов;

- parameter – параметр; связь между объектами осуществляется посредством передачи параметров; значения параметров являются содержанием передаваемого сообщения;

- local – локальная переменная метода;

- global – глобальная переменная метода, область видимости которой – вся диаграмма.

Сообщения на диаграмме сотрудничества изображаются стрелками вдоль связей. Порядок передачи сообщений может быть указан явно в виде номера возле стрелки. Сообщения могут быть разного вида. Разные виды сообщений изображаются разными стрелками. Виды сообщений следующие:

⟶ сплошная линия с треугольной стрелкой – такое сообщение означает вызов процедуры (метода объекта) или вызов другого потока управления;

⟶ сплошная линия с обычной стрелкой, простое сообщение (простой поток управления или просто передача данных);

⟶ сплошная линия с полустрелкой; асинхронное сообщение, сообщение не имеет заранее обусловленного времени передачи;

--> пунктирная линия с обычной стрелкой, такое сообщение означает возврат значения из процедуры.

На рис. 60 приведен упрощенный пример диаграммы сотрудничества, иллюстрирующей расчеты с помощью чеков.

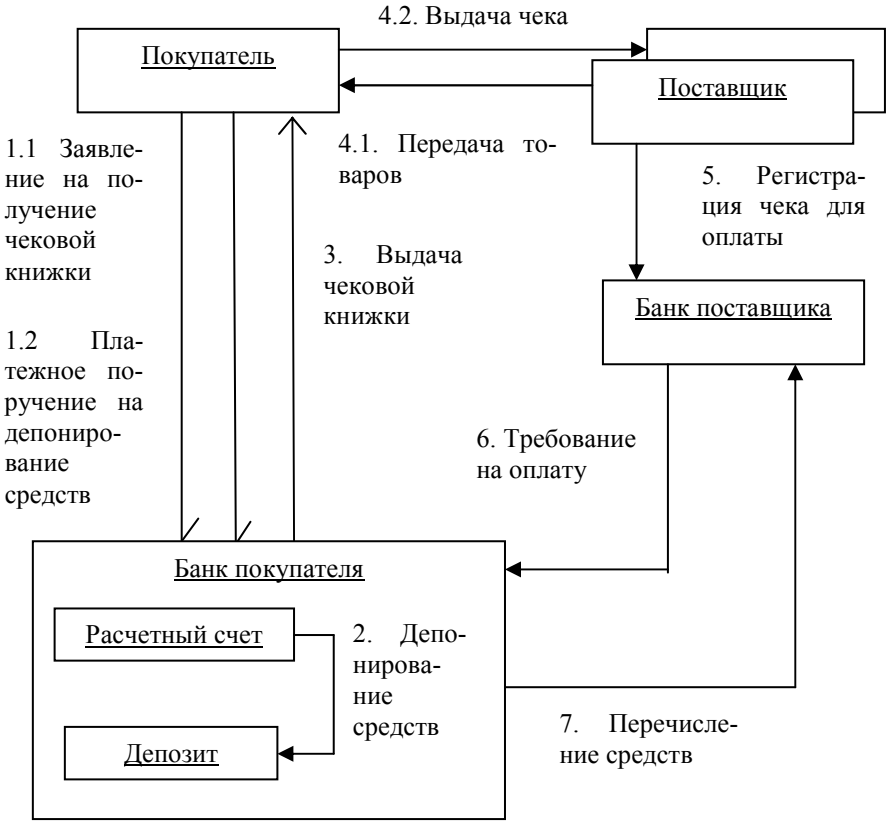


Рис. 60. Диаграмма сотрудничества

3.8. Диаграммы компонентов

Информационные системы на уровне программных кодов могут состоять из различных файлов: базы данных, приложения, справочных файлов, веб-документов, динамических библиотек, исходных текстов.

Диаграммы компонентов отображают взаимосвязи между файлами, а также распределение классов и их экземпляров по файлам [10].

Диаграммы компонентов играют существенную роль при оптимизации быстродействия системы. С их помощью выявляются наиболее используемые компоненты, определяется их распределение по модулям. Это существенные вопросы для систем с числом модулей от тысячи и выше.

Основным элементом диаграмм компонентов является компонент (component). Графически он изображается прямоугольником со встроенными слева прямоугольными секциями. Внутри прямоугольника указывается имя компонента. В качестве имени компонента могут быть указаны: имя исполняемого файла, имя файла базы данных. Может быть также указана служебная информация: версия, язык реализации, разработчик. Графическое изображение компонента показано на рис. 61.

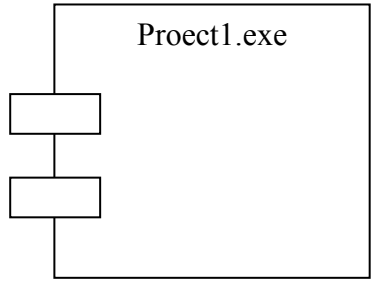


Рис. 61. Графическое изображение компонента

Иногда перед именем компонента указывают его спецификацию:

- library - библиотека;
- table – база данных, отдельная таблица;
- file – исходный текст программы;
- document - документ;
- executable – исполняемый файл.

Между компонентами на диаграмме компонентов существуют различные отношения.

Отношения зависимости, показывающие, что один компонент зависит от другого и при его изменении тоже меняется, изображаются пунктирной стрелкой (рис. 62).

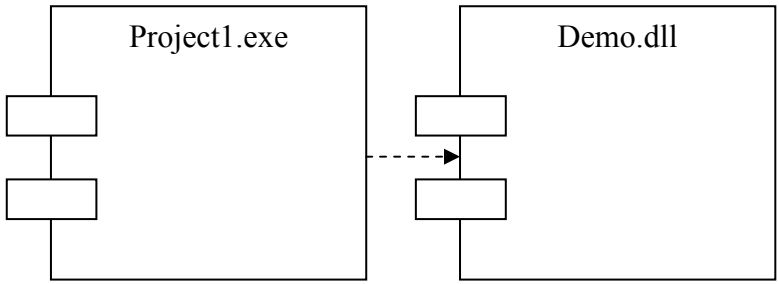


Рис. 62. Графическое изображение отношения зависимости

Отношения реализации изображаются сплошной линией или записываются в виде текста внутри компонента (рис. 63).

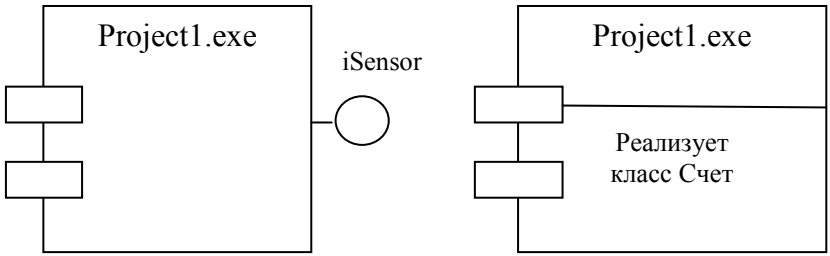


Рис. 63. Графическое изображение отношений реализации

3.9. Диаграммы развертывания

Диаграммы развертывания служат для иллюстрации физического размещения информационной системы на серверах, компьютерах пользователей, отображения физических каналов передачи информации между компонентами информационной системы.

Основным обозначением, используемым на диаграммах данного вида, является узел (node), который графически изображается проекцией куба (рис. 64).

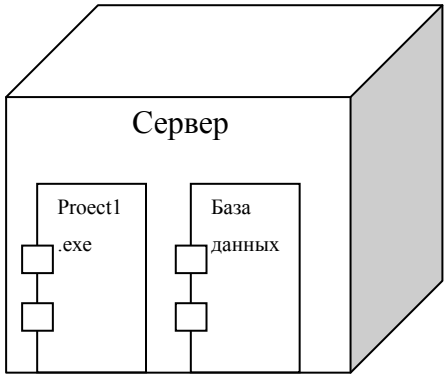


Рис. 64. Графическое изображение узла

Между узлами сплошными линиями показывают соединения, которые могут содержать пояснения, объясняющие характер реализации связи между компонентами (рис. 65).

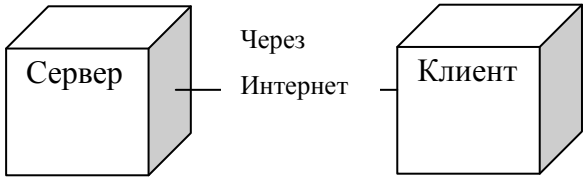


Рис. 65. Графическое изображение соединения

4. СОЗДАНИЕ ОБЪЕКТНОЙ МОДЕЛИ ДАННЫХ В ALLFUSION COMPONENT MODELER

AllFusion Component Modeler является мощным объектно-ориентированным инструментальным средством, позволяющим разрабатывать ИС разного назначения [10].

AllFusion Component Modeler поддерживает методологию CA Catalysis. Методология Catalysis основывается на стандарте объектного моделирования UML и специально ориентирована на технологию компонентной разработки.

AllFusion Component Modeler и Catalysis обеспечивают эффективные решения и минимальный риск при реализации крупномасштабных проектов, ориентированных на компонентную сборку.

AllFusion Component Modeler 4.1 призван обеспечить полный технологический цикл разработки крупных ИС.

4.1. Инструментальная среда AllFusion Component Modeler

AllFusion Component Modeler содержит несколько окон и инструментальных панелей:

- Workspace –навигатор модели;

- Property – окно (расположено слева внизу) отображает свойства элементов модели;

- Output – отображает отчеты о результатах поиска, замены и других действиях пользователя;

- Панель задач;

- Панель инструментов;

- Палитра инструментов (вид зависит от типа редактируемой диаграммы).

Каждое из окон можно скрыть или включить с помощью пункта меню View.

Панель задач (task bar) по умолчанию расположена в левой верхней части основного окна AllFusion Component Modeler. Панель задач содержит 4 раздела и позволяет организовать работу над проектом в соответствии с ролями разработчиков:

- Analyst (аналитик);

- Designer (дизайнер);

- Implementor (кодировщик);

- Reporting (последний раздел служит для генерации отчетов).

Для переключения между разделами панели задач следует щелкнуть по кнопке с названием раздела. Окно каждого раздела содержит список задач. Для создания новой задачи щелкают правой кнопкой мыши по окну панели задач и выбирают пункт меню Customize. При этом появляется диалоговое окно Task Bar.

Каждая задача представляет собой выполняемый скрипт, например на Visual Basic или Java. Для создания новой задачи необходимо щелкнуть по кнопке Create, внести в диалоговом окне имя задачи, затем щелкнуть по кнопке Browse и выбрать файл скрипта. Для включения задачи в раздел необходимо щелкнуть по кнопке Apply.

Окно Workspace имеет 3 вкладки – Models, Packages, Diagrams. Вкладка Models содержит древовидный список элементов модели, который является верхним уровнем представления

модели. Вкладка Packages содержит список пакетов. Пакет является нижним уровнем модели и может включать задачи и другие модели, используемые в проекте. Вкладка Diagrams содержит список диаграмм модели.

4.2. Диаграммы объектной модели

Модель проектируемой информационной системы представляет собой совокупность диаграмм, описывающих различные аспекты структуры и поведения ИС.

Для создания новой диаграммы следует перейти во вкладку Models окна Workspace и щелкнуть павой кнопкой мыши по модели, в которой создается новая диаграмма. В контекстном меню следует выбрать пункт New Diagram и затем тип диаграммы. AllFusion Component Modeler позволяет создавать диаграммы восьми типов:

- Activity (активности);
- Class (классов);
- Collaboration (кооперации);
- Sequence (последовательности);
- State (состояний);
- Use Case (вариантов использования);
- Component (компонентов);
- Deployment (развертывания).

После выбора нужного типа диаграммы в правой верхней части основного окна появится окно редактора диаграмм Diagram Editor).

В левой части редактора диаграмм Diagram Editor размещается палитра инструментов, содержание которой зависит от типа диаграммы. С помощью кнопок палитры инструментов в диаграмму можно внести элементы модели, описывающие структуру и поведение системы, а также связи между ними.

Для просмотра диаграмм модели служит вкладка Diagrams окна Workspace.

Для просмотра диаграммы необходимо правой кнопкой мыши щелкнуть по иконке диаграммы в списке и выбрать пункт меню Open Document.

4.3. Диаграммы вариантов использования

Диаграммы вариантов использования (Use Case) показывают, какие функции выполняет проектируемая система, с какими внешними системами (actors) взаимодействует.

Внешние системы (воздействующие объекты) – это конечные пользователи или другие программы, взаимодействующие с проектируемой ИС.

Вариант использования (use case) – это последовательность действий, выполняемых системой, которые приводят к определенным результатам.

Диаграммы Use Case включают связи между воздействующими объектами и вариантами использования. Связь изображается в виде стрелок.

Для внесения элементов (вариантов использования и объектов) в диаграмму и установления связей между ними используют кнопки панели инструментов. Для создания связи между элементами диаграммы необходимо щелкнуть по кнопке связи нужного типа в палитре инструментов, затем по краю первого элемента и по краю второго элемента. Связь будет установлена от первого ко второму элементу. Внести новый элемент в диаграмму можно, переместив его из палитры методом drag & drop.

Кнопки панели инструментов для создания элементов диаграммы Use Case имеют следующее назначение:

1 (Actor) – воздействующий объект (actor);

2 (UseCase) – вариант использования (use case);

3 (Comment) – примечание;

4 (Class) – класс;

5 (Interface) – интерфейс, именованное множество операций, описывающее поведение элемента;

6 (SubSystem) – подсистема; подсистемой может быть другая диаграмма;

7 (Collaboration) – ссылка на диаграмму кооперации (Collaboration);

8 (Association) – ассоциативная связь (связь между вариантом использования (use case) и воздействующим объектом (actor));

9 (Usage) – использование (показывает, что один вариант использования при своем выполнении обязательно использует другой);

10 (Includes) – отношение между базовым включаемым вариантом использования; базовый вариант использования может зависеть от результатов включаемого варианта использования;

11 (Extends) - расширение; один вариант использования может использовать (а может и нет) другой вариант использования;

12 (Generalization) – обобщения; показывает, что у нескольких элементов диаграммы имеются общие черты;

13 (Dependency) – отношение зависимости; показывает, что один элемент зависит от другого;

14 (Abstraction) – абстракция; показывает зависимость между двумя элементами, которые имеют разный уровень абстракции;

15 (Comments) – указывает связь между элементом модели и примечанием.

4.4. Диаграммы классов

Под объектом в UML понимается некоторое абстрактное представление конкретного объекта предметной области. Каждый объект имеет состояние, поведение и индивидуальность.

Например, объект «Проект» может иметь два состояния – «открыт» и «закрыт». Поведение объекта определяет, как объект взаимодействует с другими объектами. Индивидуальность означает, что каждый объект уникален и отличается от других объектов. Под классом понимается описание объектов, обладающих общими свойствами (атрибутами), поведением, общими взаимоотношениями с другими объектами и общей семантикой.

Каждый класс может иметь атрибуты (свойства). Например, класс «Клиент» может иметь атрибуты: фамилия, имя, отчество, место выдачи водительского удостоверения, номер водительского удостоверения. Кроме того, каждый класс может выполнять методы (operation) – некоторые действия, которые описывают поведение объектов класса.

Для определения свойств класса следует правой кнопкой мыши щелкнуть по классу и выбрать во всплывающем меню пункт Create.

Классы могут иметь взаимосвязи (relationship), называемые отношениями. В нотации UML имеется несколько типов отношений. Отношение использования (associations) показывает, что объект одного класса связан с одним или несколькими объектами другого класса. Отношение включения (aggregation) является частным случаем отношения использования. Оно показывает, что один объект является частью другого. При воздействии на один объект, связанный отношением включения, некоторые операции автоматически могут затронуть другой объект. Каждая связь может быть охарактеризована определенной фразой, называемой именем роли. Для создания имени связи следует щелкнуть по ней правой кнопкой мыши и выбрать в меню пункт Properties. В появившемся диалоговом окне Shape Properties во вкладке Label можно внести имя связи.

Каждая связь может иметь индикатор множественности, который показывает, сколько объектов одного класса соответ-

ствует объекту другого класса, например 1..* (один-ко-многим).

Наследование (inheritance) описывает взаимосвязь между классами, когда один класс (называется подклассом, subclass) наследует структуру и/или поведение одного или нескольких классов. Связь классов в иерархии наследования называется отношением наследования (generalization).

Кнопки панели инструментов для создания элементов диаграммы классов имеют следующее назначение:

1 (Class) – класс;

2 (Package) – пакет;

3 (Interface Class) – интерфейс класса; это именованное множество операций, описывающее поведение класса;

4 (Object) – объект (экземпляр класса);

5 (Data Type) – тип данных; тип может быть предопределенным (например, число или строка) и определяемым пользователем;

6 (Signal) – сигнал, описывает коммуникации между объектами;

7 (Exception) – исключение; это сигнал, который возникает как ответ на ошибку;

8 (Comment) – примечание;

9 (Generalization) – обобщения, показывает, что у некоторых элементов есть общие черты;

10 (Link) - связь;

11 (Association) – ассоциация;

12 (Binding) – связывание; приписывает значение параметру;

13 (Permission) – разрешение; обозначает зависимость между элементами, когда один элемент использует содержимое другого элемента;

14 (Abstraction) – абстракция; зависимость между двумя элементами, имеющими разные уровни абстракции;

15 (Dependency) – отношение зависимости; показывает, что один элемент зависит от другого;

16 (Usage) – использование; показывает, что один элемент при своем выполнении обязательно использует другой;

17 (XOR Const)– исключающее «или»; в случае, когда несколько ассоциаций связаны с одним и тем же классом, показывает, что объект может быть связан только с одной ассоциацией;

18 (Comments) – связь между элементом модели и примечанием.

4.5. Диаграммы последовательности

Язык UML включает диаграммы, описывающие поведение модели в динамике. К диаграммам такого типа относятся диаграммы последовательности (Sequence). На диаграмме последовательности события показываются так, как они происходят во времени.

В верхней части диаграммы последовательности располагаются объекты (экземпляры класса). Диаграмма разделена на вертикальные полосы (линии жизни, lifeline), каждая из которых соответствует одному объекту. Ось времени направлена сверху вниз. На диаграмме указывают периоды активности для каждого объекта, указывают сообщения, которыми обмениваются объекты.

Панель инструментов для создания элементов диаграммы последовательности имеет следующие кнопки:

1 (Actor) – воздействующий объект (actor);

2 (Object) – объект (экземпляр класса);

3 (Comment) – примечание;

4 (Call) – вызов; метод одного класса вызывает операцию другого класса;

5 (Send) – отправка; передача сигнала от одного элемента к другому;

6 (Return) – возвращение; возвращение сигнала;

7 9Create) – создание; классификатор-клиент создает экземпляр классификатора-поставщика;

8 (Destroy) – уничтожение; классификатор-клиент уничтожает экземпляр классификатора-поставщика;

9 (Uninterpreted) – неинтерпретируемое значение, реализация которого не определяется языком UML;

10 (Comments) – связь между элементом модели и примечанием.

4.6. Диаграммы активности

Диаграмма активности (activity) описывает динамическое поведение элементов модели и представляет собой описание потока работ (графа деятельности).

Диаграмма активности может быть разбита на «плавательные дорожки» (SwimLine). Каждая «плавательная дорожка» имеет собственное имя и служит для группировки элементов (состояний), которые реализует один объект.

При создании диаграммы активности особое внимание уделяется последовательным и параллельным шагам выполнения работ. Для описания ветвления работ или управления диаграммы активности содержат специальные элементы.

Панель инструментов для создания диаграммы активности имеет следующие кнопки:

1 (swim lane) – «плавательная дорожка»;

2 (action state) – состояние действия; это элементарное (не имеющее внутренней структуры) действие, которое переводит объект в следующее состояние;

3 (Subactivity) – подсостояние; состояние, которое имеет внутреннюю структуру; может состоять из параллельных и/или последовательных простых состояний;

4 (Object Flow) – состояние потока объектов; состояние, описывающее наличие объекта в данной точке графа деятельности;

5 (Initial State) – исходное состояние; стартовая точка графа деятельности;

6 (Final State) – конечное состояние; финишная точка графа деятельности;

7 (Decision) – разветвление; показывает, что работы могут выполняться параллельно;

8 (Join) – слияние управления; показывает слияние альтернативных путей выполнения работ;

9 (Fork) – разветвление управления; показывает альтернативные пути выполнения работ;

10 (Choice) – выбор; элемент, предназначенный для декомпозиции состояния;

11 (Comment) – примечание;

12 (Transition) – переход; отношение между двумя состояниями, когда одно состояние переходит в другое;

13 (Comments) – связь между элементом модели и примечанием.

4.7. Диаграммы состояний

Диаграмма состояний (State) описывает динамическое поведение объектов. Основным элементом диаграммы является состояние (State) – множество объектов класса, которые реагируют на внешнее воздействие одинаковым образом. Состояние может иметь имя и описывает некоторый период времени жизненного цикла объекта. Состояния могут быть связаны переходами (transition), которые представляют собой реакцию объекта на событие. Граф состояний и переходов из одного состояния в другое называется конечным автоматом.

Диаграмма состояний создается с помощью следующих кнопок панели инструментов:

1 (Simple State) – простое состояние; элементарное (не имеющее внутренней структуры состояние);

2 (Composite) – композитное состояние; состояние, в котором могут содержаться несколько последовательных или параллельных подсостояний;

3 (Region) – область; служит для выделения группы элементов на диаграмме состояний. Для выделения группы элементов надо щелкнуть по кнопке Region, затем перенести маркер в окно диаграммы, нажать на кнопку мыши и тащить маркер, захватив в область выделения необходимые элементы;

4 (Synch State) – состояние синхронизации; позволяет синхронизировать управление между параллельными областями конечного автомата;

5 (Stub State) – состояние заглушка; ссылается на определенное состояние во вложенном автомате;

6 (Initial State) – исходное состояние;

7 (Final State) – конечное состояние;

8 (Join) – слияние управления; показывает слияние альтернативных путей выполнения работ;

9 (Fork) – разветвление управления; показывает альтернативные пути выполнения работ;

10 (Choice) – выбор; элемент, предназначенный для декомпозиции состояний;

11 (Shallow History) – кратковременное историческое состояние; позволяет запомнить подсостояние композитного состояния, которое было активно непосредственно перед выходом из него;

12 (Deep History) – долговременное историческое состояние; позволяет запомнить подсостояния композитного состояния на определенную глубину;

13 (Comment) – примечание;

14 (Sub Machine) – состояние, описываемое ссылкой на вложенный автомат, который неявно подставляется на то место, откуда производится ссылка;

15 (Transition) – переход; отношение между двумя состояниями, когда одно состояние переходит в другое;

16 (Comments) – связь между элементом модели и примечанием.

4.8. Диаграммы кооперации

На диаграмме кооперации отражаются события и потоки данных. Корпоративные диаграммы (Collaboration) показывают группы элементов модели и связи, взаимодействие которых приводит к какой-либо цели (например, резервирование билетов, покупка квартиры, оформление кредита и т.д.).

С помощью кнопок панели инструментов можно создать следующие элементы на диаграмме кооперации:

1 (Actor) – воздействующий объект (actor);

2 (Class) – класс;

3 (Object) – объект (экземпляр класса);

4 (Classifier Role) – классификатор; описывает роль, которую играет в кооперации элемент диаграммы;

5 (Use Case) – вариант использования (use case);

6 (Interface) – интерфейс;

7 (Comment) – примечание;

8 (Association …) – роль в ассоциации; связь двух ролей классификаторов в кооперативной диаграмме;

9 (Link) – связь; экземпляр ассоциации или роли ассоциации;

10 (Association) – ассоциация; связь между двумя классификаторами;

11 (Generalization) – обобщение; показывает, что у нескольких классификаторов имеются общие черты;

12 (Extends) – расширение; один вариант применения при своем выполнении может использовать ругой вариант;

13 (Includes) – отношение между базовым и включаемым вариантом использования; базовый вариант использова-

ния может зависеть от результатов работы включаемого варианта использования;

14 (Abstraction) – абстракция; зависимость между двумя элементами, представляющими одну и ту же концепцию на разных уровнях абстракции;

15 (Comments) – связь между элементом модели и примечанием.

4.9. Диаграммы компонентов и диаграммы развертывания

Архитектура приложения описывается в диаграммах компонентов (Component) и диаграммах развертывания (Deployment).

На диаграммах компонентов изображается вхождение классов и объектов в программные компоненты системы (модули, библиотеки и т.д.).

При помощи диаграмм развертывания документируется размещение программных модулей на физических и логических устройствах проектируемой системы.

Диаграмма компонентов показывает используемые типы компонентов, а диаграмма развертывания – экземпляры компонентов.

Диаграмма компонентов создается с помощью следующих кнопок панели инструментов:

1 (Component) – компонент; независимая от других компонентов часть системы; взаимодействие с другими компонентами осуществляется с помощью интерфейса;

2 (Interface) – интерфейс; определяется перечень операций, реализуемый программным и аппаратным обеспечением;

3 (Package) – пакет; часть модели; каждый элемент может принадлежать только одному пакету;

4 (Comment) – примечание;

5 (Generalization) – обобщение; показывает, что у нескольких классификаторов имеются общие черты;

6 (Realization) – отношение между спецификацией и программной реализацией:

7 (Association) – ассоциация;

8 (Dependency) – отношение зависимости; показывает, что один элемент зависит от другого;

9 (Usage) – использование; показывает, что один классификатор при своем выполнении обязательно использует другой;

10 (Comments) – связь между элементом модели и примечанием.

Диаграмма развертывания создается с помощью следующих кнопок панели инструментов:

1 (Component) - компонент; независимая от других компонентов часть системы; взаимодействие с другими компонентами осуществляется с помощью интерфейса;

2 (Component …) – экземпляр компонента; реализация компонента в конкретный момент времени;

3 (Node) – узел (вычислительный ресурс системы); на узлах могут находиться объекты и компоненты;

4 (Node …) – экземпляр узла; реализация узла в конкретный момент времени;

5 (Interface) – интерфейс; определяется перечень операций, реализуемый программным и аппаратным обеспечением;

6 (Object) – объект (экземпляр класса);

7 (Package) – пакет (часть модели); каждый элемент может принадлежать только одному пакету;

8 (Comment) – примечание;

9 (Node Insta…) – отношение между экземплярами узлов;

10 (Communication) – коммуникационная ассоциация; ассоциация между узлами, осуществляющими коммуникации;

11 (Communication …) – коммуникативная связь; связь между экземплярами узлов, осуществляющими коммуникации;

12 (Flow) – поток; показывает, что две версии одного и того же объекта существуют в разные моменты времени;

13 (Dependency) – отношение зависимости; показывает, что один элемент зависит от другого;

14 (Abstraction) – абстракция; зависимость между двумя элементами, представляющими одну и ту же концепцию на разных уровнях абстракции;

15 (Comments) – связь между элементом модели и примечанием.

5. СОЗДАНИЕ МОДЕЛЕЙ ПРОЦЕССОВ

5.1. Методологии проектирования BPwin

Для проведения анализа и реорганизации бизнес-процессов фирма Computer Associates предлагает CASE-средства верхнего уровня BPwin, поддерживающее методологию IDEF0 (функциональная модель), IDEF3 (WorkFlow Diagram), DFD (DataFlow Diagram).

Функциональная модель предназначена для описания существующих бизнес-процессов на предприятии (так называемая модель AS-IS) и идеального положения вещей (модель TO-BE). Методология IDEF0 предписывает построение иерархической системы диаграмм для описания 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). Декомпозиция может быть продолжена далее. После каждого сеанса декомпозиции проводится сеанс экспертизы: каждая диаграмма проверяется экспертами предметной области, представителями заказчика, людьми, непосредственно участвующими в бизнес-процессе. Такая технология создания модели позволяет построить модель, адекватную предметной области.

Модель DFD используется для моделирования документооборота.

Методология IDEF3 позволяет описать логику взаимодействия компонентов системы.

В IDEF0 система представляется как совокупность взаимодействующих работ или функций. Взаимодействие системы с окружающим миром описывается как **вход** (нечто, что перерабатывается системой), **выход** (результат деятельности системы), **управление** (стратегии и процедуры, под управлением которых производится работа) и **механизм** (ресурсы, необходимые для проведения работы). Находясь под управлением, система преобразует входы в выходы, используя механизмы.

Диаграммы IDEF0. Основу методологии IDEF0 составляет графический язык описания бизнес-процессов. Модель в нотации IDEF0 представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Модель может содержать следующие основные диаграмм:

- контекстную диаграмму (в каждой модели может быть только одна контекстная диаграмма);
- диаграммы декомпозиции;
- диаграммы дерева узлов.

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и ее взаимодействия с внешней средой.

После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией.

Диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются **диаграммами декомпозиции**. После декомпозиции контекстной диаграммы

проводится декомпозиция каждого большого фрагмента системы на более мелкие и так далее, до достижения нужного уровня подробности описания. После каждого сеанса декомпозиции проводятся сеансы экспертизы. Эксперты предметной области указывают на несоответствие реальных бизнес-процессов созданным диаграммам. Найденные несоответствия исправляются.

Диаграмма дерева узлов показывает иерархическую зависимость работ, при этом взаимосвязи между работами не отражаются. Диаграмм дерева узлов может быть в модели сколь угодно много, поскольку дерево может быть построено на произвольную глубину.

5.2. Создание контекстной диаграммы

Запуск **BPwin** осуществляется стандартным образом. После запуска появляется основное окно, содержащее главное меню, панели инструментов и в левой части навигатор модели – **Model Explorer**.

Для создания новой модели выбирают пункты меню **File/New**. В строке **Name** задают имя работы, изображающей систему в целом, выбирают тип диаграммы (IDEF0, IDEF3 или DFD) и щелкают по кнопке **ОК** (рис. 66).

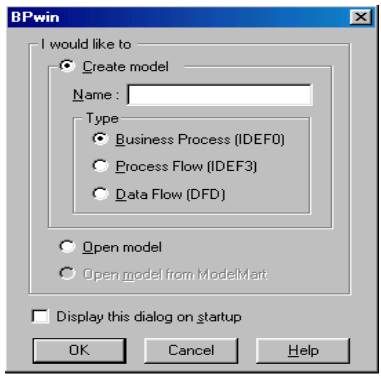


Рис. 66. Диалоговое окно создания новой модели

Работы обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Работы изображаются в виде прямоугольников. Все работы должны быть названы и определены.

Имя работы должно быть выражено отглагольным существительным, обозначающим действие (например, «Разработка информационного обеспечения»). При создании новой модели автоматически создается контекстная диаграмма с единственной работой, изображающей систему в целом.

Для внесения имени работы следует вызвать контекстное меню, выбрать пункт **Name**… и в появившемся диалоге внести имя работы.

Для описания других характеристик контекстной диаграммы в контекстном меню выбирают пункт **Model Properties**…. В закладке **General** задают имена проекта, модели, авторов. В закладке Purpose определяют цель моделирования (например, «Описать функциональность процесса разработки программного продукта с целью спецификации информационной системы»). Здесь же указывается точка зрения человека, который видит систему в нужном для моделирования аспекте. В закладке **Status** можно описать статус модели (черновой, рабочий, рекомендуемый и т.д.), время создания и последнего редактирования. В закладке **Source** описывают источники информации для построения модели (например, «Опрос экспертов предметной области и анализ документации»). В закладке **Definition** дают определение модели и описание области применения.

5.3. Создание диаграммы декомпозиции

Диаграммы декомпозиции содержат родственные работы, т.е. дочерние работы, имеющие общую родительскую работу.

Для создания диаграммы декомпозиции следует щелкнуть по кнопке .

Возникает диалог **Activity Box Count**, в котором следует указать нотацию новой диаграммы (IDEF0, DFD, IDEF3) и количество работ на ней. Обычно выбирают нотацию IDEF0 и щелкают по кнопке OK. Появляется диаграмма декомпозиции (рис. 67).

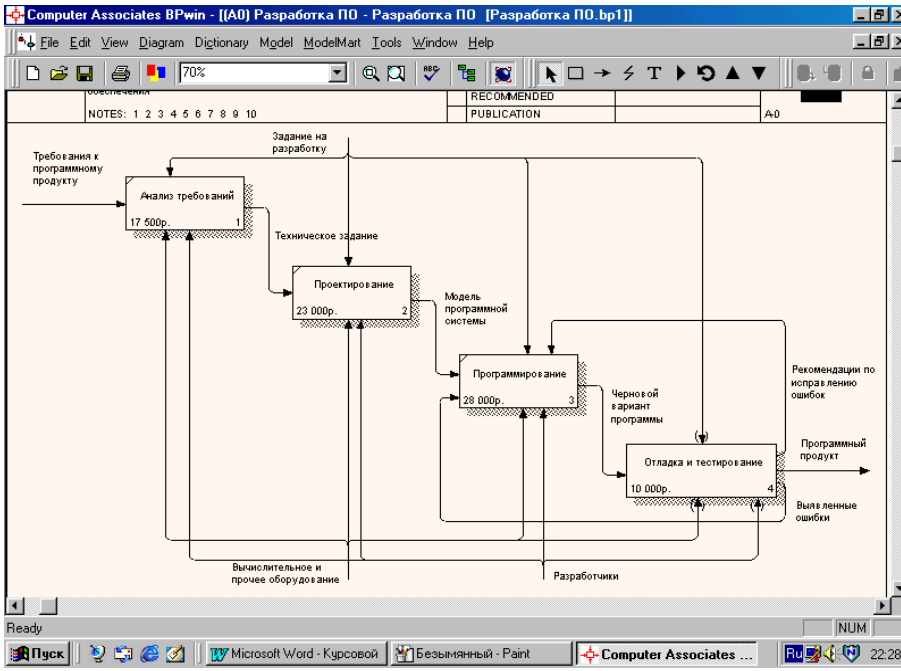


Рис. 67. Пример диаграммы декомпозиции

Допустимое количество работ 2-8. Для обеспечения наглядности и лучшего понимания моделируемых процессов рекомендуется использовать от трех до шести блоков на одной диаграмме. Работы на диаграммах декомпозиции обычно рас-

полагаются по диагонали от левого верхнего к правому нижнему. Такой порядок называется порядком доминирования. Согласно этому принципу расположения в левом верхнем углу располагается самая важная работа или работа, выполняемая по времени первой. Далее вправо вниз располагаются менее важные или выполняемые позже работы. Каждая из работ на диаграмме декомпозиции может быть в свою очередь декомпозирована (выделить работу и нажать по кнопке декомпозиции). На диаграмме декомпозиции работы нумеруются автоматически слева направо. Номер работы показывается в правом нижнем углу. В левом верхнем углу изображается небольшая диагональная черта, которая показывает, что данная работа не была декомпозирована.

5.4. Создание стрелок

Взаимодействие работ с внешним миром и между собой описывается в виде стрелок. **Стрелки** представляют собой некую информацию и именуются существительными (например, «Разработчики», «Программный продукт»).

В **IDEF0** различают пять типов стрелок.

Вход (Input) – материал или информация, которые используются или преобразуются работой для получения результата (выхода). Допускается, что работа может не иметь ни одной стрелки входа. Каждый тип стрелок подходит к определенной стороне прямоугольника, изображающего работу, или выходит из нее. Стрелка входа рисуется как входящая в левую грань работы.

Управление (Control) – правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления. Стрелка управления рисуется как входящая в верхнюю грань работы. Управление влияет на работу но не преобразуется работой.

Выход (Output) – материал или информация, которые производятся работой. Каждая работа имеет хотя бы одну

стрелку выхода. Работа без результата не имеет смысла и не должна моделироваться. Стрелка выхода рисуется как исходящая из правой грани работы.

Механизм (Mechanism) – ресурсы, которые выполняют работу, например персонал предприятия, станки, устройства и т.д. Стрелка механизма рисуется как входящая в нижнюю грань работы.

Вызов (Call) – специальная стрелка, указывающая на другую модель работы. Данная стрелка рисуется как исходящая из нижней грани работы. Стрелка вызова используется для указания того, что некоторая работа выполняется за пределами моделируемой системы.

Граничные стрелки. Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. Они могут начинаться у границы диаграммы и заканчиваться у работы, или наоборот. Такие стрелки называются граничными.

Создание граничных стрелок (стрелка входа):

- щелкнуть по кнопке с символом стрелки  в палитре инструментов, перенести курсор к левой стороне экрана пока не появится граничная темная полоса;

- щелкнуть один раз по полосе (откуда выходит стрелка) и еще раз в левой части работы со стороны входа (где заканчивается стрелка);

- выбрать в палитре инструментов  кнопку редактирования;

Для добавления имени стрелки щелкнуть правой кнопкой мыши по линии стрелки и выбрать пункт меню Name…, в закладке Name в строке Arrow Name задать имя стрелки.

Стрелки управления, выхода и механизма изображаются аналогично.

Несвязные граничные стрелки. При декомпозиции работы входящие в нее и исходящие из нее стрелки автоматически появляются на диаграмме декомпозиции, но при этом не

касаются работ. Для связывания стрелок входа, управления или механизма с работой необходимо перейти в режим редактирования стрелок, щелкнуть по наконечнику стрелки и по соответствующей стороне работы. Для связывания стрелки выхода необходимо перейти в режим редактирования стрелок, щелкнуть по стороне выхода работы, а затем по стрелке.

Внутренние стрелки. Для связи работ между собой используются внутренние стрелки, которые не касаются границ диаграммы и начинаются у одной и кончаются у другой работы.

Для рисования внутренней стрелки необходимо в режиме рисования стрелок щелкнуть по стороне (например, выхода) одной работы и затем по стороне (например, входа) другой.

В **IDEF0** различают пять типов связей работ.

Связь по входу, когда стрелка выхода вышестоящей работы направляется на вход нижестоящей.

Связь по управлению, когда выход вышестоящей работы направляется на управление нижестоящей.

Обратная связь по входу, когда выход нижестоящей работы направляется на вход вышестоящей. Такая связь, как правило, используется для описания циклов.

Обратная связь по управлению, когда выход нижестоящей работы направляется на управление вышестоящей.

Связь выход-механизм, когда выход одной работы направляется на механизм другой. Такая связь показывает, что одна работа подготавливает ресурсы, необходимые для проведения другой работы.

Разветвляющиеся и сливающиеся стрелки. Одни и те же данные или объекты, порожденные одной работой, могут использоваться сразу в нескольких других работах, а с другой стороны данные, порожденные в разных работах могут перерабатываться в одном месте. Для моделирования таких ситуаций используются разветвляющиеся и сливающиеся стрелки. Для разветвления стрелки нужно в режиме редактирования стрелки

щелкнуть по фрагменту стрелки и по соответствующей стороне работы. Для слияния двух стрелок выхода нужно в режиме редактирования стрелки сначала щелкнуть по стороне выхода работы, а затем по соответствующему фрагменту стрелки.

Тоннелирование стрелок. Вновь внесенные граничные стрелки на диаграмме декомпозиции нижнего уровня изображаются в квадратных скобках и автоматически не появляются на диаграмме верхнего уровня.

Для их перетаскивания необходимо щелкнуть правой кнопкой мыши по участку стрелки в квадратных скобках и в контекстном меню выбрать пункт Arrow Tunnel… и в появившемся диалоговом окне Border Arrow Editor щелкнуть ОК. Если в окне был выбран переключатель Resolve it to border arrow, то данная стрелка будет отображена и на верхнем уровне. Если в окне был выбран переключатель Change it to resolved rounded tunnel, то стрелка будет затоннелирована и не попадет на другую диаграмму (такое тоннелирование называется «не-в-родительской-диаграмме»).

Другим примером тоннелирования может быть ситуация, когда стрелка механизма мигрирует с верхнего уровня на нижний. В этом случае стрелка механизма на нижнем уровне может быть удалена, после чего на родительской диаграмме она может быть затоннелирована. Такое тоннелирование называется «не-в-дочерней-работе». Действия в данном случае аналогичны.

5.5. Диаграммы дерева узлов

Диаграмма дерева узлов показывает иерархию работ модели и позволяет рассмотреть всю модель целиком, но не показывает взаимосвязи между работами (стрелки).

Для создания диаграммы дерева узлов следует выбрать в меню пункты Diagram/Add Node Tree…. Возникает диалоговое окно формирования дерева узлов Node Tree Wizard. В данном окне указывают имя диаграммы, глубину дерева Number of

levels (по умолчанию 3) и корень дерева (по умолчанию родительская работа текущей диаграммы). После нажатия кнопки Готово в отдельном окне появится диаграмма дерева узлов (рис. 68).

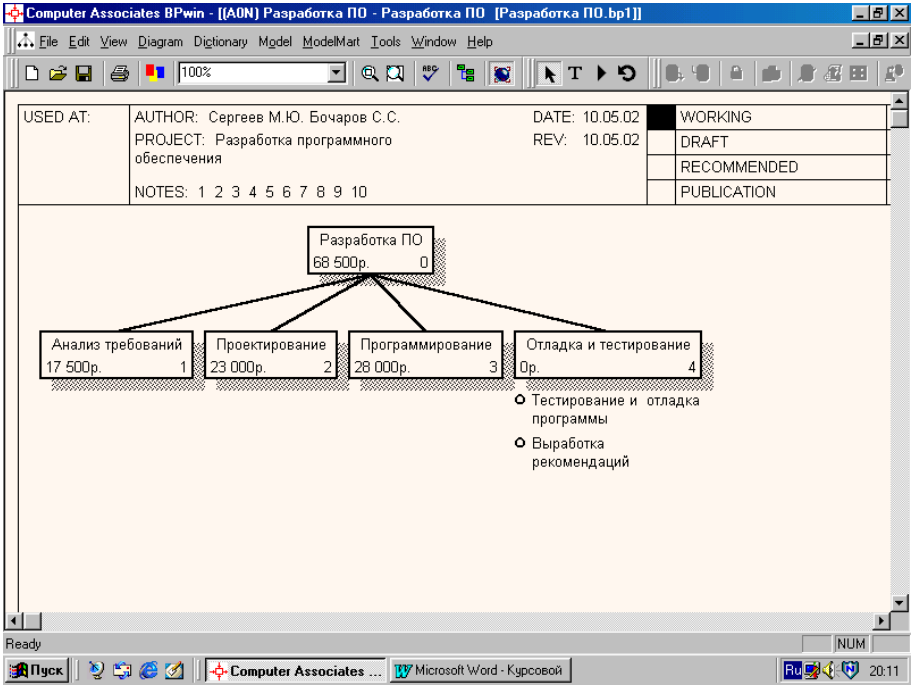


Рис. 68. Диаграмма дерева узлов

5.6. Диаграммы потоков данных DFD

Диаграммы потоков данных (Data Flow Diagramming, DFD) используются для описания документооборота и обработки информации. Подобно IDEF0, DFD представляет модельную систему как сеть связанных между собой работ. Их можно использовать как дополнение к модели IDEF0, для бо-

лее наглядного отображения текущих операций документооборота в корпоративных системах обработки информации. DFD описывает:

- функции обработки информации (работы);
- документы (стрелки), объекты, сотрудников или отделы, которые участвуют в обработке информации;
- внешние ссылки, которые обеспечивают интерфейс с внешними объектами, находящимися за границами моделируемой системы;
- таблицы для хранения документов (хранилище данных).

Работы. В DFD работы представляют функции системы, преобразующие входы в выходы. На диаграмме изображаются прямоугольниками со скругленными углами. В отличие от IDEF0 работы не имеют управления и механизмов

Внешние сущности. Изображают входы в систему и/или выходы из системы. Внешние сущности изображаются в виде прямоугольника с тенью и обычно располагаются по краям диаграммы.

Стрелки (потоки данных). Описывают движения объектов из одной части системы в другую. Поскольку DFD каждая сторона работы не имеет четкого назначения как в IDEF0, стрелки могут подходить и выходить из любой грани прямоугольника работы. В DFD также применяются двунаправленные стрелки для описания диалогов типа «команда-ответ» между работой и внешней сущностью, между работами и между внешними сущностями.

Хранилища данных. В отличие от стрелок, описывающих объекты в движении, хранилища данных изображают объекты в покое. В системах обработки информации хранилища данных являются механизмом, который позволяет сохранить данные для последующих процессов.

Слияние и разветвление стрелок. В DFD стрелки могут сливаться и разветвляться, что позволяет описать декомпо-

зицию стрелок. Каждый новый сегмент сливающейся стрелки может иметь собственное имя.

Построение диаграмм DFD. Для того чтобы дополнить модель IDEF0 диаграммой DFD, нужно выделить работу и нажать кнопку декомпозиции. В появившемся диалоге Activity Box Count активизировать переключатель DFD. Создается новая диаграмма DFD, и стрелки, которые касаются родительской работы, мигрируют на диаграмму DFD. В палитре инструментов на новой диаграмме DFD появляются новые кнопки.

Кнопка добавления в диаграмму внешней ссылки.  Внешняя ссылка является источником или приемником данных извне модели.

Кнопка добавления в диаграмму хранилища данных.  Хранилище данных позволяет описать данные, которые необходимо сохранить в памяти прежде, чем использовать в работах.

На следующем этапе необходимо добавить в диаграмму внешние сущности и хранилища данных, присоединить необходимые граничные стрелки, ненужные граничные стрелки удалить и затоннелировать их в родительской диаграмме, создать внутренние стрелки между объектами диаграммы.

Пример DFD диаграммы представлен на рис. 69.

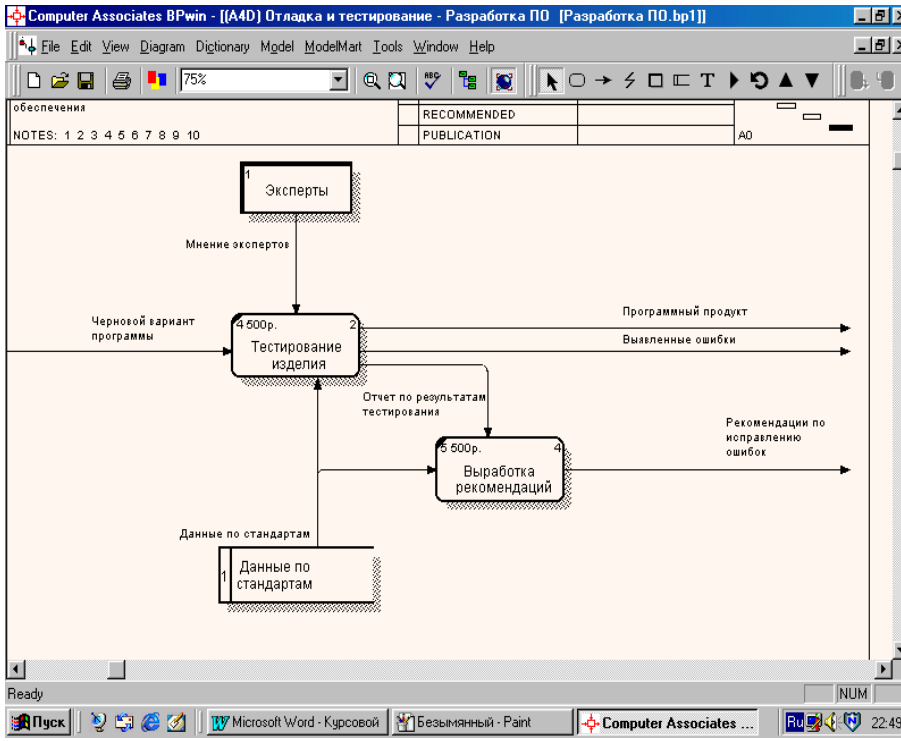


Рис. 69. Пример DFD диаграммы

6. СОЗДАНИЕ МОДЕЛИ ДАННЫХ

6.1. Отображение модели данных в ERwin

ERwin имеет два уровня представления данных – логический и физический. **Логический уровень –** это абстрактный взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире (например, «отдел» или «фамилия сотрудника»). Объекты модели, представленные на логическом уровне, называются сущностями и атрибутами. Логическая мо-

дель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных зависит от конкретной СУБД, физически являясь отображением системного каталога. Физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей.

Разделение модели данных на логические и физические позволяет решить несколько важных задач.

Документирование модели. Многие СУБД имеют ограничения на именование объектов (например, ограничения на длину имени таблицы, поля, запрет на использование специальных символов в имени). На физическом уровне объекты базы данных могут называться так, как того требует ограничение СУБД. На логическом уровне можно этим объектам дать синонимы – имена более понятные неспециалистам. Такое соответствие позволяет лучше задокументировать модель и дает возможность обсуждать структуру данных с экспертами предметной области.

Масштабирование. Создание модели данных, как правило, начинается с создания логической модели. После описания логической модели проектировщик может выбрать необходимую СУБД и ERwin автоматически создаст соответствующую физическую модель. На основе физической модели ERwin может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Этот процесс называется прямым проектированием. Тем самым достигается масштабируемость – создав одну логическую модель данных, можно сгенерировать физические модели под любую поддерживаемую ERwin СУБД. С другой стороны, ERwin способен по содержимому системного каталога или SQL-скрипту воссоздать физическую или логическую модель данных. На основе полученной логической модели данных можно сгенерировать физическую модель для другой СУБД и затем сгенерировать ее системный каталог. Следо-

вательно, ERwin позволяет решить задачу по переносу структуры данных с одного сервера на другой.

6.2. Создание логической модели данных

Запуск ERwin осуществляется стандартным образом. При запуске появляется меню, панели инструментов, в левой части навигатор модели – Model Explorer.

Этапы создания логической модели данных.

Создание новой модели. Для создания логической модели данных выбирают пункты меню File/ New…, в появившемся диалоговом окне активизировать переключатель **Logical/Physical**, в поле **Database** выбрать из списка предполагаемую СУБД и в поле **Version** – ее версию, после чего нажать кнопку **ОК**.

По умолчанию исходная модель получает имя **Main Subject Area**. На следующем этапе создают логическую диаграмму ERwin. Основные компоненты диаграммы – сущности, атрибуты и связи. Каждая сущность является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает определенные свойства объекта.

Построение модели данных предполагает определение сущностей и атрибутов, т.е. необходимо определить, какая информация будет храниться в конкретной сущности или атрибуте. Сущность можно определить как объект, событие или концепцию, информация о которых должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе.

Внесение сущностей в модель. Для внесения сущности в модель необходимо щелкнуть по кнопке  , затем щелкнуть по тому месту на диаграмме, где необходимо расположить новую сущность. Щелкнув правой кнопкой по

сущности, выбирают в контекстном меню пункт **Entity Properties…**. В появившемся диалоговом окне определяют имя (**Name**) и описание (**Definition**) сущности.

Описание атрибутов сущности. Как было указано выше, каждый атрибут хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называют первичным ключом. Для описания атрибутов следует, щелкнув правой кнопкой мыши по сущности, выбрать в появившемся контекстном меню пункт **Attributes**…. Появится диалог **Attributes** (рис. 70).

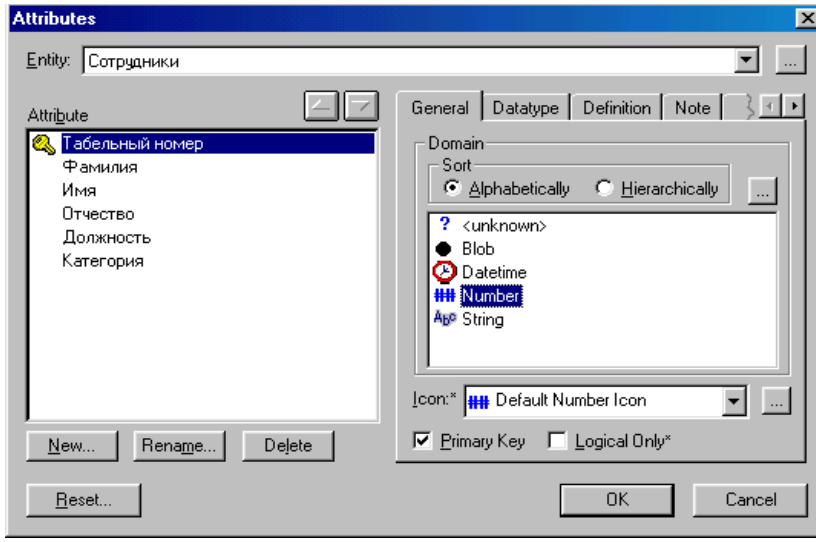


Рис. 70. Диалоговое окно Attributes

Для добавления нового атрибута щелкают по кнопке **New**…. В появившемся диалоге **New Attribute** (рис. 71) указывают имя атрибута (поле **Attribute Name**), имя атрибута в физической модели (поле **Column Name**) и выбирают домен. До-

мен атрибута будет использоваться при определении типа поля на уровне физической модели. Затем щелкают по кнопке **ОК**.

Для атрибутов первичного ключа закладки **General** диалогового окна **Attributes** необходимо выставить флаг **Primary Key**. В закладке **Definition** данного окна можно задать описание атрибута. В закладке **Datatype** можно уточнить размер атрибута.

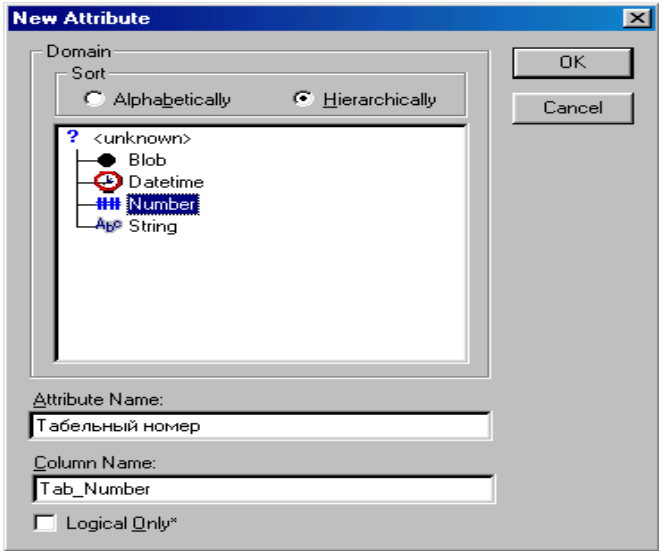


Рис. 71 Диалоговое окно New Attribute

Этап описания атрибутов повторяют для каждой новой внесенной сущности.

Установления связей между сущностями. Связь являются логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой. Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы, например:

Каждый КЛИЕНТ <*размещает*>ЗАКАЗы;

Каждый ЗАКАЗ <*выполняется*>СОТРУДНИКом.

На логическом уровне можно установить идентифицирующую связь один-ко-многим, связь многие-ко-многим и неидентифицирующую связь один-ко-многим (соответствующие кнопки располагаются в палитре инструментов слева направо). Различают зависимые и независимые сущности. Тип сущности определяется ее связью с другими сущностями. Идентифицирующая связь устанавливается между независимой (родительской) и зависимой (дочерней) сущностями. Когда рисуется идентифицирующая связь, ERwin автоматически преобразует дочернюю сущность в зависимую. Зависимая сущность изображается прямоугольником со скругленными углами. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ – (FK).

При установлении неидентифицирующей связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов дочерней сущности. Неидентифициирующая связь служит для связывания независимых сущностей.

Идентифицирующая связь показывается на диаграмме сплошной линей с жирной точкой на дочернем конце связи, не идентифицирующая – пунктирной.

Для создания новой связи следует выполнить следующие действия.

Щелкнуть по нужной кнопке установки связи в палитре инструментов (одной из трех).

Щелкнуть сначала по родительской, затем по дочерней сущности.

Форму линий можно изменить. Щелчком выделить линию и перетащить в нужное место.

Редактирование связей. Для редактирования свойств связи следует щелкнуть правой кнопкой мыши по линии связи и выбрать в контекстном меню пункт **Relationship Properties**….

В закладке **General** появившегося диалогового окна можно задать мощность, имя и тип связи (рис.72).

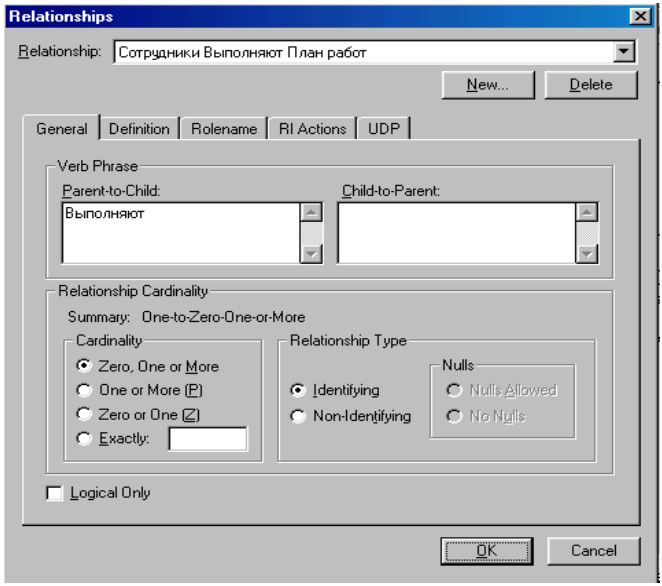


Рис. 72. Диалоговое окно Relationship

Мощность связи (Cardinality). Служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней. Различают четыре типа мощности. Конкретный тип мощности выбирают путем активизации соответствующего типа переключателя.

Имея связи (Verb Phrase). Фраза, характеризующая отношение между родительской и дочерними сущностями. Для идентифицирующей или неидентифицирующей связи один-ко-многим, достаточно указать имя, характеризующее отношение от родительской к дочерней сущности (Parent-to-Child). Для связи многие-ко-многим следует указывать имена как Parent-to-Child, так и Child-to-Parent.**Тип связи (Relationship Type)**. Необходимо установить тип связи – идентифицирующая или неидентифицирующая.

В закладке **Definition** можно задать более полное определение (описание) связи.

В закладке **Rolename** можно задать имя роли.

Имя роли (функциональное имя) – это синоним атрибута внешнего ключа, который показывает какую роль играет атрибут в дочерней сущности. Например, в сущности **Сотрудник** внешний ключ **Номер отдела** имеет функциональное имя «Где работает», которое показывает какую роль играет этот атрибут в сущности. Полное имя такого ключа – это функциональное имя и базовое имя разделенное точкой. Например, «Где работает.Номер отдела».

В закладке RI Actions (рис. 73) определяются **правила ссылочной целостности**.

Правила ссылочной целостности (Referential Integrity) – логические конструкции, которые выражают бизнес-правила использования данных и представляют собой правила вставки, замены и удаления. При генерации схемы базы данных на основе опций логической модели, задаваемых в закладке RI Actions, будут сгенерированы правила декоративной ссылочной целостности, которые должны быть предписаны для каждой связи и триггеры, обеспечивающие ссылочную целостность. Триггеры представляют собой программы, выполняемые всякий раз при выполнении команд вставки, замены и удаления (Insert, Update и Delete).

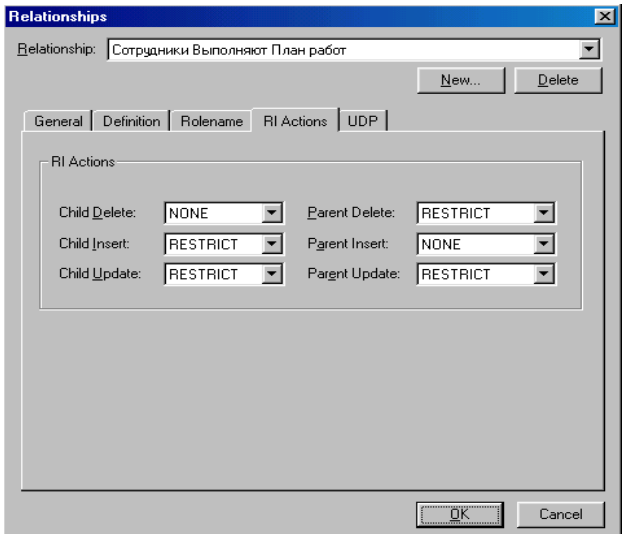


Рис. 73. Закладка RI Actions диалогового окна Relationships

Для каждого типа связи возможны различные режимы вставки, замены или удаления, осуществляемые в родительской и дочерней таблице. Режимы могут принимать следующие значения:

RESTRICT – действие ограничено;

CASCADE – действие выполняется каскадно;

NONE – действие не приводит к изменению значения внешнего ключа.

ERwin автоматически присваивает каждой связи значение ссылочной целостности, устанавливаемое по умолчанию прежде чем добавить ее в диаграмму.

Пример логической модели представлен на рис. 74.

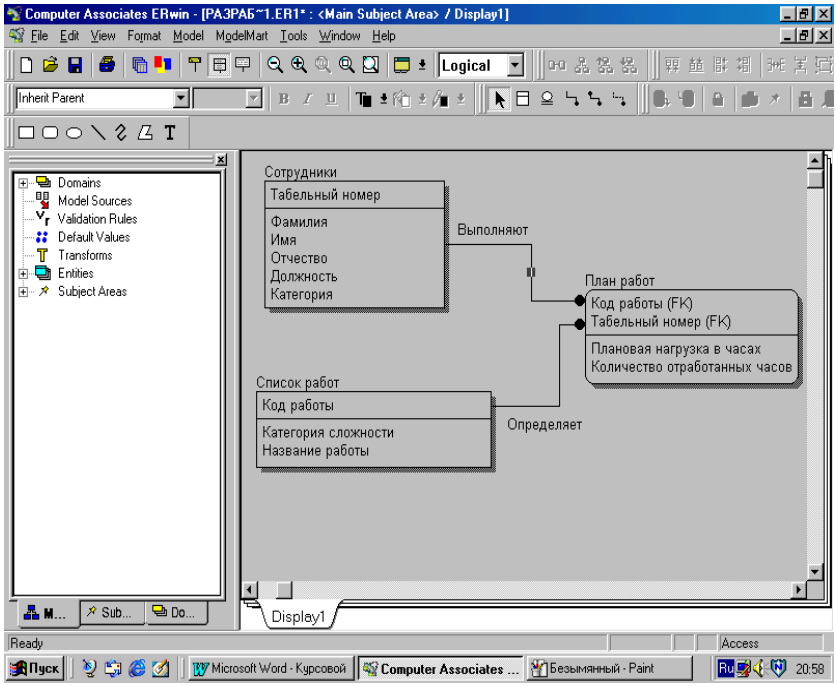


Рис. 74. Пример логической модели данных

6.3. Создание физической модели данных

Физическая модель содержит всю информацию, необходимую для реализации конкретной базы данных. Для вывода на экран физической модели нужно выбрать переключатель Physical на панели инструментов (рис. 75).

Физический уровень представления модели зависит от выбранного сервера (СУБД). Для выбора СУБД нужно переключиться на физический уровень и выбрать пункты меню Database/Choose Database… после чего откроется диалоговое окно Target Server.

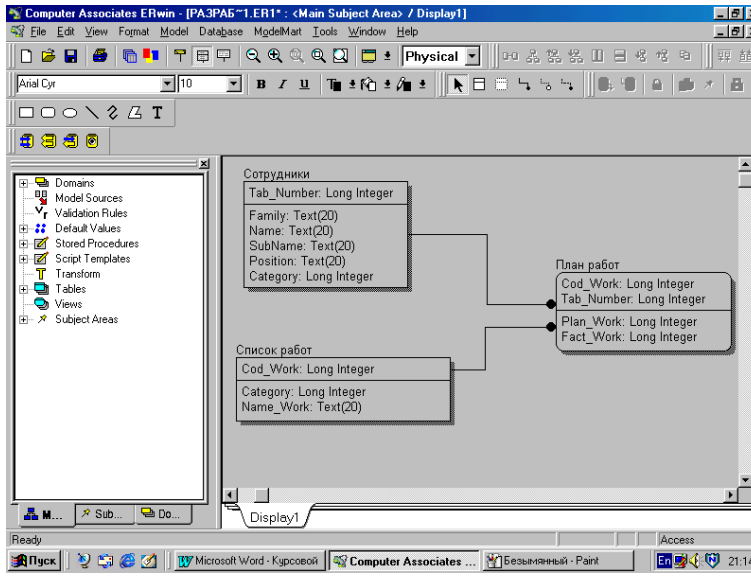


Рис. 75. Пример физической модели данных

ERwin поддерживает практически все распространенные СУБД, всего более 20 реляционных и нереляционных баз данных. Для выбора можно щелкнуть по соответствующей кнопке рядом с именем СУБД.

Процесс генерации физической схемы базы данных из логической модели данных называется прямым проектированием (Forward Engineering). Для генерации системного каталога базы данных следует перейти на физический уровень модели и выбрать пункт меню Tools/ Forward Engineering/Schema Generation… или щелкнуть по кнопке  на панели инструментов.

Откроется диалоговое окно Schema Generation (рис. 76).

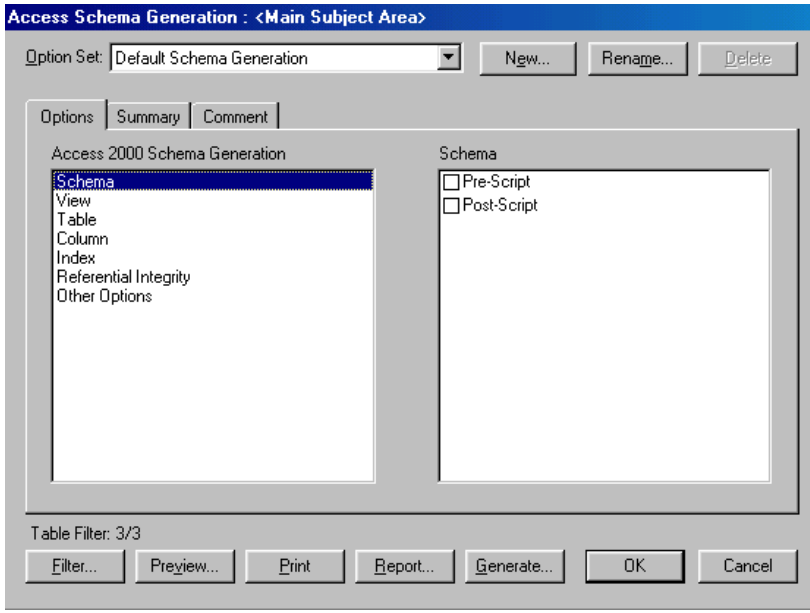


Рис. 76. Закладка Options диалогового окна Schema Generation

Данное окно имеет три закладки.

Options. Служит для задания опций генерации объектов базы данных (триггеров, таблиц, представлений, колонок, индексов и т.д.). Для заданий опций генерации какого-либо объекта следует выбрать объект в левом списке закладки, после чего включить соответствующую опцию в правом списке.

В закладке **Summary** отображаются все опции, заданные в закладке Options.

Comment. Позволяет внести комментарий для каждого набора опций.

Кнопка Preview вызывает диалог Schema Generation Preview, в котором отображается SQL-скрипт, создаваемый ERwin для генерации системного каталога СУБД. Нажатие на кнопку Generate приведет к запуску процесса генерации схемы.

Возникает диалоговое окно связи с базой данных. Например, для СУБД Access оно будет выглядеть следующим образом (рис. 77).

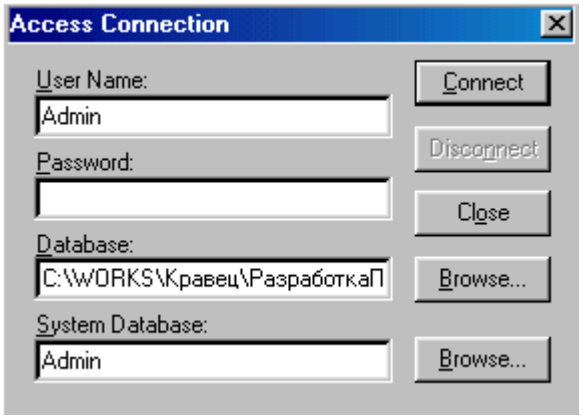


Рис. 77. Диалоговое окно связи с базой данных

В поле User Name следует ввести слово Admin. В поле Database необходимо ввести полное имя существующей базы данных, куда будет генерироваться модель данных. Затем щелкают по кнопке Connect, после чего выполняется SQL-скрипт генерации схемы данных.

ЗАКЛЮЧЕНИЕ

Учебное пособие рассматривает основы проектирования распределенных информационных систем. Пособие необходимо для выполнения лабораторных работ и курсового проекта по дисциплине «Проектирование распределенных информационных систем».

В пособии рассмотрены общие вопросы построения информационных систем, классификация ИС, состав ИС, этапы

жизненного цикла ИС, классификация CASE-средств проектирования ИС.

Вторая глава посвящена изложению основ структурного подхода к проектированию ИС и технологии построения моделей SADT, DFD, ERD, STD.

Освоение базовых принципов построения объектно-ориентированных баз данных, основ проектирования с применением языка UML позволит реализовывать объектно-ориентированный подход при создании современных информационных систем.

Особое внимание в пособии уделено построению типовых диаграмм на языке UML, позволяющих провести всесторонний анализ проектируемых информационных или программных систем.

Отдельные главы посвящены CASE-средствам проектирования ИС.

Четвертая глава описывает функциональные возможности инструментальной среды AllFusion Component Modeler, которую используют для объектно-ориентированного проектирования ИС.

Пятая глава посвящена описанию методологии и инструментальных средств BPwin для моделирования процессов.

Шестая глава описывает функциональные возможности ERwin по созданию моделей данных.

Таким образом, учебное пособие обеспечит освоение базовых понятий в сфере проектирования ИС и позволит практически освоить методы проектирования информационных систем с применением структурного и объектно-ориентированного подходов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Избачков Ю.С. Информационные системы: учебник для вузов / Ю.С. Избачков, В.Н. Петров. - СПб.: Питер, 2005. – 656 с.
2. Информационные системы и технологии в экономике и управлении: учеб. пособие / под ред. Проф. В.В. Трофимова. – М.: Высшее образование, 2007. – 480 с.
3. Саак А.Э. Информационные технологии управления / А.Э. Саак, Е.В. Пахомов, В.Н. Тюшняков. – СПб.: Питер, 2008. – 320 с.
4. Банк В.Р. Информационные системы в экономике / В.Р. Банк, В.С. Зверев. – М.: Экономистъ, 2005. – 477 с.
5. Советов Б.Я. Базы данных: теория и практика: учебник для бакалавров / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. – М.: Издательство Юрайт, 2012. – 464 с.
6. Буч Г. UML руководство пользователя / Г. Буч, Дж. Рамбо, А. Джекобсон. М.: ДМК, 2000. – 234 с.
7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином, 2001. – 278 с.
8. Леоненков А.В. Самоучитель UML / А.В. Леоненков. - СПб.: BHV, 2006. – 342 с.
9. Сергеева Т.И. Базы данных: модели данных, проектирование, язык SQL: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ВГТУ, 2012. - 233 с.
10. Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite / С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2005. – 432 с.

ОГЛАВЛЕНИЕ

Учебное издание

Сергеева Татьяна Ивановна
Сергеев Михаил Юрьевич

ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

В авторской редакции

Подписано в печать .10.2017.
Формат 60×84/16. Бумага для множительных аппаратов.
Усл. печ. л. . Уч.-изд. л. . Тираж экз.
Заказ. №

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14