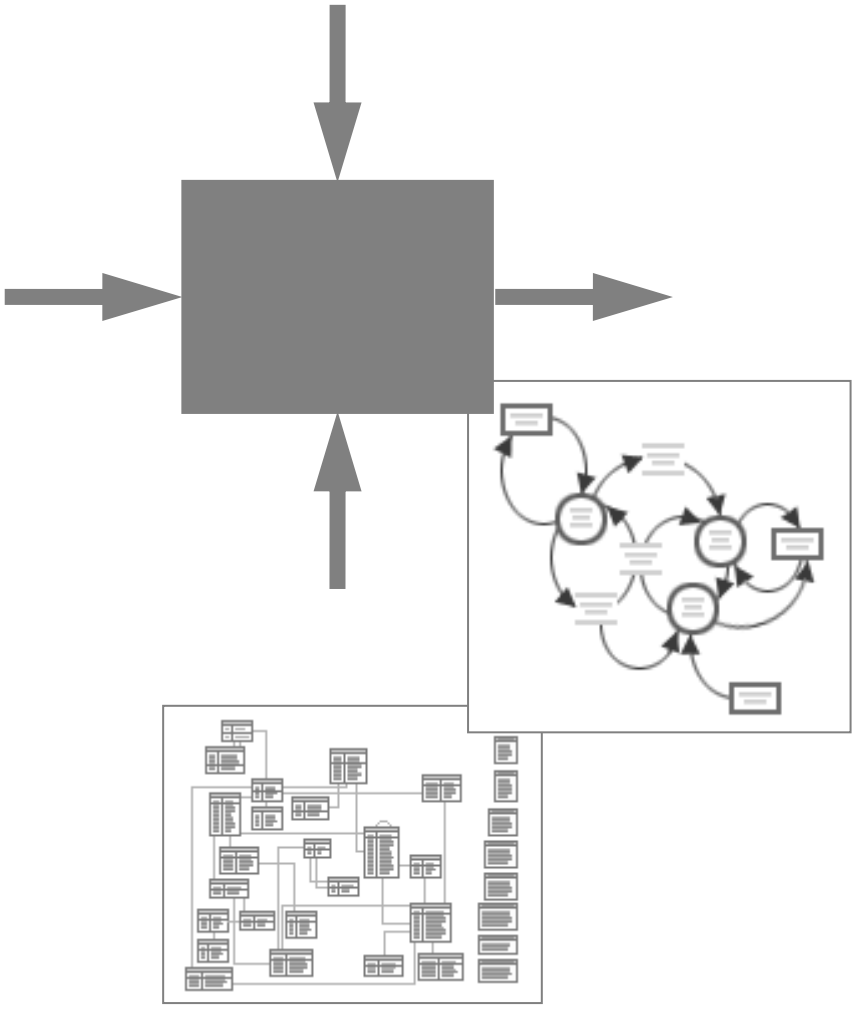
Э.Г. Галиаскаров
Д.О. Бытев
С.П. Бобков

ТЕОРИЯ

ИНФОРМАЦИОННЫХ ПРОЦЕССОВ И СИСТЕМ

Учебное пособие



ИГХТУ 2005

Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Ивановский государственный химико-технологический университет»
«Международный университет бизнеса и новых технологий (институт)»

Э.Г. Галиаскаров, Д.О. Бытев, С.П. Бобков

ТЕОРИЯ ИНФОРМАЦИОННЫХ ПРОЦЕССОВ И СИСТЕМ

Учебное пособие

Допущено учебно-методическим объединением по образованию в области прикладной информатики в качестве учебного пособия для студентов высших учебных заведений

Иваново 2005

УДК 681.3

Галиаскаров Э.Г., Бытев Д.О., Бобков С.П. Теория информационных процессов и систем: Учебное пособие/ Иван. гос. хим.-технол. ун-т. Иваново, 2005. 144 с. ISBN 5-9616-0075-0

Цель написания учебного пособия – дать студентам общее представление об информационных системах и познакомить их с современными методами описания систем и процессов. Структурно пособие состоит из четырех глав. В первой главе рассматриваются основные определения теории информационных систем, компоненты и структура информационной системы, классификация и области применения информационных систем. Во второй главе дается понятие модели и моделирования, рассматриваются основные типы информационных моделей и моделей данных. В третьей главе изложены структурные методы и средства анализа и проектирования информационных систем, приведен ряд примеров их использования. В четвертой главе рассмотрены состав, структура, функциональные особенности и классификация CASE-средств, приведены характеристики одного из популярнейших CASE-средств AllFusion Modeling Suite от компании Computer Associates.

Учебное пособие "Теория информационных процессов и систем" предназначено для студентов, обучающихся по специальности 071900 "Информационные системы и технологии". Материал пособия изложен в форме, доступной для самостоятельного изучения студентами. При написании учебного пособия использовались материалы, приведенные в списке литературы.

Табл. 7. Ил. 44. Библиогр.: 21 назв.

Печатается по решению редакционно-издательского совета Ивановского государственного химико-технологического университета.

Рецензенты:

кафедра прикладной математики и информационных технологий Ивановской государственной текстильной академии;

кандидат технических наук В.М. Вейцман (ЗАО «Информационные системы», г. Ярославль).

ISBN 5-9616-0075-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Информационные системы - область науки и техники, которая включает совокупность средств, способов и методов человеческой деятельности, направленных на создание и применение систем сбора, передачи, обработки, хранения и накопления информации.

Информационные системы предназначены для накопления сведений, хранения их и выдачи по мере необходимости. Сведения эти представляют собой описания предметов реального мира или абстрактных предметов, возникающих в различных дисциплинах науки, и представляют собой некоторые «истинные» утверждения или сообщения. С течением времени или в результате ошибок они могут становиться «ложными». Естественно, что одной из дисциплин, лежащих в основе теории информационных систем, должна быть и является *математическая логика*.

Математическими дисциплинами, пригодными для описания совокупностей предметов и их свойств, являются *теория множеств* и *реляционная алгебра* (последняя является математической теорией отношений). Сведения должны быть выражены на тех или иных языках. Для того чтобы их можно было подвергать обработке с помощью ЭВМ, сведения должны быть выражены на *формальных языках* (в которых смысл предложений однозначно определяется их формой). Для обработки сведений на ЭВМ должна быть составлена программа, являющаяся машинной формой *алгоритма*. Наконец, обработка информации машиной в соответствии с программами должна осуществляться за приемлемое время с допустимым расходованием различных ресурсов. Решение этих вопросов позволяет осуществить *теория сложных систем.*

Таким образом, в основе теории информационных процессов и систем лежат математическая логика, теория множеств, реляционная алгебра, теория формальных языков, теория алгоритмов и теория сложных систем.

Теория информационных процессов и систем изучает единицы информации, модели данных в информационных системах, конструктивные свойства описаний информационных систем.

Теория информационных процессов и систем направлена на изучение и решение проблемы организации информации в информационной системе, а также эффективной реализации процессов проектирования, эксплуатации и развития информационных систем.

Одним из современных подходов к описанию процессов и систем, их реализующих, является методология структурного анализа и проектирования. Данная методология базируется на системном подходе, прошла проверку временем и позволяет эффективно проводить построение функциональных моделей, описание бизнес-процессов, моделирование потоков данных и формирование оптимальной структуры данных.

Методология структурного анализа и проектирования имеет ряд развитых, хорошо документированных и рекомендованных для широкого использования стандартов: **IDEF0** (функциональное моделирование), **IDEF3** (моделиро-

вание бизнес-процессов), **DFD** (моделирование потоков данных), **IDEF1** (информационное моделирование), **IDEF1x** (моделирование реляционных структур данных) и некоторые другие. Большинство из этих стандартов реализованы в виде компьютерных систем поддержки процесса разработки и внедрения сложных информационных систем, так называемые **CASE**-системы, позволяющие автоматизировать и упростить процесс сопровождения информационной системы на всем протяжении ее жизненного цикла.

Для полноты картины следует отметить, что в последние годы все большей популярностью пользуется методология объектно-ориентированного анализа и проектирования. Данная методология основана на объектной парадигме и в качестве стандарта использует унифицированный язык моделирования **UML**. Язык **UML** обладает развитым инструментарием описания систем и процессов и, вероятно, представляет самую перспективную нотацию для создания систем самой различной природы.

ГЛАВА 1. ОБЩЕЕ ПРЕДСТАВЛЕНИЕ ОБ ИНФОРМАЦИОННЫХ СИСТЕМАХ

1.1. Понятие информационной системы

В истории вычислительной техники можно проследить две основных области ее использования: для выполнения сложных численных расчетов и для хранения и обработки больших объемов информации. Вторая область применения привела к созданию информационных систем (ИС). Рассмотрим понятия информации и системы [1-3].

Информация - это некоторые сведения, знания об объектах и процессах реального мира. Экономическая информация отображается, как правило, в виде документов. ***Документ*** - это материальный носитель информации, имеющий юридическую силу и оформленный в установленном порядке.

Наряду с термином "информация" часто используют слово "данные". Во время обработки смысл информации отодвигается на второй план, а основное внимание обращают на форму представления, в этом смысле ***данные*** - это информация, представленная в формализованном виде, который позволяет передавать или обрабатывать ее при помощи технических средств.

Система - множество элементов, находящихся в отношениях и связях друг с другом, которое образует определенную целостность, единство. Каждая система характеризуется структурой, входными и выходными потоками, целью и ограничениями, законом функционирования.

Структура – совокупность образующих систему элементов и связей между ними.

Элемент – объект, обладающий рядом важных свойств, для которого определён закон функционирования, и внутренняя структура которого не рассматривается.

Подсистема – часть системы, выделенная по определённому признаку, обладает некоторой самостоятельностью и допускает разложение на некоторые элементы.

Вид отношений между элементами, проявляющийся при взаимодействии называется ***связью***. Различают внешние связи, то есть связи системы с окружающей средой, и внутренние связи, т. е. связи между подсистемами и элементами.

Среда – множество объектов вне данного элемента, которое может оказывать влияние на данный элемент и само находится под его воздействием.

Цель – это ситуация или область ситуаций, которую нужно достигнуть при функционировании системы за определённый промежуток времени.

Закон функционирования – описывает процесс функционирования элемента системы во времени.

Процесс – совокупность состояний системы, упорядоченных по изменению какого-либо параметра, определяющего свойства системы.

Информационная система (ИС) - это программно-аппаратный комплекс, предназначенный для автоматизированного сбора, хранения, обработки и выдачи информации. Другими словами можно сказать, что информационная система есть средство реализации (осуществления) информационных процессов.

Информационный процесс – совокупность последовательных действий, производимых над информацией для получения какого-либо результата (достижения цели). Принято разделять информационные процессы на **общие** и **основные**. Наиболее **общими** информационными процессами являются *сбор*, *преобразование* и *использование* информации. К **основным** информационным процессам относят *поиск*, *отбор*, *хранение*, *передачу*, *кодирование*, *обработку* и *защиту* информации.

Обычно ИС имеют дело с большими объемами информации, которая имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы продажи билетов на транспорте и др.

ИС всегда специализируется на информации из определенной области реального мира: экономики, техники, медицины и т.д. Часть реального мира, отображаемая в ИС, называется ***предметной областью***. К примеру, экономические ИС - это ИС, предметной областью которых является экономика, и т.п.

Любая ИС включает в себя четыре ***компонента***: информационные средства, программные средства (обеспечение), технические средства, персонал. Информационное обеспечение реализуется в виде файловой системы или в виде базы данных. ***База данных (БД)*** - это совокупность описаний объектов предметной области и связей между ними, актуальных для конкретной предметной области.

Особенность ИС по сравнению с вычислительными системами состоит в том, что структура данных в ИС обычно сложна (а сложность определяется не столько объемом, сколько количеством взаимосвязей), а задачи по обработке данных однотипны для разных предметных областей (создание, поиск, ввод и вывод, группировка, сортировка). Поэтому все типовые функции по работе с данными были выделены в специальную систему. ***Система управления базой данных (СУБД)*** - это комплекс программных и языковых средств создания, ведения и манипулирования данными.

Программные средства делят на две части: системное программное обеспечение (СПО) и прикладное программное обеспечение (ППО). В состав СПО входит операционная система ЭВМ (ОС). ОС настолько тесно связана с техническими средствами, что их часто объединяют и называют ***программно-аппаратной платформой***, например, для ПЭВМ IBM PC используется платформа "WINTEL" (Windows + Intel).

Персонал - это специалисты, которые обслуживают и сопровождают ИС, их часто включают в состав системы, поскольку без персонала невозможна работа сколько-нибудь сложной системы.

1.2. История и направления развития информационных систем

Понятие информационной системы (ИС) на протяжении своего существования претерпело значительные изменения. Первоначально ИС считалась любая система, позволяющая собирать, хранить и обрабатывать информацию, например - система каталогов в библиотеке, телефонный справочник и т.п. С появлением ЭВМ к ИС стали относить программы, которые выполняют перечисленные функции и имеют дело с большими объемами информации. Условно можно выделить три поколения ИС. Рассмотрим основные характеристики компонентов этих ИС [3].

Первое поколение предназначалось для решения установившихся задач, которые четко определялись на этапе создания системы и затем практически не изменялись.

Основные черты 1-го поколения ИС:

- техническое обеспечение систем составляли ЭВМ 2-3 поколения;
- информационное обеспечение (ИО) представляло собой массивы (файлы) данных, структура которых определялась той программой, в которой они использовались;
- программное обеспечение - специализированные прикладные программы, например, программа начисления заработной платы;
- архитектура ИС - централизованная. Как правило, применялась пакетная обработка задач. Конечный пользователь не имел непосредственного контакта с ИС, вся предварительная обработка информации и ввод производились персоналом ИС.

Недостатки ИС 1-го поколения:

- сильная взаимосвязь между программами и данными, то есть изменения в предметной области приводили к изменению структуры данных, а это заставляло переделывать программы;
- трудоемкость разработки и модификации систем;
- сложность согласования частей системы, разработанных разными людьми в разное время.

Второе поколение. Стремление преодолеть указанные недостатки породило в 70-х годах технологию баз данных. База данных (БД) создается для группы взаимосвязанных задач для многих пользователей, и это позволяет частично решить перечисленные проблемы. Вначале СУБД разрабатывались для больших ЭВМ, и их количество не превышало десятка. Каждая система была уникальным и очень сложным произведением, но на ее основе можно было намного быстрее и эффективнее разрабатывать прикладные ИС. Однако стоимость самой развитой СУБД была и остается очень большой. Благодаря появлению ПЭВМ, технология БД стала массовой, создано большое количество инструментальных средств и СУБД для разработки ИС, что в свою очередь вызвало появление огромного количества прикладных ИС в разных областях, в том числе в области экономики, которые отличаются эффективностью, полнотой функций и уровнем сервиса.

Основные черты 2-го поколения ИС:

- основу ИО составляет база данных;
- программное обеспечение состоит из прикладных программ и СУБД;
- технические средства: ЭВМ 3-4 поколения и ПЭВМ;
- средства разработки ИС: процедурные языки программирования 3-4 поколения, расширенные языком работы с БД (SQL, QBE);
- архитектура ИС: наиболее популярны две разновидности: персональная локальная ИС, централизованная БД с сетевым доступом.

Большим шагом вперед явилось развитие принципа "дружественного интерфейса" по отношению к пользователю (как к конечному, так и к разработчику ИС). Например, повсеместно применяется графический интерфейс, развитые системы помощи и подсказки пользователю, разнообразные инструменты для упрощения разработки ИС: системы быстрой разработки приложений (RAD-системы), средства автоматизированного проектирования ИС (CASE- средства).

К концу 80-х годов выявились и недостатки систем 2-го поколения:

- большие капиталовложения в компьютеризацию предприятий не дали ожидаемого эффекта, соответствующего затратам (увеличились накладные расходы, но не произошло резкого повышения производительности);
- внедрение ИС столкнулось с инертностью людей, нежеланием конечных пользователей менять привычный стиль работы, осваивать новые технологии;
- к квалификации пользователей стали предъявляться более высокие требования (знание персонального компьютера, конкретных прикладных программ и СУБД, способность постоянно повышать свою квалификацию).

В связи с этим постепенно стало формироваться 3-е поколение ИС. Рассмотрим основные **черты современного поколения ИС**.

Техническая платформа: мощные ЭВМ 4-5 поколения, использование разных платформ в одной ИС (большие ЭВМ, мощные стационарные ПК, мобильные ПК). Наиболее характерно широкое применение вычислительных сетей - от локальных до глобальных.

Информационное обеспечение: ведутся интенсивные разработки с целью повышения интеллектуальности банка данных в следующих направлениях:

- новые модели знаний, учитывающие не только структуру информации, но и активный характер знаний;
- средства оперативного анализа информации (OLAP) и средства поддержки принятия решений (DSS);
- новые формы представления информации, более естественные для человека (мультимедиа, полнотекстовые БД, гипертекстовые БД, средства восприятия и синтеза речи).

Программное обеспечение (ПО): существенно новым является появление и развитие открытой компонентной архитектуры ИС. ***Компонент*** - это программа, выполняющая какой-либо осмысленный с точки зрения конечного пользователя набор функций и имеющая открытый интерфейс. ПО ИС собирается из готовых компонентов, как мозаика из фрагментов. С другой стороны,

компонент может функционировать на разных типах ЭВМ и связь между компонентами устанавливается не на этапе компиляции, а в реальном масштабе времени. Такой принцип построения позволяет использовать огромный накопленный опыт программистов, ускорять разработку ИС, создавать распределенные ИС.

Архитектура ИС: стала более разнообразной в связи с многоплатформенностью. Так, в настоящее время развивается трехступенчатая модель ИС. Благодаря такому построению снижаются требования к клиентским машинам и общая стоимость системы, повышается общая эффективность и производительность. Узким местом является пропускная способность и надежность вычислительных сетей.

Методы разработки ИС: при традиционном подходе сначала выявлялись информационные потоки на предприятии, а затем к этой структуре привязывалась ИС, повторяя и закрепляя тем самым недостатки организации бизнеса. В 90-93 гг. бурно обсуждалась идея бизнес-реинжиниринга, предложенная М. Хаммером. Она состоит в том, что для получения существенного эффекта от ИС необходимо одновременно с разработкой ИС пересмотреть и бизнес-процессы, удалив и упростив некоторые из них. Другая идея - создание ИС с расчетом на длительную или постоянную модернизацию, причем система в каждый период своей жизни приносит пользу и способна развиваться дальше. Наконец, при создании ИС необходим учет национальной, профессиональной и корпоративной культуры, так как человеческий фактор часто является решающим для успеха. Таким образом, современная корпоративная ИС должна создаваться как часть предприятия, включающая бизнес-архитектуру, персонал и информационные технологии.

1.3. Классификация информационных систем

По режиму работы ИС делятся на пакетные, диалоговые и смешанные [3]**.** *Пакетные ИС* работают в пакетном режиме: вначале данные накапливаются, и формируется пакет данных, а затем пакет последовательно обрабатывается рядом программ. Недостаток этого режима - низкая оперативность принятия решений и обособленность пользователя от системы. *Диалоговые ИС* работают в режиме обмена сообщениями между пользователями и системой (например, система продажи авиабилетов). Этот режим особенно удобен, когда пользователь может выбирать перспективные варианты из числа предлагаемых системой. *Смешанные ИС* сочетают оба типа режима работы.

По способу распределения вычислительных ресурсов ИС делятся на локальные и распределенные. *Локальные ИС* используют одну ЭВМ, а в *распределенных ИС* взаимодействуют несколько ЭВМ, связанных сетью. Отдельные узлы сети обычно территориально удалены друг от друга, решают разные задачи, но используют общую информационную базу.

По функциям различают три вида ИС: системы обработки данных, автоматизированные системы управления и информационно-поисковые системы [3].

Системы обработки данных (СОД) предназначены, например, для решения задач расчета заработной платы, статистической отчетности, то есть таких, которые наряду с функциями ввода, выборки, коррекции информации выполняют математические расчеты без применения методов оптимизации.

Автоматизированные системы управления (АСУ) отличаются от СОД тем, что сами выполняют управленческие функции по отношению к объекту. В АСУ включаются прикладные программы для принятия и оптимизации управленческих решений. Примером АСУ является система для оптимального управления запасами материалов на складе.

Информационно-поисковые системы (ИПС) предназначены для поиска требуемого документа или факта во множестве документов (рис. 1.1).

Поисковый образ документа (ПОД) получается в результате процесса индексирования, которое выполняется квалифицированными специалистами и состоит из двух этапов: выявление смысла документа и описание смысла на специальном информационно-поисковом языке (ИПЯ).

Запрос к ИПС описывается также на этом языке. Поиск документа состоит в сравнении множества хранящихся в системе ПОД и текущего поискового образа запроса (ПОЗ), в результате чего пользователю выдается требуемый документ или отказ. Различают два режима работы ИПС: текущее информирование пользователей о новых поступлениях и ретроспективный поиск по разовым запросам.

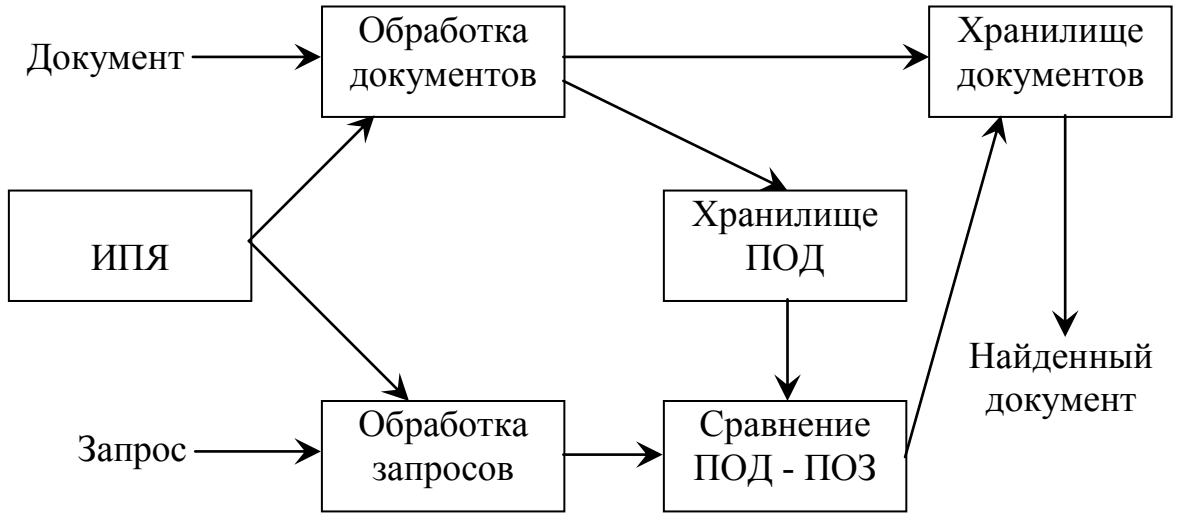


Рис. 1.1. Схема информационно-поисковой системы

По концепции построения ИС делятся на файловые системы, автоматизированные банки данных (АБД), интеллектуальные банки данных (банки знаний) и хранилища данных.

Информационное обеспечение ИС первого типа построено в виде *файловых систем.* В современных ЭВМ операционная система берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным. Программное обеспечение ИС напрямую использует функции ОС для работы с файлами. Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам. В таких системах сложно решить проблемы согласования данных в разных файлах, коллективного доступа к данным, модификации структуры данных.

Банком данных (АБД) (рис. 1.2) называют систему специальным образом организованных баз данных, программных, технических, языковых и организационно - методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных [4].

В отличие от файловых систем, структура базы данных (БД) меньше зависит от прикладных программ, а все функции по работе с БД сосредоточены в специальном компоненте - системе управления базами данных (СУБД), которая играет центральную роль в функционировании банка данных, так как обеспечивает связь прикладных программ и пользователей с данными. Сведения о структуре БД сосредоточены в словаре-справочнике (репозитории) АБД, этот вид информации называется ***метаинформацией.*** В состав метаинформации входит семантическая информация, физические характеристики данных и информация об их использовании. С помощью словарей данных автоматизируется процесс использования метаинформации в ИС.

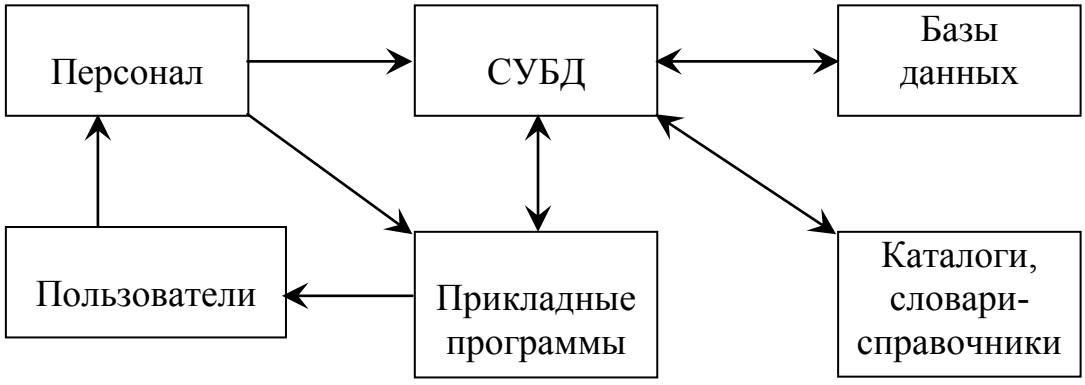


Рис. 1.2. Структура банка данных

Интеллектуальный банк данных (ИБД) (рис. 1.3) - это сравнительно новый способ построения ИС, при котором информация о предметной области условно делится между двумя базами.

База данных содержит сведения о количественных и качественных характеристиках конкретных объектов. *База знаний (БЗ)* содержит сведения о закономерностях в ПО, позволяющие выводить новые факты из имеющихся в БД, метаинформацию, сведения о структуре предметной области, сведения, обеспечивающие понимание запроса и синтез ответа.

Диалоговый процессор предназначен для понимания смысла запроса и его перевода в термины знаний, заложенных в БЗ. *Планировщик* преобразует полученный запрос в рабочую программу, составляя ее из модулей, имеющихся в БЗ. *Подсистема пополнения знаний* позволяет ИС обучаться.

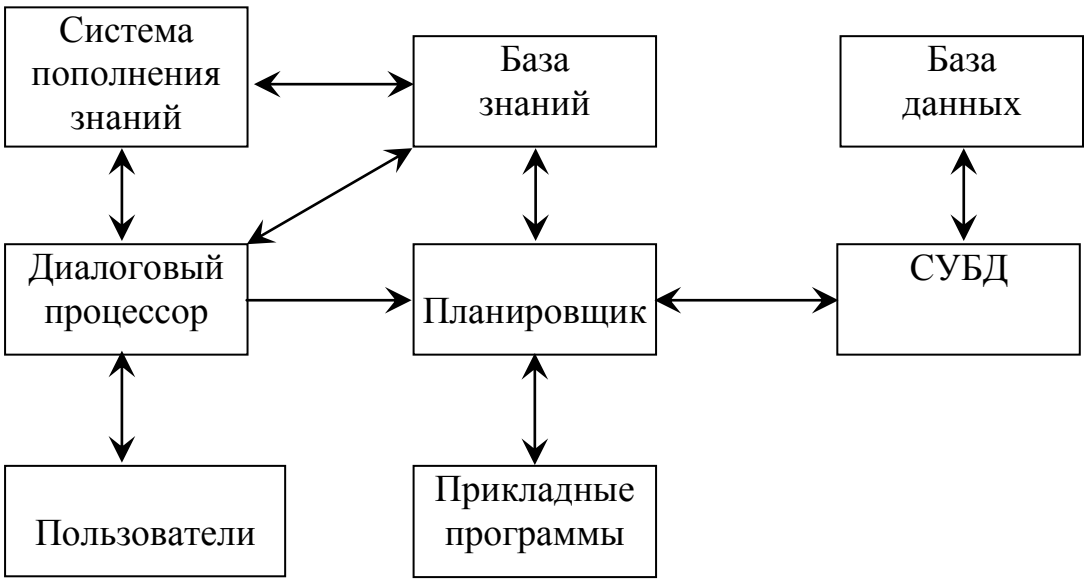


Рис. 1.3. Структура банка знаний

Если в традиционном банке данных знания о предметной области заложены программистом в каждую прикладную программу, а также в структуру БД, то в интеллектуальном банке данных они хранятся в базе знаний и отделены от прикладных программ. В отличие от данных, знания активны: на их основе формируются цели и выбираются способы их достижения. Например, ИБД в системе складского учета может автоматически реагировать на такое событие, как уменьшение количества деталей на складе до критической нормы, при этом ИБД без участия пользователя генерирует документы для заказа этих деталей и отправляет их по электронной почте поставщику.

Другое характерное отличие знаний от данных - связность, причем знания отражают как структурные взаимосвязи между объектами предметной области, так и вызванные конкретными бизнес - процессами, например такие связи, как “происходит одновременно”, “следует из...”, "если - то" и др.

Наконец, существенную роль в ИБД играет форма представления информации для пользователя: она должна быть как можно ближе к естественным для человека способам обмена данными (профессиональный естественный язык, речевой ввод / вывод, графическая форма).

В настоящее время в корпоративных базах данных накоплены гигантские объемы информации, однако она недостаточно эффективно используется в процессе управления бизнесом, поэтому бурно развивается новая форма построения ИС - склады (хранилища) данных.

Хранилище данных представляет собой АБД [5], в котором база данных разделена на два компонента: оперативная БД хранит текущую информацию,

квазипостоянная БД содержит исторические данные (рис. 1.4), например, в оперативной БД могут содержаться данные о продажах за текущий год, а в квазипостоянной БД хранятся систематизированные годовые отчеты и балансы за все время существования предприятия. Подсистема оперативного анализа данных позволяет эффективно и быстро анализировать текущую информацию. Подсистема принятия решений пользуется обобщенной и исторической информацией, применяет методы логического вывода. Для общения с пользователем служит универсальный интерфейс.

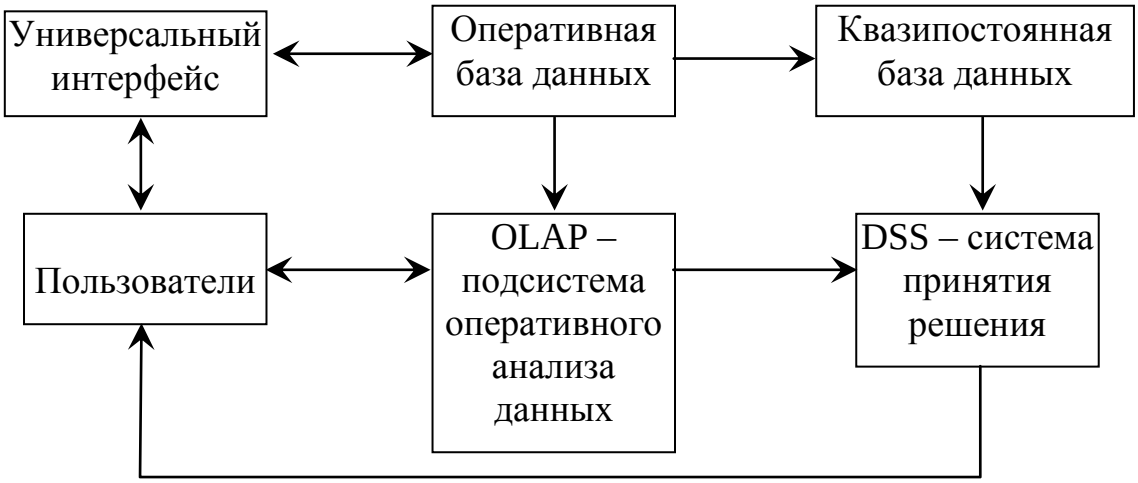


Рис. 1.4. Структура хранилища данных

По масштабу информационные системы подразделяются на следующие группы [6]: одиночные; групповые; корпоративные.

Одиночные информационные системы реализуются, как правило, на автономном персональном компьютере (сеть не используется). Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. Подобные приложения создаются с помощью так называемых *настольных или локальных систем* управления базами данных (СУБД). Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

Групповые информационные системы ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используются серверы баз данных (называемые также SQL-серверами) для рабочих групп. Существует довольно большое количество различных SQL-серверов, как коммерческих, так и свободно распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix.

Корпоративные информационные системы являются развитием систем для рабочих групп, они ориентированы на крупные компании и могут поддерживать территориально разнесенные узлы или сети. В основном они имеют

иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура клиент-сервер со специализацией серверов или же многоуровневая архитектура. При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информационных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

Для групповых и корпоративных систем существенно повышаются требования к надежности функционирования и сохранности данных. Эти свойства обеспечиваются поддержкой целостности данных, ссылок и транзакций в серверах баз данных.

По сфере применения информационные системы обычно подразделяются на четыре группы [6]:

- системы обработки транзакций;
- системы поддержки принятия решений;
- информационно-справочные системы;
- офисные информационные системы.

Системы обработки транзакций, в свою очередь, по оперативности обработки данных, разделяются на пакетные информационные системы и оперативные информационные системы. В информационных системах организационного управления преобладает режим оперативной обработки транзакций — OLTP (OnLine Transaction Processing), для отражения актуального состояния предметной области в любой момент времени, а пакетная обработка занимает весьма ограниченную часть. Для систем OLTP характерен регулярный (возможно, интенсивный) поток довольно простых транзакций, играющих роль заказов, платежей, запросов и т. п. Важными требованиями для них являются:

- высокая производительность обработки транзакций;
- гарантированная доставка информации при удаленном доступе к БД по телекоммуникациям.

Системы поддержки принятия решений — DSS (Decision Support System) — представляют собой другой тип информационных систем, в которых с помощью довольно сложных запросов производится отбор и анализ данных в различных разрезах: временных, географических и по другим показателям.

Обширный класс *информационно-справочных систем* основан на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные системы получили в сети Интернет.

Класс *офисных информационных систем* нацелен на перевод бумажных документов в электронный вид, автоматизацию делопроизводства и управление документооборотом.

Следует отметить, что приводимая классификация по сфере применения в достаточной степени условна. Крупные информационные системы очень часто обладают признаками всех перечисленных выше классов. Кроме того, корпоративные информационные системы масштаба предприятия обычно состоят из ряда подсистем, относящихся к различным сферам применения.

По способу организации групповые и корпоративные информационные системы подразделяются на следующие классы [6]:

- системы на основе архитектуры файл-сервер;
- системы на основе архитектуры клиент-сервер;
- системы на основе многоуровневой архитектуры;
- системы на основе Интернет/интранет-технологий.

В любой информационной системе можно выделить необходимые функциональные компоненты (табл. 1.1), которые помогают понять ограничения различных архитектур информационных систем.

Таблица 1.1

Типовые функциональные компоненты информационной системы [6]

Обозначение	Наименование	Характеристика
PS	Presentation Services (средства представления)	Обеспечиваются устройствами, принимающими ввод от пользователя и отображающими то, что сообщает ему компонент логики представления PL, с использованием соответствующей программной поддержки
PL	Presentation Logic (логика представления)	Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя при выборе команды в меню, нажатии кнопки или выборе элемента из списка
BL	Business or Application Logic (прикладная логика)	Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение
DL	Data Logic (логика управления данными)	Операции с базой данных (SQL-операторы), которые нужно выполнить для реализации прикладной логики управления данными
DS	Data Services (операции с базой данных)	Действия СУБД, вызываемые для выполнения логики управления данными, такие как манипулирование данными, определения данных, фиксация или откат транзакций и т. п. СУБД обычно компилирует SQL-предложения
FS	File Services (файловые операции)	Дисковые операции чтения и записи данных для СУБД и других компонентов. Обычно являются функциями операционной системы (ОС)

Архитектура файл-сервер не имеет сетевого разделения компонентов диалога **PS** и **PL** и использует компьютер для функций отображения, что об-

легчает построение графического интерфейса. Файл-сервер только извлекает данные из файлов, так что дополнительные пользователи и приложения добавляют лишь незначительную нагрузку на центральный процессор. Каждый новый клиент добавляет вычислительную мощность к сети.

Объектами разработки в файл-серверном приложении являются компоненты приложения, определяющие логику диалога **PL**, а также логику обработки **BL** и управления данными **DL.** Разработанное приложение реализуется либо в виде законченного загрузочного модуля, либо в виде специального кода для интерпретации.

Однако такая архитектура имеет существенный недостаток: при выполнении некоторых запросов к базе данных клиенту могут передаваться большие объемы данных, загружая сеть и приводя к непредсказуемости времени реакции. Значительный сетевой трафик особенно сильно сказывается при организации удаленного доступа к базам данных на файл-сервере через низкоскоростные каналы связи. Одним из вариантов устранения данного недостатка является удаленное управление файл-серверным приложением в сети. При этом в локальной сети размещается сервер приложений, совмещенный с телекоммуникационным сервером (обычно называемым сервером доступа), в среде которого выполняются обычные файл-серверные приложения. Особенность состоит в том, что диалоговый ввод-вывод поступает от удаленных клиентов через телекоммуникации. Приложения не должны быть слишком сложными, иначе велика вероятность перегрузки сервера, или же нужна очень мощная платформа для сервера приложений.

Одним из традиционных средств, на основе которых создаются файл-серверные системы, являются локальные СУБД. Однако такие системы, как правило, не отвечают требованиям обеспечения целостности данных (в частности, они не поддерживают транзакции). Поэтому при их использовании задача обеспечения целостности данных возлагается на программы клиентов, что приводит к усложнению клиентских приложений. Однако эти инструменты привлекают своей простотой, удобством использования и доступностью. Поэтому файл-серверные информационные системы до сих пор представляют интерес для малых рабочих групп и, более того, нередко используются в качестве информационных систем масштаба предприятия.

Архитектура клиент-сервер предназначена для разрешения проблем файл-серверных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать наиболее эффективно. Особенностью архитектуры клиент-сервер является использование выделенных серверов баз данных, понимающих запросы на языке структурированных запросов SQL (Structured Query Language) и выполняющих поиск, сортировку и агрегирование информации.

Отличительная черта серверов БД — наличие справочника данных, в котором записана структура БД, ограничения целостности данных, форматы и даже серверные процедуры обработки данных по вызову или по событиям в программе. Объектами разработки в таких приложениях помимо диалога и ло-

гики обработки являются, прежде всего, реляционная модель данных и связанный с ней набор SQL-операторов для типовых запросов к базе данных.

Большинство конфигураций клиент-сервер использует двухуровневую модель, в которой клиент обращается к услугам сервера. Предполагается, что диалоговые компоненты **PS** и **PL** размещаются на клиенте, что позволяет обеспечить графический интерфейс. Компоненты управления данными **DS** и **FS** размещаются на сервере, а диалог **(PS, PL),** логика **BL** и **DL** — на клиенте. Двухуровневое определение архитектуры клиент-сервер использует именно этот вариант: приложение работает у клиента, СУБД — на сервере (рис. 1.5).

Поскольку эта схема предъявляет наименьшие требования к серверу, она обладает наилучшей масштабируемостью. Однако сложные приложения, вызывающие большое взаимодействие с БД, могут жестко загрузить как клиента, так и сеть. Результаты SQL-запроса должны вернуться клиенту для обработки, потому что там находится логика принятия решения. Такая схема приводит к дополнительному усложнению администрирования приложений, разбросанных по различным клиентским узлам.

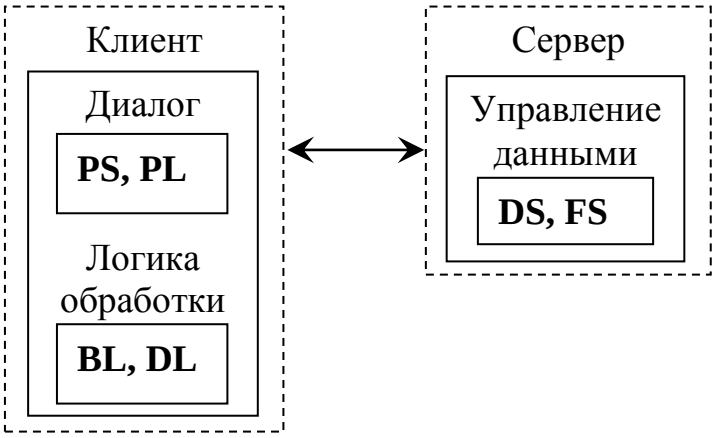


Рис. 1.5. Классический вариант клиент-серверной информационной системы

Для сокращения нагрузки на сеть и упрощения администрирования приложений компонент BL можно разместить на сервере. При этом вся логика принятия решений оформляется в виде *хранимых процедур* и выполняется на сервере БД.

Хранимая процедура — процедура с операторами SQL для доступа к БД, вызываемая по имени с передачей требуемых параметров и выполняемая на сервере БД. Хранимые процедуры могут компилироваться, что повышает скорость их выполнения и сокращает нагрузку на сервер.

Хранимые процедуры улучшают целостность приложений и БД, гарантируют актуальность коллективно используемых операций и вычислений. Улучшается сопровождение таких процедур, а также безопасность (нет прямого доступа к данным).

Следует помнить, что перегрузка хранимых процедур прикладной логикой может перегрузить сервер, что приведет к потере производительности. Эта

проблема особенно актуальна при разработке крупных информационных систем, в которых к серверу может одновременно обращаться большое количество клиентов. Поэтому в большинстве случаев следует принимать компромиссные решения: часть логики приложения размещать на стороне сервера, часть — на стороне клиента. Такие клиент-серверные системы называются системами с разделенной логикой. Данная схема при удачном разделении логики позволяет получить более сбалансированную загрузку клиентов и сервера, но при этом затрудняется сопровождение приложений.

Создание архитектуры клиент-сервер возможно и на основе многотерминальной системы. В этом случае в многозадачной среде сервера приложений выполняются программы пользователей, а клиентские узлы вырождены и представлены терминалами. Подобная схема информационной системы характерна для UNIX.

Многоуровневая архитектура стала развитием архитектуры клиент-сервер и в своей классической форме состоит из трех уровней:

- нижний уровень представляет собой приложения клиентов, выделенные для выполнения функций и логики представлений PS и PL и имеющие программный интерфейс для вызова приложения на среднем уровне;
- средний уровень представляет собой сервер приложений, на котором выполняется прикладная логика BL и с которого логика обработки данных DL вызывает операции с базой данных DS;
- верхний уровень представляет собой удаленный специализированный сервер базы данных, выделенный для услуг обработки данных DS и файловых операций FS (без риска использования хранимых процедур).

Подобную концепцию обработки данных пропагандируют, в частности, фирмы Oracle, Sun, Inprise и др.

Трехуровневая архитектура позволяет еще больше сбалансировать нагрузку на разные узлы и сеть, а также способствует специализации инструментов для разработки приложений и устраняет недостатки двухуровневой модели клиент-сервер.

Централизация логики приложения упрощает администрирование и сопровождение. Четко разделяются платформы и инструменты для реализации интерфейса и прикладной логики, что позволяет с наибольшей отдачей реализовывать их специалистам узкого профиля. Наконец, изменения прикладной логики не затрагивают интерфейса, и наоборот. Но поскольку границы между компонентами **PL, BL** и **DL** размыты, прикладная логика может появиться на всех трех уровнях. Сервер приложений с помощью монитора транзакций обеспечивает интерфейс с клиентами и другими серверами, может управлять транзакциями и гарантировать целостность распределенной базы данных. Средства удаленного вызова процедур наиболее соответствуют идее распределенных вычислений: они обеспечивают из любого узла сети вызов прикладной процедуры, расположенной на другом узле, передачу параметров, удаленную обработку и возврат результатов.

С ростом систем клиент-сервер необходимость трех уровней становится все более очевидной. Продукты для трехзвенной архитектуры, так называемые мониторы транзакций, являются относительно новыми. Эти инструменты в основном ориентированы на среду UNIX, однако прикладные серверы можно строить на базе Microsoft Windows NT с использованием вызова удаленных процедур для организации связи клиентов с сервером приложений. На практике в локальной сети могут использоваться смешанные архитектуры (двухуровневые и трехуровневые) с одним и тем же сервером базы данных. С учетом глобальных связей архитектура может иметь больше трех звеньев. В настоящее время появились новые инструментальные средства для гибкой сегментации приложений клиент-сервер по различным узлам сети.

Таким образом, многоуровневая архитектура распределенных приложений позволяет повысить эффективность работы корпоративной информационной системы и оптимизировать распределение ее программно-аппаратных ресурсов. Но пока на российском рынке по-прежнему доминирует архитектура клиент-сервер.

Интернет/интранет-технологии. В развитии технологии Интернет/интранет основной акцент пока что делается на разработке инструментальных программных средств. В то же время наблюдается отсутствие развитых средств разработки приложений, работающих с базами данных. Компромиссным решением для создания удобных и простых в использовании и сопровождении информационных систем, эффективно работающих с базами данных, стало объединение Интернет/интранет-технологии с многоуровневой архитектурой. При этом структура информационного приложения приобретает следующий вид: браузер — сервер приложений — сервер баз данных — сервер динамических страниц — web-сервер.

Благодаря интеграции Интернет/интранет-технологии и архитектуры клиент-сервер процесс внедрения и сопровождения корпоративной информационной системы существенно упрощается при сохранении достаточно высокой эффективности и простоты совместного использования информации.

1.4. Жизненный цикл ИС

Жизненный цикл (ЖЦ) любой системы - это непрерывный процесс, который начинается с момента принятия решения о ее создании и заканчивается в момент полного изъятия системы из эксплуатации [6]. Структура ЖЦ ПО ИС в соответствии с международным стандартом **ISO/IEC 12207** базируется на трех группах процессов:

- основные (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем);

- организационные (управление проектами, создание инфраструктуры, усовершенствование, обучение).

Рассмотрим определения некоторых из этих процессов.

Разработка включает все работы по созданию ИС в соответствии с заданными требованиями. Разработка состоит из 4-х этапов:

1) формирование и анализ требований к системе (в результате составляется спецификация системы);
2) концептуальное проектирование (создание информационной модели системы без привязки к типу ЭВМ и системных программных средств);
3) проектирование реализации (выбор вычислительной системы, системных программных средств, проектирование структуры данных);
4) физическая реализация (разработка прикладных программ, базы данных, их отладка и тестирование, написание документации).

Эксплуатация включает все работы по внедрению компонентов ИС, созданию рабочих мест, обучению персонала, а также собственно эксплуатацию, в том числе поиск и устранение проблем, подготовку предложений по развитию и улучшению системы. ***Модернизация ИС*** - это процесс замены отдельных компонент системы, ее проводят в связи с изменениями предметной области, для повышения качества и надежности ИС, для совместимости с другими ИС. ***Сопровождение*** - это поддержание системы в работоспособном состоянии в период эксплуатации. ***Управление проектом*** относится к организационным процессам ЖЦ и связано с планированием работ, созданием коллектива разработчиков, контролем за сроками и качеством работ. ***Верификация*** - это вспомогательный процесс, который состоит в определении того, отвечает ли промежуточный проект требованиям соответствующего этапа.

Каждый процесс ЖЦ характеризуется определенными задачами и методами их решения, исходными данными и результатами. Часто результаты более поздних процессов изменяют исходные данные более ранних этапов, поэтому ЖЦ ИС носит ***итерационный характер.***

1.5. Основные виды обеспечения информационных систем

Общую структуру информационной системы можно рассматривать как совокупность подсистем независимо от сферы применения. В этом случае говорят о ***структурном признаке*** классификации, а подсистемы называют обеспечивающими. Таким образом, структура любой информационной системы может быть представлена совокупностью обеспечивающих подсистем (рис. 1.6).

Среди обеспечивающих подсистем обычно выделяют информационное, техническое, математическое, программное, организационное и правовое обеспечение [7]. Рассмотрим наиболее важные из них более подробно.

Организационное обеспечение. Организационное обеспечение в качестве самостоятельного вида обеспечения стали выделять как один из важнейших компонентов успешной разработки и эффективного функционирования автоматизированных информационных систем (АИС), от которого зависит взаи-

модействие целей и функций системы, аппарата управления и разнообразных ресурсов.

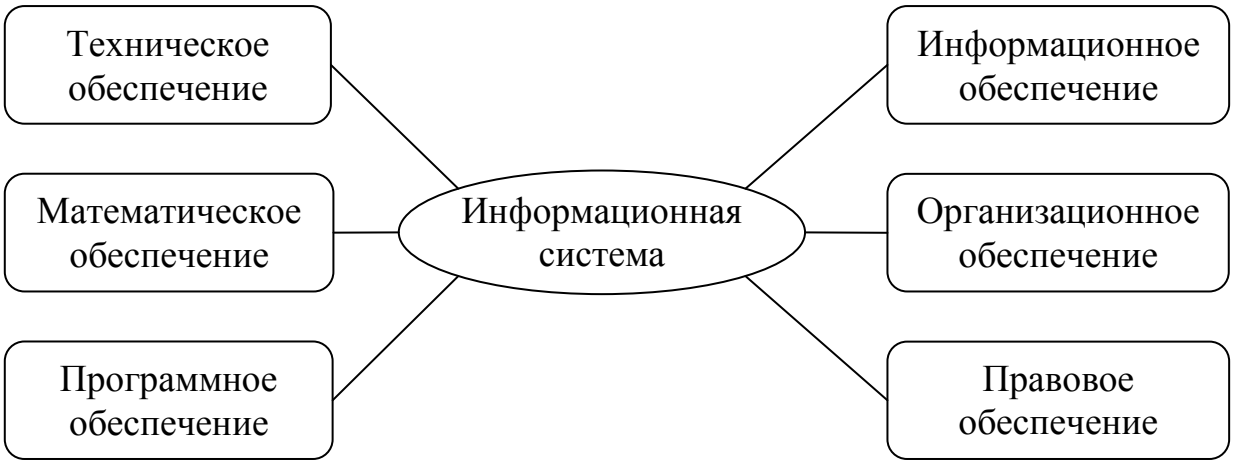


Рис. 1.6. Структура информационной системы как совокупность обеспечивающих подсистем [1,2]

Основная цель организационного обеспечения - анализ существующей системы управления и разработка комплекса организационных решений, направленных на повышение ее эффективности. Оно необходимо для обеспечения взаимодействия персонала АИС как с техническими средствами, так и между собой в процессе решения задач управления.

В составе организационного обеспечения можно выделить:

1) методические материалы, регламентирующие процесс создания и функционирования системы, средства, необходимые для эффективного проектирования и функционирования АИС - комплексы задач, включая типовые, типовые структуры управления, унифицированные формы документов;

2) общегосударственные и отраслевые классификаторы;

3) техническая документация, формируемая в процессе проектного обследования (техническое задание и технико-экономическое обоснование АИС), в ходе технического и рабочего проектирования (технический и рабочий проекты) и в период внедрения (документы, оформляющие поэтапную сдачу системы в эксплуатацию);

4) поскольку выполнение любой работы предполагает наличие исполнителей, необходим коллектив специалистов аппарата управления, осуществляющий процессы анализа данных и принятия решений, а также обработки данных, и занимающийся вопросами разработки и развития самой системы управления объектом, что и составляет четвертую группу структурной схемы организационного обеспечения.

Организационное обеспечение объединяет в единую систему техническое, программное и информационное обеспечение. Для общего руководства разработками создается координационный совет, функциями которого являются: утверждение методологии автоматизации управления, установление

этапов создания АИС, принятие решений по вопросам, возникающим на всех этапах работы и т.п. На предприятии, для которого разрабатывается АИС, создается специализированное подразделение, основное назначение которого - обеспечить организацию работ по подготовке предприятия к внедрению системы управления, контроль за ходом разработок, участие в разработках и внедрении, сопровождение в процесс функционирования системы, развитие и совершенствование системы, участие в подготовке кадров, обучение персонала предприятия методам работы в условиях функционирования АИС.

При создании локальных систем обработки данных (СОД), представляющих собой собственно базы данных или их совокупности, организационное обеспечение создается и ведется администрацией системы управления базами данных.

Информационное обеспечение. Под *информационным обеспечением в* настоящее время принято понимать совокупность данных, языковых средств описания данных, программных средств обработки информационных массивов, а также процедур и методов их организации, хранения, накопления и доступа к ним, обеспечивающих выдачу всей необходимой информации в процессе решения задач, а также справочной информации.

При разработке информационного обеспечения решается ряд проблем. Часть этих проблем связана с необходимостью получения информации, подготовки данных для обработки на ЭВМ и оперирование с ними вне ЭВМ, другая часть - с обработкой, поиском и хранением данных. В соответствии с этим в составе информационного обеспечения можно выделить *внемашинное* и *внутримашинное* информационное обеспечение.

Внемашинное информационное обеспечение. Создание внемашинного информационного обеспечения предполагает определение состава объектов предметной области, их идентификацию, установление свойств объектов, отношений между ними, возникающих в процессе функционирования предприятия, формализацию данных в соответствии с требованиями машинной обработки и разработку правил представления информации в соответствующих документах.

Внемашинное информационное обеспечение включает: *системы классификации и кодирования информации, классификаторы технико-экономической информации, системы унифицированной документации.*

Однозначное формализованное описание всех данных, используемых при решении задач АИС, обеспечивающее автоматизацию процессов хранения, поиска, обработки и выдачи данных без искажения их смысла, осуществляется с помощью средств формализованного описания экономической информации, в состав которых входит и система общегосударственных и локальных классификаторов, с помощью которых идентифицируются материальные объекты, устанавливаются их свойства. Вопросам классификации при разработке информационного обеспечения придается большое значение.

Классификация является результатом упорядоченного распределения объектов заданного множества. Совокупность правил распределения объектов множества на подмножества есть *система классификации.*

В соответствии с этой системой производится классифицирование выбранных объектов (или элементов множества) и выделение группировок, объединяющих часть объектов классификации по одному или нескольким признакам *(класс, подкласс, группа, подгруппа, вид, подвид, тип).*

Всем объектам классификации и классификационным группировкам должно быть присвоено свое кодовое обозначение в принятой системе кодирования. Каждое кодовое обозначение *(код)* есть обозначение объекта или группировки в виде знака или группы знаков по правилам, установленным данной системой кодирования. Таким образом, *классификатор* - это систематизированный свод наименований группировок объектов, признаков и их кодовых обозначений.

На практике используют в основном две системы классификации: *иерархическую* и *фасетную.*

В *иерархической* системе между классификационными группировками устанавливается строгое отношение подчиненности (иерархии). Наиболее сложной проблемой при построении иерархической системы является выбор признаков деления и порядка их следования. Система хорошо приспособлена для ручной обработки данных, обладает большой информативностью кодового обозначения и возможностью создания мнемонических кодов, несущих смысловую нагрузку. Основной ее недостаток - жесткость структуры, обусловленная фиксированностъю признаков и порядка их следования, что заставляет оставлять резерв для обеспечения долговечности классификатора.

При *фасетной* системе некоторый набор признаков сформирован в параллельные независимые фасеты. Конкретные значения признаков внутри фасетов располагаются в классифицированном иерархическом порядке или в виде перечисления. Группировки образуются путем комбинаций значений признаков, взятых из соответствующих фасетов. Последовательность расположения фасетов при образовании группировки задается структурной формулой, определяемой характером решаемых задач и алгоритмов обработки. Система лишена недостатков предыдущей, но представляется весьма сложной и рекомендуется для применения в условиях машинной обработки информации. Признаки, используемые для формирования группировок, должны выбираться соответственно задачам, решаемым в АСУ.

Информация, содержащаяся в классификаторах, представляется в закодированном виде с помощью специальных систем. Системы кодирования, основанные на предварительной классификации, называют *классификационными.* Их делят на *последовательные* и *параллельные.*

Последовательная система соответствует иерархической системе классификации. Значение признака, записанного в виде цифры или буквы на определенном разряде кодового обозначения, зависит от признаков, записанных в предыдущих разрядах, т.е. код нижестоящей группировки образуется путем до-

бавления соответствующего кода к коду вышестоящей группировки. Основной недостаток системы — сложная структура и большая длина кода.

Параллельная система используется только при фасетной классификации. Для обозначения отдельного фасета в случае нестандартного расположения их в соответствии со структурной формулой выделяется определенный разряд или группа разрядов. Значение признака на определенном разряде не зависит от значений признаков на других разрядах. Система дает многоаспектную классификацию, хорошо приспособлена к машинной обработке. Система достаточно гибкая, однако длина кода весьма значительна. Классификационные коды строятся по разрядной (позиционной) или комбинированной системами кодирования.

При *разрядной* системе кодирования каждому классификационному признаку отводится определенное число разрядов, которое зависит от количества предметов кодируемого множества. В комбинированной системе применяются комбинации рациональных систем, например, разрядной и серийной и др., поэтому она является наиболее гибкой. Разновидностью разрядной системы является шахматная, применяется к номенклатурам, характеризующимся двумя признаками, из которых один располагается по вертикали, а другой - по горизонтали.

Системы кодирования, не требующие предварительной классификации, называют регистрационными. Они бывают двух типов: *порядковые* и *серийные.*

Порядковая система применяется для кодирования однопризначных номенклатур и предусматривает присвоение объектам цифр натурального ряда чисел без пропуска номеров.

Серийная система служит для кодирования аналогичных простых номенклатур и предполагает присвоение серий номеров объектам, выделенным в группу, а в пределах серии объектам присваиваются номера по порядку.

При кодировании может применяться также система повторения, в которой используются цифровые и буквенные обозначения, непосредственно характеризующие данный объект (размер, вес и т.д.).

При построении кодов следует также предусмотреть возможность автоматического обнаружения ошибок кодирования с помощью ЭВМ. С этой целью коды объектов дополняются контрольными разрядами, определяемыми по установленному алгоритму. Такие защитные коды называют кодами обнаружения ошибок. Для расчета контрольного разряда наиболее широко используются методы контроля по модулю.

Работа по классификации и кодированию информации реализуется путем создания классификаторов технико-экономической информации. Классификатор устанавливает взаимооднозначное соответствие между кодом объекта и его наименованием. В зависимости от уровня действия все классификаторы, применяемые в АИС, делятся на *общегосударственные* и *локальные*.

При проектировании АИС для каждого предприятия составляется перечень необходимых локальных классификаторов, которые должны быть разработаны с учетом имеющихся общегосударственных классификаторов и специ-

фических особенностей конкретного предприятия, и определяются правила пользования ими. На предприятиях используются следующие основные виды классификаторов: материалов, готовой продукции, оборудования, инструмента, структурных подразделений и т.д.

Для обмена информацией между системами разных уровней управления должна быть учтена возможность перехода от кодов одного классификатора к кодам другого классификатора той же номенклатуры, т.е. требуется унификация и увязка всех применяемых классификаторов. С этой целью разработана единая система классификации и кодирования важнейших номенклатур (ЕСКК), центральным звеном которой является комплекс общегосударственных классификаторов, а для организации работы с различными классификаторами необходимо еще и создание перекодировочных таблиц.

Разработка локальных классификаторов на предприятии начинается с определения методики, в которой приводится состав признаков объектов, содержащихся в классификаторе, устанавливается наличие соподчиненных признаков, выбираются системы классификации и кодирования, разрабатывается структура классификатора и кодового обозначения, определяется система контроля кодовых обозначений, указывается форма взаимодействия с классификаторами других уровней. Затем формируется упорядоченное множество объектов, осуществляется его кодирование; после контроля правильности присвоения кодовых обозначений и анализа полноты представления классифицируемого множества классификатор готовится к печати и размножению, рассылается в подразделения, которые с ним работают. Одновременно определяются правила пользования локальными классификаторами, разрабатывается система их ведения, предназначенная для актуализации классификаторов и их корректировки в централизованном порядке.

При проектировании информационного обеспечения необходимо изучать характеристики потоков информации. Под *информационным потоком* понимается совокупность данных в процессе ее движения в пространстве и во времени. В качестве единицы потока используют *документ, показатель* или *сообщение* (подробнее см. раздел 2.4).

Документы являются основными носителями информации на предприятии и представляют совокупность некоторых элементов, называемых показателями.

Показатель - информационная совокупность, дающая характеристику объекта и определяющая все входящие в совокупности признаки количественно или качественно.

Совокупность взаимосвязанных форм документов, используемых в процессе управления предприятием, называется системой документации.

При разработке АИС документальная форма представления информации частично сохраняется, однако, при этом решаются проблемы унификации и стандартизации документов, а также максимального использования технических средств их получения. Унифицированная система документации представляет собой комплекс взаимосвязанных форм документов и процессов до-

кументирования данных, отвечающих единым правилам и требованиям документооборота, и является средством реализации информационных процессов обмена данными, имеющим нормативно-правовую силу в управлении. При разработке системы документации составляются инструкции по заполнению, ведению и использованию их. Совокупность всех маршрутов движения документов, входящих в систему документации, составляет общую схему документооборота.

Внутримашинное информационное обеспечение СОД и АИС. Внутримашинное ИО включает совокупность информационных массивов и баз данных, процедуры организации, ведения, хранения и обработки баз данных, методы и средства преобразования внешнего представления данных в машинное и обратно, описания хранимой и обрабатываемой информации.

Внутримашинная информационная база является информационным отображением предметной области автоматизируемого объекта и состоит из одной или нескольких баз данных. Под *базой данных* (БД) при этом понимается набор данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными.

Для описания хранимой и обрабатываемой информации базы используются *внешний, концептуальный* и *внутренний* уровни.

На *внешнем уровне* описываются информационные потребности конечного пользователя.

На *концептуальном* (Concept - понятие) уровне описывается полное информационное содержание базы: сведения о структуре обрабатываемой информации и о технологии ее обработки.

Логические взаимосвязи в структуре базы данных организуются в соответствии с типом модели данных. Различают три основных модели: *реляционная, сетевая* и *иерархическая* (подробнее см. раздел 2.3), обладающие разными множествами информационных конструкций [3,4,7].

Сведения о технологии обработки информации включают применяемые методы контроля информации, описание использования потоков информации и описание ограничений на доступ к информации.

Внутренний уровень определяет организацию данных в памяти ЭВМ и методов доступа к ним [8]. Под организацией данных понимается относительно устойчивый порядок расположения записей в памяти ЭВМ и способ обеспечения взаимосвязи между записями.

Например, при последовательной организации записи располагаются в памяти строго одна за другой, а при цепной физически разнесенные в памяти данные связываются в логическую последовательность с помощью специальных указателей - адресов связи, содержащих номер следующей обрабатываемой записи. В зависимости от организации возможны соответствующие методы доступа к данным. Для ускорения доступа (поиска записей) вводится дополнительная организация индексов. При использовании современных программных средств организация данных осуществляется автоматически и требует минимального описания.

В перспективных информационных системах как развитие баз данных предполагается использование *баз знаний,* которые позволят получать сведения, явно не хранящиеся в базах данных. Принципиальными различиями обладают три модели представления знаний - *продукционная модель, фреймовая* и модель *семантических сетей* [9]. Такие системы будут относиться к *интеллектуальным.*

Средства организации и ведения внутримашинной информационной базы. Основными средствами организации и ведения внутримашинной информационной базы на настоящий момент являются *системы управления базами данных* (СУБД). СУБД - это комплекс программ и языковых средств, предназначенных для создания, ведения и использования баз данных.

Создание баз данных производится в два этапа. На первом разрабатывается структура базы данных на основе *информационно-логической модели,* отражающей логическую структуру информации предметной области. На втором осуществляется создание структуры БД средствами СУБД и заполнение базы. Обслуживание данных производится администратором БД, на которого возлагаются функции координации процессов проектирования и эксплуатации БД, обеспечения защиты и целостности данных.

Техническое обеспечение обработки данных. Современное производство требует высоких скоростей обработки информации, удобных форм ее хранения и передачи. Необходимо также иметь динамичные способы обращения к информации, способы поиска данных в заданные временные интервалы, необходимо реализовывать сложную математическую и логическую обработку данных.

Управление сложными техническими комплексами, крупными предприятиями, управление экономикой на уровне страны требует участия в этом процессе достаточно больших коллективов. Такие коллективы могут располагаться в различных районах города, в различных регионах страны и даже в различных странах. Для решения задач управления, обеспечивающих реализацию экономической стратегии, актуальны скорость и удобство обмена информацией, а также возможность тесного взаимодействия всех членов, участвующих в процессе выработки управленческих решений.

В эпоху централизованного использования ЭВМ с пакетной обработкой информации пользователи вычислительной техники предпочитали приобретать компьютеры, на которых можно было бы решать почти все классы их задач. Как правило, сложность решаемых задач обратно пропорциональна их количеству, что приводило к неэффективному использованию ЭВМ при значительных материальных затратах. Доступ к ресурсам компьютеров был затруднен из-за политики централизации вычислительных средств.

Принцип централизованной обработки данных не отвечал высоким требованиям к надежности процесса обработки, затруднял развитие систем и не мог обеспечить необходимые временные параметры при диалоговой обработке данных в многопользовательском режиме. Кратковременный выход из строя центральной ЭВМ приводил к роковым последствиям для системы в целом, т.к.

приходилось дублировать функции центральной ЭВМ, существенно увеличивая затраты на создание и эксплуатацию СОД.

Появление малых ЭВМ, микроЭВМ и, наконец, персональных компьютеров потребовало нового подхода к организации систем обработки данных, к созданию новых информационных технологий. Возникло логически обоснованное требование перехода от использования отдельных ЭВМ в системах централизованной обработки данных к распределенной обработке данных, выполняемой на независимых, но связанных между собой компьютерах. Для реализации распределенной обработки данных были созданы *многомашинные ассоциации,* структура которых разрабатывается по одному из следующих направлений:

• многомашинных вычислительных комплексов;

• компьютерных (вычислительных) сетей.

Многомашинные вычислительные комплексы могут быть:

локальными при условии установки компьютеров в одном помещении и не требующие для взаимосвязи специального оборудования и каналов связи;

дистанционными, если некоторые компьютеры комплекса установлены на значительном расстоянии от центральной ЭВМ, и для передачи данных используются телефонные каналы связи.

Компьютерные сети. Вычислительные сети являются высшей формой многомашинных ассоциаций. Основные отличия вычислительной сети от многомашинного вычислительного комплекса:

1. Размерность. В состав многомашинного вычислительного комплекса входят обычно две, максимум три ЭВМ, расположенные преимущественно в одном помещении. Вычислительная сеть может состоять из десятков и даже сотен ЭВМ, расположенных на расстояниях друг от друга от нескольких метров до десятков, сотен и даже тыс. километров.

2. Разделение функций между ЭВМ. Если в многомашинном вычислительном комплексе функции обработки данных, передачи данных и управления системой могут быть реализованы в одной ЭВМ, то в вычислительных сетях эти функции распределены между различными ЭВМ.

3. Необходимость решения в сети задачи маршрутизации сообщений. Сообщение от одной ЭВМ к другой в сети может быть передано по различным маршрутам в зависимости от состояния каналов связи, соединяющих ЭВМ друг с другом.

Объединение в один комплекс средств вычислительной техники, аппаратуры связи и каналов передачи данных предъявляет специфические требования со стороны каждого элемента многомашинной ассоциации, а также требует формирования специальной терминологии.

Совокупность абонента и станции принято называть *абонентской системой.* Для организации взаимодействия абонентов необходима физическая передающая среда. Абонентами сети могут быть отдельные ЭВМ, комплексы ЭВМ, терминалы, промышленны роботы, станки с ЧПУ и т.д.

На базе физической передающей среды строится *коммуникационная сеть,* которая обеспечивает передачу информации между абонентскими системами.

Такой подход позволяет рассматривать любую вычислительную сеть как совокупность абонентских систем и коммуникационной сети.

Классификация вычислительных сетей. В зависимости от территориального расположения абонентских систем вычислительные сети можно разделить на три основных класса: глобальные сети (WAN); региональные сети (MAN); локальные сети (LAN).

Глобальная вычислительная сеть объединяет абонентов, расположенных в различных странах, на различных континентах. Взаимодействие между абонентами такой сети может осуществляться на базе телефонных линий связи, радиосвязи и систем спутниковой связи. Глобальные вычислительные сети позволят решить проблему объединения информационных ресурсов всего человечества и организации доступа к этим ресурсам.

Региональная вычислительная сеть связывает абонентов, расположенных на значительном расстоянии друг от друга. Она может включать абонентов внутри большого города, экономического региона, отдельной страны. Обычно расстояния между абонентами региональной вычислительной сети составляют десятки - сотни километров.

Локальная вычислительная сеть (ЛВС) объединяет абонентов, расположенных в пределах небольшой территории. В настоящее время не существует четких ограничений на территориальный разброс абонентов локальной вычислительной сети. Обычно такая сеть привязана к конкретному месту. К классу локальных вычислительных сетей относятся сети отдельных предприятий, фирм, банков, офисов и т.д. Протяженность такой сети можно ограничить пределами 2 - 2,5 километров.

Наиболее популярной глобальной компьютерной сетью является компьютерная сеть Internet. В ее состав входит множество свободно соединенных сетей. Внутри каждой сети, входящей в Internet, существует конкретная структура связи и определенная дисциплина управления. Внутри Internet структура и методы соединений между различными сетями для конкретного пользователя не имеют никакого значения.

Персональные ЭВМ, ставшие в настоящее время непременным элементом любой системы управления, привели к буму в области создания локальных вычислительных сетей. Это, в свою очередь, вызвало необходимость в разработке новых информационных технологий. Практика применения персональных ЭВМ в различных отраслях науки, техники и производства показала, что наибольшую эффективность от внедрения вычислительной техники обеспечивают не отдельные автономные ПЭВМ, а локальные вычислительные сети. Локальные вычислительные сети за последнее пятилетие получили широкое распространение в самых различных областях науки, техники, производства.

На базе ЛВС можно создавать системы автоматизированного проектирования - САПР. Это позволяет реализовать новые технологии проектирования изделий машиностроения, радиоэлектроники и вычислительной техники. В условиях развития рыночной экономики появляется возможность создавать конкурентно способную продукцию, быстро модернизировать ее, обеспечивая

реализацию экономической стратегии предприятия. ЛВС позволяют также реализовывать новые информационные технологии в системах организационно-экономического управления. В учебных лабораториях университетов ЛВС позволяют повысить качество обучения и внедрять современные интеллектуальные технологии обучения.

Программное обеспечение систем обработки данных. Программное обеспечение (ПО) систем обработки данных включает в себя программные средства и документацию, необходимую для эксплуатации этих программных средств. ПО разделяют на общесистемное (базовое) и прикладное.

Общесистемное (базовое) ПО предназначено для организации процесса обработки информации в СОД и включает в себя операционную систему (ОС), сервисные программы, системы программирования (комплексные средства разработки программ на языках высокого уровня) и программы технического обслуживания.

Прикладное ПО предназначено для решения конкретных задач СОД. В него входят программные средства общего назначения и специальные программные средства для данной СОД. К *средствам общего назначения* относятся системы управления базами данных (СУБД), табличные процессоры, текстовые и графические редакторы и др. *Специальные* программные средства могут быть как разработаны для конкретной системы обработки данных, так и приобретены готовыми на рынке ПО. При этом необходимое ПО может быть приобретено как «целиком» (если оно удовлетворяет всем необходимым требованиям), так и «собрано» из фрагментов готовых продуктов (возможно, с использованием услуг специалистов, называемых системными интеграторами).

Подробное рассмотрение всех составляющих ПО не является целью настоящего пособия, во-первых, потому что оно предназначено для студентов, изучивших ряд компьютерных курсов, и, во-вторых, потому что этому вопросу посвящена многочисленная литература. В настоящем разделе рассматриваются основные составляющие программного обеспечения современных и перспективных систем обработки данных, а именно:

- операционные системы (ОС);
- системы управления базами данных (СУБД);
- средства разработки приложений (системы программирования);
- инструментальные средства технологии сквозного проектирования (CASE-технологии) (подробнее см. главу 4).

Операционные системы. Поскольку наибольшее распространение в нашей стране получили IBM-совместимые персональные компьютеры, основными операционными системами в настоящий момент являются ОС семейства Windows. Это многозадачные многопоточные 32-разрядные операционные системы с графическими интерфейсами и расширенными сетевыми возможностями. В настоящее время используются Windows 95, ее версия OSR2, Windows 98, а также Windows NT.

Для *Windows 95 (98)* характерны следующие *новые решения. 32-х разрядная архитектура* ОС обеспечивает более высокую производительность системы, снимает многие ограничения на память системных ресурсов. *Механизм управления памятью* обеспечивает работу 32-разрядных приложений в защищенном адресном пространстве с автоматической очисткой памяти после завершения работы каждого приложения. *Вытесняющая многозадачность* позволяет усовершенствовать механизм управления ресурсами: приложение, нуждающееся в ресурсах, может приостановить свою работу до получения ресурса или перейти к выполнению других операций, не останавливая работу других программ. При этом *многопоточное* выполнение отдельной задачи позволяет при задержках одного потока работать со следующим. Под потоком подразумевается частная задача, решаемая внутри процесса, а процессом называется загруженная в память выполняемая прикладная программа, ее адресное пространство и ресурсы. Освоение ОС упростилось благодаря однотипности выполнения всех основных операций и наглядности выполняемых действий.

В Windows 98 интерфейс полностью ориентирован на работу в сети Интернет, а во встроенном пакете Microsoft Office 97 текстовый редактор Word позволяет просматривать и создавать HTML-файлы (файлы на языке разметки гипертекста).

Windows NT - это сетевая ОС, выпускаемая в двух модификациях: Windows NT Server и Windows NT workstation. Windows NT Server предназначена для управления сетевыми ресурсами, содержит средства для работы в глобальных сетях. Windows NT workstation предназначена для работы на локальных компьютерах и рабочих станциях. Обладает повышенной степенью защиты данных от несанкционированного доступа и высокой производительностью при анализе больших объемов данных.

Системы управления базами данных. Система управления базами данных (СУБД), по определению, это комплекс программ и языковых средств, предназначенных для создания, ведения и использования баз данных. До 1995 года значительная часть ПО ИС разрабатывалась с использованием таких СУБД реляционного типа, как Clipper и FoxPro. Для операционных систем Windows в наибольшей степени отвечающими требованиям СОД являются СУБД Visual FoxPro (версия 3.0 и выше) и СУБД MS Access из пакета Microsoft Office. Эти СУБД являются мощными и удобными средствами для разработки приложений баз данных с архитектурой клиент-сервер.

Новые решения. Осуществлен *переход к базе данных,* в которой содержатся все включенные в нее таблицы, их индексы, постоянные связи между таблицами, хранимые процедуры, правила проверки значений полей и действия, выполняемые при добавлении новой записи, удалении и обновлении записи, называемые *триггерами.* Введены *новые средства для обработки данных с помощью SQL (*Structured Query Language - Структурированного Языка Запросов). Введена поддержка значений NULL (в дополнение к FALSE и TRUE) для полей базы данных, предоставлены средства переноса баз данных на SQL-сервер и поддержки работы с удаленными источниками данных.

Одновременно с наличием возможности процедурного пошагового программирования введены средства *объектно-ориентированного программирования.* При объектно-ориентированном подходе реальные предметы и понятия заменяются их моделями, т.е. определенными формальными конструкциями. Формальный характер моделей позволяет определить формальные зависимости между ними, формальные операции над ними и в конечном итоге получить формальную модель разрабатываемой программной системы как композицию моделей ее компонентов. Такой подход обеспечивает возможность модификации отдельных компонентов программного обеспечения без изменений остальных и повторного использования отдельных компонентов при перепроектировании системы. Основными понятиями объектно-ориентированного программирования являются *класс, объект, свойство (атрибут), метод, событие.* Класс содержит информацию о внешнем виде и поведении объекта, иными словами, описывает свойства (атрибуты) и методы обработки событий. Событие же представляет собой действия пользователя или операционной системы, которые распознает объект. Таким образом, управление объектом осуществляется посредством обрабатываемых им событий. При создании нового объекта он наследует характеристики своего класса. Наследование позволяет определять также новые классы (производные, или дочерние) на основе существующих (родительских) классов и добавлять собственные свойства дочерних классов.

Дальнейшее развитие получили средства *визуального программирования.* Разработан новый подход к созданию приложения в целом и использованию мастеров и построителей. Мастера (Wizard) позволяют полностью сконструировать любую новую компоненту, включая проектирование баз данных, отчетов, экранных форм, а с помощью построителей в экранную форму может быть встроен любой элемент управления.

На новом уровне организована поддержка *OLE-технологии* (Object Linking and Embedding - Связывания и Встраивания Объектов), добавлена возможность встраивания OLE-объектов в экранные формы и сохранения их в полях базы данных.

Реализована возможность *технологии перемещения и сброса объектов* (drag-and-drop), возможность перемещения таблиц и полей данных в экранные формы непосредственно из диспетчера проекта или из окна базы данных, использование контекстного меню.

Современные средства разработки приложений. Не останавливаясь на эволюции средств программирования под Windows, можно лишь сказать, что до 1994 г. все средства, позволявшие создавать приложения под Windows, требовали от программиста глубокого знания архитектуры и принципов работы этой операционной системы. При этом не существовало систем, которые позволяли бы достаточно просто работать с базами данных, обеспечивая должный уровень интерфейса. В 1994 г. появилась созданная фирмой Borland (ныне Inprise) принципиально новая система - среда визуальной разработки приложений Delphi, основанная на использовании несколько расширенной версии языка

Borland Pascal, получившей название Object Pascal. В 1997 г. появилась и еще одна система фирмы Borland - C++Builder, использующая язык ANSI C++ с некоторыми расширениями (кроме того, в этой системе есть и встроенный компилятор языка Object Pascal), также работающая под Windows 95/NT. Эти системы имеют интегрированную среду разработки (IDE), то есть включают в себя редакторы кода, редакторы визуальных компонентов, компиляторы (в C++Builder их даже два - C++ и Object Pascal), отладчики, средства помощи и т.п. В обеих системах используются объектно-ориентированные языки программирования высокого уровня и встроенные в них возможности работы с базами данных, не уступающие по своей мощи возможностям СУБД типа Clipper или FoxPro. Существует также возможность использования языка SQL (и, следовательно, возможность создания баз данных с удаленным доступом).

Новые концепции. Основной концепцией в этих системах является концепция *объектно-ориентированного программирования.* Одним из ключевых понятий при этом является понятие компонентов, т.е. готовых шаблонов для всех стандартных элементов приложения Windows (стандартных диалогов, окон, кнопок, списков и др.) поставляемых с системами; на их основе можно создавать свои собственные компоненты. Компоненты предоставляют программисту уже готовый интерфейс с Windows API, в них введено понятие события, которое программист обрабатывает вместо перехвата сообщений Windows API (например, для обработки нажатия пользователем кнопки программисту надо написать примерно следующее: «при нажатии сделать то-то и то-то», а не перехватывать посланные откуда-то куда-то неудобоваримые сообщения Windows). При этом прямая работа с Windows API отнюдь не запрещена. Напротив, для этого программисту предоставляются более удобные методы, чем, скажем, в системе Visual C++ с MFC.

С обеими системами поставляется библиотека визуальных компонентов (VCL), в которой содержатся шаблоны всех стандартных визуальных элементов Windows (а также многих специальных), так что программисту остается лишь незначительно изменять их по своему вкусу. Сам программист может создавать подобные шаблоны, и система не будет делать никаких различий между «родными» компонентами и добавленными. Кроме того, естественно, при помощи систем можно создавать (и регистрировать) свои собственные DLL и статические библиотеки.

Важной особенностью систем является возможность использования объектов OLE (или DDE), то есть можно, например, редактировать в своем приложении документ Word средствами самого Word. Хотя Delphi и C++Builder и не создавались как системы для работы с Internet и Web-дизайна, в них есть некоторые возможности и для этого.

Совместимость. Системы Delphi и C++Builder практически полностью совместимы в одну сторону благодаря наличию в C++Builder встроенного компилятора Object Pascal приложения, созданные в Delphi, можно компилировать в C++Builder; более того, можно использовать даже отдельные модули Delphi, причем вперемешку с модулями, написанными на C++ (некоторые проблемы с

совместимостью все-таки существуют, но они несущественны). В Delphi нет компилятора C++, однако можно очень много создавать в C++Builder для последующего использования в Delphi, например, компоненты, DLL, (и, естественно, наоборот). Переносу различных блоков между системами способствует то, что в обоих применяются абсолютно одни и те же концепции и подходы. Можно смело сказать, что Delphi и C++Builder - системы уникальные по уровню совместимости.

Новые приемы программирования. Системы Delphi и C++Builder представляют собой визуальные средства разработки приложений. Это значит, что при создании приложения программист сразу же видит свое приложение именно в том виде, в котором его увидит и будущий пользователь. При этом программист может применять принципиально новые методы создания программ. Так добавление компонентов в приложение осуществляется методом drag-and-drop, то есть при помощи мыши выбираются нужные компоненты, а потом перетаскиваются в окно будущего приложения (окно также является компонентом, называемым в системах «формой», и его можно тоже выбирать по своему вкусу). Далее опять же при помощи мыши компоненты растягиваются до нужных размеров, перекладываются на форме и при этом все изменения автоматически фиксируются в коде программы, так что программу можно запускать в любой момент. Кроме того, большая часть свойств компонентов отображается при проектировании на экране в удобном страничном диалоге (называется он Object Inspector), и все изменения на форме можно видеть в нем сразу же, а изменения, вносимые непосредственно в него, сразу же отображаются на форме проектируемого приложения (в этом и состоит принцип двойственного ввода данных). Благодаря подобному подходу можно создать полноценный интерфейс даже для большого приложения, не написав ни единой строчки кода. При этом грамотно выбрав компоненты и надлежащим образом связав их (щелкая мышью в клетках Object Inspector), можно создать даже приложение, работающее с несколькими связанными таблицами баз данных, которое будет нужным образом фильтровать и отсортировывать данные из них.

Программное обеспечение для разработки ИС. Для решения задач разработки, сопровождения и модернизации информационных систем создаются технологии сквозного проектирования (ТСП). Эти технологии представляют собой набор компонент - программных продуктов и методов разработки, основные из которых и являются предметом нашего рассмотрения.

Интегрированный CASE-инструментарий. Он предназначен для коллективной разработки, охватывающей все этапы жизненного цикла системы (от подготовки технического задания до генерации программного кода, внедрения и эксплуатации приложений).

Процесс проектирования начинается с формализации общих требований, предъявляемых к информационной системе, и предусматривает постепенную конкретизацию замысла с использованием механизмов декомпозиции и перехода к детальным техническим решениям вплоть до получения готового программного кода, состава и топологии аппаратных средств проектируемой си-

стемы. В каждом продукте среда проектирования поддерживается удобным и легким в работе универсальным графическим редактором. Результатом выполнения проекта является:

• база данных проектируемой системы, включающая все необходимые физические объекты (таблицы, триггеры, хранимые процедуры), построенная с учетом проектируемой политики поддержания целостности данных;

• исходный код приложений информационной системы - программы, реализующие пользовательский интерфейс и логику приложений.

Разработка проекта поддерживается репозитарием, который обеспечивает централизованное хранение данных, проверку данных на взаимную непротиворечивость и полноту, сопровождение версий разработок. Репозитарий поддерживает реальный многопользовательский режим для групп разработчиков. В качестве репозитария используется специальная проектная база данных.

*Настраиваемые кодогенераторы***.** Позволяют с учетом возможностей CASE-инструментов получать исходный программный код приложений, доступных для последующего редактирования.

*Средства реинжиниринга и репроектирования***.** Дают возможность не только «прочитать» структуру имеющейся базы данных, но и установить значения по умолчанию для управляющих параметров кодогенерации. С помощью удобного графического интерфейса в среде MS Windows можно управлять процессом кодогенерации и получать новые версии приложений.

Среда разработки приложений. Представлена в первую очередь продуктами для информационных систем, построенных в архитектуре «клиент-сервер» с использованием языков программирования 4-го поколения,

СУБД и операционные системы. В основном используются серверы реляционных баз данных (SQL-серверы) и ОС UNIX, в качестве СУБД могут использоваться серверы Informix. ОС UNIX является основой реализации любого сложного проекта, поскольку органично сочетает все необходимые сервисы и предоставляет платформу для функционирования и интеграции современных программных продуктов. Для решения локальных задач обработки информации наряду с UNIX возможно применение и MS Windows NT.

Обеспечивающая часть АИС (АСУ). При проектировании структуры обеспечивающей части АИС (АСУ) необходимо выбрать виды обеспечения и организовать взаимодействие между ними таким образом, чтобы они обеспечивали реализацию задач, входящих в подсистемы функциональной части АИС. При выборе технического и программного обеспечения учитываются особенности предприятия, его финансовые возможности, объемы информационных массивов, квалификации сотрудников и т.п. В истории развития АИС первоначально был период, когда после разделения обеспечивающей части на составляющие отдельно разрабатывалось организационное (ОргО), информационное (ИО), техническое (ТО) и программное обеспечение (ПО). В результате такой независимой организации разработки структур этих видов обеспечения возникла проблема их совместимости. Поэтому в последующем был принят принцип единства ОргО, ТО, ИО и ПО.

1.6. Пользователи ИС

Пользователей ИС можно разделить на следующие **группы:**

- случайный пользователь, взаимодействие которого с ИС не обусловлено служебными обязанностями;
- конечные пользователи (потребители информации) - лицо или коллектив, в интересах которых работает ИС. Они работают с ИС повседневно, связаны с жестко ограниченной областью деятельности и, как правило, они не являются программистами, например, это бухгалтеры, экономисты, руководители подразделений;
- коллектив специалистов (персонал ИС), включающий администратора банка данных, системного аналитика, системных и прикладных программистов.

Рассмотрим более подробно состав и функции персонала ИС.

Администратор - это специалист (или группа специалистов), который понимает потребности конечных пользователей, работает с ними в тесном контакте и отвечает за определение, загрузку, защиту и эффективность работы банка данных. Он должен координировать процесс сбора информации, проектирования и эксплуатации БД, учитывать текущие и перспективные потребности пользователей.

Системные программисты занимаются разработкой и сопровождением базового программного обеспечения ЭВМ (ОС, СУБД, трансляторов, сервисных программ общего назначения).

Прикладные программисты разрабатывают программы для реализации запросов к БД.

Аналитик строит математическую модель предметной области, исходя из информационных потребностей конечных пользователей; ставит задачи для прикладных программистов. На практике персонал небольших ИС часто состоит из одного - двух специалистов, которые выполняют все перечисленные функции.

1.7. Области применения информационных систем

В последние несколько лет компьютер стал неотъемлемой частью управленческой системы предприятий. Однако современный подход к управлению предполагает еще и вложение денег в информационные технологии. Причем чем крупнее предприятие, тем больше должны быть подобные вложения.

Благодаря стремительному развитию информационных технологий наблюдается расширение области их применения. Если раньше чуть ли не единственной областью, в которой применялись информационные системы, была автоматизация бухгалтерского учета, то сейчас наблюдается внедрение информационных технологий во множество других областей. Эффективное использование корпоративных информационных систем позволяет делать более точные прогнозы и избегать возможных ошибок в управлении.

Из любых данных и отчетов о работе предприятия можно извлечь массу полезных сведений. Информационные системы как раз и позволяют извлекать максимум пользы из всей имеющейся в компании информации.

Именно этим фактом и объясняются жизнеспособность и бурное развитие информационных технологий — современный бизнес крайне чувствителен к ошибкам в управлении, и для принятия грамотного управленческого решения в условиях неопределенности и риска необходимо постоянно держать под контролем различные аспекты финансово-хозяйственной деятельности предприятия (независимо от профиля его деятельности). Поэтому можно вполне обоснованно утверждать, что в жесткой конкурентной борьбе большие шансы на победу имеет предприятие, использующее в управлении современные информационные технологии.

Рассмотрим наиболее важные задачи, решаемые с помощью специальных программных средств [6].

Бухгалтерский учет. Это классическая область применения информационных технологий и наиболее часто реализуемая на сегодняшний день задача. Такое положение вполне объяснимо. Во-первых, ошибка бухгалтера может стоить очень дорого, поэтому очевидна выгода использования возможностей автоматизации бухгалтерии. Во-вторых, задача бухгалтерского учета довольно легко формализуется, так что разработка систем автоматизации бухгалтерского учета не представляет технически сложной проблемы.

Управление финансовыми потоками. Внедрение информационных технологий в управление финансовыми потоками также обусловлено критичностью этой области управления предприятия к ошибкам. Неправильно построив систему расчетов с поставщиками и потребителями, можно спровоцировать кризис наличности даже при налаженной сети закупки, сбыта и хорошем маркетинге. И наоборот, точно просчитанные и жестко контролируемые условия финансовых расчетов могут существенно увеличить оборотные средства фирмы.

Управление складом, ассортиментом, закупками. Далее, можно автоматизировать процесс анализа движения товара, тем самым, отследив и зафиксировав те двадцать процентов ассортимента, которые приносят восемьдесят процентов прибыли. Это же позволит ответить на главный вопрос — как получать максимальную прибыль при постоянной нехватке средств? «Заморозить» оборотные средства в чрезмерном складском запасе — самый простой способ сделать любое предприятие, производственное или торговое, потенциальным инвалидом. Можно просмотреть перспективный товар, вовремя не вложив в него деньги.

Управление производственным процессом. Управление производственным процессом представляет собой очень трудоемкую задачу. Основными механизмами здесь являются планирование и оптимальное управление производственным процессом.

Автоматизированное решение подобной задачи дает возможность грамотно планировать, учитывать затраты, проводить техническую подготовку

производства, оперативно управлять процессом выпуска продукции в соответствии с производственной программой и технологией.

Очевидно, что чем крупнее производство, тем большее число бизнес-процессов участвует в создании прибыли, а значит, использование информационных систем жизненно необходимо.

Управление маркетингом. Управление маркетингом подразумевает сбор и анализ данных о фирмах-конкурентах, их продукции и ценовой политике, а также моделирование параметров внешнего окружения для определения оптимального уровня цен, прогнозирования прибыли и планирования рекламных кампаний. Решение большинства этих задач могут быть формализованы и представлены в виде информационной системы, позволяющей существенно повысить эффективность управления маркетингом.

Документооборот. Документооборот является очень важным процессом деятельности любого предприятия. Хорошо отлаженная система учетного документооборота отражает реально происходящую на предприятии текущую производственную деятельность и дает управленцам возможность воздействовать на нее. Поэтому автоматизация документооборота позволяет повысить эффективность управления.

Оперативное управление предприятием. Информационная система, решающая задачи оперативного управления предприятием, строится на основе базы данных, в которой фиксируется вся возможная информация о предприятии. Такая информационная система является инструментом для управления бизнесом и обычно называется корпоративной информационной системой.

Информационная система оперативного управления включает в себя массу программных решений автоматизации бизнес-процессов, имеющих место на конкретном предприятии. Одно из наиболее важных требований, предъявляемых к таким информационным системам, — гибкость, способность к адаптации и дальнейшему развитию.

Предоставление информации о фирме. Активное развитие сети Интернет привело к необходимости создания корпоративных серверов для предоставления различного рода информации о предприятии. Практически каждое уважающее себя предприятие сейчас имеет свой web-сервер. Web-сервер предприятия решает ряд задач, из которых можно выделить две основные:

- создание имиджа предприятия;
- максимальная разгрузка справочной службы компании путем предоставления потенциальным и уже существующим абонентам возможности получения необходимой информации о фирме, предлагаемых товарах, услугах и ценах.

Кроме того, использование web-технологий открывает широкие перспективы для электронной коммерции и обслуживания покупателей через Интернет.

В настоящее время архитектура клиент-сервер получила признание и широкое распространение как способ организации приложений для рабочих групп и информационных систем корпоративного уровня. Подобная организация работы повышает эффективность выполнения приложений за счет использования

возможностей сервера БД, разгрузки сети и обеспечения контроля целостности данных.

Двухуровневые схемы архитектуры клиент-сервер могут привести к некоторым проблемам в сложных информационных приложениях с множеством пользователей и запутанной логикой. Решением этих проблем может стать использование многоуровневой архитектуры.

1.8. Контрольные вопросы

1. Дайте определение понятия "информационная система".
2. Перечислите основные информационные процессы.
3. Дайте характеристику ИС первого поколения.
4. Дайте характеристику ИС второго поколения.
5. Дайте характеристику ИС третьего поколения.
6. Перечислите признаки классификации ИС.
7. Используя схему на рис. 1.1, опишите принципы функционирования ИПС.
8. Опишите структуру банка данных.
9. Опишите структуру банка знаний и хранилищ данных.
10. Дайте характеристику основных функциональных компонентов ИС.
11. Какова типовая структура ИС?
12. Что понимают под информационным обеспечением ИС и каково его содержание?
13. Что понимают под техническим обеспечением ИС и каково его содержание?
14. Что понимают под программным обеспечением ИС и каково его содержание?
15. Дайте классификацию пользователей ИС.
16. Приведите примеры областей применения ИС.

ГЛАВА 2. ОБЩИЕ ПРИНЦИПЫ МОДЕЛИРОВАНИЯ ПРОЦЕССОВ И СИСТЕМ

2.1. Типы информационных моделей

Понятие модели столь широко используется в повседневной жизни, что приобрело очень много смысловых оттенков. Это и "Дом моделей" известного кутюрье, и модель престижной марки автомобиля, и модель политического руководства, и математическая модель колебаний маятника. Применительно к ИС под ***моделью будем понимать некоторое представление о системе, отражающее наиболее существенные закономерности ее структуры и процесса функционирования и зафиксированное на некотором языке или в другой форме.***

Наиболее общей моделью системы является так называемая **модель "черного ящика"** [10]. В этом случае система представляется в виде прямоугольника, внутреннее устройство которого скрыто от аналитика или неизвестно. Однако система не является полностью изолированной от внешней среды, поскольку последняя оказывает на систему некоторые информационные или материальные воздействия. Такие воздействия получили название *входных воздействий*. В свою очередь, система также оказывает на среду или другие системы определенные информационные или материальные воздействия, которые получили название *выходных воздействий*. Такая модель также часто называется моделью окружающей среды. Ценность моделей, подобных модели "черного ящика", весьма условна. Они позволяют дать только самое общее представление о системе, выделить ее из внешней среды.

Процесс разработки адекватных моделей и их последующего конструктивного применения требует не только знания общей методологии системного анализа, но и наличия соответствующих изобразительных средств или языков для фиксации результатов моделирования и их документирования. Очевидно, что естественный язык не вполне подходит для этой цели, поскольку обладает неоднозначностью и неопределенностью. Для построения моделей были разработаны достаточно серьезные теоретические методы, основанные на развитии математических и логических средств моделирования, а также предложены различные формальные и графические нотации, отражающие специфику решаемых задач.

Информационные модели представляют собой формальные или полуформальные способы описания информации, включающие совокупность правил структурирования данных, правил выполнения операций над ними, а также ограничений целостности, которые описывают допустимые связи и значения данных, допустимые последовательности их изменения.

Примеров моделей можно привести достаточно много. Например, аэродинамическая модель гоночного автомобиля или проектируемого самолета, модель ракетного двигателя, модель колебательной системы, модель системы электроснабжения региона, модель избирательной компании и др. Общим

свойством всех моделей является их подобие оригинальной системе или системе-оригиналу. Важность построения моделей заключается в возможности их использования для получения информации о свойствах или поведении системы-оригинала. При этом процесс построения и последующего применения моделей для получения информации о системе-оригинале получил название ***моделирование***.

Сама ИС является своеобразной моделью некоторой системы реального мира, поэтому при создании ИС всегда решается задача сбора и представления в том или ином виде информации о деятельности, функционировании этой системы. При этом обычно рассматриваются следующие аспекты:

- объекты, с которыми оперирует система («что, какие данные должны отображаться в ИС»);
- процессы, которые она выполняет («как должна обрабатываться информация в ИС»);
- события, которые влияют на изменение процессов и объектов («когда происходит изменение системы»);
- место, где должны размещаться части системы.

В соответствии с этим можно выделить несколько классов информационных моделей: **структурные, функциональные, поведенческие, архитектурные.**

2.1.1. Структурные модели

Структурная модель - это описание структуры предметной области в виде совокупности взаимосвязанных объектов или процессов (состав объектов, их свойства и связи). В этом классе моделей внимание разработчиков ИС сосредоточено на том, КАКАЯ информация должна обрабатываться системой, а не на том, ГДЕ она размещается и КАК обрабатывается.

Среди этих моделей чаще всего выделяют следующие разновидности:

- диаграммы сущность – связи (**ER**-диаграммы, **IDEF1x**),
- диаграммы потоков данных (**Data Flow Diagrams** - **DFD**),
- семантические модели (модели знаний),
- объектно-ориентированные модели.

ER-диаграммы (Entity - Relationship Diagrams, **ERD**) являются одним из основных методов концептуального проектирования баз данных, к ним относятся диаграммы Чена, **IDEF1x** - для моделирования реляционных баз данных и др. разновидности. При построении этих моделей используют такие понятия, как сущности, атрибуты и связи. *Сущность* – это отдельный элемент предметной области (сотрудник, предмет, событие, абстрактное понятие), который должен быть представлен в базе данных, то есть информация о нем должна *храниться* в системе. *Атрибут* – это некоторое свойство объекта, а *связь* отображает ассоциативное отношение между сущностями.

DFD применяются для описания структуры потоков информации внутри системы; примером таких диаграмм служат диаграммы Гэйна – Сарсона, модель **IDEF1 (Information Modeling).**

Семантические модели призваны отобразить смысл (семантику) данных [9], их главное отличие от других моделей состоит в том, что наряду со структурой хранимых данных они описывают правила получения такой информации, которая в явном виде не хранится в системе. Эти модели используются обычно при разработке интеллектуальных ИС. Например, модель **IDEF5 (Ontology Description Capture)** описывает онтологию моделируемой системы («**Онтология**» включает в себя список всех терминов предметной области; правила комбинирования терминов в разных ситуациях; правила получения новых знаний). Среди семантических моделей обычно выделяют следующие разновидности**:**

- логическая модель описывает объекты и операции над ними в виде предикатов первого порядка, является строго формальной, применяет метод логического вывода новых знаний “от цели к данным”;
- продукционная модель описывает знания в виде набора фактов и правил вида “если - то”, позволяет учитывать неопределенность знаний, использует эвристические правила вывода;
- семантическая сеть описывает знания в виде бинарных типизированных отношений между объектами и наглядно изображается в виде графа (вершина - объект, дуга - связь между двумя объектами);
- фреймовая модель - это семантическая сеть с N-арными отношениями между объектами и присоединенными процедурами, которые реализуют операционные знания.

Объектно-ориентированная модель является дальнейшим развитием фреймовой модели и ER-модели, она позволяет описывать как состояние объекта, так и его поведение. Говорят, что объект «инкапсулирует» состояние и поведение. Имеется несколько разновидностей этих моделей, например, **IDEF4 (Object-Oriented Design)** - метод объектно-ориентированного проектирования системы.

2.1.2. Функциональные модели

Функциональные модели описывают действия над объектами и методы их преобразования. Этот класс моделей описывает, КАК части системы взаимодействуют друг с другом, КАК работают отдельные части ИС. Например, функциональная диаграмма **IDEF0** (**Function Modeling)** представляет изучаемую систему в виде набора взаимосвязанных функций, обычно эта модель строится на самом первом этапе изучения любой системы. Другая модель - **IDEF3 (Process Description Capture)** - описывает сценарий и последовательность операций для каждого процесса, а также показывает, какие пользователи работают с системой.

2.1.3. Поведенческие модели

Этот класс моделей отражает изменение состояния объектов ВО ВРЕМЕНИ в результате некоторых событий. Состояние объекта в какой-либо момент времени описывается набором значений его свойств. Поведение объекта описывается в виде набора действий, связанных с событиями в предметной

области. Для описания поведения ИС применяют событийные графы и матрицы, диаграммы потоков событий, а также «раскрашенные сети Петри», получаемые из **IDEF0**.

2.1.4. Архитектурные модели

Этот вид моделей позволяет показать, ГДЕ размещаются отдельные части системы и описать механизм их взаимодействия. Например, к этому виду можно отнести модель «клиент-сервер», модель «логистические сети» и др.

В заключение заметим, что существуют смешанные модели, сочетающие разные аспекты, например, структурно-функциональные, структурно-событийные и т.п.

2.2. МОДЕЛИ ДАННЫХ

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

Разные люди могут иметь разное представление об одном и том же предмете, поэтому для разных классов пользователей ИС можно выделить несколько видов представлений об информации. Эти представления отличаются как профессиональными потребностями пользователей (см. раздел 1.6), так и уровнем детальности.

В настоящее время чаще всего используется **трехуровневое представление** данных, предусматривающее

- уровень бизнеса (внешнее представление, характерное для конечного пользователя, для «заказчика» ИС),
- системный уровень (концептуальное представление, свойственное администратору ИС и системному аналитику);
- программный уровень (внутреннее, или физическое представление, которое характерно для разработчика ИС, программиста).

Таким образом, система представляется Вам в разном виде в зависимости от того, кем Вы являетесь. Кроме того, каждое из представлений может быть дано в разных аспектах (структура, функции, поведение и т.д.).

Внешнее представление данных - это описание информационных потребностей конечного пользователя (КП). Каждая группа конечных пользователей имеет свое представление о нужной для них информации, поэтому внешних представлений столько, сколько групп пользователей работает с ИС. Внешние представления отличаются друг от друга составом, формой отображения данных, детальностью.

Концептуальное представление данных это наиболее полное представление, отражающее смысл информации, оно может быть только одно и не должно содержать противоречий и двусмысленностей. Описание предметной области, выполненное без ориентации на используемые в дальнейшем про-

граммные и технические средства, называется **инфологической моделью предметной области (ИЛМ).** Фактически – это такое описание системы, которое не зависит от любых подробностей реализации ИС (таких, как тип СУБД, состав прикладных программ, язык программирования, вычислительная платформа и т.п.). Обычно инфологическую модель строят с помощью специальных формализованных методик. Основные требования к ИЛМ:

- адекватность отображения предметной области;
- легкость расширения и модификации;
- возможность иерархического представления данных;
- возможность автоматизированного проектирования;
- однозначность понимания и легкость восприятия всеми лицами, участвующими в разработке и эксплуатации системы.

ИЛМ включает в себя следующие компоненты:

- структурную информационную модель предприятия;
- описание потребностей пользователей (список запросов, объем данных, частота обращения, режим работы пользователей);
- описание ограничений целостности данных;
- семантическую информацию о данных (онтология);
- информацию о мерах обеспечения безопасности и целостности данных,
- описание алгоритмов вычислений, последовательности выполнения операций и т.д.

Разработчики ИС обычно используют более детальное описание концептуальной модели, тем не менее, еще считающееся логическим.

Для описания логики работы программ, их структуры, их взаимодействия с данными и друг с другом применяют схемы программ, схемы структуры программ, схемы данных, схемы взаимодействия программ. Правила изображения этих схем описаны государственными стандартами ЕСПД [21].

При создании БД выделяют как особое представление *даталогическую модель базы данных* - это модель логического уровня, использующая одну из типовых **моделей данных,** и учитывающая требования конкретной СУБД.

Модели данных на основе записей представляют базу данных в виде совокупности записей фиксированного формата, которые могут иметь разные типы. Каждый тип записей содержит фиксированное количество полей заданной длины. Существует несколько типовых моделей данных, которые используются современными СУБД [8]: реляционная, сетевая, иерархическая, модель на основе инвертированных списков, многомерная, объектно-реляционная.

Внутреннее представление (физическая модель) – описывает представление данных на физическом носителе информации в компьютере [8]. Этот уровень служит для достижения оптимальной производительности и экономного использования дисковой памяти. Так, например, в даталогической модели БД может быть указан тип и допустимая длина единиц информации, но только в физической модели становится известен их реальный размер и размещение на дисках. На этом уровне описывают:

- распределение дискового пространства для хранения данных и программ;
- подробности хранения информации с указанием реального объема элементов данных;
- сведения о сжатии данных и методах их шифрования и т.д.

В настоящее время внутренние модели все чаще строятся автоматически, с помощью автоматизированных средств разработки ИС, современных языков программирования, СУБД.

Таким образом, многоуровневое представление информации ИС обусловлено потребностями различных групп пользователей и современным уровнем развития инструментальных средств создания ИС. Оно позволяет разделить работу по созданию и обслуживанию ИС на относительно независимые части.

На рис. 2.1 представлена классификация моделей данных [8].

Типы инфологических моделей, использующихся для описания структур, более подробно мы рассмотрим ниже, в разделе 3.7. Здесь же рассмотрим основные виды даталогических моделей.

2.2.1. Документальные модели данных

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

Модели, основанные на языках разметки документов, связаны прежде всего со стандартным общим языком разметки — SGML (Standart Generalised Markup Language), который был утвержден ISO в качестве стандарта еще в 80-х годах. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тегов (ссылок), их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования тегов осуществляется при помощи специального набора правил, называемых DTD-описаниями, которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду некоторой своей сложности SGML использовался в основном для описания синтаксиса других языков (наиболее известным из которых является HTML), и немногие приложения работали с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор инструкций — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране программы-клиента и определяют этим самым способ представления документа, но не его структуру.

Рис. 2.1. Классификация моделей

В качестве элемента гипертекстовой базы данных, описываемой HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеет возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его сегодня главным механизмом представления информации в Интернете.

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

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

Тезаурусные модели основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

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

2.2.2. Теоретико-графовые модели данных

Иерархическая модель данных является наиболее простой среди всех даталогических моделей. Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает как родительский, а с ним может быть связано множество подчиненных объектов. Иерархия проста и естественна в отображении взаимосвязи между классами объектов.

Основными информационными единицами иерархической модели являются *база данных (БД), сегмент и поле*.

Поле данных определяется как минимальная, неделимая единица данных доступная пользователю с помощью СУБД.

Сегмент называют *записью*, которая содержит два понятия: *тип сегмента (записи)* и *экземпляр сегмента*.

Тип сегмента – поименованная совокупность типов элементов данных в него входящих.

Экземпляр сегмента образован из конкретных значений полей или элементов данных в него входящих.

Схема иерархической базы данных (ИБД) представляет собой совокупность отдельных деревьев, каждое дерево называется *физической БД*. Каждая физическая БД удовлетворяет следующим иерархическим ограничениям:

- в каждой физической БД существует один корневой сегмент, т. е. сегмент у которого нет родительского исходного типа сегмента;
- каждый логический исходный сегмент может быть связан с произвольным числом логически подчиненных сегментов;
- каждый логически подчиненный сегмент связан только с одним логически исходным родительским сегментом.

Каждый тип сегмента может иметь множество соответствующих ему экземпляров. Между экземплярами сегмента также существует иерархическая связь.

При работе с иерархической моделью каждая программа, пользователь или приложение определяют свою внешнюю модель. Внешняя модель представляет собой совокупность поддеревьев для физической БД, с которой работает пользователь. Каждый подграф внешней модели в обязательном порядке должен содержать корневой тип сегмента соответствующей БД.

Достоинства: простота понимания и использования иерархической модели, наивысшая скорость поиска информации, т.е. ответы на запросы.

Недостатки: взаимосвязи типа "многие-ко-многим" трудно реализуемы, из-за строгой иерархичности усложняется операция добавления и удаления, доступ к любому порожденному узлу возможен только через корневой.

Сетевая модель. Сетевая модель во многом подобна иерархической, которая собственно является подмножеством сетевых моделей данных. Базовым объектом является *элемент, агрегат, запись и набор данных*.

Элемент данных имеет тоже самое понятие, что и в иерархической модели.

Агрегат данных соответствует следующему уровню обобщения и делится на агрегат типа *вектор* и агрегат типа *повторяющаяся группа*.

Агрегат типа *вектор* соответствует линейному набору элементов данных. Например, агрегат **Адрес: Город, Улица, Дом, Квартира**.

Агрегат типа повторяющаяся группа соответствует совокупности векторов данных. Например, агрегат **Зарплата: Месяц, Сумма** – соответствует повторяющейся группе с числом повторений 12.

Запись – совокупность агрегатов или элементов данных, моделирующая некоторый класс объектов реального мира. Данное понятие соответствует понятию *сегмент* в иерархической модели.

Набор – двухуровневый граф, связывающий отношением "один-ко-многим" два типа записи. Набор отражает иерархическую связь между двумя записями. Родительский тип записи в данном наборе называется владельцем набора, а дочерний тип записи – членом того же набора.

Достоинства: простота реализации связи "многие-ко-многим", доступ к данным может быть осуществлен различными путями.

Недостатки: потеря независимости данных при реализации базы данных, сложность управления данными.

Для получения более подробной информации по теоретико-графовым моделям отправляем читателя к [4,5,8].

2.2.3. Теоретико-множественные модели данных

Среди теоретико-множественных моделей данных следует выделить реляционную, модель, основанную на инвертированных списках, постреляционную, многомерную [4,5,8].

Реляционная модель данных. Реляционную модель данных отличает простота и наглядность со стороны программистов и серьезное теоретическое обоснование. Кроме того, развитие формального аппарата представления и манипуляция данными сделали данную модель наиболее перспективной для использования.

Теоретической основой модели стала теория отношений, сформулированная американцем Чарльзом Содерсом Пирсом и немцем Эрнстом Шредером. Однако основные понятия реляционной модели были заложены в 1970 г. Э.Ф. Коддом [11].

Основной структурой данных в данной модели является отношение, именно поэтому модель получила название *реляционной* (от английского *relation* – отношение). *N-арным отношением* R называют подмножество декартова произведения $D_1 \times D_2 \times \ldots \times D_n$ множеств D_1, D_2, … , D_n ($n \geq 1$), необязательно различных. Исходные множества D_i называют в модели *доменами*.

Вхождение домена в отношение называют *атрибутом*. Строки отношения называются *кортежами*. Количество атрибутов в отношении называется *степенью*, или *рангом,* отношения.

Любое отношение является динамической моделью некоторого реального объекта внешнего мира. Поэтому вводится понятие *экземпляра отношения* и *схемы отношения*.

Экземпляр отношения – состояние объекта в данный момент времени.

Схемой отношения называют перечень имен атрибутов данного отношения с указанием домена, к которому они относятся: $S_R = (A_1, A_2, A_n)$, $A_i \subseteq D_i$.

Если атрибуты принимают значения из одного и того же домена, то они называются *θ-сравнимыми*, где θ - множество допустимых операций сравнения.

Схемы двух отношений называются *эквивалентными*, если они имеют одинаковую степень и возможно такое упорядочение имен атрибутов в схемах, что на одинаковых местах будут находиться сравнимые атрибуты, то есть атрибуты, принимающие значения из одного домена.

Отношение имеет простую графическую интерпретацию и может быть представлено в виде таблицы. Любая таблица реляционной базы данных состоит из *строк* (называемых также *записями*) и *столбцов* (называемых также *полями*). Строки таблицы содержат сведения о представленных в ней фактах (или документах, или людях, одним словом, — об однотипных объектах). На пересечении столбца и строки находятся конкретные значения содержащихся в таблице данных.

Данные в таблицах удовлетворяют следующим принципам:

1. Каждое значение, содержащееся на пересечении строки и колонки, должно быть *атомарным* (то есть не расчленяемым на несколько значений).
2. Значения данных в одной и той же колонке должны принадлежать к одному и тому же типу, доступному для использования в данной СУБД.
3. Каждая запись в таблице уникальна, то есть в таблице не существует двух записей с полностью совпадающим набором значений ее полей.
4. Каждое поле имеет уникальное имя.
5. Последовательность полей в таблице несущественна.
6. Последовательность записей также несущественна.

Несмотря на то, что строки таблиц считаются неупорядоченными, любая система управления базами данных позволяет сортировать строки и колонки в выборках из нее нужным пользователю способом.

Поскольку последовательность колонок в таблице несущественна, обращение к ним производится по имени, и эти имена для данной таблицы уникальны (но не обязаны быть уникальными для всей базы данных).

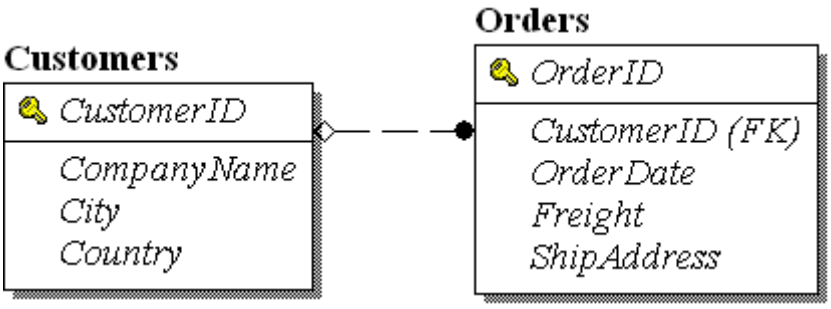
Ключи и связи. Поскольку строки в таблице неупорядочены, нам нужна колонка (или набор из нескольких колонок) для уникальной идентификации каждой строки. Такая колонка (или набор колонок) называется *первичным ключом (primary key)*. Первичный ключ любой таблицы обязан содержать уникальные непустые значения для каждой строки. Если первичный ключ состоит из более чем одной колонки, он называется *составным первичным ключом* (*composite primary key*).

Типичная база данных обычно состоит из нескольких связанных таблиц. Колонка, указывающая на запись в другой таблице, связанную с данной записью, называется *внешним ключом* (*foreign key*). Как видим, в случае таблицы Orders внешним ключом является колонка CustomerID:

Иными словами, *внешний ключ* — это колонка или набор колонок, чьи значения совпадают с имеющимися значениями первичного ключа другой таблицы.

Подобное взаимоотношение между таблицами называется *связью* (*relationship*). Связь между двумя таблицами устанавливается путем присваива-

ния значений внешнего ключа одной таблицы значениям первичного ключа другой.



Если каждый клиент в таблице Customers может разместить только один заказ, говорят, что эти две таблицы связаны соотношением *один-к-одному* (*one-to-one relationship*). Если же каждый клиент в таблице Customers может разместить ноль, один или много заказов, говорят, что эти две таблицы связаны соотношением *один-ко-многим* (*one-to-many relationship*) или соотношением *master-detail*. Подобные соотношения между таблицами используются наиболее часто. В этом случае таблица, содержащая внешний ключ, называется *detail—таблицей*, а таблица, содержащая первичный ключ, определяющий возможные значения внешнего ключа, называется *master-таблицей*.

Группа связанных таблиц называется *схемой* базы данных (*database schema*). Информация о таблицах, их колонках (имена, тип данных, длина поля), первичных и внешних ключах, а также иных объектах базы данных, называется *метаданными* (*metadata*).

Любые манипуляции с данными в базах данных, такие как выбор, вставка, удаление, обновление данных, изменение или выбор метаданных, называются *запросом* к базе данных (*query*). Обычно запросы формулируются на каком-либо языке, который может быть как стандартным для разных СУБД, так и зависящим от конкретной СУБД.

Процесс проектирования данных представляет собой определение метаданных в соответствии с задачами информационной системы, в которой будет использоваться будущая база данных. Подробности о том, как производить анализ предметной области, создавать диаграммы «сущность-связь» (*ERD — entity-relationship diagrams*) и модели данных, будут даны ниже (см. разделе 3.7). Здесь мы обсудим лишь один из основных принципов проектирования данных — принцип *нормализации*.

Нормализация представляет собой процесс реорганизации данных путем ликвидации повторяющихся групп и иных противоречий в хранении данных с целью приведения таблиц к виду, позволяющему осуществлять непротиворечивое и корректное редактирование данных.

Теория нормализации основана на концепции *нормальных форм*. Говорят, что таблица находится в данной нормальной форме, если она удовлетворяет определенному набору требований. Теоретически существует пять нормальных форм [8], но на практике обычно используются только первые три. Более того,

первые две нормальные формы являются по существу промежуточными шагами для приведения базы данных к третьей нормальной форме.

Первая нормальная форма. Чтобы таблица соответствовала первой нормальной форме, все значения ее полей должны быть атомарными и все записи — уникальными. Поэтому любая реляционная таблица по определению уже находится в первой нормальной форме.

Таблица OrderedProducts

OrderID	ProductID	CustomerID	Address	Quantity	OrderDate
10265	17	BLONP	24, place Kleber	30	07.25.96
10265	70	BLONP	24, place Kleber	20	07.25.96
10278	44	BERGS	Berguvsvagen 8	16	08.12.96
10278	59	BERGS	Berguvsvagen 8	15	08.12.96
10280	24	BERGS	Berguvsvagen 8	12	08.14.96
10280	55	BERGS	Berguvsvagen 8	20	08.14.96
10289	3	BSBEV	Faunt Circus	30	08.26.96
10289	64	BSBEV	Faunt Circus	9	08.26.96

Тем не менее, эта таблица содержит избыточные данные, например, одни и те же сведения о клиенте повторяются в записи о каждом заказанном продукте. Результатом избыточности данных являются аномалии модификации данных — проблемы, возникающие при добавлении, изменении или удалении записей. Например, при редактировании данных в таблице OrderedProducts могут возникнуть следующие проблемы:

- адрес конкретного клиента может содержаться в базе данных только тогда, когда клиент заказал хотя бы один продукт;
- при удалении записи о заказанном продукте одновременно удаляются сведения о самом заказе и о клиенте, его разместившем;
- если, не дай бог, заказчик сменил адрес, придется обновить все записи о заказанных им продуктах.

Некоторые из этих проблем могут быть решены путем приведения базы данных ко *второй нормальной форме*.

Вторая нормальная форма. Говорят, что реляционная таблица находится во *второй нормальной форме*, если она находится в первой нормальной форме, и ее неключевые поля *полностью зависят* от всего первичного ключа.

Таблица OrderedProducts находится в первой, но не во второй нормальной форме, так как поля CustomerID, Address и OrderDate зависят только от поля OrderID, являющегося частью составного первичного ключа (OrderID, ProductID).

Чтобы перейти от первой нормальной формы ко второй, нужно выполнить следующие шаги:

1. Определить, на какие части можно разбить первичный ключ так, чтобы некоторые из неключевых полей зависели от одной из этих частей (*эти части не обязаны состоять из одной колонки!*).

2. Создать новую таблицу для каждой такой части ключа и группы зависящих от нее полей и переместить их в эту таблицу. Часть бывшего первичного ключа станет при этом первичным ключом новой таблицы.

3. Удалить из исходной таблицы поля, перемещенные в другие таблицы, кроме тех их них, которые станут внешними ключами.

Например, для приведения таблицы OrderedProducts ко второй нормальной форме, нужно переместить поля CustomerID, Address и OrderDate в новую таблицу (назовем ее OrdersInfo), при этом поле OrderID станет первичным ключом новой таблицы. В результате новые таблицы приобретут такой вид.

Таблица OrdersInfo

OrderID	**CustomerID**	**Address**	**OrderDate**
10265	BLONP	24, place Kleber	07.25.96
10278	BERGS	Berguvsvagen 8	08.12.96
10280	BERGS	Berguvsvagen 8	08.14.96
10289	BSBEV	Faunt Circus	08.26.96
10297	BLONP	24, place Kleber	09.04.96

Таблица OrderDetails

OrderID	**ProductID**	**Quantity**
10265	17	30
10265	70	20
10278	44	16
10278	59	15
10280	24	12
10280	55	20
10289	3	30
10289	64	9

Однако таблицы, находящиеся во второй, но не в третьей нормальной форме, по-прежнему содержат аномалии модификации данных. Вот каковы они, например, для таблицы OrdersInfo:

- Адрес конкретного клиента по-прежнему может содержаться в базе данных только тогда, когда клиент заказал хотя бы один продукт.
- Удаление записи о заказе в таблице OrdersInfo приведет к удалению записи о самом клиенте.
- Если заказчик сменил адрес, придется обновить несколько записей (хотя, как правило, их меньше, чем в предыдущем случае).

Устранить эти аномалии можно путем перехода к *третьей нормальной форме*.

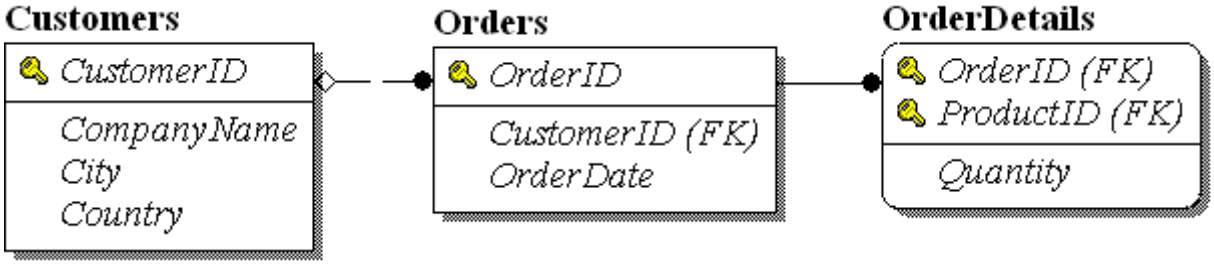
Третья нормальная форма. Говорят, что реляционная таблица находится в *третьей нормальной форме*, если она находится во второй нормальной форме, и все ее неключевые поля зависят только от первичного ключа.

Таблица OrderDetails уже находится в третьей нормальной форме. Неключевое поле Quantity полностью зависит от составного первичного ключа (OrderID, ProductID). Однако таблица OrdersInfo в третьей нормальной форме не находится, так как содержит зависимость между неключевыми полями (она называется *транзитивной зависимостью — transitive dependency*) — поле Address зависит от поля CustomerID.

Чтобы перейти от второй нормальной формы к третьей, нужно выполнить следующие шаги:

- Определить все поля (или группы полей), от которых зависят другие поля.
- Создать новую таблицу для каждого такого поля (или группы полей) и группы зависящих от него полей и переместить их в эту таблицу. Поле (или группа полей), от которого зависят все остальные перемещенные поля, станет при этом первичным ключом новой таблицы.
- Удалить перемещенные поля из исходной таблицы, оставив лишь те из них, которые станут внешними ключами.

Для приведения таблицы OrdersInfo к третьей нормальной форме создадим новую таблицу Customers и переместим в нее поля CustomerID и Address. Поле Address из исходной таблицы удалим, а поле CustomerID оставим — теперь это внешний ключ:



Итак, после приведения исходной таблицы к третьей нормальной форме таблиц стало три — Customers, Orders и OrderDetails.

Преимущества нормализации. Нормализация устраняет избыточность данных, что позволяет снизить объем хранимых данных и избавиться от описанных выше аномалий их изменения. Например, после приведения рассмотренной выше базы данных к третьей нормальной форме налицо следующие улучшения:

- Сведения об адресе клиента можно хранить в базе данных, даже если это только потенциальный клиент, еще не разместивший ни одного заказа.
- Сведения о заказанном продукте можно удалять, не опасаясь удаления данных о клиенте и заказе.

Изменение адреса клиента или даты регистрации заказа теперь требует изменения только одной записи.

Многомерная модель [5]. Задачи, решаемые OLTP и аналитическими системами, существенно различаются, поэтому их БД тоже построены на разных принципах. Критерием эффективности для систем операционной обработки служит число транзакций, которое они способны выполнить в единицу времени.

Для аналитических систем важнее скорость выполнения сложных запросов и прозрачность структуры хранения информации для пользователей. Важная особенность СППР на основе ХД состоит в том, что загрузка данных выполняется сравнительно редко, но большими порциями (до нескольких миллионов записей за один раз), поэтому в таких системах обычно не предусматриваются развитые средства обеспечения целостности, восстановления и устранения взаимных блокировок. Это не только существенно облегчает и упрощает сами средства реализации, но значительно снижает внутренние накладные расходы при доступе к информации и, следовательно, повышает производительность анализа.

В настоящее время существуют два в чем-то конкурирующих, а в чем-то взаимодополняющих друг друга подхода к построению хранилищ данных: подход, основанный на использовании многомерной модели БД (Multidimensional OLAP - MOLAP), и подход, использующий реляционную модель БД (Relational OLAP - ROLAP).

Прежде чем рассказать о каждом из них, попытаемся разобраться, какие данные могут находиться в хранилище и как они могут быть представлены. Чаще всего там содержатся сведения о значении некоторых параметров, характеризующих предметную область в определенные моменты или за определенные промежутки времени. Пусть, например, требуется создать хранилище, накапливающее информацию об изменении социально-экономической обстановки в России. Эта обстановка характеризуется многими параметрами, в числе которых: объем промышленного производства, индекс потребительских цен и др. Госкомстат России собирает их значения для различных субъектов Российской Федерации помесячно, поквартально или за год. В хранилище должны попадать факты вида: *Название параметра* в *субъекте Российской Федерации* в *момент времени* был равен *{значение}*. Например, индекс потребительских цен в городе Москве в декабре 1996 года был равен 101%. В рассматриваемом примере каждое значение связано с точкой в трехмерном пространстве (N_9, S, Г) с измерениями: N - название параметра; *S* - субъект федерации; Г- момент времени. Число возможных параметров, субъектов РФ, а также рассматриваемых моментов времени конечно, поэтому все возможные значения можно представить в виде гиперкуба (см. рис. 2.2).

В этом гиперкубе каждое значение находится в строго определенной ячейке, что значительно упрощает обращение к ней.

Представленный пример, конечно, упрощен, но он позволяет понять, что такое многомерный взгляд на данные. В реальной задаче число измерений может быть больше трех. Представление данных в виде гиперкуба более наглядно, чем совокупность нормализованных таблиц, оно понятно не только администратору БД, но и рядовым сотрудникам. Это дает им дополнительные возможности построения аналитических запросов к системе, использующей хранилище данных. Кроме того, использование многомерной модели данных позволяет резко уменьшить время поиска в ХД, обеспечивая выполнение аналитических запросов в реальном времени.

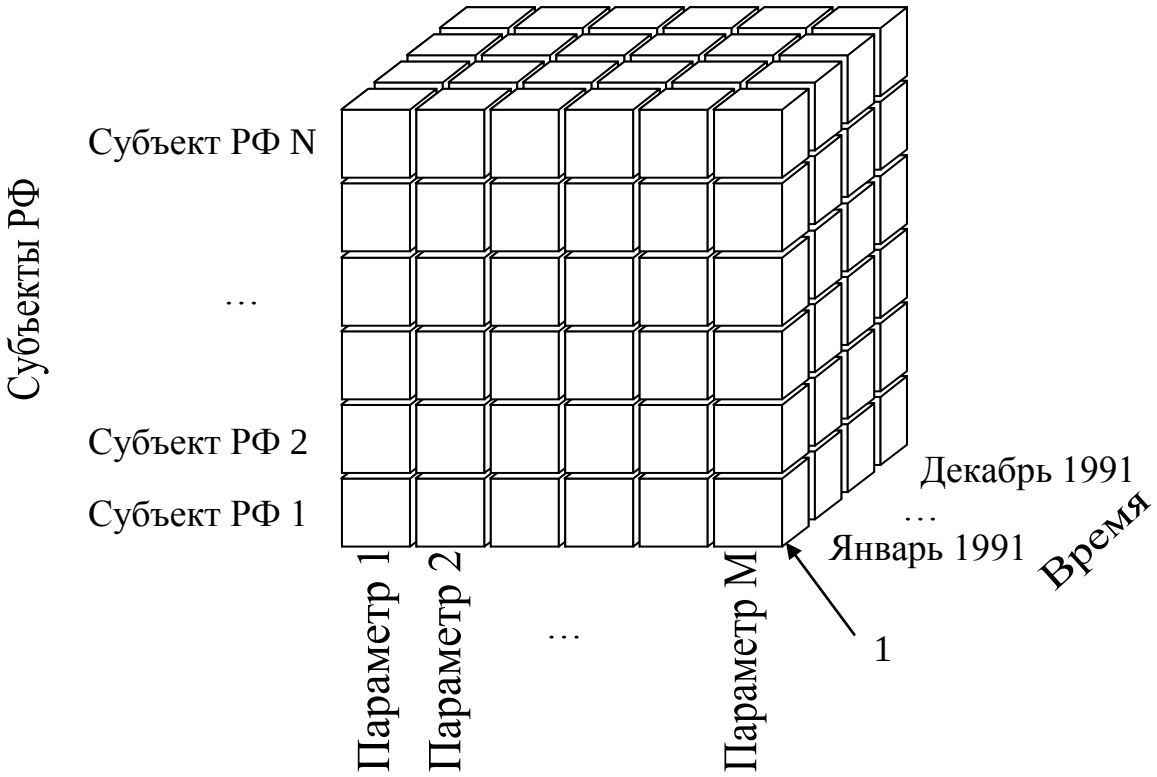


Рис. 2.2. Представление данных в виде гиперкуба:
1 - значение "Параметра М" для "Субъекта РФ 1" в январе 1991 года

Гиперкуб может быть реализован в рамках реляционной модели или существовать как отдельная БД специальной многомерной структуры. В зависимости от этого и принято различать реляционный (ROLAP) и многомерный (MOLAP) подходы к построению ХД.

Многомерная модель хранилища данных. Многомерная модель БД появилась довольно давно, однако в силу присущих ей ограничений применение получила лишь в последнее время. При использовании этой модели данные хранятся не в виде плоских таблиц, как в реляционных БД, а в виде гиперкубов - упорядоченных многомерных массивов. То есть многомерное представление данных здесь реализуется физически. Конечно, такой подход требует большего объема памяти для хранения данных, при его использовании сложно модифицировать структуру данных. Например, добавление еще одного измерения приводит к необходимости полной перестройки гиперкуба. Однако многомерные СУБД обеспечивают более быстрый по сравнению с реляционными системами поиск и чтение данных, а также избавляют от необходимости многократно соединять таблицы. Среднее время ответа на сложный аналитический запрос при использовании многомерных СУБД обычно в 10-100 раз меньше, чем в случае реляционной СУБД с нормализованной структурой.

Основные понятия многомерной модели - *измерение* и *значение* (ячейка). Измерение - это множество, образующее одну из граней гиперкуба (аналог домена в реляционной модели). Измерения играют роль индексов, используемых для идентификации конкретных значений в ячейках гиперкуба. Значения - это подвер-

гаемые анализу количественные или качественные данные, которые находятся в ячейках гиперкуба (см. рис. 2.2).

В многомерной модели вводятся следующие основные операции манипулирования измерениями: 1) сечение; 2) вращение; 3) детализация; 4) свертка.

При выполнении операции сечения формируется подмножество гиперкуба, в котором значение одного или более измерений фиксировано. Например, если на рис. 2.2 зафиксировать значение измерения "Время" равным "январь 1991 года", то мы получим двухмерную таблицу с информацией о значениях всех параметров для всех субъектов РФ в январе 1991 года.

Операция вращения изменяет порядок представления измерений. Она обычно применяется к двухмерным таблицам, обеспечивая представление их в более удобной для восприятия форме. Если в исходной таблице по горизонтали были расположены субъекты РФ, а по вертикали - параметры социально-экономической сферы, то после операции вращения параметры будут размещены по горизонтали, а названия субъектов РФ по вертикали.

Для выполнения операций свертки и детализации должна существовать иерархия значений измерения, то есть некоторая подчиненность одних значений другим. Например, 12 месяцев образуют год, субъекты РФ образуют федеральные округа. При выполнении операции свертки одно из значений измерения заменяется значением более высокого уровня иерархии. Допустим, аналитик, узнав значения параметров для января 1991 года, желает получить их значения за весь 1991 год. Чтобы это сделать, необходимо выполнить операцию свертки. Операция детализации - это операция, обратная свертке. Она обеспечивает переход от обобщенных к детализированным данным.

Основное назначение СУБД, поддерживающих многомерную модель, - реализация систем, ориентированных на аналитическую обработку. Многомерные СУБД лучше других справляются с задачами выполнения сложных нерегламентированных запросов.

Однако у многомерных БД имеются серьезные недостатки, сдерживающие их применение. Многомерные СУБД не эффективно, по сравнению с реляционными, используют память. В многомерной СУБД заранее резервируется место для всех значений, даже если часть из них заведомо будет отсутствовать. Другой недостаток состоит в том, что выбор высокого уровня детализации при реализации гиперкуба может очень сильно увеличить размер многомерной БД. В силу этих, а также некоторых других причин доступные на рынке многомерные СУБД не в состоянии оперировать данными большого объема. Объем, доступный им для хранения, ограничен 10-20 гигабайтами памяти.

Целесообразно использовать многомерную модель, если объем БД невелик и гиперкуб использует стабильный во времени набор измерений.

Реляционная модель хранилища данных. Основой при построении хранилища данных может служить и традиционная реляционная модель данных. В этом случае гиперкуб эмулируется СУБД на логическом уровне. В отличие от многомерных реляционные СУБД способны хранить огромные объемы данных, однако они проигрывают по скорости выполнения аналитических запросов.

При использовании РСУБД для организации хранилища данные организуются специальным образом. Чаще всего используется так называемая радиальная схема. Другое ее название - "*звезда*" (*star*). В этой схеме используются два типа таблиц: *таблица фактов* (*фактологическая таблица*) и несколько *справочных таблиц* (*таблицы измерений*).

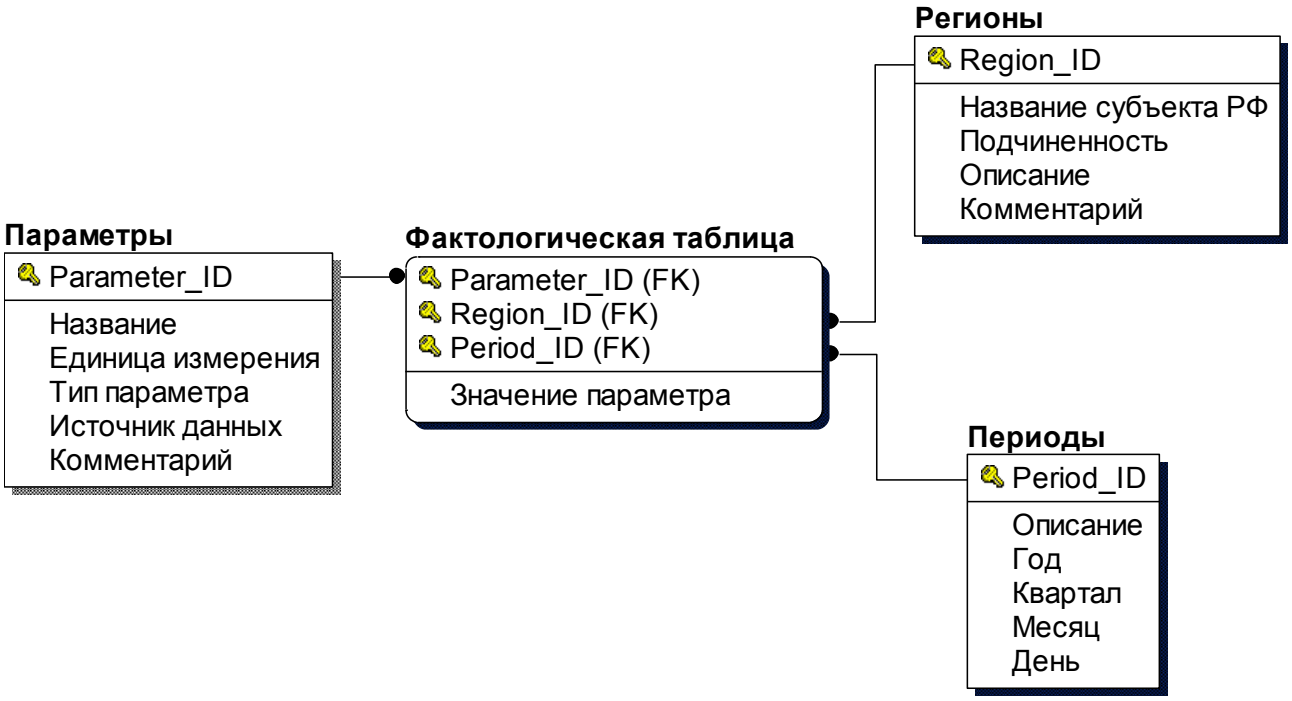


Рис. 2.3. Пример БД с радиально связанными таблицами (схема *звезда*)

В таблице фактов обычно содержатся данные, наиболее интенсивно используемые для анализа. Если проводить аналогию с многомерной моделью, то запись фактологической таблицы соответствует ячейке гиперкуба. В справочной таблице перечислены возможные значения одного из измерений гиперкуба. Каждое измерение описывается своей собственной справочной таблицей. Фактологическая таблица индексируется по сложному ключу, скомпонованному из индивидуальных ключей справочных таблиц. Это обеспечивает связь справочных таблиц с фактологической по ключевым атрибутам. В качестве примера на рис. 2.3 приведена упрощенная схема структуры хранилища данных, используемого для накопления информации из рассмотренного ранее примера (см. рис. 2.2).

В реальных системах количество строк в фактологической таблице может составлять десятки и сотни миллионов. Число справочных таблиц обычно не превышает двух десятков. Для увеличения производительности анализа в фактологической таблице могут храниться не только детализированные, но и предварительно вычисленные агрегированные данные.

Если БД включает большое число измерений, можно использовать схему "снежинка" (snowflake). В этой схеме атрибуты справочных таблиц могут быть детализированы в дополнительных справочных таблицах (см. рис. 2.4).

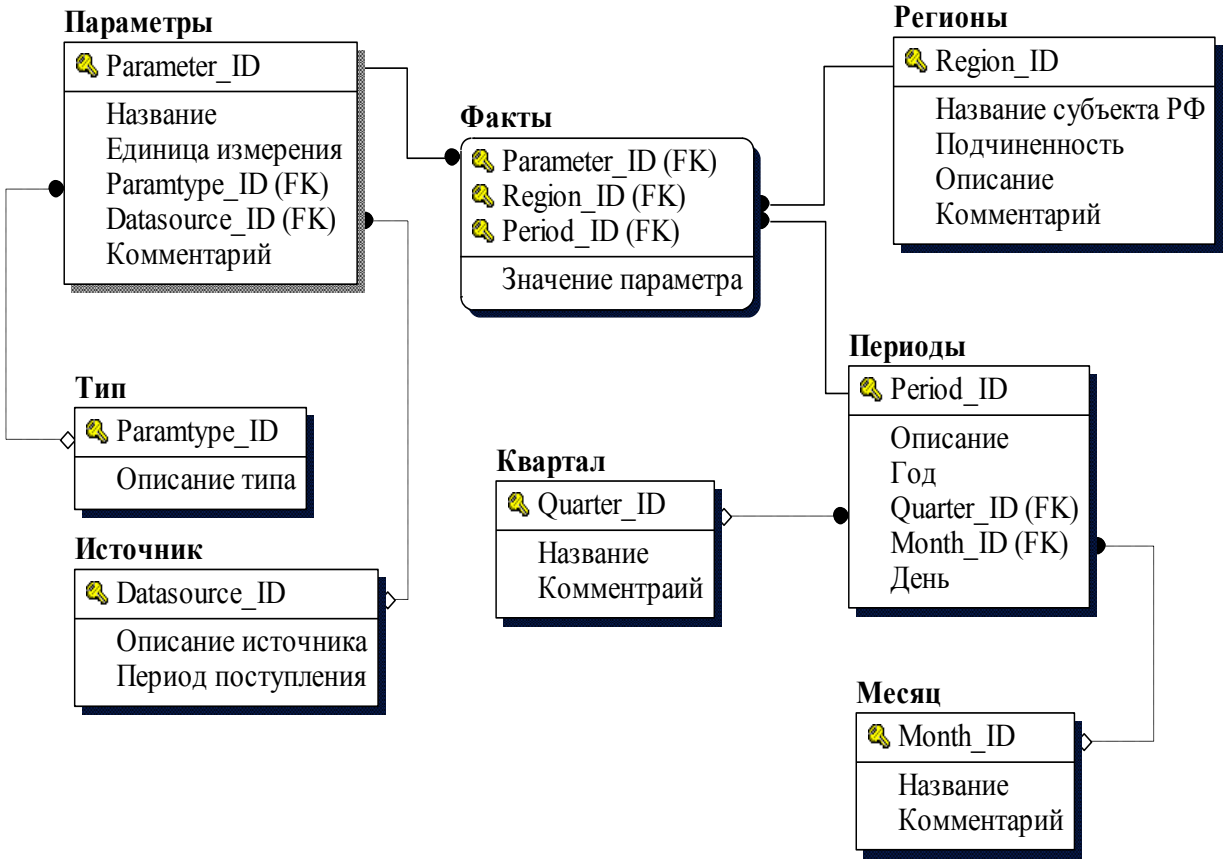


Рис. 2.4. Пример БД со схемой *снежинка*

Для сокращения времени, требуемого для получения отклика от аналитической системы, можно использовать некоторые специальные средства. В состав мощных реляционных СУБД обычно входят оптимизаторы запросов. При создании хранилищ данных на основе РСУБД их наличие приобретает особое значение. Оптимизаторы анализируют запрос и определяют лучшую, с позиции некоторого критерия, последовательность операций обращения к БД для его выполнения. Например, может минимизироваться число физических обращений к дискам при выполнении запроса. Оптимизаторы запросов используют сложные алгоритмы статической обработки, которые оперируют числом записей в таблицах, диапазоном ключей и т.д.

2.2.4. Объектно-ориентированная модель данных

В последнее время концепция объектно-ориентированных баз данных все больше привлекает внимание специалистов по обработке информации во всем мире. Однако, как правило, разработчики ставят перед собой цель создать средства поддержки объектно-ориентированного проектирования или программирования, и почти никто не пытается сделать объектно-ориентированной саму модель данных, т. е. непосредственно представлять в базе данных (БД) реальные объекты и связи описываемой предметной области. Наиболее близка к этому модель “объект—отношение” (entity-relationship model, **ER**-модель), предложенная П. Ченом [12], но даже она, по существу, представляет собой мостик,

связывающий концептуальное проектирование и обычную реляционную модель.

Объектно-ориентированная модель данных (ООМД) сочетает в себе лучшие черты **ER**-модели и реляционной модели [8].

При проектировании и эксплуатации прикладных систем в реляционной среде и разработчикам, и пользователям приходится держать в голове перечень полей различных отношений (таблиц) для организации связей между таблицами. Иначе говоря, реляционная модель плохо поддерживает семантику данных. Кроме того, при идентификации объектов и организации связей возникают неудобства с внешними и составными первичными ключами.

ООМД основана на принципиально иной технологии. Предположим, что у нас есть объект “изделия” и объект “детали”. При организации связи нас совершенно не интересует, какими характеристиками обладают указанные объекты (в терминах реляционной модели - какие поля имеются в таблицах “изделия” и “детали”). Мы просто указываем, что “изделия” “содержат” “детали”, а “детали” “входят в” “изделия” с типом связи “многие-ко-многим”. При этом всякая связь в ООМД автоматически является двусторонней - в отличие от ER-модели, где потребовалось бы организовать две разные связи и, соответственно, создать для их описания два отношения в смысле реляционной модели.

Связь может иметь свои характеристики: скажем, характеристики “позиция” (т. е. позиция детали на чертеже изделия) и “количество” (количество деталей в изделии) относятся не к изделию и не к детали, а к конкретной связи между ними. Естественно, между двумя объектами разрешается установить несколько связей. Например, “оснастка” “содержит” “детали” и “оснастка” “служит для изготовления” “детали”. Или: “читатель” “читает” “книги” и “читатель” “мечтает прочитать” “книги”.

Простая реализация такой технологии возможна благодаря тому, что каждому экземпляру объекта поставлено в соответствие уникальное значение некоторой автоматически поддерживаемой ключевой характеристики (идентификатора).

Итак, ООМД опирается на четыре основных понятия: *объект*, *характеристика объекта*, *связь*, *характеристика связи*. Если при проектировании ограничиваться рассмотренными выше бинарными связями, большинство задач можно будет решать в естественной объектной среде без дополнительного программирования. Однако существуют две ситуации, в которых для адекватного отражения в БД предметной области бинарных связей недостаточно.

Ситуация 1. Связь является множественной (обычно во времени). Например, имеется два объекта: “врачи” и “пациенты”. Можно, конечно, организовать между этими объектами связь типа “многие-ко-многим” с определенной семантикой (“врачи” “лечат” “пациенты”, “пациенты” “лечатся у” “врачи”). Однако каждый конкретный врач может оказывать помощь конкретному пациенту много раз, и каждая возникающая при этом связь будет иметь собственные характеристики. Подобную картину нетрудно было бы свести к бинарным связям, но мы пошли другим путем из-за наличия второй ситуации.

Ситуация 2. Необходимо связать между собой более двух объектов. Например, при формировании планов связываются “изделия”, “заказы” и “цехи”.

Для описания этих ситуаций в ООМД введен специальный тип характеристики (и объекта, и связи) - *ссылка*. Например, вы создаете объект “визиты” и среди прочих его характеристик используете ссылки на “врачи” и на “пациенты”. Их значениями для каждого конкретного визита будут идентификаторы конкретного врача и конкретного пациента. Следует подчеркнуть, что ссылки в ООМД даются на экземпляры конкретных объектов, а не на поля других таблиц, как в реляционной модели. Любая реализация ООМД должна эффективно поддерживать все операции со ссылками точно так же, как и с бинарными связями.

В примере с врачами и пациентами “визиты” - это объект, описывающий связь между другими объектами, т. е. объект-связь. Комбинация на основе ссылок бинарных связей и объектов-связей дает мощный и гибкий инструмент проектирования БД, максимально сближая этапы разработки на концептуальном и на логическом уровнях.

ООМД разумно применять не только при проектировании БД, но и при эксплуатации прикладной системы, ориентируясь тем самым на объектный, а не на функциональный пользовательский интерфейс.

Так, одним из самых сложных навыков работы с БД не без оснований считается умение правильно составить запрос для получения нужной информации. ООМД же позволяет обойтись вообще без запросов. Для быстрого поиска объектов в каталогах используются динамически поддерживаемые инвертированные списки по соответствующим характеристикам, благодаря чему скорость доступа к нужной информации слабо зависит от числа элементов в БД.

На этапе эксплуатации прикладных систем проявляется еще одно очень важное достоинство ООМД. Поскольку все, работающие с этой моделью, независимо от предметной области имеют дело в основном с объектами и связями между ними (и лишь изредка - со специфическими для некоторой области прикладными задачами), пользователи разных прикладных систем получают удобную возможность для обмена опытом и обучения новичков.

2.3. Документы и их структура

При проектировании информационного обеспечения необходимо изучать характеристики потоков информации. Под *информационным потоком* понимается совокупность данных в процессе ее движения в пространстве и во времени. В качестве единицы потока используют документ. *Документы* являются основными носителями информации на предприятии и представляют совокупность некоторых элементов, называемых показателями.

Состав документов, которые обрабатываются ИС, определяется на этапе формулировки требований к ИС. При этом могут использоваться формальные модели типа **DFD**, а может быть составлен и простой перечень входных и вы-

ходных документов. В любом случае после определения состава документов необходим их анализ, чтобы определить, как же информация об объектах предметной области будет отображаться в ИС [3].

Единица информации – это набор символов, которым придается определенный смысл. Значительная часть информации, циркулирующей в предметной области, отображается в документах. Документ, как правило – это свидетельство какого-либо события реальной жизни, при котором взаимодействуют два или несколько объектов. Поэтому при проектировании ИС необходимо выяснить состав документов, участвующих в бизнес-процессах, описать структуру каждого документа, выяснить правила вычисления каждого поля.

Документ состоит из отдельных полей, называемых *реквизитами,* которые считаются *элементарными единицами информации* и отображают свойства того или иного реального объекта или процесса. Например, документ "Приходная накладная" сопровождает процесс приема товаров на склад, при котором взаимодействуют такие объекты, как товары, склад, поставщик товаров. В накладной можно выделить такие реквизиты, как "Номер документа", "Наименование товара", "Цена" и др.

Каждый реквизит имеет *имя, значение и область определения.* Имя – это условное обозначение реквизита в процессах обработки данных. Значение – это величина, характеризующая некоторое свойство объекта, явления, процесса в конкретных обстоятельствах. Имя реквизита должно отражать его смысл и постоянно для одного вида документов, значения реквизита могут различаться в каждом экземпляре документа. Область определения значений реквизита (домен) может задаваться перечислением значений, типом данных (текстовый, числовой, календарный, логический) или границами. Например, реквизит "Номер документа" может принимать целые значения от 1 до 5000.

Реквизиты делятся на два вида: *признаки*, отображающие обстоятельства события (место, время, действующих лиц, единицы измерения, номера документов или идентификаторы объектов) и *основания,* отображающие количественные свойства объектов и процессов. Например, «Номер документа», «Дата», «Наименование товара» – это признаки; «Цена товара», «Сумма», «Итоговая сумма» – это основания.

Некоторые признаки документа называются *ключевыми,* так как именно по их значению можно отличить один документ от другого (принято говорить, что ключ идентифицирует документ). Чаще всего ключ документа образуется из двух признаков “Номер документа” и “Дата документа”, так как нумерация большинства документов начинается с № 1 в начале каждого года.

Реквизиты могут группироваться и образовывать *составные единицы информации* (СЕИ). Например, каждую таблицу, имеющуюся в документе, часто выделяют в отдельную СЕИ. Каждая СЕИ обычно соответствует какому-то реальному объекту или процессу. Например, в документе «Накладная» СЕИ "Список товаров" описывает свойства товаров, отпущенных по накладной. СЕИ, кроме имени и значения, характеризуются также *структурой,* которую обычно описывают перечнем имен реквизитов после имени составной едини-

цы. Как правило, этот список помещается в скобки, а имена перечисляются через запятую.

Среди составных единиц информации особо выделяют *показатель* – это элементарный осмысленный фрагмент документа, содержащий один атрибут - основание Q и несколько логически связанных с ним признаков i, j, k, … n.

Основание показателя обозначают заглавной буквой, а признаки записывают в виде индексов $Q_{ijk\ldots n}$. Минимальный набор реквизитов показателя (реквизитный минимум) должен включать такие признаки, значения которых позволяют однозначно определить значение показателя в массе однотипных документов. К ним относятся имена объектов, участвующих в процессе определения основания, и время действия.

Наименование п.п. АО “New Technology” Адрес: Сюзьва, ул.Пушкина,10				Код поставщика 34
Номенкл. №	Наименование товара	Цена (тыс. руб.)	Количество	Сумма
11 22	Canon LaserJet Бумага А4	6 0.1	2 20	12 2
ИТОГО:				14

Рис. 2.5. Приходный ордер № 123 от 1.09.98

Например, выделим показатели в документе «Накладная», рис. 2.5. Сначала запишем обозначения оснований: Ц (цена), К (количество), С (сумма), И (итого). Затем обозначим признаки: i (номер документа), n (номенклатурный номер товара), k (дата), j (наименование предприятия), a (адрес), p (код поставщика), l (фамилия ответственного лица). Показатели: $Ц_{ink}$, $К_{ink}$, $С_{ink}$, $И_{ik}$.

Значения цены, количества и суммарной стоимости n-го товара, оформленного по i-ой накладной, зависят от даты оформления, поэтому в показатель включены все три признака i, n, k. Итоговая сумма вычисляется для всех товаров, оформленных по данной накладной, значит номенклатурный номер отдельного товара не определяет значение основания И, и поэтому последний показатель включает только два признака.

Рассмотрим, почему реквизиты j, a, p, l не вошли в состав показателей. Действительно, в бухгалтерии предприятия, для которого мы описываем приходные накладные, все накладные будут иметь одно и то же значение признаков «Название и адрес предприятия», то есть их значение постоянно и не оказывает влияния на другие реквизиты. Признак p (код поставщика) влияет на цену товара, поэтому можно иначе определить показатель $Ц_{pnk}$ и выбрать один из этих вариантов (выбран одинаковый состав признаков для всех трех показателей). И, наконец, признак «Фамилия ответственного лица» не влияет на значения оснований и не относится к существенным для расчетов.

Для наглядного описания логических и расчетных связей между показателями и показа последовательности их расчета служит *граф взаимосвязей* по-

казателей, в котором вершины соответствуют показателям, а дуга идет от S_i к S_j, если при расчете основания S_j используется основание S_i.

Например, в i-й накладной от k-го числа можно выявить следующие арифметические взаимосвязи оснований:

$$C_n = Ц_n \cdot K_n, \qquad И = \sum_n C_n$$

В заключение построим граф взаимосвязи показателей (рис. 2.6).

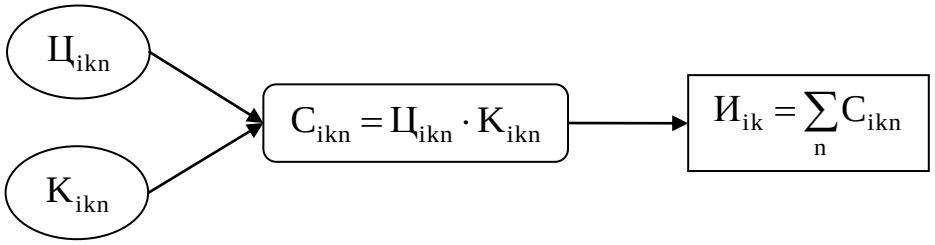


Рис. 2.6. Пример графа взаимосвязи показателей

Для показателей, описывающих бизнес-процессы, можно выделить следующие составные части:

1. Название процесса.
2. Название системы, в которой происходит процесс.
3. Название функции управления.
4. Определение момента или периода времени.
5. Перечень объектов, участвующих в процессе.
6. Единицы измерения атрибутов-оснований.

Сообщение – это составная единица информации, характеризующая в целом какой-либо объект или процесс и представляющая собой набор значений показателей.

2.4. Контрольные вопросы

1. Какие классы информационных моделей существуют?
2. Каково назначение структурных моделей?
3. В каком случае используют функциональные модели?
4. Для чего используют поведенческие модели?
5. Что показывают архитектурные модели?
6. Опишите уровни представления информации в ИС.
7. Что такое инфологическая модель, и каким требованиям она должна отвечать?
8. Дайте классификацию моделей данных.
9. Что собой представляет иерархическая модель данных?
10. Дайте характеристику сетевой модели данных.
11. Каковы причины широкого распространения реляционной модели данных?

12. Перечислите и дайте характеристику основным элементам реляционной модели данных.
13. Что такое нормализация отношений? Сформулируйте виды нормальных форм.
14. В каком случае используется многомерная модель данных, и каковы основные ее характеристики?
15. Каковы основные понятия объектно-ориентированной модели данных? Каковы ее преимущества?
16. Как называется элементарная единица информации в документе?
17. Какие характеристики имеет реквизит?
18. Какие реквизиты относят к признакам, к основаниям?
19. Что такое показатель?
20. Как и для чего строится граф взаимосвязей показателей?

ГЛАВА 3. МЕТОДОЛОГИЯ СТРУКТУРНОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ

3.1. Сущность структурного подхода

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

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов [13]. В качестве двух базовых принципов используются следующие:

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

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

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

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

- **SADT** (Structured Analysis and Design Technique) или **IDEF0** (Integration DEFinition for function modeling) модели и соответствующие функциональные диаграммы [14];
- **IDEF3** (Workflow) диаграммы потоков работ или процессов [15-17];

- **DFD** (Data Flow Diagrams) диаграммы потоков данных [15];
- **ERD** (Entity-Relationship Diagrams) диаграммы "сущность-связь" [12].

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей (AS-IS) или вновь разрабатываемой (TO BE). Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

3.2. Методология функционального моделирования IDEF0

Методология функционального моделирования **IDEF0** — это технология описания системы в целом как множества взаимозависимых действий, или функций [14-17]. Важно отметить функциональную направленность **IDEF0** — функции системы исследуются независимо от объектов, которые обеспечивают их выполнение. "Функциональная" точка зрения позволяет четко отделить аспекты назначения системы от аспектов ее физической реализации.

Наиболее часто **IDEF0** применяется как технология исследования и проектирования систем на логическом уровне. По этой причине он, как правило, используется на ранних этапах разработки проекта, до **IDEF3** моделирования для сбора данных и моделирования процесса "как есть". Результаты **IDEF0** анализа могут применяться при проведении проектирования с использованием моделей **IDEF3** и диаграмм потоков данных.

3.2.1. Синтаксис и семантика моделей IDEF0

Модели IDEF0. IDEF0 сочетает в себе небольшую по объему графическую нотацию (она содержит только два обозначения: блоки и стрелки) со строгими и четко определенными рекомендациями, в совокупности предназначенными для построения качественной и понятной модели системы.

Методология **IDEF0** в некоторой степени напоминает рекомендации, существующие в книгоиздательском деле, часто набор напечатанных моделей **IDEF0** организуется в брошюру (называемую в терминах **IDEFO** *комплект),* имеющую содержание, глоссарий и другие элементы, характерные для законченной книги.

Первый шаг при построении модели **IDEF0** заключается в определении *назначения* модели — набора вопросов, на которые должна отвечать модель. Набор вопросов можно сравнить с предисловием, в котором раскрывается назначение книги.

Границы моделирования предназначены для обозначения ширины охвата предметной области и глубины детализации и являются логическим продолжением уже определенного назначения модели. Как читающий модель, так и

непосредственно ее автор должны понимать степень детальности ответов на поставленные в назначении модели вопросы.

Следующим шагом указывается предполагаемая *целевая аудитория,* для нужд которой создается модель. Зачастую от выбора целевой аудитории зависит уровень детализации, с которым должна создаваться модель. Перед построением модели необходимо иметь представление о том, какие сведения о предмете моделирования уже известны, какие дополнительные материалы и (или) техническая документация для понимания модели могут быть необходимы целевой аудитории, какие язык и стиль изложения являются наиболее подходящими.

Под *точкой зрения* понимается перспектива, с которой наблюдалась система при построении модели. Точка зрения выбирается таким образом, чтобы учесть уже обозначенные границы моделирования и назначение модели. Однажды выбранная точка зрения остается неизменной для всех элементов модели. При необходимости могут быть созданы другие модели, отображающие систему с других точек зрения. Вот несколько примеров точек зрения при построении моделей: клиент, поставщик, владелец, редактор.

Действия. Действие, обычно в **IDEF0** называемое функцией, обрабатывает или переводит входные параметры (сырье, информацию и т.п.) в выходные. Поскольку модели **IDEF0** представляют систему как множество иерархических (вложенных) функций, в первую очередь должна быть определена функция, описывающая систему в целом — *контекстная функция.* Функции изображаются на диаграммах как поименованные прямоугольники, или функциональные блоки. Имена функций в **IDEF0** подбираются с использованием глаголов или отглагольных существительных. Важно подбирать имена таким образом, чтобы они отражали систему так, как если бы она обозревалась с точки зрения, выбранной для моделирования.

Выше мы определяли **IDEF0** модели как иерархическое множество вложенных блоков. Любой блок может быть *декомпозирован* на составляющие его блоки. Декомпозицию часто ассоциируют с моделированием "сверху вниз", однако это не совсем верно. Функциональную декомпозицию корректнее определять как моделирование "снаружи вовнутрь", в котором мы рассматриваем систему наподобие луковицы, с которой последовательно снимаются слои.

Границы и связи. Чтобы быть полезным, описание любого блока должно, как минимум, включать в себя описание объектов, которые блок создает в результате своей работы ("выхода"), и объектов, которые блок потребляет или преобразует ("вход").

В **IDEF0** также моделируются *управление* и *механизмы исполнения.* Под управлением понимаются объекты, воздействующие на способ, которым блок преобразует вход в выход. Механизм исполнения — объекты, которые непосредственно выполняют преобразование входа в выход, но не потребляются при этом сами по себе.

Для отображения категорий информации, присутствующих на диаграммах **IDEF0**, существует аббревиатура **ICOM**, отображающая четыре возможных типа стрелок:

I (Input) — вход — нечто, что потребляется в ходе выполнения процесса;

C (Control) — управление — ограничения и инструкции, влияющие на ход выполнения процесса;

O (Output) — выход — нечто, являющееся результатом выполнения процесса;

M (Mechanism) — исполняющий механизм — нечто, что используется для выполнения процесса, но не потребляется само по себе. Рис. 3.1 показывает 4 возможных типа стрелок в **IDEF0**, каждый из типов соединяется со своей стороной функционального блока.

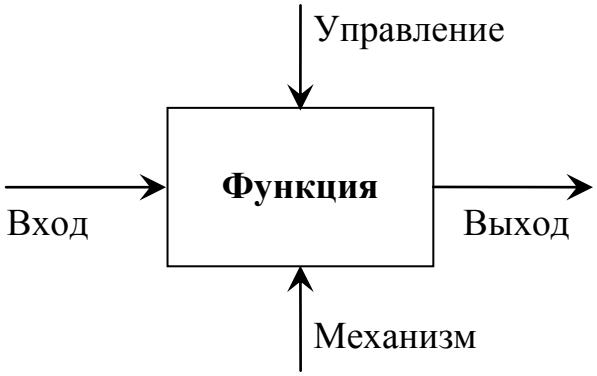


Рис. 3.1. Каждый тип стрелки соединяется со своей стороной функционального блока

Для названия стрелок, как правило, употребляются имена существительные. Стрелки могут представлять собой людей, места, вещи, идеи или события. Как и в случае с функциональными блоками, присвоение имен всем стрелкам на диаграмме является только необходимым условием для понимания читателем сути изображенного. Отдельное описание каждой стрелки в текстовом виде может оказаться критическим фактором для построения точной и полезной модели.

Стрелки входа. *Вход* представляет собой сырье, или информацию, потребляемую или преобразуемую функциональным блоком для производства *выхода.* Стрелки входа всегда направлены в левую сторону прямоугольника, обозначающего в **IDEF0** функциональный блок. *Наличие входных стрелок на диаграмме не является обязательным,* так как возможно, что некоторые блоки ничего не преобразуют и не изменяют. Примером блока, не имеющего входа, может служить "принятие решения руководством", где для принятия решения анализируется несколько факторов, но ни один из них непосредственно не преобразуется и не потребляется в результате принятия какого-либо решения.

Стрелки управления. Стрелки управления отвечают за регулирование того, как и когда выполняется функциональный блок, и, если он выполняется, какой выход получается в результате его выполнения. Так как управление кон-

тролирует поведение функционального блока для обеспечения создания желаемого выхода, *каждый функциональный блок должен иметь, как минимум, одну стрелку управления.* Стрелки управления *всегда* входят в функциональный блок сверху.

Управление часто существует в виде правил, инструкций, законов, политики, набора необходимых процедур или стандартов. Влияя на работу блока, оно непосредственно не потребляется и не трансформируется в результате. Может оказаться, что целью функционального блока является как раз изменение того или иного правила, инструкции, стандарта и т.п. В этом случае стрелка, содержащая соответствующую информацию, должна рассматриваться не как управление, а как вход функционального блока.

Управление можно рассматривать как специфический вид входа. В случаях, когда неясно, относить ли стрелку к входу или к управлению, предпочтительно относить ее к управлению до момента, пока неясность не будет разрешена.

Стрелки выхода. Выход — это продукция или информация, получаемая в результате работы функционального блока. *Каждый блок должен иметь, как минимум, один выход.* Действие, которое не производит никакого четко определяемого выхода, не должно моделироваться вообще (по меньшей мере, должно рассматриваться в качестве одного из первых кандидатов на исключение из модели).

При моделировании непроизводственных предметных областей выходами, как правило, являются данные, в каком-либо виде обрабатываемые функциональным блоком. В этом случае важно, чтобы названия стрелок входа и выхода были достаточно различимы по своему смыслу. Например, блок "Прием пациентов" может иметь стрелку "Данные о пациенте" как на входе, так и на выходе. В такой ситуации входящую стрелку можно назвать "Предварительные данные о пациенте", а *исходящую* — "Подтвержденные данные о пациенте".

Стрелки механизма исполнения. Механизмы являются ресурсом, который непосредственно исполняет моделируемое действие. С помощью механизмов исполнения могут моделироваться: ключевой персонал, техника и (или) оборудование. Стрелки механизма исполнения могут отсутствовать в случае, если оказывается, что они не являются необходимыми для достижения поставленной цели моделирования.

Комбинированные стрелки. В **IDEF0** существуют пять основных видов комбинированных стрелок: выход — вход, выход — управление, выход — механизм исполнения, выход — обратная связь на управление и выход — обратная связь на вход.

Стрелка выход—вход применяется, когда один из блоков должен полностью завершить работу перед началом работы другого блока. Так, на рис. 3.2 формирование счета должно предшествовать приему заказа.

Стрелка выход — управление отражает ситуацию преобладания одного блока над другим, когда один блок управляет работой другого (рис. 3.3).

Стрелки выход—механизм исполнения встречаются реже и отражают ситуацию, когда выход одного функционального блока применяется в качестве оборудования для работы другого блока. На рис. 3.4 зажим, устройство, используемое для закрепления детали во время ее сборки, должно быть собрано для того, чтобы выполнить сборку детали.

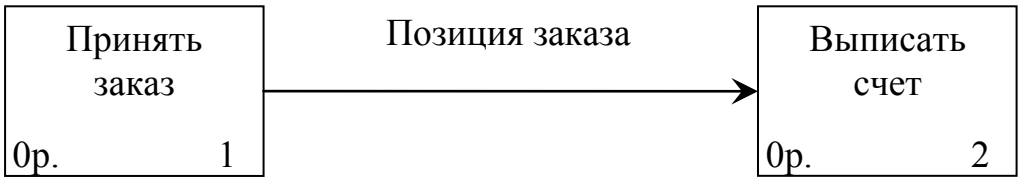


Рис. 3.2. Комбинация стрелок выход — вход

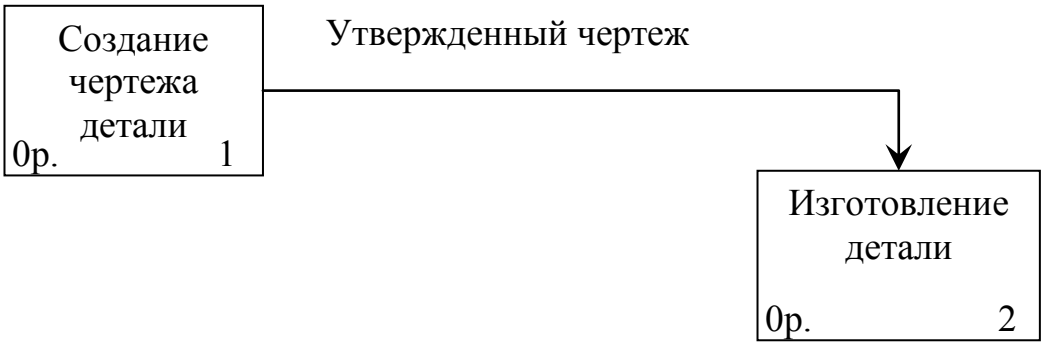


Рис. 3.3. Комбинированная стрелка выход — управление

Рис. 3.4. Комбинированная стрелка выход — механизм исполнения

Обратные связи на вход и на управление применяются в случаях, когда зависимые блоки формируют обратные связи для управляющих ими блоков. На рис. 3.5 получаемые из отдела качества рекомендации применяются для корректировки технологических параметров переработки сырья.

Стрелка выход—обратная связь на вход обычно применяется для описания циклов повторной обработки чего-либо. Рис. 3.6 может служить примером применения стрелки такого типа. Кроме того, связи выход — обратная связь на

вход могут применяться в случае, если бракованная продукция может заново использоваться в качестве сырья, как это происходит, например, при производстве оконного стекла, когда разбитое в процессе производства стекло перемалывается и переплавляется заново вместе с обыкновенным сырьем.

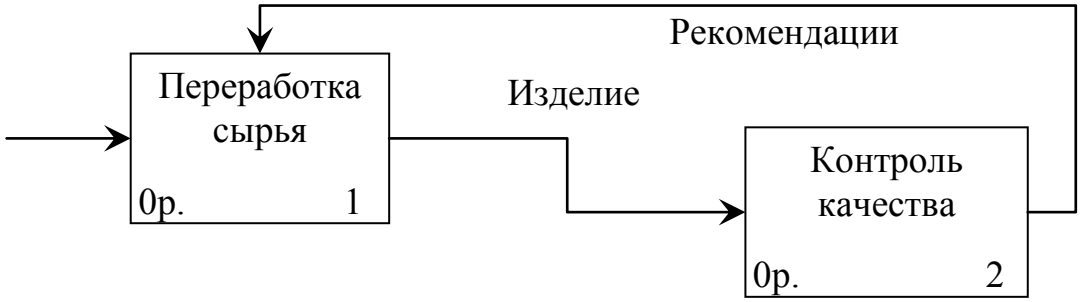


Рис. 3.5. Комбинированная стрелка выход — обратная связь на управление

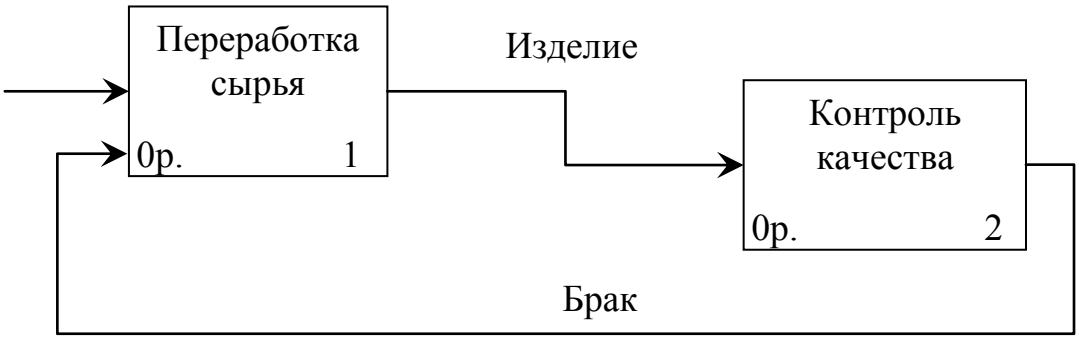


Рис. 3.6. Комбинированная стрелка выход — обратная связь на вход

Разбиение и соединение стрелок. Выход функционального блока может использоваться в *нескольких* других блоках. Фактически чуть ли не главная ценность **IDEF0** заключается в том, что эта методология помогает выявить взаимозависимости между блоками системы. Соответственно **IDEF0** предусматривает как разбиение, так и соединение стрелок на диаграмме. Разбитые на несколько частей стрелки могут иметь наименования, отличающиеся от наименования исходной стрелки. Исходная и разбитые (или объединенные) стрелки в совокупности называются *связанными.* Такая техника обычно применяется для того, чтобы отразить использование в процессе только части сырья или информации, обозначаемых исходной стрелкой. Аналогичный подход применяется и к объединяемым стрелкам.

Туннели. Понятие *связанные стрелки* используется для управления уровнем детализации диаграмм. Если одна из стрелок диаграммы отсутствует на родительской диаграмме (например, ввиду своей несущественности для родительского уровня) и не связана с другими стрелками той же диаграммы, точка входа этой стрелки на диаграмму или выхода с нее обозначается *туннелем.* На рис. 3.7, например, стрелка "корпоративная информационная система" — важ-

ный механизм исполнения для данной диаграммы, но, возможно, она более нигде не используется в модели. Туннель в данном случае используется как альтернатива загромождению родительских диаграмм помещением на них несущественных для их уровня стрелок.

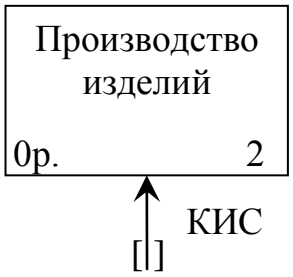


Рис. 3.7. Пример применения туннеля

Кроме того, туннели применяются для отражения ситуации, когда стрелка, присутствующая на родительской диаграмме, отсутствует в диаграмме декомпозиции соответствующего блока. На рис. 3.8 туннель у стрелки "модуль производственного отдела" обозначает, что на диаграмме декомпозиции производственного отдела отсутствует стрелка механизма управления с соответствующим наименованием.

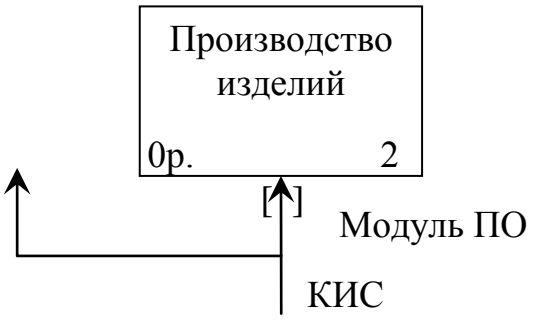


Рис. 3.8. Еще один пример применения туннеля

3.2.2. Построение моделей IDEF0

Диаграммы. На рис. 3.9 типовая диаграмма **IDEF0** показана вместе с находящейся на ее полях служебной информацией. Служебная информация состоит из хорошо выделенных верхнего и нижнего колонтитулов (заголовка и "подвала"). Элементы заголовка используются для отслеживания процесса создания модели. Элементы "подвала" отображают наименование модели, к которой относится диаграмма, и показывают ее расположение относительно других диаграмм модели.

Рис. 3.9. **IDEF0**-диаграмма со служебной информацией на полях

Все элементы заголовка диаграммы перечислены в табл. 3.1.

Таблица 3.1

Элементы заголовка диаграммы **IDEF0**

Поле	**Назначение**
USED AT	Используется для отражения внешних ссылок на данную диаграмму (заполняется на печатном документе вручную)
Author, date, project	Содержит ФИО автора диаграммы, дату создания, дату последнего внесения изменений, наименование проекта, в рамках которого она создавалась
Notes 1 ... 10	При ручном редактировании диаграмм пользователи могут зачеркивать цифру каждый раз, когда они вносят очередное исправление
Status	Статус отражает состояние разработки или утверждения данной диаграммы. Это поле используется для реализации формального процесса публикации с шагами пересмотра и утверждения
Working	Новая диаграмма, глобальные изменения или новый автор для существующей диаграммы
Draft	Диаграмма достигла некоторого приемлемого для читателей уровня и готова для представления на утверждение
Recommended	Диаграмма одобрена и утверждена. Какие-либо изменения не предвидятся
Publication	Диаграмма готова для окончательной печати и публикации
Reader	ФИО читателя
Date	Дата знакомства читателя с диаграммой
Context	Набросок расположения функциональных блоков на родительской диаграмме, на котором подсвечен декомпозируемый данной диаграммой блок. Для диаграммы самого верхнего уровня (контекстной диаграммы) в поле помещается контекст TOP

Все элементы подвала перечислены в табл. 3.2.

Таблица 3.2

Элементы "подвала" диаграммы **IDEF0**

Поле	Назначение
Node	Номер диаграммы, совпадающий с номером родительского функционального блока.
ТкЦ	Имя родительского функционального блока.
Number (еще называют C-Number)	Уникальный идентификатор данной версии данной диаграммы. Таким образом, каждая новая версия данной диаграммы будет иметь новое значение в этом поле. Как правило, C-Number состоит из инициалов автора (которые предполагаются уникальными среди всех аналитиков проекта) и последовательного уникального идентификатора, например SDO005. При публикации эти номера могут быть заменены стандартными номерами страниц. Если диаграмма замещает другую диаграмму, номер заменяемой диаграммы может быть заключен в скобки -— SDO005 (SDOU04). Это обеспечивает хранение истории изменений всех диаграмм модели.

Цикл "эксперт-аналитик". Подобно циклу автор-редактор, применяющемуся в книгоиздательском деле, диаграммы **IDEF0** пересматриваются и изменяются для обеспечения точности отражения предметной области и улучшения своего качества.

Для каждого рецензента автором, как правило, подготавливается свой набор диаграмм. Предложения по изменениям и исправлениям возвращаются рецензентами автору для внесения их в модель. При возникновении разногласий между автором и рецензентом спорная диаграмма обычно рассылается всем рецензентам для достижения группового консенсуса.

Формально механизм рецензирования и модификации диаграмм поддерживается полями Status и нумерацией диаграмм, контроль истории изменений — полем Field (см. табл. 3.1).

Построение моделей. Ни одна модель не должна строиться без ясного осознания объекта и целей моделирования. Выбранное определение цели моделирования должно отвечать на следующие вопросы:

- Почему моделируется данный процесс?
- Что выявит данная модель?
- Как ознакомившиеся с этой моделью смогут ее применить?

Следующее предложение может служить примером формулирования цели моделирования. Выявить задачи каждого работника компании и понять в целом взаимосвязь между отдельно взятыми задачами для разработки руковод-

ства по обучению новых сотрудников.

Модели строятся для того, чтобы ответить на набор доставленных вопросов. Такие вопросы формулируются на ранних стадиях моделирования и впоследствии служат основой для четкого и краткого определения цели моделирования. Примерами таких вопросов могут быть:

- Каковы задачи менеджера?
- Каковы задачи клерка?
- Кто контролирует работу?
- Какая технология нужна для выполнения каждого шага? и т.п.

Точка зрения. С методической точки зрения при моделировании полезно использовать мнение экспертов, имеющих разные взгляды на предметную область, однако каждая отдельно взятая модель должна разрабатываться исходя из единственной заранее определенной точки зрения. Часто другие точки зрения вкратце документируются в прикрепленных диаграммах **FEO** (см. ниже) исключительно для наглядности изложения.

Точку зрения нужно подбирать достаточно аккуратно, основой для выбора должна служить поставленная цель моделирования. Наименованием точки зрения может быть наименование должности, подразделения или роли (например, руководитель отдела или менеджер по продажам). Как и в случае с определением цели моделирования, четкое определение точки зрения необходимо для обеспечения внутренней целостности модели и предотвращения постоянного изменения ее структуры. Может оказаться необходимым построение моделей с разных точек зрения для детального отражения всех особенностей выделенных в системе функциональных блоков

Границы моделирования. Одним из положительных результатов построения функциональных моделей оказывается прояснение границ моделирования системы в целом и ее основных компонентов. Хотя и предполагается, что в процессе работы над моделью будет происходить некоторое изменение границ моделирования, их вербальное (словесное) описание должно поддерживаться с самого начала для обеспечения координации работы участвующих в проекте аналитиков. Как и при определении цели моделирования, отсутствие границ затрудняет оценку степени завершенности модели, поскольку границы моделирования имеют тенденцию к расширению с ростом размеров модели.

Границы моделирований имеют два компонента: ширину охвата и глубину детализации. Ширина охвата обозначает внешние границы моделируемой системы. Глубина детализации определяет степень подробности, с которой нужно проводить декомпозицию функциональных блоков.

Чтобы облегчить правильное определение границ моделирования при разработке моделей **IDEF0**, существенные усилия затрачиваются на разработку и рецензирование контекстной диаграммы **IDEF0** (диаграммы "самого верхнего" уровня). Иногда даже прибегают к построению дополнительной диаграммы для отображения уровня более высокого, чем контекстный, для данной модели, что позволяет обозначить систему, внутри которой располагается объект для моделирования. Существенные затраты на разработку контекстной диаграммы

вполне оправданы, поскольку она является своего рода "точкой отсчета" для остальных диаграмм модели и вносимые в нее изменения каскадом отражаются на все лежащие ниже уровни.

Когда границы моделирования понятны, становятся ясными и причины, по которым некоторые объекты системы не вошли в модель.

Выбор наименования контекстного блока. Рекомендуемой последовательностью действий при построении модели "с нуля" являются: формулирование цели моделирования, выбор точки зрения, определение границ моделирования. Наименование контекстного блока — функционального блока самого высокого уровня — обобщает определение границ моделирования.

Правила подбора имени для контекстного блока в целом не отличаются от общих правил наименования функциональных блоков, поэтому для них обычно подбирают обобщающие названия, типа "Управление отделом по работе с клиентами", "Обработка заказов" и т.п.

Определение стрелок на контекстной диаграмме. Стрелки диаграмм **IDEF0** обычно проще проектировать в следующем порядке: выход, вход, механизм исполнения, управление. Каждый функциональный блок обозначает отдельную функцию, и эта функция часто имеет ясно и кратко описываемые результаты работы. Наличие неясностей при анализе выходов того или иного функционального блока — возможный сигнал необходимости проведения реинжиниринга рассматриваемого бизнес-процесса.

Определение выходов. После идентификации возможных выходов полезно провести анализ модели на предмет покрытия ею *всех возможных* сценариев поведения процесса. Это означает, что если существует вероятность возникновения той или иной ситуации в ходе процесса, модель отражает возможность возникновения такой ситуации. Многие начинающие аналитики забывают отразить негативные результаты работы функциональных блоков. Например, блок "Провести экзамен по вождению" определенно произведет поток водителей, только что получивших права, но вполне правомерно ожидать и потока лиц, не сдавших экзамен. Негативные результаты часто используются в качестве обратных связей, анализ на их наличие должен проводиться для каждого блока. Важным также является необходимость включения в модель спорных стрелок, принятие решения о наличии которых в модели вполне можно переложить на плечи рецензирующих модель экспертов.

Определение входов. Входы можно рассматривать как особым образом преобразуемые функциональными блоками для производства выхода сырье или информацию. В производственных отраслях определить, как входное сырье преобразуется в готовую продукцию, обычно довольно просто. Однако при моделировании информационных потоков входной поток данных может представляться не потребляемым и не обрабатываемым вообще. Случаи, когда входящие и исходящие стрелки называются в точности одинаково, крайне редки и в основном указывают на бесполезность данного блока для системы в целом или на некорректный выбор имени для исходящей стрелки. Решением может служить применение более подробного описания для входящих и исходящих

потоков данных. Например, вход может иметь название "Предварительный диагноз пациента", а выход—"Уточненный диагноз пациента".

Определение механизмов исполнения. После создания входов и выходов можно приступить к рассмотрению механизмов исполнения, или ресурсов, относящихся к функциональному блоку. В понятие механизма исполнения входят: персонал, оборудование, информационные системы и т.п. Например, функциональный блок "Собрать деталь" может потребовать использования какого-либо оборудования, например гаечного ключа. При приеме экзаменов на водительские права механизмом исполнения является инспектор ГИБДД. Как правило, определить механизмы исполнения для функциональных блоков довольно просто.

Определение управления. Должно быть определено управление, контролирующее ход работы функционального блока. Все функциональные блоки в **IDEF0** должны иметь хотя бы одно управление. В случаях, когда не ясно, относить ли стрелку к входу или к управлению, следует ее рисовать как управление. Важно помнить, что управление можно рассматривать как особую форму входа функционального блока.

Когда контекстная диаграмма представляется завершенной, попробуйте задать следующие вопросы:

- Обобщает ли диаграмма моделируемый бизнес-процесс?
- Согласуется ли диаграмма с границами моделирования, точкой зрения и целью моделирования?
- Подходит ли выбранный уровень детализации стрелок для контекстного блока? (Обычно на контекстной диаграмме рекомендуется рисовать не более шести стрелок каждого типа.)

Нумерация блоков и диаграмм. Все функциональные блоки **IDEF0** нумеруются. В номерах допускается использование префиксов произвольной длины, но в подавляющем большинстве моделей используется префикс А. Номер блока проставляется за префиксом. Контекстный блок всегда имеет номер А0.

Префикс повторяется для каждого блока модели. Номера используются для отражения уровня декомпозиции, на котором находится блок. Блок А0 декомпозируется в блоки А1, А2, А3 и т.д. А1 декомпозируется в А11, А12, А13 и т.д. А11 декомпозируется в А111, А112, А113 и т.д. Для каждого уровня декомпозиции в конец номера добавляется одна цифра.

Связь между диаграммой и ее родительским функциональным блоком. Функциональный блок декомпозируется, если необходимо детально описать его работу. При декомпозиции блока полезно рассмотреть его жизненный цикл, это поможет определить функциональные блоки получающейся "детской" диаграммы. Например, жизненный цикл блока "Поджарить бифштекс" может выглядеть как следующая последовательность: "Подготовить продукты", "Отбить мясо", "Разогреть масло" и т.д.

При моделировании **IDEF0** важно иметь в виду, что граница детской диаграммы есть граница родительского функционального блока. Это означает, что

вся работа выполняется блоками самого нижнего уровня. В отличие от иерархии, применяемой в структурном программировании, блоки верхнего уровня не являются субъектами управления для блоков нижнего уровня. Это означает, что в **IDEF0** дети — это те же объекты, что и их родители, только показанные с большей детализацией. Действия генерального директора компании на диаграммах **IDEF0** могут отражаться рядом с действиями простых рабочих.

На концах *граничных стрелок* (начинающихся или заканчивающихся за пределами диаграммы) детских диаграмм помещаются коды **ICOM**, чтобы показать, где находится соответствующая стрелка на родительской диаграмме. Они нужны для проверки целостности модели и могут быть полезны, когда порядок расположения стрелок на детской диаграмме отличается от порядка их расположения на родительской диаграмме. Код **ICOM** состоит из латинской буквы **I, C, O** или **M** и числа, показывающего расположение стрелки на родительской диаграмме в порядке сверху вниз или слева направо.

Два подхода к началу моделирования ("в ширину" и "в глубину"). Модели могут проектироваться с использованием подхода "в ширину", когда каждая диаграмма максимально детализируется перед своей декомпозицией, и с подходом "в глубину", когда сначала определяется иерархия блоков, а затем создаются соединяющие их стрелки. Естественно, возможно применение комбинации этих подходов, причем иерархия блоков может иногда немного измениться после того, как нарисованы стрелки. Это происходит из-за того, что создание стрелок может изменить понимание внутренней архитектуры моделируемого объекта.

Когда остановиться? Сформулированная цель моделирования содержит вопросы, на которые должна отвечать модель. Когда становится возможным получение ответов на них с помощью модели, модель считается удовлетворяющей поставленным требованиям и рассматривается как завершенная. При построении декомпозиции первого уровня нужно следить за тем, чтобы все блоки на диаграмме лежали внутри определенных ранее границ моделирования. Перед декомпозированием блока нужно удостовериться, не приведет ли это к превышению установленной ранее глубины детализации для данной модели. Еще одно правило состоит в том, что моделирование **IDEF0** должно продолжаться до тех пор, пока стрелки предшествования (вход и выход) преобладают на диаграммах.

При необходимости дальнейшей детализации отдельных процессов могут быть использованы диаграммы **IDEF3.**

Другие диаграммы IDEF0. В дополнение к контекстным диаграммам и диаграммам декомпозиции при разработке и представлении моделей могут применяться другие виды **IDEF0**-диаграмм.

Дерево модели. Это обзорная диаграмма, показывающая структуру всей модели. На рис. 3.10 приведен фрагмент такой диаграммы. Обычно вершина дерева соответствует контекстному блоку, под вершиной выстраивается вся иерархия блоков модели. Однако не запрещается назначать вершиной произвольный блок, помещая под ним все его детские блоки. Из-за высокой итера-

тивности функционального моделирования можно ожидать, что дерево модели будет неоднократно изменяться существенным образом до тех пор, пока не будет получена его стабильная версия. Обзор модели с использованием дерева помогает сконцентрироваться на функциональной декомпозиции модели.

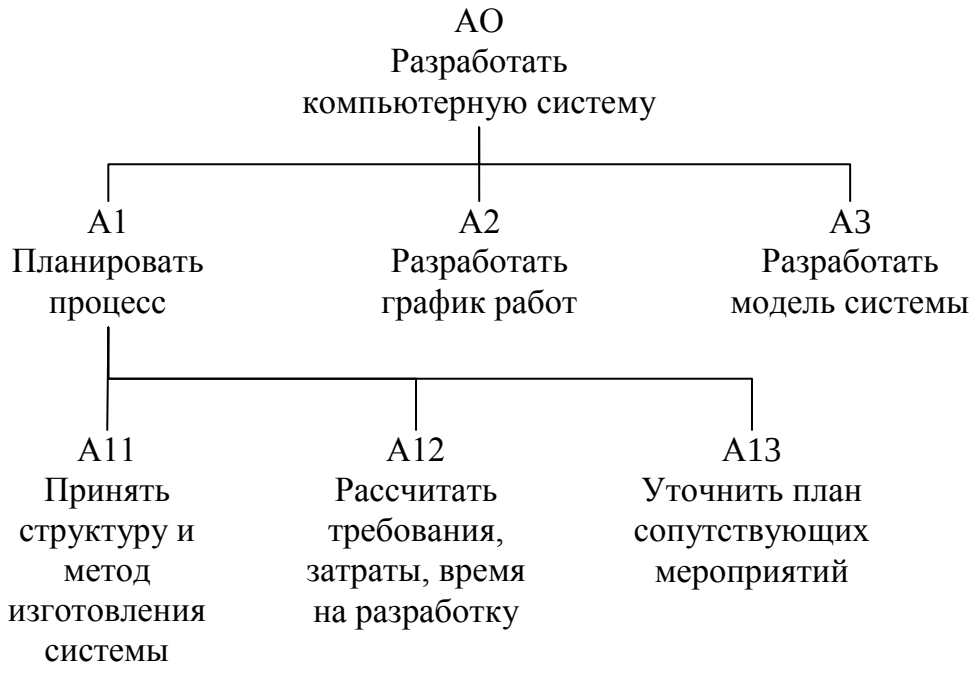


Рис. 3.10. Фрагмент дерева модели

Презентационные диаграммы. Презентационные диаграммы (For Exposition Only diagrams — **FEO** diagrams) часто включают в модели, чтобы проиллюстрировать другие точки зрения или детали, выходящие за рамки традиционного синтаксиса **IDEF0**. Диаграммы **FEO** допускают нарушение любых правил построения диаграмм **IDEF0** в целях выделения важных с точки зрения аналитика частей модели. Естественно, если диаграмма **FEO** включена в модель исключительно для отображения другой точки зрения на систему, она скорее всего будет выглядеть как обыкновенная диаграмма **IDEF0**, удовлетворяя всем ограничениям **IDEF0.**

Один из способов использования **FEO**-диаграмм состоит в отделении функционального блока от его окружения посредством создания диаграммы с единственным блоком и всеми относящимися к нему стрелками наподобие контекстной диаграммы. Это может оказаться полезным в ситуациях, когда необходимо быстро получить информацию об интерфейсе (стрелках) функционального блока, а соответствующая диаграмма декомпозиции содержит слишком много объектов.

Кроме того, встречаются следующие виды презентационных диаграмм:

• копия диаграммы **IDEF0**, которая содержит все функциональные блоки, и стрелки, относящиеся только к одному из функциональных блоков, — это позволяет отразить взаимодействие между этим блоком и другими объектами

диаграммы;

• копия диаграммы **IDEF0**, которая содержит все функциональные блоки, и стрелки, непосредственно относящиеся только к входу и (или) к выходу родительского блока;

• различные точки зрения, как правило, на глубину одного уровня декомпозиции.

3.2.3. Взаимосвязь моделей IDEF0 и IDEF3

Действия, выполняемые в функциональных блоках. Как правило, при работе с пластиковой картой клиент не производит всех доступных ему при этом действий, выполняя ограниченный набор операций. Например, при оплате покупки не производится снятие наличных, а при проверке баланса состояние счета вообще не изменяется (это верно, конечно, только в случае, если карта обслуживается приличным банком). Мы можем декомпозировать функциональный блок "Обработка операций с пластиковыми картами", создав дополнительные блоки для оплаты покупок, снятия наличных, проверки баланса и т.п. Вместо этого можно создать отдельные модели **IDEF3** для каждого из этих действий. Это, в частности, полезно, если в дальнейшем предполагается заняться оцениванием соответствующих операций по тем или иным параметрам.

Более простой альтернативой предложенным выше двум подходам может служить так называемая *таблица вызова* (activation table), описывающая различные комбинации входов, выходов, управлений и механизмов исполнения для каждого способа вызова функционального блока на исполнение. *Вызов* — это уникальная конфигурация значений входа, управления и требований к механизмам исполнения (табл. 3.3). Каждому вызову присваивается уникальное имя в пределах блока и перечисляются значения различных стрелок. Комбинация значений стрелок должна быть уникальной для каждого вызова, из чего следует, что для каждого вызова любые две одинаковые стрелки не могут иметь одинаковых значений.

Таблица 3.3

Таблица вызовов для блока "Подсчитать наличные"

Вызов	Стрелка	Значение стрелки
Значительная сумма наличных денег	Наличные деньги	Более 1000 руб.
	Счетчик банкнот	1 требуется
Мелкая сумма наличных денег	Наличные деньги	Не более 1000 руб.
	Счетчик банкнот	0 требуется

Информация о вызовах из табл. 3.3 также дает определенную информацию о стрелках управления данного функционального блока. Например, мы можем предположить, что политика банка при подсчете сумм наличных за-

ключается в использовании счетчиков банкнот для сумм, превышающих 1000 руб.

Создание моделей IDEF3 для отображения блоков IDEF0. Для иллюстрирования вызовов листовых функциональных блоков **IDEF0** (т.е. блоков, не имеющих диаграмм декомпозиции) может быть применено построение моделей **IDEF3**. Если развитие модели **IDEF0** предполагается аналитиками именно таким способом, моделями **IDEF3** должен быть тщательно документирован каждый возможный вызов функционального блока. Соответствующие таблицы вызовов (наподобие табл. 3.3) можно будет получить впоследствии из соответствующих диаграмм **IDEF3.**

Итак, методология функционального моделирования **IDEF0** — это технология описания системы в целом как множества взаимозависимых действий, или функций. **IDEF0** имеет функциональную направленность. **IDEF0** — функции системы исследуются независимо от объектов, которые обеспечивают их выполнение. Одной из основных идей моделей **IDEF0** является построение двух видов моделей: "как есть" и "как должно быть". Это нужно при проведении реинжиниринга бизнес-процессов организации. Кроме того, **IDEF0** обеспечивает удобный язык обмена информацией о моделируемой системе.

3.3. Методология описания бизнес-процессов IDEF3

IDEF3 — способ описания процессов, основной целью которого является обеспечение структурированного метода, используя который эксперт в предметной области может описать положение вещей как упорядоченную последовательность событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу [15-17].

Технология **IDEF3** хорошо приспособлена для сбора данных, требующихся для проведения структурного анализа системы. В отличие от большинства технологий моделирования бизнес-процессов, **IDEF3** не имеет жестких синтаксических или семантических ограничений, делающих неудобным описание неполных или нецелостных систем. Кроме того, автор модели (системный аналитик) избавлен от необходимости смешивать свои собственные предположения о функционировании системы с экспертными утверждениями в целях заполнения пробелов в описании предметной области. На рис. 3.11 изображен пример описания процесса с использованием методологии **IDEF3.**

Технология **IDEF3** также может быть использована как метод проектирования бизнес-процессов. **IDEF3**-моделирование органично дополняет традиционное моделирование с использованием стандарта **IDEF0**. В настоящее время оно получает все большее распространение как вполне жизнеспособный путь построения моделей проектируемых систем для дальнейшего анализа имитационными методами. Имитационное тестирование часто используют для оценки эксплуатационных качеств разрабатываемой системы.

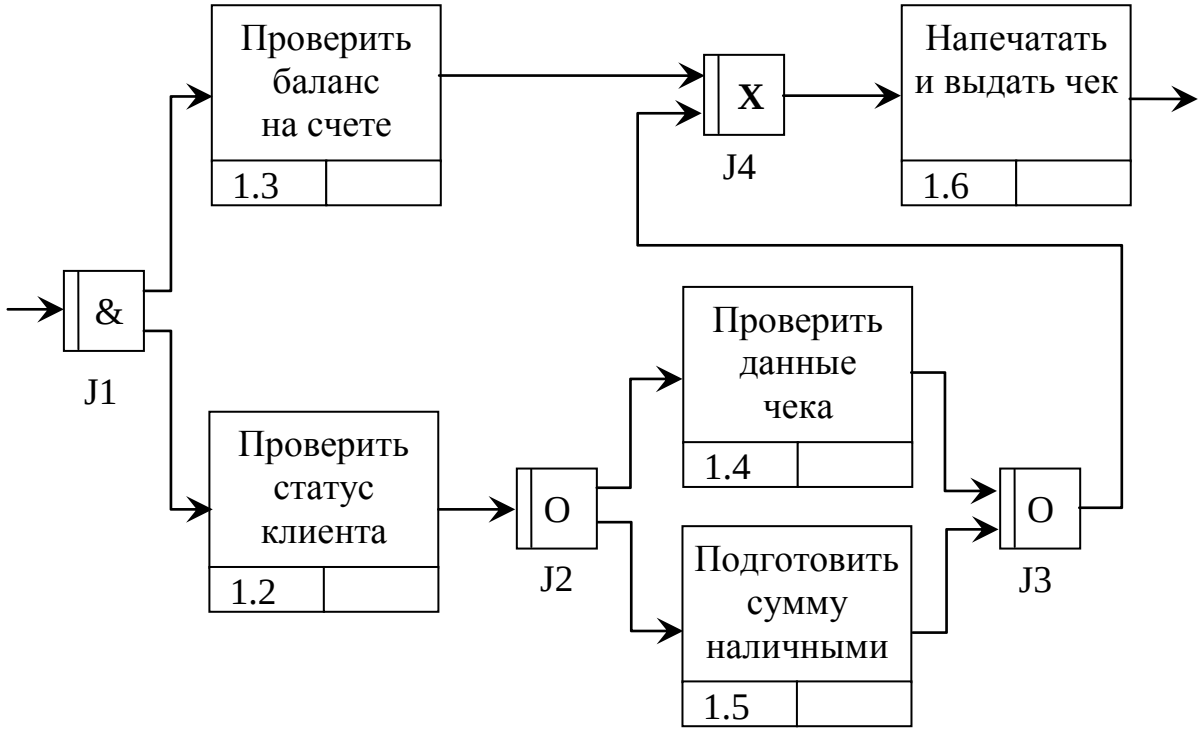


Рис. 3.11. Описание процесса в методологии **IDEF3**

3.3.1. Синтаксис и семантика моделей IDEF3

Модели IDEF3. Основой модели IDEF3 служит так называемый *сценарий* бизнес-процесса, который выделяет последовательность действий или подпроцессов анализируемой системы. Поскольку сценарий определяет назначение и границы модели, довольно важным является подбор подходящего наименования для обозначения действий. Для подбора необходимого имени применяются стандартные рекомендации по предпочтительному использованию глаголов и отглагольных существительных. Например, "Обработать заказ клиента" или "Применить новый дизайн" — вполне подходящие названия сценариев.

Точка зрения для большинства моделей должна быть явным образом документирована. Обычно это название набора должностных обязанностей человека, являющегося источником информации о моделируемом процессе.

Для системного аналитика также важно понимание цели моделирования — набора вопросов, ответами на которые будет служить модель, границ моделирования (какие части системы войдут в модель, а какие не будут в ней отображены) и целевой аудитории (для кого разрабатывается модель).

Диаграммы. Как и в любой рассматриваемой в этой книге технологии моделирования действий, главной организационной единицей модели **IDEF3** является диаграмма. Взаимная организация диаграмм внутри модели **IDEF3** особенно важна в случае, когда модель заведомо создается для последующего опубликования или рецензирования, что является вполне обычной практикой при проектировании новых систем. В этом случае системный аналитик должен

позаботиться о таком информационном наполнении диаграмм, чтобы каждая из них была самодостаточной и в то же время понятной читателю.

Единица работы. Действие. Аналогично другим технологиям моделирования действие, или в терминах **IDEF3** "единица работы" (Unit of Work — UOW) — другой важный компонент модели. Диаграммы **IDEF3** отображают действие в виде прямоугольника. Как уже отмечалось, действия именуются с использованием глаголов или отглагольных существительных, каждому из действий присваивается уникальный идентификационный номер. Этот номер не используется вновь даже в том случае, если в процессе построения модели действие удаляется. В диаграммах **IDEF3** номер действия обычно предваряется номером его родителя (рис. 3.12).

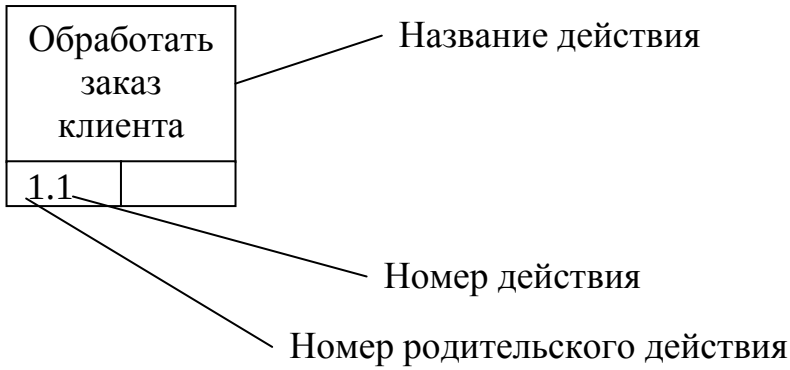


Рис. 3.12. Изображение и нумерация действия в диаграмме **IDEF3**

Связи. Связи выделяют существенные взаимоотношения между действиями. Все связи в **IDEF3** являются однонаправленными, и, хотя стрелка может начинаться или заканчиваться на любой стороне блока, обозначающего действие, диаграммы **IDEF3** обычно организовываются слева направо таким образом, что стрелки начинаются на правой и заканчиваются на левой стороне блоков. В табл. 3.4 приведены три возможных типа связей.

Связь типа "Временное предшествование". Как видно из названия, связи этого типа отражают, что исходное действие должно полностью завершиться, прежде чем начнется выполнение конечного действия. Связь должна быть поименована таким образом, чтобы человеку, просматривающему модель, была понятна причина ее появления. Во многих случаях завершение одного действия инициирует начало выполнения другого, как показано на рис. 3.13. В этом примере автор должен принять рекомендации рецензентов, прежде чем начать вносить соответствующие изменения в работу.

Связь типа "Объектный поток". Одной из наиболее часто встречающихся причин использования связи типа "объектный поток" состоит в том, что некоторый объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия. Такая связь отличается от связи временного предшествования двойным концом обозначающей ее стрелки. Наименования потоковых связей должны четко идентифицировать

объект, который передается с их помощью. Временная семантика объектных связей аналогична связям предшествования. Это означает, что порождающее объектную связь исходное действие должно завершиться, прежде чем конечное действие начнет выполняться, как показано на рис. 3.14. В приведенном примере счет на оплату услуг является результатом выполнения действия 1.1. Счет необходим для проведения оплаты услуг.

Таблица 3.4

Типы связей в модели **IDEF3**

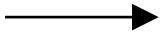
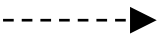
Изображение	**Название**	**Назначение**
	Временное предшествование (Temporal precedence)	Исходное действие должно завершиться прежде, чем конечное действие сможет начаться
	Объектный поток (Object flow)	Выход исходного действия является входом конечного действия. Из этого, в частности, следует, что исходное действие должно завершиться прежде, чем конечное действие
	Нечеткое отношение (Relationship)	Вид взаимодействия между исходным и конечным действиями задается аналитиком отдельно для каждого случая использования такого отношения

Рис. 3.13. Связь типа "Предшествование" между действиями 1.1 и 1.2

Связь типа "Нечеткое отношение". Связи этого типа используются для выделения отношений между действиями, которые невозможно описать с использованием предшественных или объектных связей. Значение каждой такой связи должно быть определено, поскольку связи типа "Нечеткое отношение" сами по себе не предполагают никаких ограничений. Одно из применений нечетких отношений — отображение взаимоотношений между параллельно выполняющимися действиями. Рис. 3-15 иллюстрирует фрагмент процесса запуска бензопилы с водяным охлаждением и нечеткое отношение между действиями "Запустить двигатель" и "Запустить водяной насос". Название стрелки

может быть использовано для описания природы отношения, более подробное объяснение может быть приведено в виде отдельной ссылки.

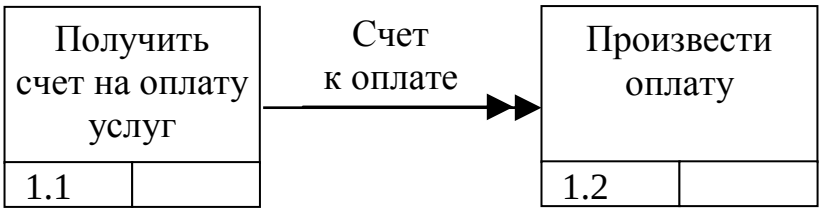


Рис. 3.14. Объектная связь между действиями 1.1 и 1.2

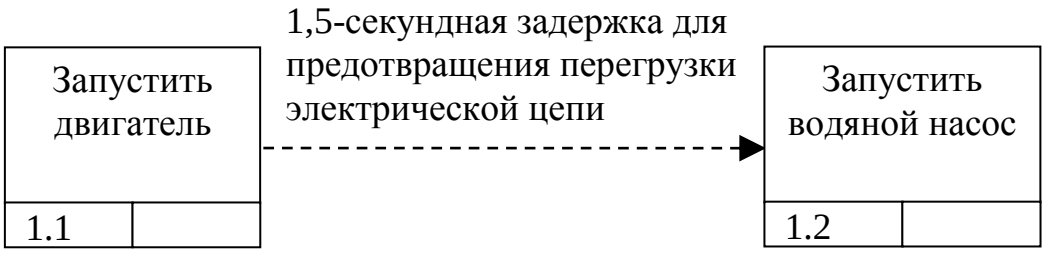


Рис. 3.15. Связь типа "Нечеткое отношение"

Наиболее часто нечеткие отношения используются для описания специальных случаев связей предшествования, например для описания альтернативных вариантов временного предшествования. Обратимся еще раз к рис. 3.13. На рис. 3.16 вертикальные линии показывают начало и окончание действий 1.1 и 1.2, имеющих предшественную связь. В соответствии с рисунком внесение исправлений в работу начинается ПОСЛЕ принятия всех замечаний от рецензентов.

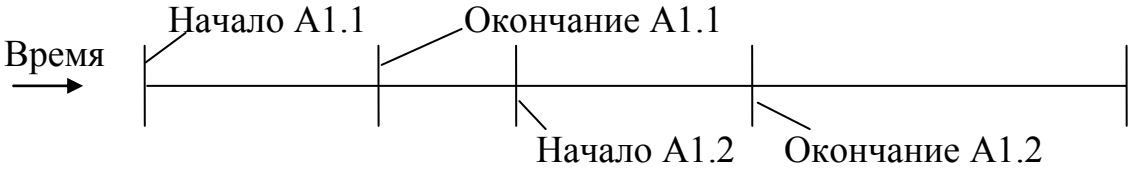


Рис. 3.16. Временная шкала выполнения действия для рис. 3-13

Альтернативная предшественной связи с рис. 3.13 связь нечеткого отношения представлена на рис. 3.17. В этом примере внесение исправлений начинается по мере получения замечаний от рецензентов, т.е. до непосредственного окончания действия по принятию замечаний.

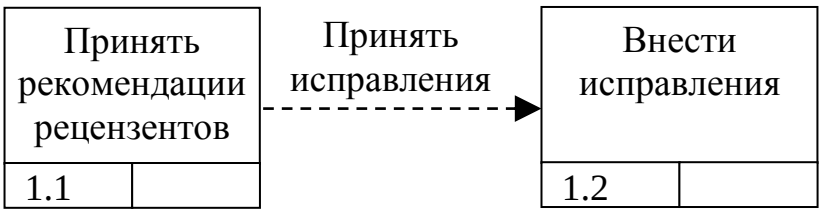


Рис. 3.17. Альтернатива связи предшествования

На рис. 3.18 приведена соответствующая этой ситуации временная шкала.

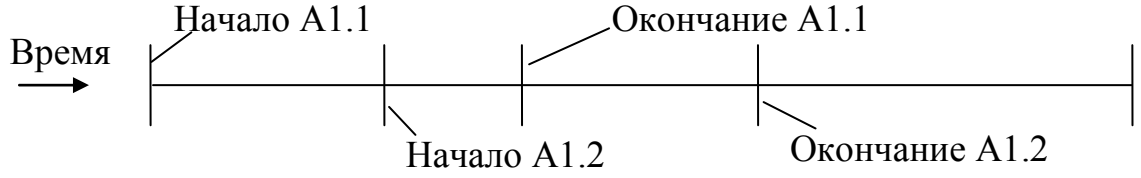


Рис. 3.18. Альтернативная временная шкала

Отметим еще раз необходимость четкого документирования временных ограничений между действиями, соединенными нечетким отношением. В качестве примера рассмотрим еще одну временную шкалу (рис. 3.19) для рис. 3.13.

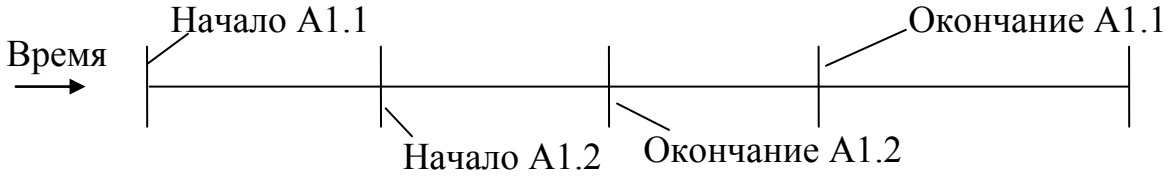


Рис. 3.19. Другой вариант альтернативной временной шкалы

В случае, изображенном на рис. 3.19, внесение исправлений будет начато после получения первых замечаний, однако будет закончено ПЕРЕД тем, как все замечания от рецензентов будут получены и обработаны.

Оба рассмотренных выше варианта временной альтернативной шкалы могут иметь место в реальности, поэтому корректная интерпретация нечеткого отношения должна быть документирована в модели. Важно отметить, что *корректность* в этом случае означает именно интерпретацию, которая в точности отображает документируемую ситуацию, а не интерпретацию, более эффективную для работы системы с точки зрения аналитика

Соединения. Завершение одного действия может инициировать начало выполнения сразу нескольких других действий, или, наоборот, определенное действие может требовать завершения нескольких других действий для начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используются для описания *ветвления* процесса.

- Разворачивающие соединения используются для разбиения потока. Завершение одного действия вызывает начало выполнения нескольких других.
- Сворачивающие соединения объединяют потоки. Завершение одного или нескольких действий вызывает начало выполнения только одного другого действия.

Существуют три типа соединений.

"И"-соединения. Соединения этого типа инициируют выполнение всех своих конечных действий. Все действия, присоединенные к сворачивающему "И"-соединению, должны завершиться, прежде чем может начать выполняться

следующее действие. На рис. 3.20 после обнаружения пожара инициируются включение пожарной сигнализации, вызов пожарной охраны и начинается тушение пожара. Запись в журнал производится только тогда, когда все три перечисленных действия завершены.

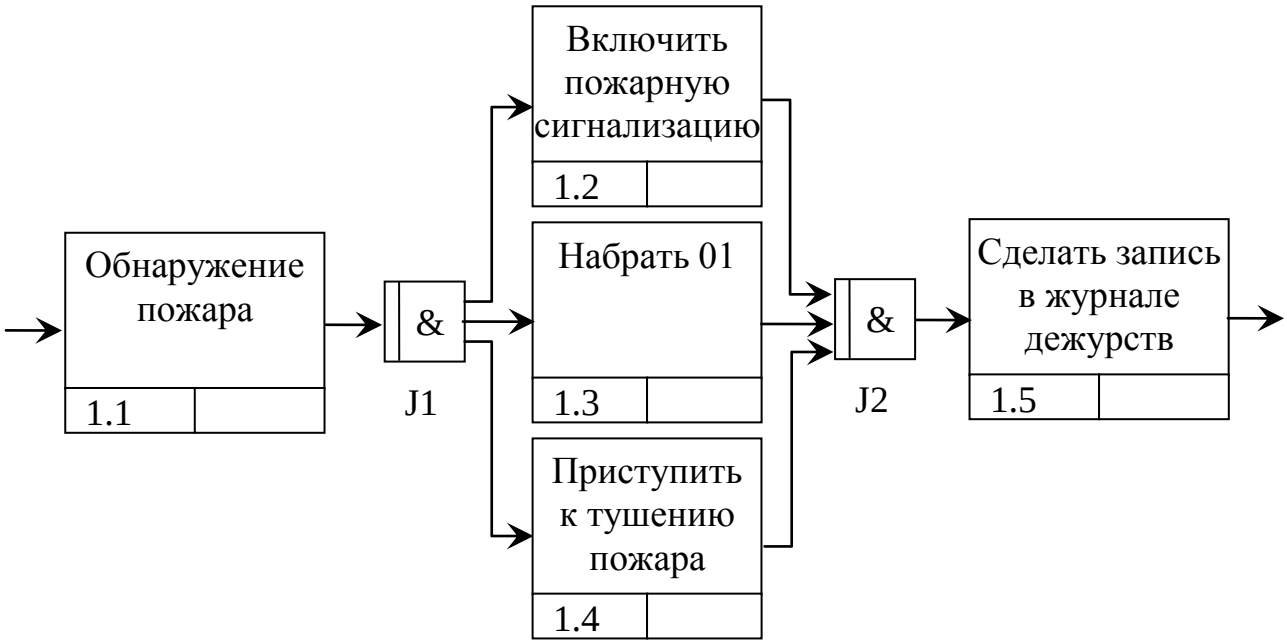


Рис. 3.20. Асинхронное "И" - соединение

Соединение "Эксклюзивное ИЛИ". Вне зависимости от количества действий, прицепленных к сворачивающему или разворачивающему соединению "Эксклюзивное ИЛИ", инициировано будет только одно из них, и поэтому только одно из них будет завершено перед тем, как любое действие, следующее за сворачивающим соединением "Эксклюзивное ИЛИ", сможет начаться.

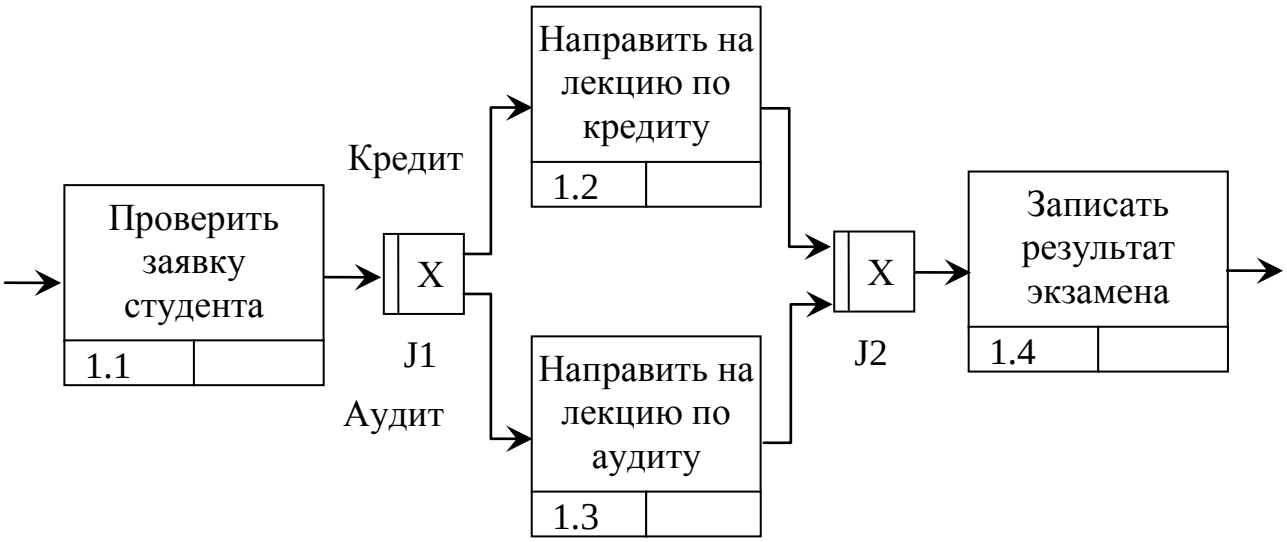


Рис. 3.21. Соединение "Эксклюзивное ИЛИ"

Если правила активации соединения известны, они обязательно должны быть документированы либо в его описании, либо пометкой стрелок, исходящих из разворачивающего соединения, как показано на рис. 3.21.

На рис. 3.21 соединение "Эксклюзивное ИЛИ" используется для отображения того факта, что студент не может одновременно быть направлен на лекции по двум разным курсам.

Соединение "ИЛИ". Соединения этого типа предназначены для описания ситуаций, которые не могут быть описаны двумя предыдущими типами соединений. Аналогично связи нечеткого отношения соединение "ИЛИ" в основном определяется и описывается непосредственно системным аналитиком. На рис. 3.22 соединение J2 может активировать проверку данных чека и (или) проверку суммы наличных. Проверка чека инициируется, если покупатель желает расплатиться чеком, проверка суммы наличных — при оплате наличными. И то, и другое действие инициируется при частичной оплате чеком и частичной – наличными.

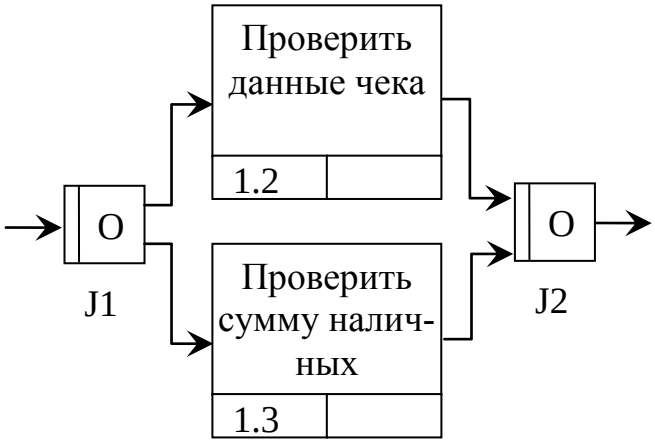


Рис. 3.22. Асинхронное соединение "ИЛИ"

Синхронные и асинхронные соединения. В рассмотренных примерах связей "И" и "ИЛИ" мы не затрагивали отношений между началом и окончанием действий, инициируемых разворачивающими соединениями. Все действия в этих примерах выполнялись асинхронно, т.е. они не должны были начинать выполняться одновременно. Однако есть случаи, когда время начала или окончания параллельно выполняемых действий должно быть одинаковым, т.е. действия должны выполняться синхронно. Для моделирования такого поведения системы используются синхронные соединения. В табл. 3.5 приведены виды синхронных соединений.

Синхронное соединение обозначается двумя вертикальными линиями внутри обозначающего его прямоугольника в отличие от одной вертикальной линии в асинхронном соединении.

Во многих спортивных состязаниях выстрел стартового пистолета, запуск секундомера и начало состязания должны произойти одновременно. В противном случае состязание будет нечестным.

Таблица 3.5

Синхронные и асинхронные соединения модели IDEF3

Обозначение	Наименование	Смысл в случае слияния стрелок	Смысл в случае разветвления стрелок
&	Асинхронное "И"	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
&	Синхронное "И"	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
O	Асинхронное "ИЛИ"	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
O	Синхронное "ИЛИ"	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
X	Эксклюзивное "ИЛИ"	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Рис. 3.23 иллюстрирует модель этого примера, построенную с использованием синхронного соединения.

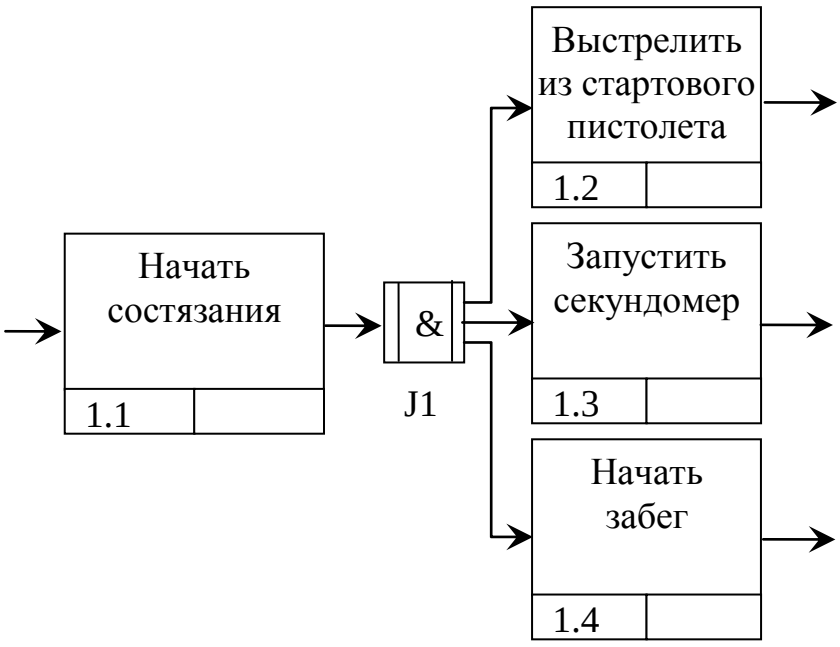


Рис. 3.23. Синхронное соединение "И"

Заметим, что синхронное разворачивающее соединение не обязательно должно иметь парное себе сворачивающее соединение. Действительно, начинающиеся одновременно действия вовсе не обязаны оканчиваться одновременно, как это видно из примера с состязаниями. Также возможны ситуации синхронного окончания асинхронно начавшихся действий.

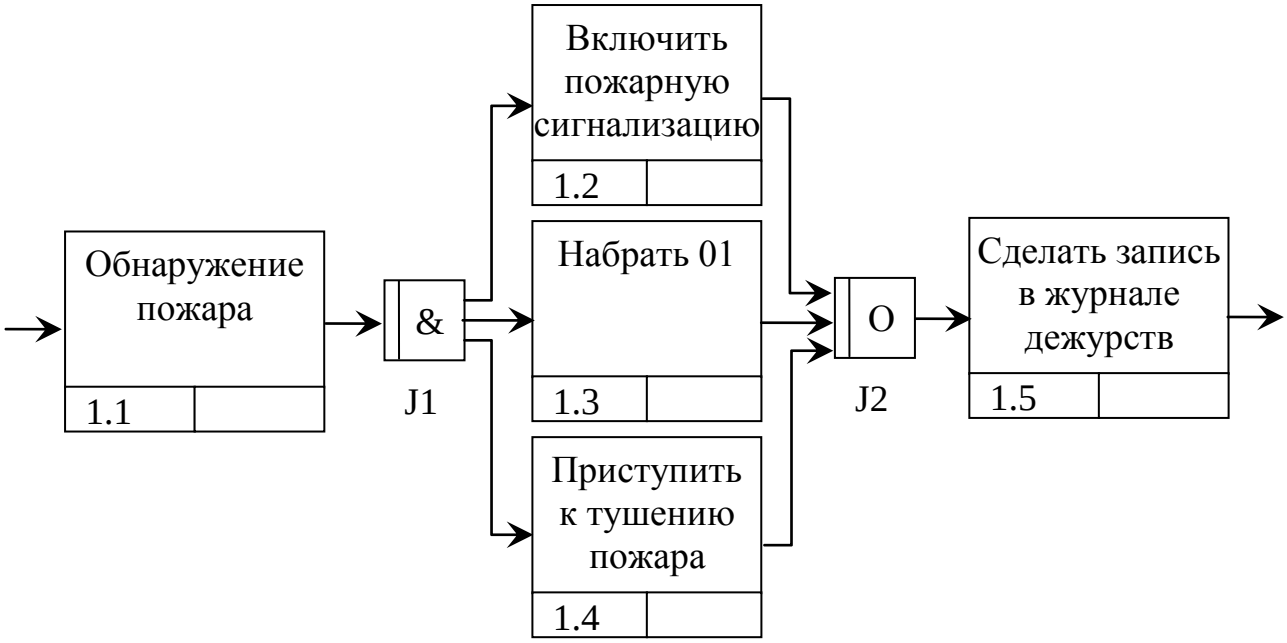


Рис. 3.24. Пример комбинации двух типов соединений

Парность соединений. Все соединения на диаграммах должны быть парными, из чего следует, что любое разворачивающее соединение имеет парное себе сворачивающее. Однако типы соединений вовсе не обязательно должны совпадать. На рис. 3.24 разворачивающее "И"-соединение имеет парное сворачивающее "ИЛИ"-соединение. Интерпретация соединения J1 аналогична случаю, показанному на рис. 3.20. Соединение J2 интерпретируется следующим образом: после включения пожарной сигнализации и (или) вызова пожарных и (или) начала тушения производится запись в журнал.

Комбинации соединений. Соединения могут комбинироваться для создания более сложных правил ветвления (рис. 3.25). Комбинации соединений следует использовать с осторожностью, поскольку перегруженные ветвлением диаграммы могут оказаться сложными для восприятия.

Указатели — это специальные символы, которые ссылаются на другие разделы описания процесса. Они выносятся на диаграмму для привлечения внимания читателя к каким-либо важным аспектам модели.

Указатель изображается на диаграмме в виде прямоугольника, похожего на изображение действия. Имя указателя обычно включает его тип (например, ОБЪЕКТ, UOB и т.п.) и идентификатор. На рис. 3.26 изображен указатель типа ОБЪЕКТ.

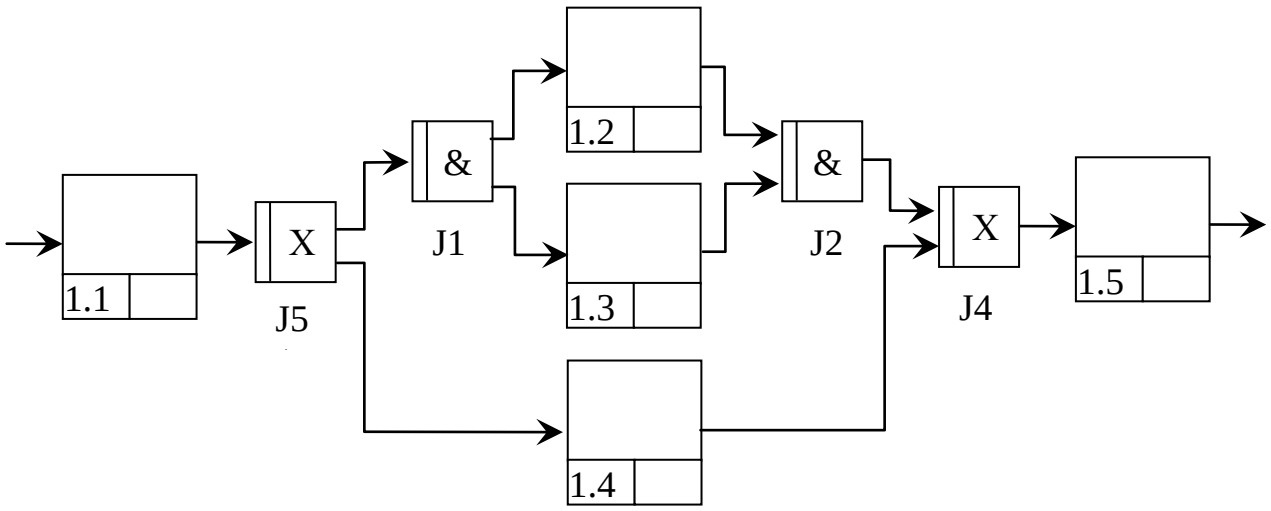


Рис. 3.25. Диаграмма **IDEF3** с комбинацией соединений

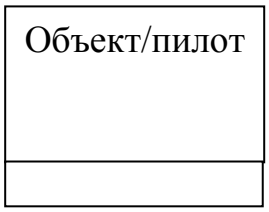


Рис. 3.26. Указатель ОБЪЕКТ

Существуют следующие типы указателей:

- ОБЪЕКТ (OBJECT) - служит для описания того, что в действии принимает участие какой-либо заслуживающий отдельного внимания объект;
- ССЫЛКА (GOTO) – служит для реализации цикличности выполнения действий. Указатель ССЫЛКА может относиться и к соединению;
- ЕДИНИЦА ДЕЙСТВИЯ (Unit of Behavior —UOB) – служит для помещения на диаграмму дополнительного экземпляра уже существующего действия без зацикливания. Например, если действие "Подсчет наличных" выполняется несколько раз, в первый раз оно создается как действие, а последующие его появления на диаграмме оформляются указателями UOB;
- ЗАМЕТКА (NOTE) – служит для документирования любой важной информации общего характера, относящейся к изображенному на диаграммах. В этом смысле ССЫЛКА служит альтернативой методу помещения текстовых заметок непосредственно на диаграммах;
- УТОЧНЕНИЕ (Elaboration — ELAB) – служит для уточнения или более подробного описания изображенного на диаграмме. Указатели УТОЧНЕНИЕ обычно используются для описания логики ветвления у соединений.

На рис. 3.27 показан пример отображения важного с точки зрения модели отношения между действием и объектом.

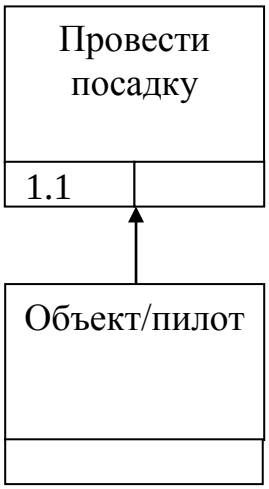


Рис. 3.27. Указатель ОБЪЕКТ ссылается на действие

Декомпозиция действий. Действия в **IDEF3** могут быть декомпозированы, или разложены на составляющие, для более детального анализа. Декомпозировать действие можно *несколько* раз. Это позволяет документировать альтернативные потоки процесса в одной модели.

Для корректной идентификации действий в модели с множественными декомпозициями схема нумерации действий расширяется и наряду с номерами действия и его родителя включает в себя порядковый номер декомпозиции. Например, в номере действия 1.2.5: 1 — номер родительского действия, 2 — номер декомпозиции, 5 — номер действия.

3.3.2. Требования IDEF3 к описанию бизнес-процессов

В этом подразделе мы рассмотрим построение **IDEF3** - диаграммы на основе выраженного в текстовом виде описания процесса. Предполагается, что в построении диаграммы принимают участие ее автор (в основном как системный аналитик) и один или несколько экспертов предметной области, которые будут представлять нам описание процесса.

Определение сценария, границ моделирования, точки зрения. Перед тем как попросить экспертов предметной области подготовить описание моделируемого процесса, должны быть документированы границы моделирования, чтобы экспертам была понятна необходимая глубина и полнота требуемого от них описания. Кроме того, если точка зрения аналитика на процесс отличается от обычной точки зрения для эксперта, это должно быть ясно и аккуратно описано.

Вполне возможно, что эксперты не смогут сделать приемлемое описание без применения формального опроса автором модели. В таком случае автор должен заранее приготовить набор вопросов таким же образом, как журналист заранее подготавливает вопросы для интервью

Определение действий и объектов. Результатом работы экспертов обычно является текстовый документ, описывающий интересующий аналитика круг вопросов. В дополнение к нему может иметься письменная документация, позволяющая пролить свет на природу изучаемого процесса. Вне зависимости от того, является ли информация текстовой или вербальной, она анализируется и разделяется частями речи для идентификации списка действий (глаголы и отглагольные существительные), составляющих процесс, и объектов (имена существительные), участвующих в процессе.

В некоторых случаях возможно создание графической модели процесса в присутствии экспертов. Такая модель также может быть разработана после сбора всей необходимой информации, что позволяет не отнимать время экспертов на детали форматирования получающихся диаграмм.

Поскольку модели **IDEF3** могут одновременно разрабатываться несколькими командами, **IDEF3** поддерживает простую схему резервирования номеров действий в модели. Каждому аналитику выделяется уникальный диапазон номеров действий, что обеспечивает их независимость друг от друга.

Последовательность и параллельность. Если модель создается после проведения интервью, аналитик должен принять решения по построению иерархии участвующих в модели диаграмм, например, насколько подробно будет детализироваться каждая отдельно взятая диаграмма. Если последовательность или параллельность выполнения действий окончательно не ясна, эксперты могут быть опрошены вторично (возможно, с использованием черновых вариантов незаконченных диаграмм) для получения недостающей информации. Важно, однако, различать предполагаемую (появляющуюся из-за недостатка информации о связях) и явную (ясно указанную в описании эксперта) параллельности.

Итак, **IDEF3** — это способ описания бизнес-процессов, который нужен дня описания положения вещей как упорядоченной последовательности событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу. **IDEF3** хорошо приспособлен для сбора данных, требующихся для проведения структурного анализа системы. Кроме того, **IDEF3** применяется при проведении стоимостного анализа поведения моделируемой системы.

3.4. Моделирование потоков данных (процессов)

3.4.1. Назначение диаграммы поток данных

Так же, как и диаграммы **IDEF0**, диаграммы потоков данных **DFD** моделируют систему как набор действий, соединенных друг с другом стрелками [13, 15-18]. Диаграммы потоков данных также могут содержать два новых типа объектов: объекты, собирающие и хранящие информацию — *хранилища(накопители) данных* и *внешние сущности* — объекты, которые моделируют взаимодействие с теми частями системы (или другими системами), которые выходят за границы моделирования.

В отличие от стрелок в **IDEF0**, которые иллюстрируют отношения, стрелки в **DFD** показывают, как объекты (включая и данные) реально перемещаются от одного действия к другому. Это представление потока вкупе с хранилищами данных и внешними сущностями обеспечивает отражение в **DFD**-моделях таких физических характеристик системы, как *движение* объектов (потоки данных), *хранение* объектов (хранилища данных), *источники* и *потребители* объектов (внешние сущности).

Построение **DFD**-диаграмм в основном ассоциируется с разработкой программного обеспечения, поскольку нотация **DFD** изначально была разработана для этих целей. В частности, графическое изображение объектов на **DFD**-диаграммах этой главы соответствует принятому Крисом Гейном (Chris Gane) и Тришем Сарсоном (Trish Sarson), авторами **DFD**-метода, известного как метод Гейна — Сарсона. Другой распространенной нотацией **DFD** является так называемый метод Йордана — Де Марко (Yourdon — DeMarco).

3.4.2. Синтаксис и семантика DFD

В отличие от **IDEF0**, рассматривающего систему как множество взаимопересекающихся действий, в названиях объектов **DFD**-диаграмм преобладают имена существительные. Контекстная **DFD**-диаграмма часто состоит из одного функционального блока и нескольких внешних сущностей. Функциональный блок на этой диаграмме обычно имеет имя, совпадающее с именем всей системы.

Добавление на диаграмму внешних ссылок не изменяет фундаментального требования, что модель должна строиться с единственной точки зрения и должна иметь четко определенные цель и границы, что уже обсуждалось ранее.

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

Внешние сущности обеспечивают необходимые входы для системы и/или являются приемниками для ее выходов. Одна внешняя сущность может одновременно предоставлять входы (функционируя как поставщик) и принимать выходы (функционируя как получатель). Внешние сущности изображаются как прямоугольники (рис. 3.28), расположенные как бы "над" диаграммой и бросающим на нее тень, для того, чтобы можно было выделить этот символ среди других обозначений, и обычно размещаются у краев диаграммы. Одна внешняя сущность может быть размещена на одной и той же диаграмме в нескольких эк-

земплярах. Этот прием полезно применять для сокращения количества линий, соединяющих объекты на диаграмме.

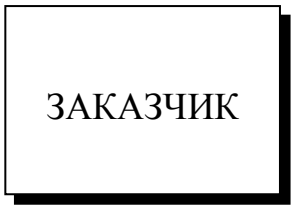


Рис. 3.28. Внешняя сущность

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

Подсистема (или система) на контекстной диаграмме изображается следующим образом (рис. 3.29).

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

Процессы. Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно реализованное логическое устройство и т.д. Процесс на диаграмме потоков данных изображается, как показано на рис. 3.30.

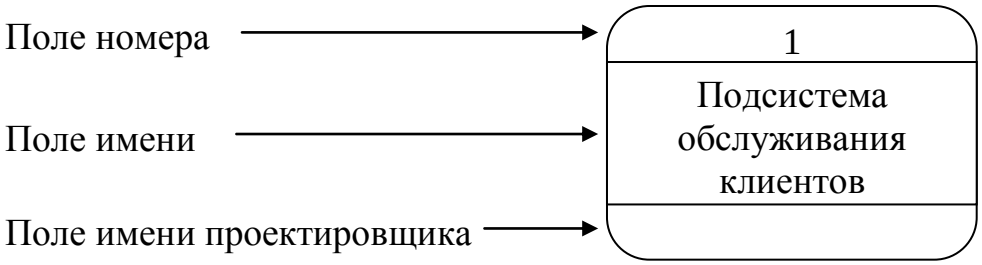


Рис. 3.29. Система или подсистема

Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например:

- "Ввести сведения о клиентах";
- "Выдать информацию о текущих расходах";

- "Проверить кредитоспособность клиента".

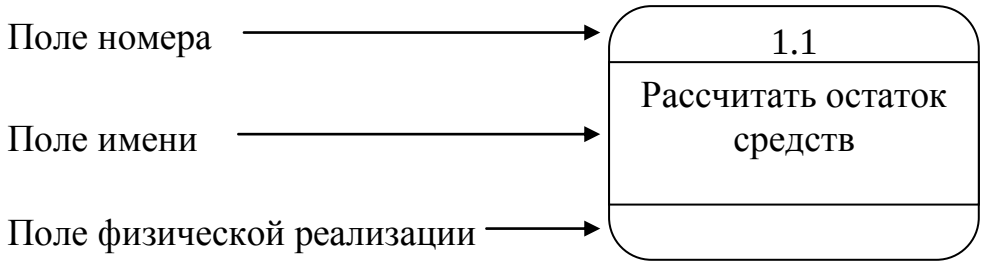


Рис. 3.30. Процесс

Использование таких глаголов, как "обработать", "модернизировать" или "отредактировать" означает, как правило, недостаточно глубокое понимание данного процесса и требует дальнейшего анализа.

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

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

Накопитель данных может быть реализован физически в виде микрофиши, ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных изображается, как показано на рис. 3.31.

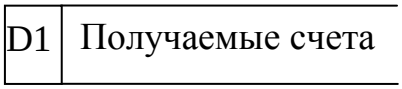


Рис. 3.31. Накопитель

Накопитель данных идентифицируется буквой "D" и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика.

Накопитель данных в общем случае является прообразом будущей базы данных и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Потоки данных. Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рис. 3.32). Каждый поток данных имеет имя, отражающее его содержание.

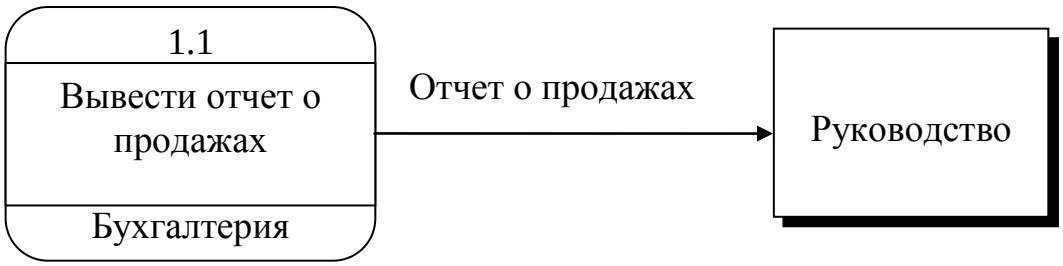


Рис. 3.32. Поток данных

Поскольку все стороны обозначающего функциональный блок **DFD** прямоугольника равнозначны (в отличие от **IDEF0**), стрелки могут начинаться и заканчиваться в любой части блока. В **DFD** также используются двунаправленные стрелки, которые нужны для отображения взаимодействия между блоками (например, диалога типа приказ — результат выполнения).

Стрелки на **DFD**-диаграммах могут быть разбиты (разветвлены) на части, и при этом каждый получившийся сегмент может быть переименован таким образом, чтобы показать декомпозицию данных, переносимых данным потоком. Стрелки могут и соединяться между собой (объединяться) для формирования так называемых комплексных объектов.

3.4.3. Построение иерархии диаграмм потоков данных

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

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы. Признаками сложности (в смысле контекста) могут быть:

- наличие большого количества внешних сущностей (десять и более);
- распределенная природа системы;
- многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных ИС строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диа-

граммы следующего уровня детализируют контекст и структуру подсистем. Иерархия контекстных диаграмм определяет взаимодействие основных функциональных подсистем проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС. Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры ИС на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в разработке которых участвуют разные организации и коллективы разработчиков. После построения контекстных диаграмм полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстных диаграммах, выполняется ее детализация при помощи **DFD**. Каждый процесс на **DFD**, в свою очередь, может быть детализирован при помощи **DFD** или миниспецификации. При детализации должны выполняться следующие правила:

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

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

Миниспецификация является конечной вершиной иерархии **DFD**. Решение о завершении детализации процесса и использовании миниспецификации принимается аналитиком исходя из следующих критериев:

- наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- возможности описания преобразования данных процессом в виде последовательного алгоритма;
- выполнения процессом единственной логической функции преобразования входной информации в выходную;
- возможности описания логики процесса при помощи миниспецификации небольшого объема (не более 20-30 строк).

При построении иерархии **DFD** переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных

конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений. После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные недетализированные объекты следует детализировать, вернувшись на предыдущие шаги разработки. В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть записаны.

3.4.4. Пример банковской задачи

В качестве примера создания модели рассмотрим фрагмент проекта системы, организующей работу банкомата по обслуживанию клиента по его кредитной карте [18]. Этот пример будет строиться поэтапно, на нем будут продемонстрированы базовые техники структурного анализа и проектирования по мере их определения.

На рис. 3.33 приведена контекстная диаграмма системы с единственным процессом ОБСЛУЖИТЬ, идентифицирующая внешние сущности КЛИЕНТ и КОМПЬЮТЕР БАНКА, хранящий информацию о счетах всех клиентов. Опишем потоки данных, которыми обменивается проектируемая система с внешними объектами.

Для банковского обслуживания клиенту необходимо предоставить системе свою КРЕДИТНУЮ КАРТУ для автоматического считывания с нее информации (ПАРОЛЬ, ЛИМИТ ДЕНЕГ, ДЕТАЛИ КЛИЕНТА), а также сообщить свои КЛЮЧЕВЫЕ ДАННЫЕ, а именно ПАРОЛЬ и ЗАПРОС НА ОБСЛУЖИВАНИЕ, т.е. требуемую ему услугу (например, снятие со счета наличных денег). Банковское обслуживание с позиций клиента, в свою очередь, должно обеспечить следующее:

- выдать СООБЩЕНИЕ, приглашающее клиента ввести КЛЮЧЕВЫЕ ДАННЫЕ;
- выдать клиенту ДЕНЬГИ;
- выдать клиенту ВЫПИСКУ по проведенному обслуживанию, включающую ВЫПИСКУ О ДЕНЬГАХ, ВЫПИСКУ ПО БАЛАНСУ и ВЫПИСКУ ПО ОПЕРАЦИИ, проведенной банком.

Рис. 3.33. Контекстная диаграмма банковской задачи

Контекстный процесс и КОМПЬЮТЕР БАНКА должны обмениваться следующей информацией:

- ДАННЫЕ ПО СЧЕТУ клиента в банке;
- ПРОТОКОЛ ОБСЛУЖИВАНИЯ, включающей информацию об ОБРАБОТАННОЙ ДОКУМЕНТАЦИИ, изымаемой ДЕНЕЖНОЙ СУММЕ и ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА.

Контекстный процесс может быть детализирован **DFD** первого уровня как показано на рис. 3.34. Эта диаграмма содержит 4 процесса и хранилище ДАННЫЕ КРЕДИТНОЙ КАРТЫ, которое изображено дважды на диаграмме с целью избежания пересечений линий потоков данных.

Процесс 1.1 (ПОЛУЧИТЬ ПАРОЛЬ) осуществляет прием и проверку пароля клиента и имеет на входе/выходе следующие потоки:

- внешний выходной поток СООБЩЕНИЕ для информирования клиента о своей готовности принять пароль;
- входной поток ВВЕДЕННЫЙ ПАРОЛЬ как элемент внешнего потока КЛЮЧЕВЫЕ ДАННЫЕ;
- входной поток ПАРОЛЬ из хранилища ДАННЫЕ КРЕДИТНОЙ КАРТЫ для проверки вводимого клиентом пароля.

Процесс 1.2 (ПОЛУЧИТЬ ЗАПРОС НА ОБСЛУЖИВАНИЕ) осуществляет прием и проверку запроса клиента на проведение необходимой ему банковской операции и имеет на входе/выходе следующие потоки:

- внешний выходной поток СООБЩЕНИЕ для информирования клиента о своей готовности принять запрос на обслуживание;
- входной поток ЗАПРОС НА ОБСЛУЖИВАНИЕ как элемент внешнего потока КЛЮЧЕВЫЕ ДАННЫЕ;
- входной поток ЛИМИТ ДЕНЕГ из хранилища ДАННЫЕ КРЕДИТНОЙ КАРТЫ для контроля наличия денег на счете клиента.

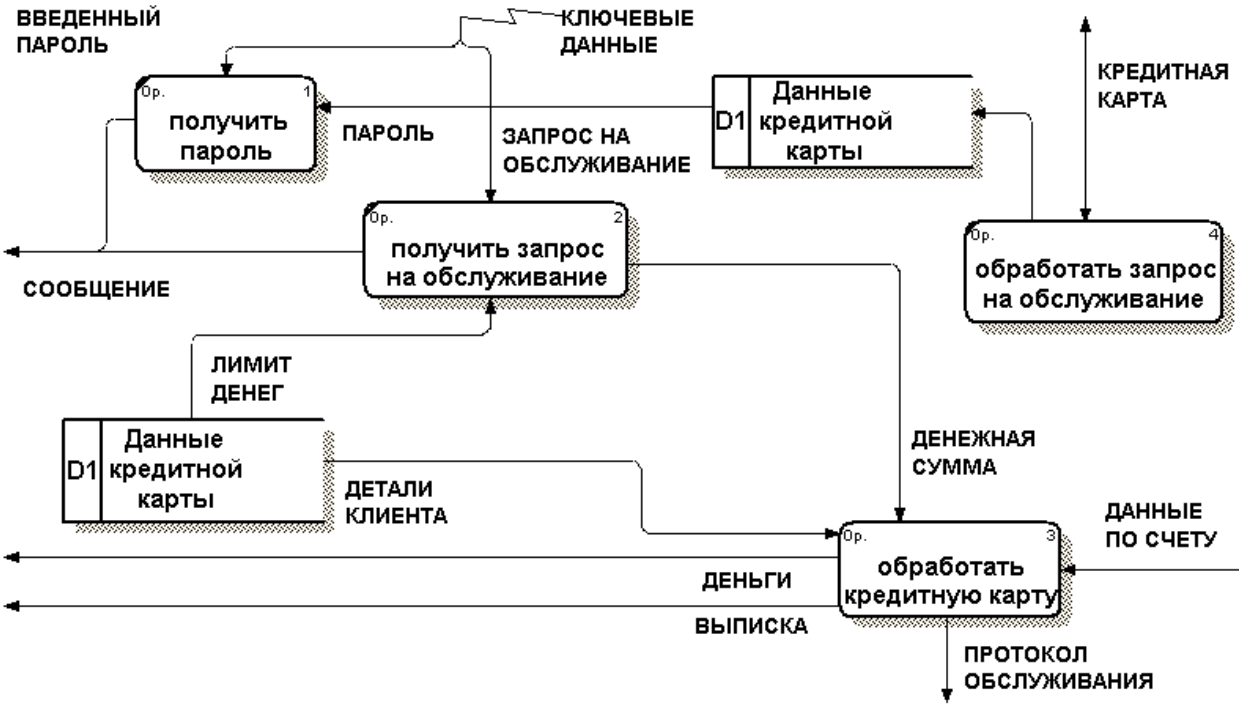


Рис. 3.34. Детализация процесса ОБСЛУЖИТЬ

Процесс 1.3 (ОБРАБОТАТЬ ЗАПРОС НА ОБСЛУЖИВАНИЕ) имеет внешний входной поток ДАННЫЕ ПО СЧЕТУ (из внешней сущности КОМПЬЮТЕР БАНКА), входной поток ДЕТАЛИ КЛИЕНТА (из хранилища), а также внешние выходные потоки ВЫПИСКА, ДЕНЬГИ и ПРОТОКОЛ ОБСЛУЖИВАНИЯ.

Процесс 1.4 (ОБРАБОТАТЬ КРЕДИТНУЮ КАРТУ) осуществляет считывание информации с кредитной карты и имеет на входе внешний поток КРЕДИТНАЯ КАРТА, а на выходе поток ДАННЫЕ КРЕДИТНОЙ КАРТЫ. Отметим, что нет необходимости в идентификации последнего потока, т.к. идентифицировано соответствующее хранилище.

Процессы 1.1, 1.2 и 1.4 являются элементарными, поэтому нет необходимости в их детализации с помощью **DFD** уровня 2 (они будут раскрыты с помощью спецификаций процессов в разделе 3.6.5). Процесс 1.3 может быть детализирован с помощью **DFD** второго уровня как показано на рис. 3.35. Эта диаграмма содержит 4 элементарных процесса, спецификации которых также будут приведены в разделе 3.6.5.

Процесс 1.3.1 (ОБРАБОТАТЬ ДОКУМЕНТАЦИЮ БАНКА) осуществляет обработку внутренней банковской документации по клиенту и имеет входной поток ДЕТАЛИ КЛИЕНТА и выходной поток ОБРАБОТАННАЯ ДОКУМЕНТАЦИЯ (часть внешнего потока ПРОТОКОЛ СДЕЛКИ).

Процесс 1.3.2 (РАСПЕЧАТАТЬ БАЛАНС КЛИЕНТА) выдает справку по истории счета клиента и по балансу клиента. Входные потоки - ДЕТАЛИ КЛИЕНТА и ДАННЫЕ ПО БАЛАНСУ (часть внешнего потока ДАННЫЕ ПО

СЧЕТУ), выходные потоки - ВЫПИСКА ПО БАЛАНСУ (часть внешнего потока ВЫПИСКА) и ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА (часть внешнего потока ПРОТОКОЛ ОБСЛУЖИВАНИЯ).

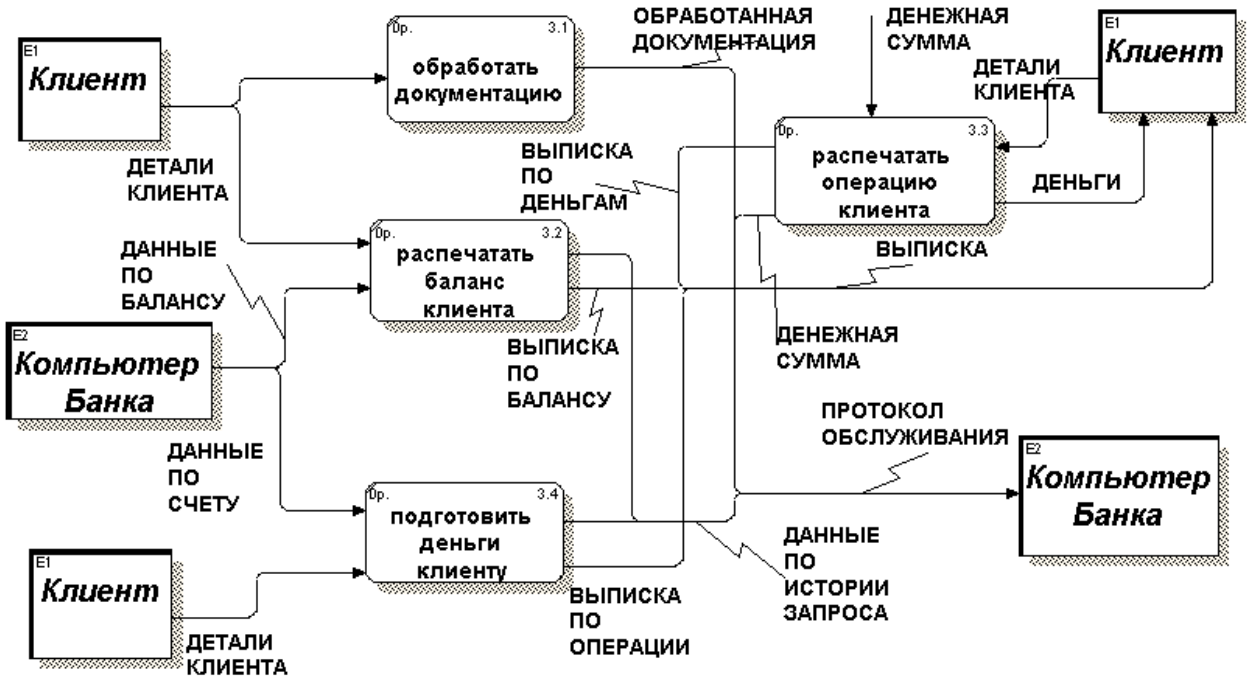


Рис. 3.35. Детализация процесса ОБРАБОТАТЬ ЗАПРОС НА ОБСЛУЖИВАНИЕ

Процесс 1.3.3 (ПРИГОТОВИТЬ ДЕНЬГИ КЛИЕНТУ) обеспечивает выдачу наличных денег и информирование компьютера банка об изъятых из банка деньгах. Он имеет входные потоки ДЕНЕЖНАЯ СУММА и ДЕТАЛИ КЛИЕНТА и выходные потоки ДЕНЬГИ и ДЕНЕЖНАЯ СУММА (часть потока ПРОТОКОЛ ОБСЛУЖИВАНИЯ).

Процесс 1.3.4 (РАСПЕЧАТАТЬ ОПЕРАЦИЮ КЛИЕНТА) выдает справку по истории счета и уведомление по проведенной операции. Входные потоки ДАННЫЕ ПО СЧЕТУ и ДЕТАЛИ КЛИЕНТА, выходные потоки - ВЫПИСКА ПО ОПЕРАЦИИ (часть потока ВЫПИСКА) и ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА (часть потока ПРОТОКОЛ ОБСЛУЖИВАНИЯ).

3.5. Словарь данных

Диаграммы потоков данных обеспечивают удобное описание функционирования компонент системы, но не снабжают аналитика средствами описания деталей этих компонент, а именно, какая информация преобразуется процессами и как она преобразуется. Для решения первой из перечисленных задач предназначены текстовые средства моделирования, служащие для описания структуры преобразуемой информации и получившие название словарей данных [18].

Словарь данных представляет собой определенным образом организованный список всех элементов данных системы с их точными определениями, что дает возможность различным категориям пользователей (от системного аналитика до программиста) иметь общее понимание всех входных и выходных потоков и компонент хранилищ. Определения элементов данных в словаре осуществляются следующими видами описаний:

- описанием значений потоков и хранилищ, изображенных на **DFD**;
- описанием композиции агрегатов данных, движущихся вдоль потоков, т.е. комплексных данных, которые могут расчленяться на элементарные символы (например, АДРЕС ПОКУПАТЕЛЯ содержит ПОЧТОВЫЙ ИНДЕКС, ГОРОД, УЛИЦУ и т.д.);
- описанием композиции групповых данных в хранилище;
- специфицированием значений и областей действия элементарных фрагментов информации в потоках данных и хранилищах;
- описанием деталей отношений между хранилищами.

Содержимое словаря данных. Для каждого потока данных в словаре необходимо хранить имя потока, его тип и атрибуты. Информация по каждому потоку состоит из ряда словарных статей, каждая из которых начинается с ключевого слова - заголовка соответствующей статьи, которому предшествует символ "**@**".

По типу потока в словаре содержится информация, идентифицирующая:

- простые (элементарные) или групповые (комплексные) потоки;
- внутренние (существующие только внутри системы) или внешние (связывающие систему с другими системами) потоки;
- потоки данных или потоки управления;
- непрерывные (принимающие любые значения в пределах определенного диапазона) или дискретные (принимающие определенные значения) потоки.

Атрибуты потока данных включают:

- имена-синонимы потока данных в соответствии с узлами изменения имени;
- БНФ-определение в случае группового потока;
- единицы измерения потока;
- диапазон значений для непрерывного потока, типичное его значение и информацию по обработке экстремальных значений;
- список значений и их смысл для дискретного потока;
- список номеров диаграмм различных типов, в которых поток встречается;
- список потоков, в которые данный поток входит (как элемент БНФ-определения);
- комментарий, включающий дополнительную информацию (например о цели введения данного потока).

БНФ-нотация. **БНФ-нотация** позволяет формально описать расщепление/ объединение потоков. Поток может расщепляться на собственные отдельные ветви, на компоненты потока-предка или на то и другое одновременно. При

расщеплении/объединении потока существенно, чтобы каждый компонент потока-предка являлся именованным. Если поток расщепляется на подпотоки, необходимо, чтобы все подпотоки являлись компонентами потока-предка. И наоборот, при объединении потоков каждый компонент потока-предка должен по крайней мере однажды встречаться среди подпотоков. Отметим, что при объединении подпотоков нет необходимости осуществлять исключение общих компонент, а при расщеплении подпотоки могут иметь такие общие (одинаковые) компоненты.

Важно понимать, что точные определения потоков содержатся в словаре данных, а не на диаграммах. Например, на диаграмме может иметься групповой узел с входным потоком *X* и выходными подпотоками *Y* и *Z*. Однако это вовсе не означает, что соответствующее определение в словаре данных обязательно должно быть *X=Y+Z*. Это определение может быть следующим:

$$X=A+B+C;\ Y=A+B;\ Z=B+C$$

Такие определения хранятся в словаре данных в так называемой **БНФ-статье**. БНФ-статья используется для описания компонент данных в потоках данных и в хранилищах. Ее синтаксис имеет вид:

@БНФ = <простой оператор> / <БНФ-выражение>,

где <простой оператор> есть текстовое описание, заключенное в "/", а <БНФ-выражение> есть выражение в форме Бэкуса-Наура, допускающее следующие операции отношений:

- **=** - означает "композиция из",
- **+** - означает "И",
- **[!]** - означает "ИЛИ",
- **( )** - означает, что компонент в скобках не обязателен,
- **{ }** - означает итерацию компонента в скобках,
- **" "** - означает литерал.

Итерационные скобки могут иметь нижний и верхний предел, например:

3{болт}7 - от 3 до 7 итераций
1{болт} - 1 и более итераций
{шайба}3 - не более 3 итераций

БНФ-выражение может содержать произвольные комбинации операций:

@БНФ = [винт ! болт + 2{гайка}2 + (прокладка) ! клей]

Ниже приведен пример описания потока данных с помощью БНФ:

@ИМЯ = ВОСЬМЕРИЧНАЯ ЦИФРА
@ТИП = дискретный поток
@БНФ = ["0"!"1"!"2"!"3"!"4"!"5"!"6"!"7"]

Посмотрим, как некоторые потоки, присутствующие на вышеприведенных диаграммах потоков данных, представляются в словаре данных.

@ИМЯ = ВВЕДЕННАЯ КРЕДИТНАЯ КАРТА
@ТИП = управляющий поток
@БНФ = /указывает, что кредитная карта введена/
@ИМЯ = ДАННЫЕ КРЕДИТНОЙ КАРТЫ
@ТИП = дискретный поток
@БНФ = ПАРОЛЬ + ДЕТАЛИ КЛИЕНТА + ЛИМИТ ДЕНЕГ
@ИМЯ = ДАННЫЕ ПО БАЛАНСУ
@ТИП = дискретный поток
@БНФ = /текущий баланс счета клиента/
@ЕДИНИЦА ИЗМЕРЕНИЯ = доллар
@ДИАПАЗОН = +/- 100000
@ТОЧНОСТЬ = .01
@ИМЯ = ДЕНЬГИ
@ТИП = дискретный поток
@БНФ = /деньги, выдаваемые клиенту/
@ЕДИНИЦА ИЗМЕРЕНИЯ = доллар
@НОРМА = 5..1000
@КОММЕНТАРИЙ Сумма выдаваемых денег должна делиться на 5
@ИМЯ = ПРОТОКОЛ ОБСЛУЖИВАНИЯ
@ТИП = дискретный поток
@БНФ = (ОБРАБОТАННАЯ ДОКУМЕНТАЦИЯ)+
+ (ДЕНЕЖНАЯ СУММА) +
+ (ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА)

3.6. Методы задания спецификаций процессов

Спецификация процесса (СП) используется для описания функционирования процесса в случае отсутствия необходимости детализировать его с помощью **DFD** (т.е. если он достаточно невелик, и его описание может занимать до одной страницы текста). Фактически СП представляют собой алгоритмы описания задач, выполняемых процессами: множество всех СП является полной спецификацией системы. СП содержат номер и/или имя процесса, списки входных и выходных данных и тело (описание) процесса, являющееся спецификацией алгоритма или операции, трансформирующей входные потоки данных в выходные. Известно большое число разнообразных методов, позволяющих задать тело процесса, соответствующий язык может варьироваться от **структурированного естественного языка** или псевдокода до **визуальных языков проектирования** (типа **FLOW-форм** и **диаграмм Насси-Шнейдермана**) и формальных компьютерных языков [18].

Независимо от используемой нотации спецификация процесса должна начинаться с ключевого слова (например, **@СПЕЦПРОЦ**). Требуемые входные и выходные данные должны быть специфицированы следующим образом:
@ВХОД = <имя символа данных>
@ВЫХОД = <имя символа данных>

@ВХОДВЫХОД = <имя символа данных>,
где <имя символа данных> - соответствующее имя из словаря данных.

Эти ключевые слова должны использоваться перед определением СП, например:

@ВХОД = СЛОВА ПАМЯТИ
@ВЫХОД = ХРАНИМЫЕ ЗНАЧЕНИЯ
@СПЕЦПРОЦ
Для всех СЛОВ ПАМЯТИ выполнить:
Распечатать ХРАНИМЫЕ ЗНАЧЕНИЯ
@

Ситуация, когда символ данных является одновременно входным и выходным, может быть описана двумя способами: либо символ описывается два раза с помощью **@ВХОД** и **@ВЫХОД**, либо один раз с помощью **@ВХОДВЫХОД**.

Иногда в СП задаются **пред-** и **постусловия** выполнения данного процесса. В предусловии записываются объекты, значения которых должны быть истинны перед началом выполнения процесса, что обеспечивает определенные гарантии безопасности для пользователя. Аналогично, в случае наличия постусловия гарантируется, что значения всех входящих в него объектов будут истинны при завершении процесса.

Спецификации должны удовлетворять следующим требованиям:

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

Ниже рассматриваются некоторые наиболее часто используемые методы задания спецификаций процессов.

3.6.1. Структурированный естественный язык

Структурированный естественный язык применяется для читабельного, строгого описания спецификаций процессов [18]. Он является разумной комбинацией строгости языка программирования и читабельности естественного языка и состоит из подмножества слов, организованных в определенные логические структуры, арифметических выражений и диаграмм.

В состав языка входят следующие основные символы:

- глаголы, ориентированные на действие и применяемые к объектам;

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

Управляющие структуры языка имеют один вход и один выход. К ним относятся:

1) последовательная конструкция:
   ВЫПОЛНИТЬ функция1
   ВЫПОЛНИТЬ функция2
   ВЫПОЛНИТЬ функция3
2) конструкция выбора:
   ЕСЛИ <условие> **ТО**
   ВЫПОЛНИТЬ функция1
   ИНАЧЕ
   ВЫПОЛНИТЬ функция2
   КОНЕЦЕСЛИ
3) итерация:
   ДЛЯ <условие>
   ВЫПОЛНИТЬ функция
   КОНЕЦДЛЯ
   или
   ПОКА <условие>
   ВЫПОЛНИТЬ функция
   КОНЕЦПОКА

При использовании структурированного естественного языка приняты следующие соглашения:

1. Логика процесса выражается в виде комбинации последовательных конструкций, конструкций выбора и итераций.
2. Ключевые слова **ЕСЛИ, ВЫПОЛНИТЬ, ИНАЧЕ** и т.д. должны быть написаны заглавными буквами.
3. Слова или фразы, определенные в словаре данных, должны быть написаны заглавными буквами.
4. Глаголы должны быть активными, недвусмысленными и ориентированными на целевое действие (заполнить, вычислить, извлечь, а не модернизировать, обработать).
5. Логика процесса должна быть выражена четко и недвусмысленно.

Ниже приведен пример спецификации процесса 1 (ПОЛУЧИТЬ ПАРОЛЬ) для диаграммы, изображенной на рис. 3.34.

@ВХОД = ВВЕДЕННЫЙ ПАРОЛЬ
@ВХОД = ПАРОЛЬ

@ВЫХОД = СООБЩЕНИЕ
@ВЫХОД = КОРРЕКТНЫЙ ПАРОЛЬ
@СПЕЦПРОЦ 1.1 ПОЛУЧИТЬ ПАРОЛЬ
ВЫПОЛНИТЬ выдать СООБЩЕНИЕ клиенту,
запрашивающее ввод пароля
принять ВВЕДЕННЫЙ ПАРОЛЬ
ДОТЕХПОРПОКА ВВЕДЕННЫЙ ПАРОЛЬ = ПАРОЛЬ
или были сделаны три попытки ввода
КОНЕЦВЫПОЛНИТЬ
ВЫПОЛНИТЬ установить флаг КОРРЕКТНЫЙ
ПАРОЛЬ в случае равенства
@ КОНЕЦ СПЕЦИФИКАЦИИ ПРОЦЕССА 1.1

3.6.2. Таблицы и деревья решений

Структурированный естественный язык неприемлем для некоторых типов преобразований. Например, если действие зависит от нескольких переменных, которые в совокупности могут продуцировать большое число комбинаций, то его описание будет слишком запутанным и с большим числом уровней вложенности. Для описания подобных действий традиционно используются таблицы и деревья решений [18].

Проектирование спецификаций процессов с помощью **таблиц решений** (ТР) заключается в задании матрицы, отображающей множество входных **условий** в множество **действий.**

ТР состоит из двух частей. Верхняя часть таблицы используется для определения условий. Обычно условие является **ЕСЛИ**-частью оператора **ЕСЛИ-ТО** и требует ответа "да-нет". Однако иногда в условии может присутствовать и ограниченное множество значений, например, ЯВЛЯЕТСЯ ЛИ ДЛИНА СТРОКИ БОЛЬШЕЙ, МЕНЬШЕЙ ИЛИ РАВНОЙ ГРАНИЧНОМУ ЗНАЧЕНИЮ?

Нижняя часть ТР используется для определения действий, т.е. **ТО**-части оператора **ЕСЛИ-ТО**. Так, в конструкции - **ЕСЛИ** ИДЕТ ДОЖДЬ, **ТО** РАСКРЫТЬ ЗОНТ - ИДЕТ ДОЖДЬ является условием, а РАСКРЫТЬ ЗОНТ - действием.

Левая часть ТР содержит собственно описание условий и действий, а в правой части перечисляются все возможные комбинации условий и, соответственно, указывается, какие конкретно действия и в какой последовательности выполняются, когда определенная комбинация условий имеет место.

Поясним вышесказанное на примере спецификации процесса выбора символов из входного потока. При выборе символов необходимо руководствоваться следующими правилами:

1) если очередной символ является управляющим, то подать звуковой сигнал и вернуть код ошибки;

2) если буфер формируемой строки заполнен, то подать звуковой сигнал и вернуть код ошибки;
3) если очередной символ не находится в заданном диапазоне, то подать звуковой сигнал и вернуть код ошибки;
4) иначе поместить символ в буфер, увеличить значение счетчика выбранных символов и вернуть новое значение счетчика.

Таблица решений для данного примера выглядит следующим образом (табл. 3.6):

Таблица 3.6

	УСЛОВИЯ	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
C1	isctrl(c)	Д	Д	Д	Д	Н	Н	Н	Н
C2	I > max_lenght	Д	Д	Н	Н	Д	Д	Н	Н
C3	out_of_range(c)	Д	Н	Д	Н	Д	Н	Д	Н
	ДЕЙСТВИЯ								
D1	beep()	1	1	1	1	1	1	1	
D2	return(ERROR)	2	2	2	2	2	2	2	
D3	return(++i)								2
D4	putchar(c)								1

Заметим, что если выполняется условие C1, то нет необходимости в проверке условий C2 и C3. Поэтому комбинации условий 1, 2, 3, 4 могут быть заменены обобщающей комбинацией (Д,-,-), где "-" означает любую из возможных альтернатив (в данном случае, Д или Н). Аналогично, комбинации условий 5 и 6 могут быть заменены обобщающей комбинацией (Н,Д,-). Редуцированная таким образом таблица решений будет иметь следующий вид (табл. 3.7).

Таблица 3.7

	УСЛОВИЯ	**1**	**2**	**3**	**4**
C1	isctrl(c)	Д	Н	Н	Н
C2	I > max_lenght	-	Д	Н	Н
C3	out_of_range(c)	-	-	Д	Н
	ДЕЙСТВИЯ				
D1	beep()	1	1	1	
D2	return(ERROR)	2	2	2	
D3	return(++i)				2
D4	putchar(c)				1

Отметим, что на основе таблицы решений легко осуществляется автоматическая кодогенерация. Для вышеприведенного примера соответствующий код может выглядеть следующим образом:

```
IF (isctrl(c)) { beep(); return(ERROR) }
   ELSE {
   IF (i>max_length) { beep(); return(ERROR) }
   ELSE {
   IF (out_of_range(c)) { beep(); return(ERROR) }
   ELSE { putchar(c); return(++i) }
   }    }
```

Построение ТР рекомендуется осуществлять по следующим шагам:

1. Идентифицировать все условия (или переменные) в спецификации. Идентифицировать все значения, которые каждая переменная может иметь.
2. Вычислить число комбинаций условий. Если все условия являются бинарными, то существует 2^N комбинаций N переменных.
3. Идентифицировать каждое из возможных действий, которые могут вызываться в спецификации.
4. Построить пустую таблицу, включающую все возможные условия и действия, а также номера комбинаций условий.
5. Выписать и занести в таблицу все возможные комбинации условий.
6. Редуцировать комбинации условий.
7. Проверить каждую комбинацию условий и идентифицировать соответствующие выполняемые действия.
8. Выделить комбинации условий, для которых спецификация не указывает список выполняемых действий.
9. Обсудить построенную таблицу.

Вариантом таблицы решений является дерево решений (ДР), позволяющее взглянуть на процесс условного выбора с позиции схемы. Дерево решений для вышерассмотренного примера приведено на рис. 3.36.

Обычно ДР используется при малом числе действий и когда не все комбинации условий возможны, а ТР - при большом числе действий и когда возможно большинство комбинаций условий.

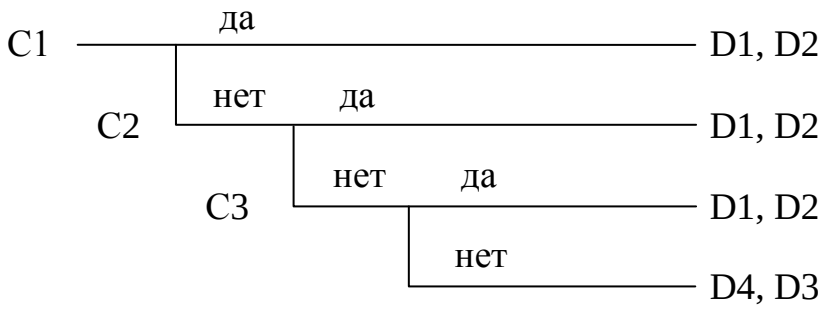


Рис. 3.36. Дерево решений

3.6.3. Визуальные языки проектирования спецификаций

Визуальные языки проектирования являются относительно новой, оригинальной методикой разработки спецификаций процесса [18]. Они базируются на основных идеях структурного программирования и позволяют определять потоки управления с помощью специальных иерархически организованных схем.

Одним из наиболее известных подходов к визуальному проектированию спецификаций является подход с использованием **FLOW-форм**. Каждый символ FLOW-формы имеет вид прямоугольника и может быть вписан в любой внутренний прямоугольник любого другого символа. Символы помечаются с помощью предложений на естественном языке или с использованием математической нотации.

Символы FLOW-форм приведены на рис. 3.37. Каждый символ является блоком обработки. Каждый прямоугольник внутри любого символа также представляет собой блок обработки.

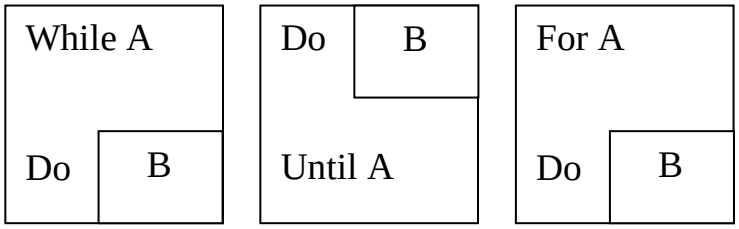


Рис. 3.37. Символы FLOW-форм

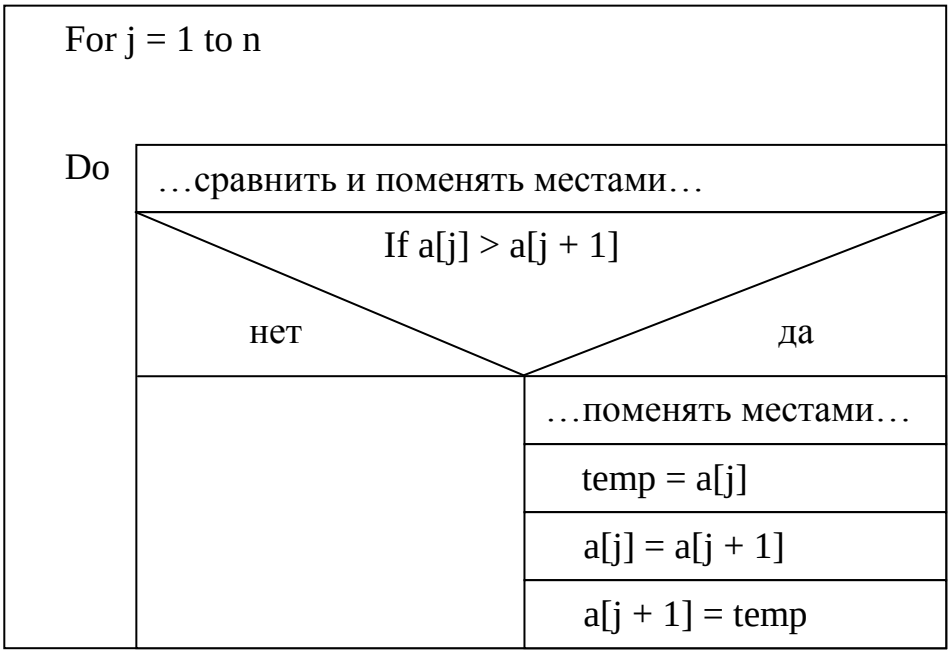


Рис. 3.38. Пример FLOW-формы

На рис. 3.38 приведен пример использования данного подхода при проектировании спецификации процесса, обеспечивающего упорядочивание определенным образом элементов массива и являющегося фрагментом алгоритма сортировки методом "поплавка".

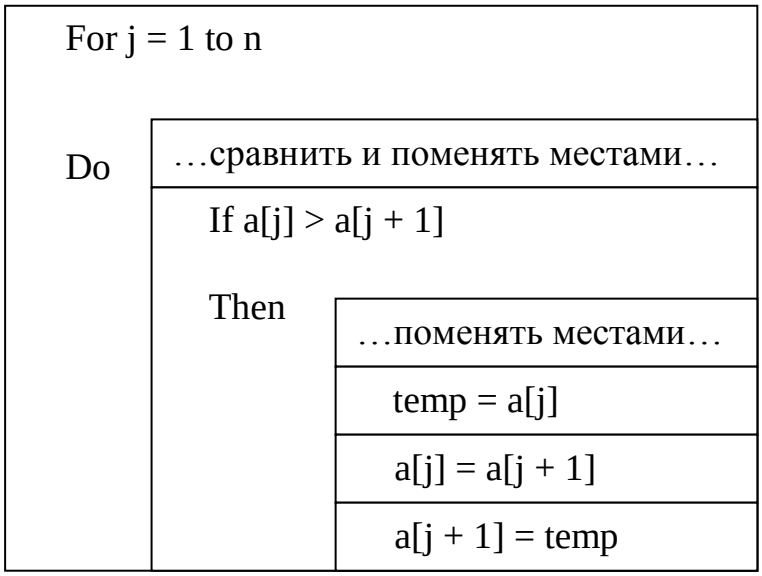


Рис. 3.39. Диаграмма Насси-Шнейдермана

Дальнейшее развитие FLOW-формы получили в диаграммах Насси-Шнейдермана. На этих диаграммах символы последовательной обработки и цикла изображаются также, как и соответствующие символы FLOW-форм. В символах условного выбора и case-выбора собственно условие располагается в верхнем треугольнике, выбираемые варианты - на нижних сторонах треугольника, а блоки обработки - под выбираемыми вариантами. Диаграмма Насси-Шнейдермана для вышеприведенного примера изображена на рис. 3.39.

3.6.4. Сравнение методов

Наиболее трудным методом задания СП являются языки программирования (C, COBOL, FORTRAN и др.). Сложность заключается в том, что языки программирования концентрируют внимание на деталях реализации, а потоки данных в **DFD** представляются абстрактно (их фактическая композиция определяется в словаре данных). Поэтому сложность - не в написании СП, а в их синхронизации и согласовании с **DFD**, поскольку при редактировании **DFD**, вообще говоря, должны корректироваться и спецификации процессов.

Перечислим некоторые положительные и отрицательные стороны рассмотренных методов задания СП.

Структурированный естественный язык применяется в случаях, когда детали СП известны не полностью. Он обеспечивает быстрое проектирование СП, прост в использовании, легко понимаем проектировщиками и программистами, а также конечным пользователем. К его недостаткам относятся отсутствие про-

цедурных возможностей и неспособность к автоматической кодогенерации из-за наличия неоднозначностей.

Таблицы и деревья решений позволяют управлять сложными комбинациями условий и действий, обеспечивают визуальное (табличное и графическое, соответственно) представление СП и легко понимаемы конечным пользователем. Кроме этого, таблицы решений позволяют легко идентифицировать несущественности и бреши в СП. Главным недостатком методов является отсутствие процедурных возможностей.

Визуальные языки проектирования поддерживаются автоматической кодогенерацией, позволяют осуществлять декомпозицию СП. Их недостаток - трудность модификации СП при изменении деталей.

3.6.5. Спецификации процессов для примера банковской задачи

Приведем спецификации процессов для рассмотренного в разделе 3.4.4 примера банковской задачи с использованием структурированного естественного языка.

1) Спецификация процесса 1.1 приведена ранее в разделе 3.6.1.

2) Спецификация процесса 1.2.

@ВХОД = ЛИМИТ ДЕНЕГ
@ВХОД = ЗАПРОС НА ОБСЛУЖИВАНИЕ
@ВЫХОД = ДЕНЕЖНАЯ СУММА
@ВЫХОД = СООБЩЕНИЕ
@ВЫХОД = ТРЕБУЕМОЕ ОБСЛУЖИВАНИЕ
@СПЕЦПРОЦ 1.2 ПОЛУЧИТЬ ЗАПРОС НА ОБСЛУЖИВАНИЕ
ВЫПОЛНИТЬ
выдать СООБЩЕНИЕ клиенту по вводу запроса на обслуживание
принять ЗАПРОС НА ОБСЛУЖИВАНИЕ
обновить данные ТРЕБУЕМОЕ ОБСЛУЖИВАНИЕ (а именно, ЗАПРОС ДОКУМЕНТАЦИИ, ЗАПРОС ДЕНЕГ, ЗАПРОС БАЛАНСА, ЗАПРОС НА ОПЕРАЦИЮ)
ЕСЛИ
был сделан ЗАПРОС ДЕНЕГ
ТО
ВЫПОЛНИТЬ
запросить ДЕНЕЖНУЮ СУММУ
выдать требуемую ДЕНЕЖНУЮ СУММУ с учетом того, что она не должна превышать ЛИМИТ ДЕНЕГ
КОНЕЦЕСЛИ
ДОТЕХПОРПОКА запрашивается продолжение обслуживания или не все обслуживание было выполнено
КОНЕЦВЫПОЛНИТЬ
@ КОНЕЦ СПЕЦИФИКАЦИИ ПРОЦЕССА 1.2

3) Спецификация процесса 1.3.

3.1) Спецификация процесса 1.3.1.
@ВХОД = ЗАПРОС ДОКУМЕНТАЦИИ
@ВХОД = ДЕТАЛИ КЛИЕНТА
@ВЫХОД = ОБРАБОТАННАЯ ДОКУМЕНТАЦИЯ
@СПЕЦПРОЦ 1.3.1 ОБРАБОТАТЬ ДОКУМЕНТАЦИЮ БАНКА
по получении ЗАПРОСА ДОКУМЕНТАЦИИ выдать ОБРАБОТАННУЮ ДОКУМЕНТАЦИЮ, содержащую ДЕТАЛИ КЛИЕНТА, КОМПЬЮТЕРУ БАНКА
@КОНЕЦ СПЕЦИФИКАЦИИ ПРОЦЕССА 1.3.1

3.2) Спецификация процесса 1.3.2.
@ВХОД = ДАННЫЕ ПО БАЛАНСУ
@ВХОД = ЗАПРОС БАЛАНСА
@ВХОД = ДЕТАЛИ КЛИЕНТА
@ВЫХОД = ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА
@ВЫХОД = ВЫПИСКА ПО БАЛАНСУ
@СПЕЦПРОЦ 1.3.2 РАСПЕЧАТАТЬ БАЛАНС КЛИЕНТА
по получении ЗАПРОСА БАЛАНСА выдать ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА
затем выдать ВЫПИСКУ ПО БАЛАНСУ, содержащую ДАННЫЕ ПО БАЛАНСУ
@ КОНЕЦ СПЕЦИФИКАЦИИ ПРОЦЕССА 1.3.2

3.3) Спецификация процесса 1.3.3.
@ВХОД = ДЕНЕЖНАЯ СУММА
@ВХОД = ЗАПРОС ДЕНЕГ
@ВХОД = ДЕТАЛИ КЛИЕНТА
@ВЫХОД = ДЕНЬГИ
@ВЫХОД = ВЫПИСКА О ДЕНЬГАХ
@ВЫХОД = ДЕНЕЖНАЯ СУММА
@СПЕЦПРОЦ 1.3.3 ПРИГОТОВИТЬ ДЕНЬГИ ДЛЯ КЛИЕНТА
по получении ЗАПРОСА ДЕНЕГ выдать ДЕНЬГИ по значению ДЕНЕЖНОЙ СУММЫ
выдать ВЫПИСКУ О ДЕНЬГАХ, содержащую ДЕНЕЖНУЮ СУММУ
передать КОМПЬЮТЕРУ БАНКА информацию о ДЕНЕЖНОЙ СУММЕ
@ КОНЕЦ СПЕЦИФИКАЦИИ ПРОЦЕССА 1.3.3

3.4) Спецификация процесса 1.3.4.
@ВХОД = ДАННЫЕ ПО СЧЕТУ
@ВХОД = ЗАПРОС НА ОПЕРАЦИЮ
@ВХОД = ДЕТАЛИ КЛИЕНТА
@ВЫХОД = ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА
@ВЫХОД = ВЫПИСКА ПО ОПЕРАЦИИ
@СПЕЦПРОЦ 1.3.4 РАСПЕЧАТАТЬ ОПЕРАЦИЮ КЛИЕНТА
по получении ЗАПРОСА НА ОПЕРАЦИЮ выдать ДАННЫЕ ПО ИСТОРИИ ЗАПРОСА для специфицирования ДЕТАЛЕЙ КЛИЕНТА, чтобы получить те-

кущие ДАННЫЕ ПО СЧЕТУ
выдать ВЫПИСКУ ПО ОПЕРАЦИИ, содержащую ДАННЫЕ ПО СЧЕТУ
@ **КОНЕЦ СПЕЦИФИКАЦИИ** ПРОЦЕССА 1.3.4

4) Спецификация процесса 1.4.
@ВХОД = УДАЛЕННАЯ КРЕДИТНАЯ КАРТА
@ВХОДВЫХОД = КРЕДИТНАЯ КАРТА
@ВЫХОД = ДАННЫЕ КРЕДИТНОЙ КАРТЫ
@ВЫХОД = ВВЕДЕННАЯ КРЕДИТНАЯ КАРТА
@СПЕЦПРОЦ 1.4 ОБРАБОТАТЬ КРЕДИТНУЮ КАРТУ
ВЫПОЛНИТЬ считать КРЕДИТНУЮ КАРТУ
записать в хранилище ДАННЫЕ КРЕДИТНОЙ КАРТЫ
выдать управляющий поток ВВЕДЕННАЯ КРЕДИТНАЯ КАРТА
по получении управляющего потока УДАЛЕННАЯ КРЕДИТНАЯ КАРТА удалить КРЕДИТНУЮ КАРТУ
@ **КОНЕЦ СПЕЦИФИКАЦИИ** ПРОЦЕССА 1.4

3.7. Моделирование данных

Прежде чем приступать к созданию системы автоматизированной обработки информации, разработчик должен сформировать понятия о предметах, фактах и событиях, которыми будет оперировать данная система. Для того, чтобы привести эти понятия к той или иной модели данных, необходимо заменить их информационными представлениями. Одним из наиболее удобных инструментов унифицированного представления данных, независимого от реализующего его программного обеспечения, является *модель "сущность-связь"* (*entity - relationship model, ER - model*), предложенная П. Ченом [12].

Модель "сущность-связь" основывается на некой важной семантической информации о реальном мире и предназначена для *логического* представления данных. Она определяет значения данных в контексте их взаимосвязи с другими данными. Важным для нас является тот факт, что из **модели "сущность-связь"** могут быть порождены все существующие **модели данных** (иерархическая, сетевая, реляционная, объектная), поэтому она является наиболее общей.

3.7.1. Элементы ER-модели

Любой фрагмент предметной области может быть представлен как *множество сущностей*, между которыми существует некоторое *множество связей*. Дадим определения [13]:

Сущность (entity) - это объект, который может быть идентифицирован неким способом, отличающим его от других объектов. Примеры: *конкретный человек, предприятие, событие* и т.д.

Набор сущностей (entity set) - множество сущностей одного типа (обладающих одинаковыми свойствами). Примеры: *все люди, предприятия, праздни-*

ки и т.д. Наборы сущностей не обязательно должны быть непересекающимися. Например, сущность, принадлежащая к набору МУЖЧИНЫ, также принадлежит набору ЛЮДИ.

Сущность фактически представляет из себя множество *атрибутов*, которые описывают свойства всех членов данного набора сущностей.

Пример: рассмотрим множество работников некого предприятия. Каждого из них можно описать с помощью характеристик *табельный номер, имя, возраст*. Поэтому, сущность СОТРУДНИК имеет атрибуты ТАБЕЛЬНЫЙ НОМЕР, ИМЯ, ВОЗРАСТ.

Множество значений (область определения) атрибута называется *доменом*. Например, для атрибута ВОЗРАСТ домен (назовем его ЧИСЛО ЛЕТ) задается интервалом целых чисел больших нуля, поскольку людей с отрицательным возрастом не бывает.

В статье П. Чена [12] атрибут определяется как *функция, отображающая набор сущностей в набор значений или в декартово произведение наборов значений*. Так атрибут ВОЗРАСТ производит отображение в набор значений (домен) ЧИСЛО ЛЕТ. Атрибут ИМЯ производит отображение в декартово произведение наборов значений ИМЯ, ФАМИЛИЯ и ОТЧЕСТВО.

Отсюда определяется *ключ сущности* - группа атрибутов, такая, что отображение набора сущностей в соответствующую группу наборов значений является взаимнооднозначным отображением. Другими словами: ключ сущности - это один или более атрибутов уникально определяющих данную сущность. В нашем примере ключем сущности СОТРУДНИК является атрибут ТАБЕЛЬНЫЙ НОМЕР (конечно, только в том случае, если все табельные номера на предприятии уникальны).

Связь (relationship) - это ассоциация, установленная между несколькими сущностями. Примеры:

- поскольку каждый сотрудник работает в каком-либо отделе, между сущностями СОТРУДНИК и ОТДЕЛ существует связь "работает в" или ОТДЕЛ-РАБОТНИК;
- так как один из работников отдела является его руководителем, то между сущностями СОТРУДНИК и ОТДЕЛ имеется связь "руководит" или ОТДЕЛ-РУКОВОДИТЕЛЬ;
- могут существовать и связи между сущностями одного типа, например связь РОДИТЕЛЬ-ПОТОМОК между двумя сущностями ЧЕЛОВЕК.

(Следует отметить, что в методике проектирования данных есть своеобразное правило хорошего тона, согласно которому сущности обозначаются с помощью имен существительных, а связи - глагольными формами. Данное правило, однако, не является обязательным).

Связь также может иметь атрибуты. Например, для связи ОТДЕЛ-РАБОТНИК можно задать атрибут СТАЖ РАБОТЫ В ОТДЕЛЕ.

Роль сущности в связи - функция, которую выполняет сущность в данной связи. Например, в связи РОДИТЕЛЬ-ПОТОМОК сущности ЧЕЛОВЕК могут

иметь роли "родитель" и "потомок". Указание ролей в модели "сущность-связь" не является обязательным и служит для уточнения семантики связи.

Набор связей (relationship set) - это отношение между *n* (причем *n* не меньше 2) сущностями, каждая из которых относится к некоторому набору сущностей.

В случае *n=2*, т.е. когда связь объединяет две сущности, она называется *бинарной*. Доказано, что *n*-арный набор связей (*n>2*) всегда можно заменить множеством бинарных, однако первые лучше отображают семантику предметной области.

То число сущностей, которое может быть ассоциировано через набор связей с другой сущностью, называют *степенью связи*. Рассмотрение степеней особенно полезно для бинарных связей. Могут существовать следующие степени бинарных связей:

- **один-к-одному** (*1:1*). Это означает, что в такой связи сущности с одной ролью всегда соответствует не более одной сущности с другой ролью.

Другой важной характеристикой связи помимо ее степени является *класс принадлежности* входящих в нее сущностей или *кардинальность* связи. Так как в каждом отделе обязательно должен быть руководитель, то каждой сущности "ОТДЕЛ" непременно должна соответствовать сущность "СОТРУДНИК". Однако не каждый сотрудник является руководителем отдела, следовательно, в данной связи не каждая сущность "СОТРУДНИК" имеет ассоциированную с ней сущность "ОТДЕЛ".

Таким образом, говорят, что сущность "СОТРУДНИК" имеет *обязательный класс принадлежности* (этот факт обозначается также указанием интервала числа возможных вхождений сущности в связь, в данном случае это 1,1), а сущность "ОТДЕЛ" имеет *необязательный класс принадлежности* (0,1). Теперь данную связь мы можем описать как *0,1:1,1*.

- **один-ко-многим** (*1:n*). В данном случае сущности с одной ролью может соответствовать любое число сущностей с другой ролью. Такова связь ОТДЕЛ-СОТРУДНИК. В каждом отделе может работать произвольное число сотрудников, но сотрудник может работать только в одном отделе.

Здесь также необходимо учитывать класс принадлежности сущностей. Каждый сотрудник должен работать в каком-либо отделе, но не каждый отдел (например, вновь сформированный) должен включать хотя бы одного сотрудника. Поэтому сущность "ОТДЕЛ" имеет обязательный, а сущность "СОТРУДНИК" необязательный классы принадлежности.

- **много-к-одному** (*n:1*). Эта связь аналогична отображению *1:n*. Предположим, что рассматриваемое нами предприятие строит свою деятельность на основании контрактов, заключаемых с заказчиками. Этот факт отображается в модели "сущность-связь" с помощью связи КОНТРАКТ-ЗАКАЗЧИК, объединяющей сущности КОНТРАКТ(НОМЕР, СРОК ИСПОЛНЕНИЯ, СУММА) и ЗАКАЗЧИК(НАИМЕНОВАНИЕ, АДРЕС). Так как с одним заказчиком может

быть заключено более одного контракта, то связь КОНТРАКТ-ЗАКАЗЧИК между этими сущностями будет иметь степень *n:1*.

В данном случае, по совершенно очевидным соображениям (каждый контракт заключен с конкретным заказчиком, а каждый заказчик имеет хотя бы один контракт, иначе он не был бы таковым), каждая сущность имеет обязательный класс принадлежности.

- **многие-ко-многим** (*n:n*). В этом случае каждая из ассоциированных сущностей может быть представлена любым количеством экземпляров. Пусть на рассматриваемом нами предприятии для выполнения каждого контракта создается рабочая группа, в которую входят сотрудники разных отделов. Поскольку каждый сотрудник может входить в несколько (в том числе и ни в одну) рабочих групп, а каждая группа должна включать не менее одного сотрудника, то связь между сущностями СОТРУДНИК и РАБОЧАЯ ГРУППА имеет степень *n:n*.

3.7.2. Методология IDEF1x

Метод **IDEF1**, разработанный Т. Рэмей (T. Ramey), основан на подходе П. Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии **IDEF1** создана ее новая версия - методология **IDEF1x**. **IDEF1x** разработана с учетом таких требований, как простота изучения и возможность автоматизации. **IDEF1x**-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF) [12,14].

Сущность в методологии **IDEF1x** является *независимой* от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. На схемах изображается в виде **прямоугольника** (рис. 3.40).

Сущность называется *зависимой* от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности. На схемах изображается в виде **прямоугольника с закругленными краями** (рис. 3.40).

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком: Имя сущности/Номер сущности, например, Сотрудник/44.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В **IDEF1x** могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка (на диаграмме изоб-

ражается латинской буквой P со стороны экземпляра сущности-потомка);

- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка (на диаграмме изображается латинской буквой Z со стороны экземпляра сущности-потомка);
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка (на диаграмме изображается целым положительным числом со стороны экземпляра сущности-потомка).

Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка рис. 3.40. Мощность связи может принимать следующие значения: N – 0, 1 или более, Z – 0 или 1, P – 1 или более, точно – любая цифра (мощность по умолчанию - N).

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

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (рис. 3.40). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

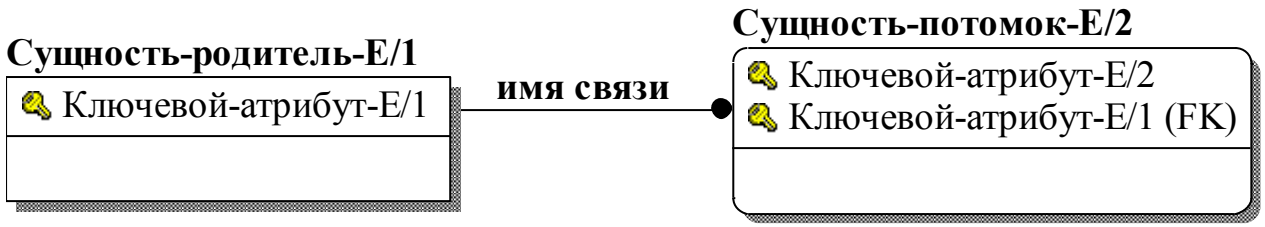


Рис. 3.40. Идентифицирующая связь

Пунктирная линия изображает неидентифицирующую связь (рис. 3.41). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

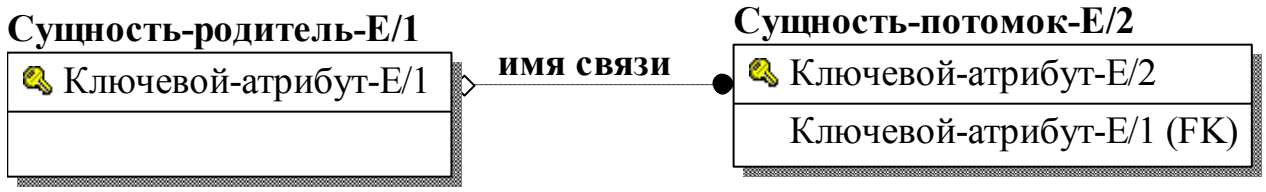


Рис. 3.41. Неидентифицирующая связь

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой (рис. 3.42).

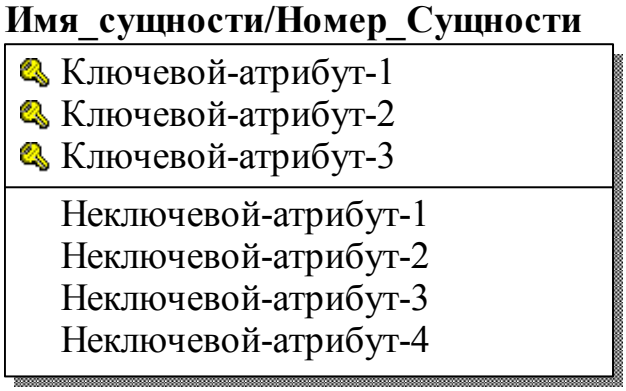


Рис. 3.42. Ключевые и неключевые атрибуты сущности

Сущности могут иметь также внешние ключи (Foreign Key - FK), которые используются в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках (рис. 3.40, 3.41).

Методология IDEFIX подразделяется на уровни, соответствующие проектируемой модели данных системы. Каждый такой уровень соответствует определенной фазе проекта. Такой подход полезен при создании систем по принципу «сверху вниз».

Верхний уровень состоит из Entity Relation Diagram (Диаграмма сущность-связь) и the Key-Based model (Модель данных, основанная на ключах).

Нижний уровень состоит из the Fully Attributed model (Полная атрибутивная модель) и Transformation Model (Трансформационная модель).

Диаграмма сущность-связь (Entity Relation Diagram) является самым высоким уровнем в модели данных и определяет набор сущностей и отношений между ними. Целью этой диаграммы является формирование общего взгляда на систему для ее дальнейшей детализации.

Модель данных, основанная на ключах (the Key-Based model). Этот тип модели описывает структуру данных системы, в которую включены все сущности и атрибуты, в том числе ключевые. Целью этой модели является детализация модели сущность-связь, после чего модель данных может начать реализовываться.

Полная атрибутивная модель (the Fully Attributed model). Эта модель включает в себя все сущности, атрибуты, связи и является наиболее детальным представлением структуры данных. Полная атрибутивная модель представляет данные в третьей нормальной форме.

Трансформационная модель (Transformation Model) содержит всю информацию для реализации проекта, который может быть частью общей ин-

формационной системы и описывать предметную область. Трансформационная модель позволяет проектировщикам и администраторам БД представлять, какие объекты БД хранятся в словаре данных, и проверить, насколько физическая модель данных удовлетворяет требованиям информационной системы. Фактически из трансформационной модели автоматически можно получить модель СУБД, которая является точным отображением системного каталога СУБД.

3.7.3. Правила формирования отношений

Правила формирования отношений основываются на учете степени связи между сущностями и классе принадлежности (КП) экземпляров сущности.

Формирование отношений для связи 1:1.

Правило 1. Если для обеих сущностей класс принадлежности обязательный, то формируется одно отношение, при этом первичный ключ отношения может быть любым ключом этих двух сущностей.

Правило 2. Если класс принадлежности одной из сущностей обязателен, а другой необязательный, то под каждую сущность формируется по отношению, в каждое отношение включается первичный ключ каждой сущности и к сущности, которая имеет обязательный класс принадлежности, добавляется в качестве атрибута ключ сущности с необязательным классом принадлежности.

Правило 3. Если класс принадлежности обеих сущностей необязательный, то необходимо создать 3 отношения: по одному на каждую сущность и еще одно, первичный ключ которого состоит из первичных ключей каждой сущности.

Формирование отношений для связи 1:M

Правило 4. Если класс принадлежности M-связной сущности обязателен (класс принадлежности 1-связной сущности не имеет значения), то формируются два отношения, при этом M-связной сущности соответствует отношение с первичным ключом этой сущности и внешним ключом 1-связной сущности.

Правило 5. Если класс принадлежности M-связной сущности необязательный, то подобно правилу 3 формируются 3 отношения.

Формирование отношений для связи M:M

Правило 6. Независимо от класса принадлежности сущностей формируются 3 отношения, как описано выше.

3.7.4. Пример построения ER-модели

В гипотетическом пункте обмена валюты создается локальная ИС, призванная автоматизировать процесс учета сделок купли-продажи валюты. Создаваемая система должна обеспечить ввод, хранение и поиск информации о сделках, совершенных в данном пункте обмена. Каждой сделке присваивается уникальный цифровой код. Информация о сделке должна содержать сведения о да-

те и времени сделки, суммах покупаемой и продаваемой валют, фамилии, имени, отчестве и номере паспорта клиента, а также о фамилии, инициалах и учетном номере личного дела кассира в отделе кадров. Система должна иметь возможность вычислять денежный оборот за один или несколько дней. А также осуществлять поиск информации о сделках по номеру паспорта клиента. Задача состоит в проектировании структуры базы данных, разрабатываемой автоматизированной ИС [5].

После проведения анализа предметной области можно выделить следующие основные сущности: СДЕЛКА, ВАЛЮТА, КЛИЕНТ, КАССИР. Для задания связей между указанными сущностями составим формальное описание предметной области при помощи ряда высказываний.

Любой КЛИЕНТ должен совершать одну или несколько СДЕЛОК.

Каждую СДЕЛКУ должен совершать только один КЛИЕНТ.

Любой КАССИР может обслуживать одну или несколько СДЕЛОК, но может не обслуживать и ни одной.

Каждую СДЕЛКУ должен обслуживать только один КАССИР.

Любая ВАЛЮТА может покупаться и продаваться при разных СДЕЛКАХ.

При совершении СДЕЛКИ должна покупаться одна ВАЛЮТА и продаваться другая ВАЛЮТА.

Анализ данных высказываний позволяет выделить четыре связи:

КЛИЕНТ совершает СДЕЛКУ.

КАССИР обслуживает СДЕЛКУ.

ВАЛЮТА покупается при СДЕЛКЕ.

ВАЛЮТА продается при СДЕЛКЕ.

Используя методологию **IDEF1x**, построим **ER**-модель описанной выше предметной области (рис.3.43)

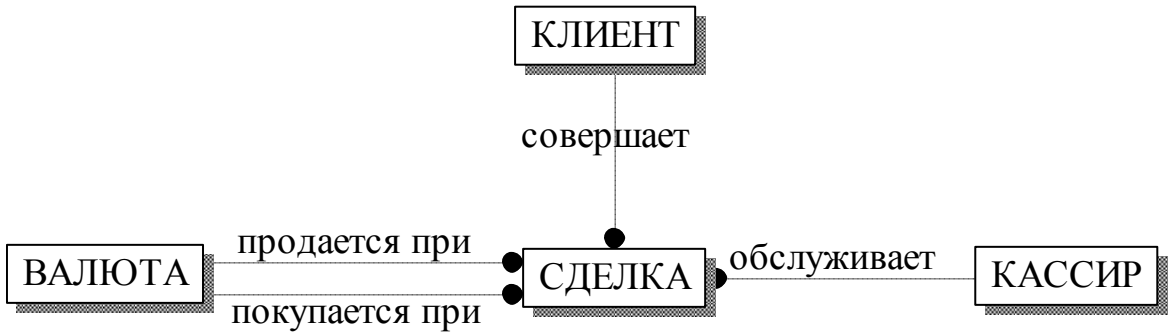


Рис. 3.43. **ER**-модель пункта обмена валюты

Все четыре связи являются связями "один-ко-многим". Во всех случаях сущность СДЕЛКА является дочерней. Все связи неидентифицирующие, т.к. любой экземпляр сущности СДЕЛКА может быть однозначно идентифицирован по коду сделки, т.е. вне зависимости от экземпляров других сущностей.

Во всех связях родительские сущности не могут принимать пустые значения, поскольку при отсутствии экземпляра хотя бы одной из родительских сущностей экземпляр сущности СДЕЛКА перестает описывать сделку по обмену валюты.

Теперь для каждой сущности укажем первичные и неключевые атрибуты. Полезным источником информации в этом случае может стать перечень требований к хранимой информации, приведенный в задании.

Сведения о клиенте должны содержать фамилию, имя, отчество и номер его паспорта. В качестве первичного ключа можно выбрать номер паспорта клиента, т.к. он является уникальным. Однако номер паспорта не является числом и для его хранения используется строка минимум из 10 символов, что не совсем удобно. Поэтому введем для каждого КЛИЕНТА уникальный числовой номер, который и будет первичным ключом. Атрибут *номер паспорта* сделаем альтернативным ключом, чтобы обеспечить возможность быстрого поиска информации о сделках по его значению.

Сведения о кассире включают фамилии и инициалы, а также учетный номер личного дела кассира. По тем же соображениям в качестве первичного ключа числовой номер кассира, а не учетный номер личного дела.

Сведения о валюте будут содержать два атрибута: *код валюты* в качестве первичного ключа и *название валюты*.

Сущность СДЕЛКА часть атрибутов унаследует от родительских сущностей и остается добавить следующие: *дата сделки, время сделки, сумма покупаемой валюты, сумма продаваемой валюты*. Очевидно, что первичным ключом следует выбрать уникальный цифровой код сделки. Поскольку по заданию нам требуется вычислять денежный оборот за определенный период времени, полезно сделать атрибут *дата сделки* инверсным входом, т.к. он часто будет использоваться для доступа к данным. Поскольку между сущностями ВАЛЮТА и СДЕЛКА двойная связь, а имена атрибутов в одной сущности должны быть уникальны, то каждому внешнему ключу в сущности СДЕЛКА от сущности ВАЛЮТА следует задать разное имя роли.

На этом процесс создания логический модели завершается, а сама модель приобретает вид, представленный на рис. 3.44.

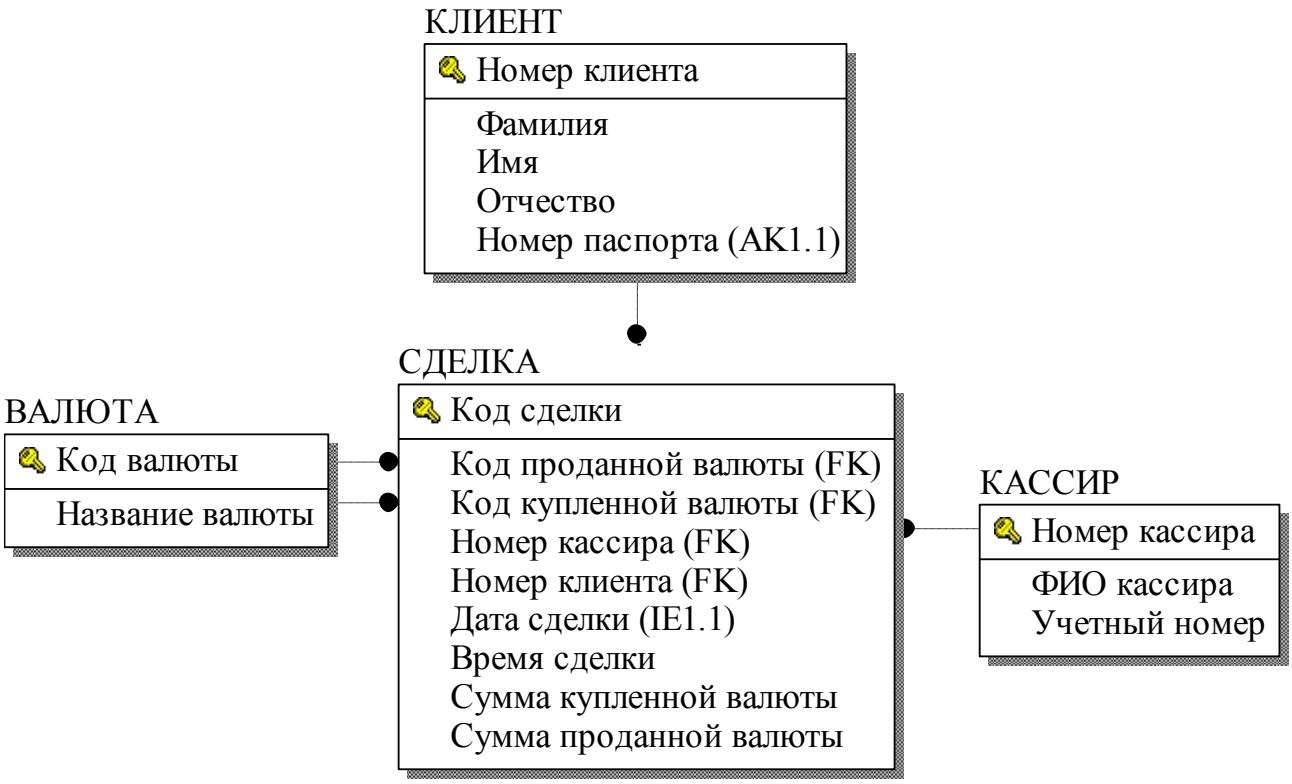


Рис. 3.44. Полная атрибутивная модель пункта обмена валюты

3.8. Контрольные вопросы

1. В чем заключается сущность структурного подхода?
2. Перечислите основные принципы структурного подхода.
3. Какие методологии структурного анализа и проектирования существуют?
4. Дайте характеристику основных элементов методологии IDEF0.
5. Каково назначение методологии IDEF0?
6. Сформулируйте правила построения контекстной диаграммы IDEF0.
7. Перечислите типы стрелок в IDEF0 и сформулируйте их назначение.
8. Какие типы диаграмм реализуются в рамках методологии IDEF0?
9. Каковы правила наименования функциональных блоков и стрелок в IDEF0?
10. Каковы правила построения диаграмм декомпозиции IDEF0?
11. Некоторое предприятие выпускает пластмассовые изделия для населения. Сырьевые полимеры и красители предприятие закупает у сторонних производителей. Предприятие планирует свое производство исходя из конъюнктуры рынка. Предприятие работает только с оптовыми клиентами. Используя это описание, постройте функциональную модель с точки зрения директора предприятия.
12. Каково назначение методологии IDEF3?

13. Дайте характеристику основных элементов методологии IDEF3.
14. Какие типы связей существуют в IDEF3?
15. Какие типы соединений используются в IDEF3?
16. Каковы правила построения диаграмм IDEF3?
17. Процесс производства некоторого изделия можно представить в виде такой последовательности действий: подготовка сырья, изготовление изделия (засыпка сырья, контроль параметров изготовления, выемка изделия), проверка качества сырья, приемка изделия – отправка на склад, браковка изделия – отправка на переработку. Постройте модель процесса, используя методологию IDEF3.
18. Каково назначение диаграмм DFD?
19. Дайте характеристику основных элементов диаграмм DFD.
20. Каковы правила построения диаграмм DFD?
21. Что такое миниспецификации и каково их назначение?
22. Для чего используют БНФ-нотации?
23. Ваша цель – разработать эффективную программную систему для обеспечения электронной продажи книг (через Интернет). Система должна предоставлять следующие возможности потенциальному клиенту: i) регистрация клиента; ii) просмотр каталога книг, представленных к продаже; iii) оформление заказа; iv) просмотр ранее сделанных заказов для зарегистрированных клиентов. Постройте DFD диаграмму, описывающую процесс взаимодействия клиента с системой электронных заказов.
24. Какие методы задания спецификаций процессов существуют?
25. Каково назначение методологии ERD?
26. Дайте характеристику основных элементов методологии ERD.
27. В чем отличие методологии IDEF1x от ERD?
28. Дайте характеристику основных элементов методологии IDEF1x.
29. Используя текст задания 23, постройте инфологическую модель данных, отражающую предметную область задачи.

ГЛАВА 4. СРЕДСТВА АВТОМАТИЗАЦИИ МЕТОДОЛОГИЙ СТРУКТУРНОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ

4.1. СОСТАВ, СТРУКТУРА И ФУНКЦИОНАЛЬНЫЕ ОСОБЕННОСТИ CASE-СРЕДСТВ

CASE-средства служат инструментарием для поддержки и усиления методов структурного анализа и проектирования. Эти инструменты поддерживают работу пользователей при создании и редактировании графического проекта в интерактивном режиме, они способствуют организации проекта в виде иерархии уровней абстракции, выполняют проверки соответствия компонентов. Фактически **CASE**-средства представляют собой новый тип графически-ориентированных инструментов, восходящих к системе поддержки жизненного цикла программного обеспечения (ЖЦ ПО). Обычно к ним относят любое программное средство, обеспечивающее автоматическую помощь при разработке ПО, его сопровождении или деятельности по управлению проектом, и проявляющее следующие дополнительные черты [18]:

- мощная графика для описания и документирования систем ПО, а также для улучшения интерфейса с пользователем, развивающая творческие возможности специалистов и не отвлекающая их от процесса проектирования на решение второстепенных вопросов;
- интеграция, обеспечивающая легкость передачи данных между средствами и позволяющая управлять всем процессом проектирования и разработки ПО непосредственно через процесс планирования проекта;
- использование компьютерного хранилища (репозитария) для всей информации о проекте, которая может разделяться между разработчиками и исполнителями как основа для автоматического продуцирования ПО и повторного его использования в будущих системах.

Помимо перечисленных основополагающих принципов графической ориентации, интеграции и локализации всей проектной информации в репозитарии в основе концептуального построения **CASE**-средств лежат следующие положения:

1. Человеческий фактор, определяющий разработку ПО как легкий, удобный и экономичный процесс.

2. Широкое использование базовых программных средств, получивших массовое распространение в других приложениях (БД и СУБД, компиляторы с различных языков программирования, отладчики, документаторы, издательские системы, оболочки экспертных систем и базы знаний, языки четвертого поколения и др.).

3. Автоматизированная или автоматическая кодогенерация, выполняющая несколько видов генерации кодов: преобразования для получения документации, формирования БД, ввода/модификации данных, получения выполняемых машинных кодов из спецификаций ПО, автоматической сборки модулей из словарей и моделей данных и повторно используемых программ, автоматической конверсии ранее используемых файлов в форматы новых требований.

4. Ограничение сложности, позволяющее получать компоненты, поддающиеся управлению, обозримые и доступные для понимания, а также обладающие простой и ясной структурой.

5. Доступность для разных категорий пользователей.

6. Рентабельность.

7. Сопровождаемость, обеспечивающая способность адаптации при изменении требований и целей проекта.

Интегрированный **CASE**-пакет содержит четыре основные компоненты [18]:

1) Средства централизованного хранения всей информации о проектируемом ПО в течение всего ЖЦ (**репозитарий**) являются основой **CASE**-пакета. Соответствущая БД должна иметь возможность поддерживать большую систему описаний и характеристик и предусматривать надежные меры по защите от ошибок и потерь информации. Репозитарий должен обеспечивать:

- инкрементный режим при вводе описаний объектов;
- распространение действия нового или скорректированного описания на информационное пространство всего проекта;
- синхронизацию поступления информации от различных пользователей;
- хранение версий проекта и его отдельных компонент;
- сборку любой запрошенной версии;
- контроль информации на корректность, полноту и состоятельность.

2) Средства **ввода** предназначены для ввода данных в репозитарий, а также для организации взаимодействия с **CASE**-пакетом. Эти средства должны поддерживать различные методологии и использоваться на всем ЖЦ разными категориями разработчиков: аналитиками, проектировщиками, инженерами, администраторами и т. д.

3) Средства **анализа, проектирования и разработки** предназначены для того, чтобы обеспечить планирование и анализ различных описаний, а также их преобразования в процессе разработки.

4) Средства **вывода** служат для документирования, управления проектами кодовой генерации.

Все перечисленные компоненты в совокупности должны:

- поддерживать графические модели;
- контролировать ошибки;
- организовывать и поддерживать репозитарий;
- поддерживать процесс проектирования и разработки.

Поддержка графических моделей. Графическая ориентация **CASE** заключается в том, что программы являются схематическими проектами и формами, которые много проще в использовании, чем многостраничные описания. Для представления программ применяются структурные диаграммы различных типов, дополнительное достоинство которых заключается в их использовании и в качестве наглядной “двумерной” документации по проекту.

Для **CASE** существенны 4 типа диаграмм: диаграммы функционального проектирования (для этих целей наиболее часто употребляются **DFD**-

диаграммы потоков данных), диаграммы моделирования данных (как правило, **ERD**-диаграммы “сущность-связь”), диаграммы моделирования поведения (как правило, **STD**-диаграммы переходов состояний) и структурные диаграммы (карты), применяющиеся на этапе проектирования и описывающие отношения между модулями и внутримодульную структуру. Создание и модификация подобных диаграмм осуществляется с помощью специальных графических редакторов (диаграммеров), являющихся сервисными средствами на этапах анализа требований и проектирования спецификаций. Современные диаграммеры обеспечивают:

- создание иерархически связанных диаграмм, в которых комбинируются графические и текстовые объекты;
- создание и редактирование объектов в любом месте диаграммы;
- создание, перемещение и выравнивание групп объектов, изменение их размеров, масштабирование;
- сохранение связей между объектами при их перемещении и изменении размеров;
- автоматический контроль ошибок и др.

Реализация подобных возможностей позволяет пользователю целиком сосредоточиться на собственно проектировании, не отвлекаясь на решение второстепенных вопросов, связанных с размещением элементов диаграмм, их компоновкой и т.п.

Полученные диаграммы дают ясное понимание и решение проблемы, позволяют проанализировать функционирование создаваемого ПО, фиксируют связи между разработчиками, пользователями и руководителями, обеспечивают стандартизацию представления структуры программы и данных.

Контроль ошибок. Важность контроля ошибок на этапах анализа требований и проектирования спецификаций обуславливается возможностью их автоматического обнаружения на ранних этапах ЖЦ. **CASE** обеспечивает автоматическую верификацию и контроль проекта на полноту и состоятельность на ранних этапах ЖЦ, что влияет на успех разработки в целом. В подтверждение этого можно привести следующие статистические данные, основанные на отчетах фирмы **TRW** по анализу 5 крупных проектов:

- при традиционной организации работ ошибки проектирования и кодирования составляют, соответственно, 64% и 32% от общего числа ошибок;
- ошибки проектирования в 100 раз труднее обнаружить на этапе сопровождения ПО, чем на этапах анализа требований и проектирования спецификаций.

В **CASE** диаграммеры и верификаторы способны осуществлять следующие типы контроля:

1) *Контроль синтаксиса диаграмм и типов их элементов*. Обычно такой контроль осуществляется при вводе и редактировании элементов диаграмм. Примеры контролируемых ситуаций:

- *по синтаксису:* любой функциональный элемент диаграммы должен иметь по крайней мере один входной и один выходной поток; два элемента данных не могут быть непосредственно связаны;
- *по типам:* функциональный элемент должен всегда использоваться для представления процедурной компоненты; поток данных всегда должен быть представлен компонентой данных.

2) *Контроль полноты и состоятельности диаграмм:* все элементы диаграмм должны быть идентифицированы и отражены в репозитарии. Например, для **DFD** контролируются неименованные или несвязанные потоки данных, процессы и хранилища данных; источники и стоки данных (внешние сущности) вне контекстной диаграммы; хранилища данных на контекстной диаграмме и т.п. При анализе словаря данных необходимо выявлять циклические определения, эквивалентные определения, неопределенные объекты.

3) *Контроль декомпозиции функций* включает оценку качества на основе различных метрик ПО и частичный семантический контроль.

4) Сквозной контроль диаграмм одного или различных типов на предмет их состоятельности по уровням - *вертикальное и горизонтальное балансирование диаграмм*. При вертикальном балансировании (диаграммы одного типа) выявляются несбалансированные потоки данных между детализируемой и детализирующей диаграммами. Горизонтальное балансирование определяет некорректности между **DFD**, **ERD**, **STD**, словарями данных и миниспецификациями процессов. Так при балансировании **DFD**-**ERD** контролируется соответствие каждого хранилища данных на **DFD** сущности или отношению на **ERD**. Контроль **DFD-STD** осуществляется по следующим правилам: каждый управляющий процесс на **DFD** детализируется спецификацией управления **STD**, и наоборот, каждой **STD** должен соответствовать управляющий процесс; каждое условие (действие) в **STD** должно соответствовать входному (выходному) управляющему потоку на **DFD**, и наоборот, каждому управляющему потоку в зависимости от его направленности должно соответствовать условие/действие на **STD**. При балансировании **DFD**-словаря данных и миниспецификаций должны проверяться следующие правила:

- каждый поток и хранилище данных должны быть определены в словаре данных (контроль неопределенных значений) и, наоборот, каждое определение в словаре должно быть отражено на диаграмме, в миниспецификации или другом определении (контроль неиспользуемых значений);
- каждый процесс на **DFD** должен детализироваться с помощью **DFD** или миниспецификации (но не тем и другим одновременно) и, наоборот, каждая миниспецификация должна соответствовать единственному процессу;
- ссылки к данным в миниспецификациях должны соответствовать объектам на диаграммах и в словаре данных;

- по возможности должна контролироваться семантика миниспецификации: например, если входные и/или выходные потоки связаны с хранилищем данных, то это должно быть отражено в миниспецификации.

Организация и поддержка репозитария. Основные функции средств организации и поддержки репозитария - хранение, доступ, обновление, анализ и визуализация всей информации по проекту ПО. Содержимое репозитария включает не только информационные объекты различных типов, но и отношения между их компонентами, а также правила использования или обработки этих компонент. Репозитарий может хранить свыше 100 типов объектов, примерами которых являются структурные диаграммы, определения экранов и меню, проекты отчетов, описания данных, логика обработки, модели данных, модели организации, модели обработки, исходные коды, элементы данных и т.п.

Каждый информационный объект в репозитарии описывается перечислением его свойств: идентификатор, имена-синонимы, тип, текстовое описание, компоненты, файл-хранилище, область значений. Кроме этого, хранятся все отношения с другими объектами (например, все объекты, в которых данный объект используется; все перекрестные ссылки), правила формирования и редактирования объекта, а также контрольная информация о времени порождения объекта, времени его последнего обновления, кем и в каком проекте он был порожден, номере версии, возможности обновления и т.п.

На основе репозитария осуществляется интеграция **CASE**-средств и разделение системной информации между разработчиками. При этом возможности репозитария обеспечивают несколько уровней интеграции: общий пользовательский интерфейс по всем средствам, передачу данных между средствами, интеграцию этапов разработки через единую систему представлений фаз ЖЦ, передачу данных и средств между аппаратурными платформами.

Репозитарий является базой для стандартизации документации по проекту и контроля состоятельности проектных спецификаций. Все отчеты строятся автоматически по репозитарию, ниже перечислены основные их типы:

1. *Отчеты по содержимому* включают сводки потоков данных и их компонент, сводки всех пар интерфейсов в описывающих межмодульные отношения структурных диаграммах, списки входных и выходных потоков для каждого функционального блока диаграмм, списки измененных за определенный период объектов, истории всех изменений объектов, описания модулей, планы тестирования модулей и подпрограмм, списки всех данных и их атрибутов, а также отношений между их компонентами и правил их обработки.

2. *Отчеты по перекрестным ссылкам* включают списки всех вызывающих и вызываемых модулей; списки объектов репозитария, к которым имеет доступ конкретный разработчик; сводки диаграмм, использующих конкретные данные; маршруты движения данных от входа к выходу.

3. *Отчеты по результатам анализа* включают сводки балансирования диаграмм по уровням, списки неопределенных информационных объектов, списки неполных диаграмм, сводки результатов анализа структуры проекта,

списки несогласованных в диаграммах и репозитарии объектов, списки неиспользуемых объектов, списки удаленных объектов.

4. *Отчеты по декомпозиции объектов* включают таблицы иерархии всех объектов модели.

Важные функции управления и контроля проекта также реализуются на основе репозитария. В частности, через репозитарий может осуществляться контроль безопасности (ограничения доступа, привилегии доступа), контроль версий, контроль изменений и др.

Поддержка процесса проектирования и разработки. При поддержке процесса проектирования и разработки основную роль играют следующие возможности **CASE**-пакетов: покрытие ЖЦ, поддержка прототипирования, поддержка структурных методологий, автоматическая кодогенерация.

При *покрытии ЖЦ* наибольшее внимание уделяется его наиболее критичным этапам - анализу требований и проектированию спецификаций. Последние являются основой всего проекта, поэтому их полнота и корректность влияют на успех разработки в целом.

Важную роль при автоматизации ранних этапов ЖЦ играют возможности *поддержки прототипирования.* Соответствующие средства используются для определения системных требований и ответа на вопросы об ожидаемом поведении системы. Такие средства как генераторы меню, экранов и отчетов позволяют быстро построить прототипы пользовательских интерфейсов и снабдить моделью функционирования системы с позиций конечного пользователя. Использование языков четвертого поколения (**4GL**) позволяет строить более сложные модели, при этом прототип позволяет промоделировать основные функции системы, но не способен контролировать ее ожидаемое поведение. Исполняемые языки спецификаций преобразуют процесс разработки в следующий итеративный процесс: спецификации определяются и выполняются, затем производится переопределение или корректировка. Созданные таким образом прототипы позволяют определять, является ли проектируемая система полной и корректной.

Поддержка структурных методологий осуществляется за счет средств их автоматизации на следующих двух уровнях:

- подготовка документации, графическая поддержка построения структурных диаграмм различных типов, продуцирование спецификаций для детализации функциональных блоков в диаграммах и структур данных на нижних уровнях (для таких спецификаций введен специальный термин - “миниспецификация”);
- корректное использование шагов обработки в методологиях.

Кодогенерация осуществляется на основе репозитария и позволяет автоматически построить до 80-90% объектных кодов или текстов программ на языках высокого уровня. При этом различными **CASE**-пакетами поддерживаются практически все известные языки программирования. Средства кодогенерации по отношению к полноте целевого продукта разделяются на средства генерации каркаса ПО и средства генерации полного продукта. В первом случае

автоматически строится откомментированная логика (потоки управления) ПО, а также коды для БД, файлов, экранов, отчетов и т.п., остальные фрагменты ПО кодируются вручную. Во втором случае из проектных спецификаций генерируется полная документированная программа, включая выполняемый код, пользовательскую и программную документацию, наборы тестов и т.д. Все эти компоненты полной программы связываются в единый объект, хранящийся в репозитарии для облегчения доступа и сопровождения.

Идея автоматической кодогенерации на основе модели заключается в следующем. Любая программа схематически может быть представлена в виде тройки: обрабатываемые данные, логический каркас обработки, линейные участки обработки. Схема базы данных может быть легко сгенерирована на основании модели “сущность-связь”, и современные средства информационного моделирования обеспечивают такую генерацию. Логика обработки генерируется на основе диаграмм потоков данных: известны алгоритмы автоматического преобразования иерархии **DFD** в структурные карты, а с задачей получения из структурных карт соответствующих кодов легко справляется теория компиляции. Наконец, линейным участкам соответствуют миниспецификации модели. И именно здесь лежит ключ к высокому проценту автоматически сгенерированного кода, существенно зависящему от метода задания миниспецификаций.

4.2. Классификация CASE-средств

Воспользуемся классификацей, приведенной в книге Г.Н. Калянова [18].

Все **CASE**-средства делятся на типы, категории и уровни. Классификация ***по типам*** отражает функциональную ориентацию **CASE**-средств в технологическом процессе.

Анализ и проектирование. Средства данной группы используются для создания спецификаций системы и ее проектирования; они поддерживают широко известные методологии проектирования. К таким средствам относятся: **AllFusion Process Modeler** (Computer Associates), **Oracle Designer** (Oracle), **PowerDesigner** (Sybase), **System Architect** (Popkin Software & Systems), **MS Visio** (Microsoft), **Rational Rose** (IBM), **Together** (Borland), **CASE.Аналитик** (Эйтэкс), **The Developer** (ASYST Technologies), **POSE** (Computer Systems Advisers), **ProKit*Workbench** (McDonnell Douglas), **Excelerator** (Index Technology), **Design-Aid** (Nastec), **Design Machine** (Optima), **MicroStep** (Meta Systems), **vsDesigner** (Visual Software), **Analist/Designer** (Yourdon), **Design/IDEF** (Meta Software Corporation), **SELECT** (Select Software Tools), **Westmount I-CASE Yourdon** (Westmount Technology B.V. & CADRE Technologies), **CASE/4/0** (microTOOL GmbH), **Case Studio**. Их целью является определение системных требований и свойств, которыми система должна обладать, а также создание проекта системы, удовлетворяющей этим требованиям и обладающей соответствующими свойствами. На выходе продуцируются спецификации компонент системы и интерфейсов, связывающих эти компоненты, а также “калька” архи-

тектуры системы и детальная “калька” проекта, включающая алгоритмы и определения структур данных.

Проектирование баз данных и файлов. Средства данной группы обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в третью нормальную форму, автоматическую генерацию схем БД и описаний форматов файлов на уровне программного кода: **AllFusion ERwin Data Modeler** (Computer Associates), **Chen Toolkit** (Chen & Asssociates), **PowerDesigner** (Sybase), **Oracle Designer** (Oracle), **Silverrun** (Computer Systems Advisers), **Case Studio, xCase Professional (**RESolution**).**

Программирование. Средства этой группы поддерживают этапы программирования и тестирования, а также автоматическую кодогенерацию из спецификаций, получая полностью документированную выполняемую программу: **Visual Studio** (Microsoft), **Borland Delphi** (Borland), **COBOL 2/Workbench** (Micro Focus), **DECASE** (DEC), **NETRON/CAP** (Netron), **APS** (Sage Software) и др.. Помимо диаграммеров различного назначения и средств поддержки работы с репозитарием, в эту группу средств включены и традиционные генераторы кодов, анализаторы кодов (как в статике, так и в динамике), генераторы наборов тестов, анализаторы покрытия тестами, отладчики.

Сопровождение и реинжениринг. К таким средствам относятся документаторы, анализаторы программ, средства реструктурирования и реинжениринга: **Adpac CASE Tools** (Adpac), **Scan/COBOL** u **SuperStructure** (Computer Data Systems), **Inspector/Recoder** (Language Technology). Их целью является корректировка, изменение, анализ, преобразование и реинжениринг существующей системы. Средства позволяют осуществлять поддержку всей системной документации, включая коды, спецификации, наборы тестов; контролировать покрытие тестами для оценки полноты тестируемости; управлять функционированием системы и т.п. Особый интерес представляют средства обеспечения мобильности (в **CASE** они получили название средств миграции) и реинжиниринга. К средствам миграции относятся трансляторы, конверторы, макрогенераторы и др., позволяющие обеспечить перенос существующей системы в новое операционное или аппаратурное окружение. Средства реинжиниринга включают:

- статические анализаторы для продуцирования схем системы ПО из ее кодов, оценки влияния модификаций (например, ”эффекта ряби” - внесение изменений с целью исправления ошибок порождает новые ошибки);
- динамические анализаторы (обычно, компиляторы и интерпретаторы с встроенными отладочными возможностями);
- документаторы, позволяющие автоматически получать обновленную документацию при изменении кода;
- редакторы кодов, автоматически изменяющие при редактировании и все предшествующие коду структуры (например, спецификации);
- средства доступа к спецификациям, их модификации и генерации нового (модифицированного) кода;

- средства реверсного инжиниринга, транслирующие коды в спецификации.

Окружение. Средства поддержки платформ для интеграции, создания и придания товарного вида **CASE**-средствам: **Multi/Cam** (AGS Management Systems), **Design/OA** (Meta Software).

Управление проектами. Средства, поддерживающие планирование, контроль, руководство, взаимодействие, т.е. функции, необходимые в процессе разработки и сопровождения проектов: **Project Workbench** (Applied Business Technology), **Microsoft Project** (Microsoft), **Borland StarTeam** (Borland).

Классификация ***по категориям*** определяет уровень интегрированности по выполняемым функциям и включает вспомогательные программы (**tools**), пакеты разработчика (**toolkit**) и инструментальные средства (**workbench**).

Категория **tools** обозначает вспомогательный пакет, решающий небольшую автономную задачу, принадлежащую проблеме более широкого масштаба.

Категория **toolkit** представляет совокупность интегрированных программных средств, обеспечивающих помощь для одного из классов программных задач; использует репозитарий для всей технической и управляющей информации о проекте, концентрируясь при этом на поддержке, как правило, одной фазы или одного этапа разработки ПО.

Категория **workbench** представляет собой интеграцию программных средств, которые поддерживают системный анализ, проектирование и разработку ПО; используют репозитарий, содержащий всю техническую и управляющую информацию о проекте; обеспечивают автоматическую передачу системной информации между разработчиками и этапами разработки; организуют поддержку практически полного ЖЦ (от анализа требований и проектирования ПО до получения документированной выполняемой программы). **Workbench**, по сравнению с **toolkit**, обладает более высокой степенью интеграции выполняемых функций, большей самостоятельностью и автономностью использования, а также наличием тесной связи с системными и техническими средствами аппаратно-вычислительной среды, на которой **workbench** функционирует. По существу, **workbench** может рассматриваться как автоматизированная рабочая станция, используемая как инструментарий для автоматизации всех или отдельных совокупностей работ по созданию ПО.

Классификация ***по уровням*** связана с областью действия CASE в пределах жизненного цикла ПО. Однако четкие критерии определения границ между уровнями не установлены, поэтому данная классификация имеет, вообще говоря, качественный характер.

Верхние (Upper) CASE часто называют средствами компьютерного планирования. Они призваны повышать эффективность деятельности руководителей фирмы и проекта путем сокращения затрат на определение политики фирмы и на создание общего плана проекта. Этот план включает цели и стратегии их достижения, основные действия в свете целей и задач фирмы, установление стандартов на различные виды взаимосвязей и т.д. Использование верхних **CASE** позволяет построить модель предметной области, отражающую всю су-

ществующую специфику. Она направлена на понимание общего и частного механизмов функционирования, имеющихся возможностей, ресурсов, целей проекта в соответствии с назначением фирмы. Эти средства позволяют проводить анализ различных сценариев (в том числе наилучших и наихудших), накапливая информацию для принятия оптимальных решений.

Средние (Middle) CASE считаются средствами поддержки этапов анализа требований и проектирования спецификаций и структуры ПО. Их использование существенно сокращает цикл разработки проекта; при этом важную роль играет возможность накопления и хранения знаний, обычно имеющихся только в голове разработчика-аналитика, что позволит использовать накопленные решения при создании других проектов. Основная выгода от использования среднего **CASE** состоит в значительном облегчении проектирования систем, проектирование превращается в итеративный процесс, включающий следующие действия:

- пользователь обсуждает с аналитиком требования к проектируемой системе;
- аналитик документирует эти требования, используя диаграммы и словари входных данных;
- пользователь проверяет эти диаграммы и словари, при необходимости модифицируя их;
- аналитик отвечает на эти модификации, изменяя соответствующие спецификации.

Кроме того, средние **CASE** обеспечивают возможности быстрого документирования требований и быстрого прототипирования.

Нижние (Lower) CASE являются средствами разработки ПО (при этом может использоваться до 30% спецификаций, созданных средствами среднего **CASE**). Они содержат системные словари и графические средства, исключающие необходимость разработки физических спецификаций. Имеются системные спецификации, которые непосредственно переводятся в программные коды разрабатываемой системы (при этом автоматически генерируется до 80-90% кодов). На эти средства возложены также функции тестирования, управления конфигурацией, формирования документации. Главными преимуществами нижних **CASE** являются: значительное уменьшение времени на разработку, облегчение модификаций, поддержка возможностей прототипирования (совместно со средними **CASE**).

4.3. AllFusion Modeling Suite

AllFusion Modeling Suite (ранее: ERwin Modeling Suite) от компании **Computer Associates** - интегрированный комплекс CASE-средств, обеспечивающий все потребности компаний-разработчиков ПО. Данный пакет служит для проектирования и анализа баз данных, бизнес-процессов и информационных систем и включает продукты, использование которых позволяет сократить расходы и повысить продуктивность процесса разработки:

- **AllFusion Process Modeler** (ранее носивший название **BPwin**) — средство моделирования бизнес-процессов;
- **AllFusion ERwin Data Modeler** (ранее называвшийся **ERwin**) — средство моделирования данных, являющееся самым популярным в мире в этой категории продуктов;
- **AllFusion Data Model Validator** (бывший **ERwin Examiner**) — средство проверки корректности моделей данных и их соответствия правилам нормализации;
- **AllFusion Model Manager** (бывший **ModelMart**) — серверный продукт, обеспечивающий коллективную работу пользователей **ERwin** и/или **BPwin** над моделями;
- **AllFusion Component Modeler** (бывший **Paradigm Plus**) — средство моделирования компонентов программного обеспечения.

AllFusion ERwin Data Modeler позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных с помощью инструментов визуального моделирования. Этот продукт широко используется разработчиками, аналитиками, проектировщиками баз данных, руководителями проектов. **AllFusion ERwin Data Modeler** поддерживает прямое и обратное проектирование (то есть как создание базы данных на основе модели, так и генерацию модели по имеющейся базе данных) для 20 типов **СУБД**, реализует методологию структурного моделирования **SADT** и нотации **IDEF1x**, **IE**, **Dimensional**, позволяет документировать структуру базы данных. Это одно из немногих средств проектирования данных, которое позволяет создавать шаблоны триггеров и хранимых процедур, на основании которых можно сгенерировать соответствующий серверный код для различных **СУБД**.

Среди возможностей данного продукта, отметим следующие:

- поддержка **Oracle9i, Sybase 12, DB2 UDB v. 7**;
- увеличение размера хранимых процедур, включаемых в модель данных;
- слияние моделей при помощи технологии полного сравнения, что позволяет объединять независимые модели для интеграции нескольких проектов в один, для миграции от отдельных моделей **AllFusion ERwin Data Modeler** к моделям **AllFusion Model Manager** или для создания корпоративных моделей;
- интеграция с **AllFusion Model Manager**, что позволяет трансформировать старые базы данных в новые и преобразовать свои модели **AllFusion ERwin Data Modeler** и **AllFusion Process Modeler** в новые версии.

AllFusion Data Model Validator — инструмент для проверки структуры баз данных и создаваемых в **ERwin** моделей, позволяющий выявлять ошибки проектирования. Этот продукт дополняет функциональность **AllFusion ERwin Data Modeler**, автоматизируя трудоемкую задачу поиска и исправления ошибок и одновременно повышая квалификацию проектировщиков баз данных благодаря встроенной системе обучения. **AllFusion Data Model Validator** помогает пользователю оптимизировать структуры данных и предлагает действия для улучшения структуры базы данных. Он может эффективно использоваться

для широкого спектра задач — от разработки первоначальной схемы до поддержки эксплуатации базы данных. Отметим, что проверка качества модели данных до генерации кода приложений при помощи **AllFusion Data Model Validator** позволяет свести к минимуму влияние вносимых изменений на код приложения, что снижает стоимость разработки приложений.

AllFusion Process Modeler представляет собой один из самых популярных инструментов визуального моделирования бизнес-процессов, который позволяет наглядно представить любую деятельность или структуру в виде модели, что дает возможность оптимизировать работу организации, проверить ее на соответствие стандартам **ISO 9000**, спроектировать ее организационную структуру, снизить издержки, исключить ненужные операции, повысить гибкость и эффективность. **BPwin** поддерживает сразу три нотации моделирования бизнес-процессов: **IDEF0** (функциональное моделирование), **DFD** (моделирование потоков данных) и **IDEF3** (моделирование потоков работ), а также методы расчета себестоимости по объему хозяйственной деятельности (функционально-стоимостной анализ, **ABC**). Данный продукт используется разработчиками, системными интеграторами, аналитиками, руководителями, специалистами по логистике, специалистами по маркетингу.

AllFusion Model Manager — сервер для обеспечения совместной деятельности группы проектировщиков при использовании **AllFusion ERwin Data Modeler** и/или **AllFusion Process Modeler** в работе над одним проектом. Сервер обеспечивает совместное создание и редактирование моделей в соответствии с заданным уровнем доступа и корпоративными стандартами, принятыми в компании, позволяет контролировать версии моделей и координировать весь ход работы над проектом. **AllFusion Model Manager** является одной из самых надежных многопользовательских сред моделирования и предназначен для выполнения крупномасштабных задач, требующих координации действий различных участников проекта.

Возможности этого продукта таковы:

- поддержка моделей прежних версий и различных типов диаграмм;
- гибкая структура метаданных;
- улучшение работы с библиотечными объектами и возможность их перемещения;
- поддержка хранения моделей в нескольких базах данных (Multi-Mart);
- управление изменениями и версиями моделей при синхронизации объектов;
- расширенная поддержка баз данных (**Microsoft SQL Server, Oracle, Sybase, Informix Modeling Client**);
- возможность синхронизации моделей **AllFusion ERwin Data Modeler** и **AllFusion Process Modeler**.

AllFusion Component Modeler представляет собой средство моделирования компонентов программного обеспечения и генерации кода приложений на основе созданных моделей. Продукт можно использовать как при создании новых приложений, так и при изменении или объединении существующих. Бла-

годаря интеграции с **AllFusion Process Modeler** этот продукт обладает возможностью использования функциональной модели вместе с объектной. **AllFusion Component Modeler** поддерживает около десятка стандартных нотаций, таких как **UML** и нотация Буча, интегрируется с технологиями **COM/DCOM, CORBA**, с продуктами **CA, Microsoft, Rational Software** и др. Этот продукт поддерживает работу в многопользовательском режиме, что обеспечивает доступ многих пользователей к одному и тому же репозитарию моделей. В целом средство, как и другие продукты семейства **AllFusion Modeling Suite**, позволяет значительно повысить производительность труда разработчиков благодаря возможности многократного использования бизнес-моделей, архитектур, интерфейсов, кода и процессов, основанных на шаблонах.

В дополнение к вышеперечисленным продуктам, в состав **AllFusion Modeling Suite** входит также новый продукт под названием **AllFusion Model Navigator** — инструмент для просмотра моделей, созданных в **AllFusion Process Modeler** и **AllFusion ERwin Data Modeler**, в режиме «только для чтения», что позволяет предотвратить несанкционированные изменения моделей. **Model Navigator** поддерживает функции просмотра моделей, их печати, а также редактирования их оформления. Продукт позволяет пользоваться информацией, содержащейся в моделях, тем сотрудникам, которые не занимаются напрямую разработкой моделей, но используют их в своей работе, например при подготовке докладов и презентаций, разработке приложений, создании технической документации.

СПИСОК ЛИТЕРАТУРЫ

1. Информатика: Учебник. 3-е переработанное издание / Под ред. проф. Н.В. Макаровой. М.: Финансы и статистика, 1999. 768 с.
2. Информационные технологии [Электронный ресурс]: разработка Сибирского государственного университета путей сообщения. Режим доступа: www.isuct.ru/~ivt/books/IS/IS1/inform/index.html
3. Шигина Н.А. Теория экономических информационных систем. Пенза: Изд. ПГУ, 2000. 84 с.
4. Базы и банки данных: Учеб. для вузов по спец. "АСУ"/В.Н. Четвериков, Г.И. Ревунков, Э.Н. Самохвалов. Под ред. В.Н. Четверикова. М.: Высш. шк., 1987. 248 с.: ил.
5. Корнеев В.В., Гареев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. М.: Нолидж, 2001. 496 с.: ил.
6. Петров В.Н. Информационные системы: Учебник. СПб.: Питер, 2002. 688 с.: ил.
7. Информационные системы: пособие для студентов вузов по специальности 071900 – "Информационные системы в экономике" / Под ред. В.Н. Волковой, Б.И. Кузина. СПб.: Изд-во СПбГТУ, 1998. 213 с.
8. Карпова Т.С. Базы данных: модели, разработка, реализация: Учебник для вузов. СПб.: Питер, 2001. 304 с.: ил.
9. Гаврилова Т. А., Хорошевский В. Ф. Базы знаний интеллектуальных систем. СПб.: Питер, 2001. 384 с.: ил.
10. Перегудов Ф.И., Тарасенко Ф.П. Введение в системный анализ: Учеб. Пособие для вузов. М.: Высш. шк., 1989. 367 с.: ил.
11. Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных. 1995, СУБД, №1. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/DBMS/DBMS7/dbms/1995/01/01.htm
12. Чен П.П-Ш. Модель "сущность-связь" – шаг к единому представлению данных. 1995, СУБД, №3. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/DBMS/DBMS7/dbms/1995/03/271.htm
13. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. М.: Финансы и статистика, 2000. 352 с.: ил. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/CASE/case1/index.htm
14. Марка Д.А., МакГоуэн К. Методология структурного анализа и проектирования. М.: МетаТехнология, 1993. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/CASE/case8/sadt_index.htm
15. Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite. М.: ДИАЛОГ-МИФИ, 2003. 432 с.: ил. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/CASE/case7/caseall6.htm
16. Структурный анализ систем: IDEF-технологии / С.В. Черемных, И.О. Семенов, В.С Ручкин. М.: Финансы и статистика, 2003. 208 с.: ил. (прикладные информационные технологии).

17. Моделирование и анализ систем. IDEF-технологии: практикум / С.В. Черемных, И.О. Семенов, В.С Ручкин. М.: Финансы и статистика, 2002. 192 с.: ил. (прикладные информационные технологии).
18. Калянов Г.Н. Консалтинг при автоматизации предприятий: Научно-практическое издание. Серия "Информатизация России на пороге XXI века". М.: СИНТЕГ, 1997. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/IS/IS8/defs0.HTM
19. Базы данных: теория и практика. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/DBMS/index.htm
20. CASE-технологии: теория и практика. [Электронный ресурс] – Режим доступа: http://www.isuct.ru/~ivt/books/CASE/index.htm
21. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

Редактор В. Л. Родичева

Подписано в печать 10.03.05 г. Усл. п.л. 8.37 Уч. изд. л. 9.29
Формат 60х84 1/16. Тираж _____ экз. Заказ _____.

Государственное образовательное учреждение высшего профессионального образования «Ивановский государственный химико-технологический университет»
153000, г. Иваново, пр-т Ф. Энгельса, 7

Отпечатано на полиграфическом оборудовании кафедры экономики и финансов ГОУ ВПО «ИГХТУ»