Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Тамбовский государственный технический университет»

Институт «Юридический»

**А. В. Платёнкин, И. П. Рак, А. В. Терехов,
В. Н. Чернышов**

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ. ПРОЕКТНЫЙ ПРАКТИКУМ

Утверждено Учёным советом ТГТУ в качестве учебного пособия для студентов дневного и заочного отделений, изучающих курсы «Проектирование информационных систем», «Проектный практикум», обучающихся по направлению 230700.62 (09.03.03) «Прикладная информатика» (профиль «Прикладная информатика в юриспруденции»)

Учебное электронное издание комплексного распространения

Тамбов
Издательство ФГБОУ ВПО «ТГТУ»
2015

УДК 004.42.(075.8)
ББК $_{3}$973.233я73-3
П37

Р е ц е н з е н т ы :

Кандидат технических наук,
технический директор филиала ОАО «МТС» в г. Тамбове
С. Б. Ушанёв

Кандидат технических наук,
доцент кафедры УКиС ФГБОУ ВПО «ТГТУ»
М. Ю. Серегин

П37 Проектирование информационных систем. Проектный практикум [Электронный ресурс] : учебное пособие для студентов дневного и заочного отделений, изучающих курсы «Проектирование информационных систем», «Проектный практикум», обучающихся по направлению 230700.62 (09.03.03) / А. В. Платёнкин, И. П. Рак, А. В. Терехов, В. Н. Чернышов. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2015. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод 35,5 Mb RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана. – ISBN 978-5-8265-1409-2.

Рассматриваются методологические основы проектирования информационных систем, основы визуального моделирования, средства объектно-ориентированного анализа и проектирования информационных систем, основные принципы разработки информационных систем, а также вопросы обеспечения информационной безопасности и защиты информации в информационных системах.

Предназначено для студентов дневного и заочного отделений, изучающих курсы «Проектирование информационных систем», «Проектный практикум», обучающихся по направлению 230700.62 (09.03.03) «Прикладная информатика» (профиль «Прикладная информатика в юриспруденции»).

УДК 004.42.(075.8)
ББК $_{3}$973.233я73-3

ISBN 978-5-8265-1409-2

ВВЕДЕНИЕ

Индустрия программного обеспечения (ПО) начала зарождаться в середине 50-х гг. прошлого столетия, когда университеты и фирмы начали использовать возможности компьютерной техники для решения определённых вычислительных задач. Многие из этих программ были написаны для своих нужд штатными программистами и распространялись между пользователями машины конкретной марки бесплатно. Самые первые отдельные фирмы, занимающиеся разработкой ПО, были созданы в США в 1959–1960 гг.

Однако до конца 1960-х гг. индустрии ПО не уделялось серьёзного внимания, поскольку её доля в компьютерном бизнесе была очень мала. Например, в 1970 г. годовой оборот всех фирм-разработчиков ПО в США был менее 4% от всего оборота компьютерного бизнеса. Серьёзный рост начался в 1970-х гг., после принятия фирмой IBM в 1969 г. решения о раздельном назначении цен на аппаратуру, ПО и услуги. Индустрия ПО очень выросла с появлением персонального компьютера в середине 1970-х, который создал растущий рынок за счёт игр, пользовательских приложений и утилит.

На сегодня производство ПО является крупнейшей отраслью мировой экономики, в которой заняты миллионы различных специалистов по всему миру. Согласно данным журнала Software Magazine сумма общемировых годовых доходов 500 самых крупных компаний-разработчиков ПО составила в 2007 г. $ 451,8 миллиарда, это выше на 14,7%, чем было в 2006 г., когда эта сумма составила $ 394 миллиарда.

1. МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

1.1. СВОЙСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программное обеспечение (ПО) – это всё или часть программ, процедур, правил и соответствующей документации системы обработки информации. ПО является одним из видов обеспечения вычислительной системы, наряду с техническим (аппаратным), математическим, информационным, лингвистическим, организационным и методическим обеспечением.

По своему назначению ПО можно разделить на:

- системное;
- прикладное;
- инструментальное.

Системное программное обеспечение – комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой – приложения пользователя.

Прикладное программное обеспечение – программа, предназначенная для выполнения определённых пользовательских задач и рассчитанная на непосредственное взаимодействие с пользователем.

Инструментальное программное обеспечение – программное обеспечение, предназначенное для использования в ходе проектирования, разработки и сопровождения программ.

В 1975 году американский учёный в области теории вычислительных систем Фредерик Брукс, проанализировав свой опыт руководства проектом разработки операционной системы OS/360 для мейнфрейма IBM System/360, определил основные свойства ПО:

- сложность;
- согласованность;
- изменяемость;
- незримость.

Сложность. Существует множество программ, которые задумываются, разрабатываются, сопровождаются и используются одним человеком. Нельзя сказать, что всё такое ПО плохо сделано, но оно имеет, как правило, очень ограниченную область применения и короткое время жизни. Обычно такое ПО легче заменить новым средством, чем переделывать или расширять.

Промышленное ПО, как правило, характеризуется длительным временем жизни и большим число пользователей. Существенной чертой та-

кого ПО является сложность. Один разработчик не в состоянии охватить все аспекты такого программного средства. Сложность подобного рода присуща всем большим программным системам.

Причинами, по которым сложность является неотъемлемым свойством ПО, являются:

– сложность реальной предметной области, из которой исходит заказ на разработку;

– трудность управления процессом разработки;

– неудовлетворительные способы описания поведения больших дискретных систем.

Сложность реального мира заключается в том, что проблемы, решаемые с помощью ПО, часто содержат сложные элементы, а к соответствующим программам предъявляется множество различных, порой взаимоисключающих требований (удобство, производительность, стоимость, надёжность и т.п.).

ПО имеет тенденцию увеличиваться в размере, при этом не просто увеличивается в размере сами элементы, а обязательно увеличивается число различных элементов. Размер современного ПО может превышать десятки тысяч строк на языках высокого уровня. Человек не в состоянии полностью понять такую систему. Поэтому для выполнения такого объёма работ требуется привлечение команды разработчиков. При этом всегда возникают значительные трудности, связанные с организацией работы большого коллектива различных специалистов.

Программные продукты являются дискретными, т.е. события, внешние по отношению к ним, могут перевести систему в новое состояние. При этом следует учитывать то, что элементы системы взаимодействуют между собой нелинейным образом, и сложность целого также возрастает нелинейно. Это служит причиной трудности понимания всех возможных состояний ПО, что ведёт к снижению её надёжности.

Согласованность. Во многих случаях новые программы должны согласовываться с уже существующим ПО. Поэтому значительная часть сложности происходит от необходимости согласования разрабатываемых программных средств с различными уже существующими интерфейсами.

Изменяемость. Программные продукты постоянно подвержены изменениям. Конечно, это относится и к другим системам (автомобилям, компьютерам и т.д.), но произведённые вещи редко подвергаются изменениям после изготовления. Их заменяют новые модели, или существенные изменения включают в более поздние серийные экземпляры того же базового проекта. Изменения здесь случаются значительно реже, чем модификация ПО. Отчасти это происходит потому, что программный продукт гораздо легче изменить.

Все удачные программные продукты подвергаются изменениям. При этом действуют два процесса. Во-первых, как только обнаруживается польза системы, начинаются попытки применения её на грани или за пределами первоначальной области. Требование расширения функций исходит в основном от пользователей, которые удовлетворены основным назначением и изобретают для неё новые применения.

Во-вторых, удачное ПО живёт дольше обычного срока существования компьютера, для которого оно первоначально было создано. Приходят новые компьютеры, и программа должна быть согласована с их возможностями.

Незримость. Программный продукт невидим. Для материальных объектов геометрические абстракции являются мощным инструментом (рис. 1.1). Например, план здания помогает архитектору и заказчику оценить пространство, возможности перемещения, виды. Становятся очевидными противоречия, можно заметить упущения. Масштабные чертежи механических деталей и объёмные модели молекул, будучи абстракциями, служат той же цели [1].

По моделям (рис. 1.2), которые строятся в процессе разработки программного продукта, невозможно представить, как он будет выглядеть и какие функции будет выполнять.

Незримость ПО не только затрудняет индивидуальный процесс проектирования, но и серьёзно затрудняет общение между специалистами, участвующими в разработке.

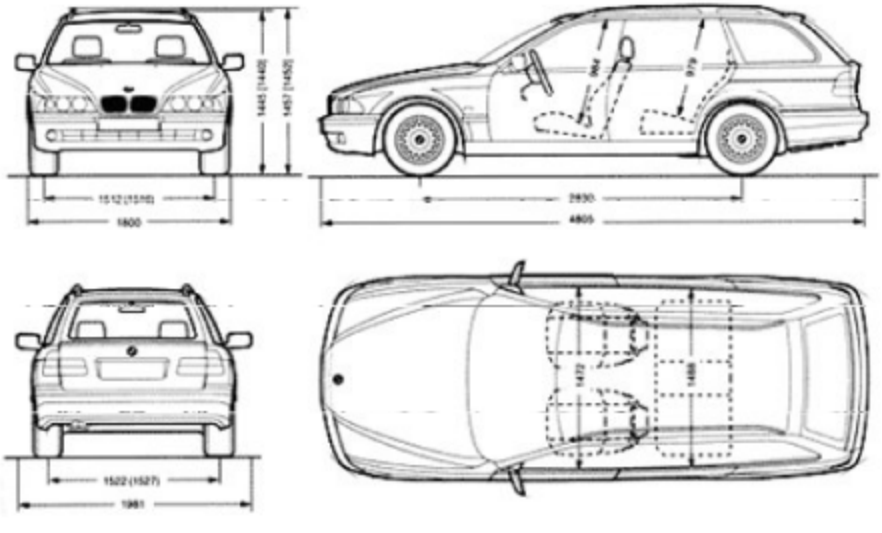

Рис. 1.1. Модель материального объекта

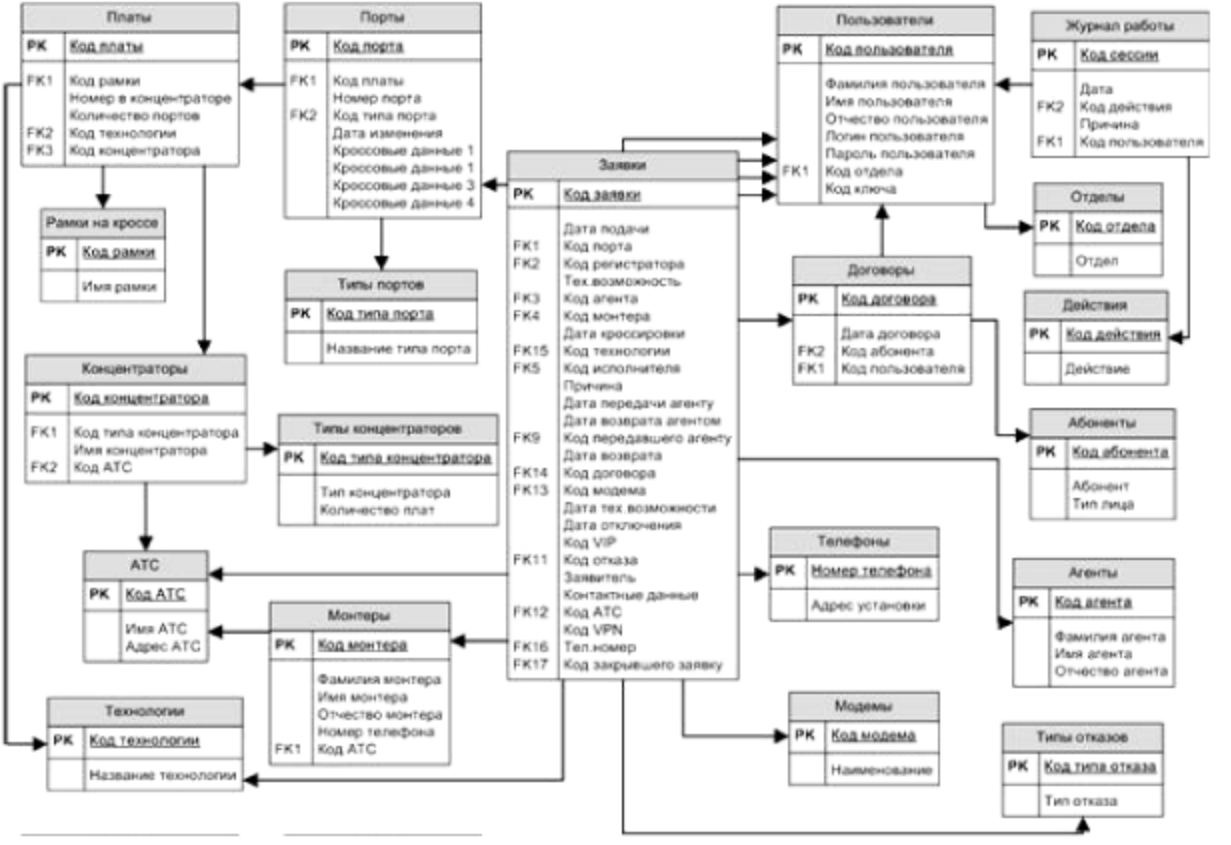

Рис. 1.2. Концептуальная модель информационной системы

1.2. ОБЩИЕ ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Система – это совокупность взаимодействующих компонентов, работающих совместно для достижения определённых целей.

Информационная система (ИС) – взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.

Основополагающим признаком системы является то, что свойства и поведение её компонентов влияют друг на друга сложным образом. Корректное функционирование отдельного компонента зависит от функционирования многих других компонентов. Системы часто имеют иерархическую структуру, т.е. в качестве компонентов содержат другие системы (подсистемы). Подсистемы могут функционировать самостоятельно, независимо от тех систем, в состав которых они входят. Вместе с тем их поведение в составе конкретной системы зависит от взаимодействия с другими подсистемами.

Сложность взаимодействия между системными компонентами означает, что система не сводится просто к сумме её составных частей. Она имеет определённые свойства, которые присущи ей именно как целостной системе. Такие интеграционные свойства системы не могут сводиться к отдельным её частям. Такие свойства проявляются тогда, когда система рассматривается как единое целое. Некоторые из этих свойств можно вывести из аналогичных свойств отдельных подсистем, но чаще они являются комплексным результатом взаимодействия подсистем и их невозможно оценить исходя из анализа отдельных системных компонентов.

Для решения проблемы сложности системы широко используется метод иерархической декомпозиции. Согласно данному подходу сложная система разбивается на более простые части, каждая из которых, в свою очередь, строится из частей меньшего размера, и т.д., до тех пор, пока самые небольшие части можно будет строить из имеющегося материала.

По отношению к проектированию ИС это означает, что её необходимо разделить (декомпозировать) на небольшие модули (подсистемы), каждый из которых можно разрабатывать независимо от других. Это позволяет при разработке модулей любого уровня иметь дело только с ним, а не со всеми остальными частями системы. Правильная декомпозиция является главным способом преодоления сложности разработки ИС. Понятие «правильная» по отношению к декомпозиции означает следующее:

– количество связей между отдельными модулями должно быть минимальным (принцип «слабой связанности» – Low Coupling);

– связность отдельных частей внутри каждого модуля должна быть максимальной (принцип «сильного сцепления» – High Cohesion).

Структура системы должна быть такой, чтобы все взаимодействия между её модулями укладывались в стандартные рамки, т.е.:

– каждый модуль должен инкапсулировать своё содержимое, т.е. скрывать его от других модулей;

– каждый модуль должен иметь чётко определённый интерфейс с другими модулями.

Инкапсуляция позволяет рассматривать структуру каждого модуля независимо от других. Интерфейсы позволяют строить систему более высокого уровня, рассматривая каждый модуль как единое целое и игнорируя его внутреннее устройство.

Модульность является важным качеством инженерных процессов и продуктов. Большинство промышленных процессов являются модульными и составлены из комплексов работ, которые комбинируются простыми способами для достижения требуемого результата.

Программные модули решают относительно небольшие функциональные задачи, и для их реализации необходимо 10 – 100 операторов языка программирования. Каждый модуль может использовать на входе около десятка типов переменных. Если для решения небольшой функциональной задачи требуется более 100 операторов, то целесообразней проводить декомпозицию задачи на несколько более простых модулей.

Функциональные группы программ (***компоненты***) формируются на базе нескольких программных модулей и решают достаточно сложные задачи. Как правило, на их реализацию используется до десятка тысяч строк программного кода. Соответственно возрастают число используемых типов переменных и разнообразие выходных данных.

Комплексы программ – программный продукт, создаваемый для решения сложных задач управления и обработки информации. В комплекс объединяется несколько функциональных компонентов для решения общей целевой задачи информационной системы. Размер программного комплекса может исчисляться сотнями модулей и сотнями тысяч операторов языка программирования.

Проектирование модулей включает в себя разработку локальных функций и подробного описания алгоритмов обработки данных; межмодульных интерфейсов; внутренних структур данных; структурных схем передачи управления; средств управления в исключительных ситуациях. С их помощью определяются функции: порядок следования отдельных шагов обработки, ситуации и типы данных, вызывающие изменения процесса обработки, а также повторно используемые функции программы [4].

На данный момент в технологии разработки информационных систем используется объектно-ориентированный подход к декомпозиции. При этом структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

1.3. ЖИЗНЕННЫЙ ЦИКЛ ИНФОРМАЦИОННЫХ СИСТЕМ

Одним из базовых понятий методологии проектирования ИС является понятие ***жизненного цикла*** (ЖЦ). ЖЦ ИС – это непрерывный процесс, начинающийся с момента принятия решения о необходимости создания ИС и заканчивающийся в момент полного её изъятия из эксплуатации.

ЖЦ ИС представлен на рис. 1.3. В данной схеме отражён тот факт, что, будучи однажды созданной, ИС входит в цикл, включающий её использование и модификацию.

ИС, в отличие от промышленных изделий, не подвержены износу. Поэтому они проходят фазу модификации из-за обнаружения ошибок, возникновения изменений в области их применения, требующих внесения соответствующих корректировок в программы, или из-за того, что прежняя модификация вызвала появление проблем в других частях ИС. Например, изменения в налоговом законодательстве могут потребовать внесения корректировок в программы подготовки платёжных ведомостей, связанные с расчётом удерживаемого налога и т.п. Однако внесённые изменения, в свою очередь, могут неблагоприятно повлиять на другие части программы, причём это может быть замечено не сразу.

По особенностям и свойствам ЖЦ ИС можно условно разделить два крупных класса – малые и большие.

К ***первому классу*** относятся небольшие программы, создаваемые одиночками или небольшими коллективами (3 – 5) специалистов, и имеют следующие свойства:

– создаются преимущественно для получения конкретных результатов автоматизации научных исследований или для анализа относительно простых процессов самими разработчиками ИС;

– не предназначены для массового тиражирования и распространения как программного продукта на рынке;

– не имеют конкретного независимого заказчика-потребителя, определяющего требования к ИС и их финансирование;

– не ограничиваются заказчиком допустимой стоимостью, трудоёмкостью и сроками их создания, требованиями заданного качества и документирования;

– не подлежат независимому тестированию, гарантированию качества и/или сертификации.

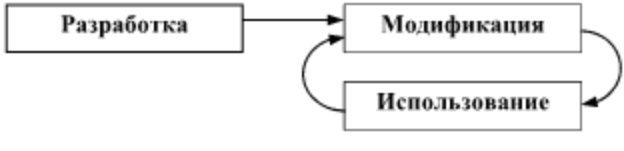

Рис. 1.3. Жизненный цикл информационной системы

Ко ***второму классу*** относятся крупномасштабные программные продукты с гарантированным качеством и отличаются следующими особенностями и свойствами их ЖЦ:

– большая размерность, высокая трудоёмкость и стоимость создания таких программных средств определяют необходимость тщательного анализа экономической эффективности всего их ЖЦ и возможной конкурентоспособности на рынке;

– от заказчика, финансирующего проект программного средства, разработчикам необходимо получать квалифицированные конкретные требования к функциям и характеристикам проекта и продукта, соответствующие выделенному финансированию и квалификации исполнителей проекта;

– для организации и координации деятельности специалистов-разработчиков при наличии единой, крупной целевой задачи, создания и совершенствования программного продукта необходимы квалифицированные менеджеры проектов;

– в проектах таких сложных программных средств с множеством различных функциональных компонентов участвуют специалисты разной квалификации и специализации, от которых требуется высокая ответственность за качество результатов деятельности каждого из них;

– от разработчиков проектов требуются гарантии высокого качества, надёжности функционирования и безопасности применения компонентов и поставляемых программных продуктов, в которые недопустимо прямое вмешательство заказчика и пользователей для изменений, не предусмотренных эксплуатационной документацией разработчиков;

– необходимо применять индустриальные, регламентированные стандартами процессы, этапы и документы, а также методы, методики и комплексы, средства автоматизации, технологии обеспечения ЖЦ ИС.

Основным нормативным документом, регламентирующим ЖЦ ИС, является международный стандарт ISO/IEC 12207 (его российский аналог ГОСТ Р ИСО/МЭК 12207–2010). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания и эксплуатации программного средства.

В соответствии с этими стандартами все процессы ЖЦ ИС делятся на три группы:

– основные процессы (приобретение, поставка, разработка, эксплуатация, сопровождение);

– вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем);

– организационные процессы (управление проектом, создание инфраструктуры проекта, усовершенствование, обучение).

Разработка включает в себя все работы по созданию ИС и её компонентов в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д.

Эксплуатация включает в себя работы по внедрению ИС, конфигурирование баз данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ИС, подготовку предложений по совершенствованию, развитию и модернизации программного средства [4].

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

Каждый процесс характеризуется определёнными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах.

2. ВИЗУАЛЬНОЕ МОДЕЛИРОВАНИЕ

2.1. ОСНОВНЫЕ ПОНЯТИЯ ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ

Под ***моделью ИС*** в общем случае понимается формализованное описание системы на определённом уровне абстракции. Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точки зрения различных людей с конкретными интересами, ролями или задачами.

Модель должна давать полное, точное и адекватное описание системы, имеющее конкретное назначение. Формальное определение модели:

M *есть модель системы* ***S****, если* ***M*** *может быть использована для получения ответов на вопросы относительно* ***S*** *с точностью* ***A****.*

Под термином «***моделирование***» понимается процесс создания формализованного описания системы в виде совокупности моделей.

Модели строятся для того, чтобы понять и осмыслить структуру и поведение будущей системы, облегчить управление процессом её разработки и уменьшить возможный риск, а также документировать принимаемые проектные решения.

Визуальное моделирование – это способ восприятия проблем с помощью зримых абстракций, воспроизводящих понятия и объекты реального мира. Моделирование способствует более полному усвоению требований, улучшению качества системы и повышению степени её управляемости.

С помощью модели можно упростить проблему, отбросив несущественные детали и сосредоточить внимание на главном. Способность к абстрагированию – это фундаментальное свойство человеческого интеллекта, помогающее справиться со сложностью. Чтобы создать ИС, разработчики должны абстрагировать её свойства с разных точек зрения, с помощью точных нотаций (систем обозначений) сконструировать адекватные модели, удостовериться, удовлетворяют ли они исходным требованиям, а затем реализовать модели на практике, постепенно пополняя систему новыми функциями.

К моделированию сложных систем приходится прибегать ввиду того, что мы не в состоянии постичь их единовременно во всей полноте. Способность человека к восприятию сложных вещей имеет свои естественные границы. Предварительное моделирование позволяет проектировщику увидеть общую картину взаимодействий компонентов проекта без необходимости анализа особых свойств каждого компонента.

Графические* (*визуальные*) *модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры системы. Под ***архитектурой*** понимается набор основных правил, определяющих организацию системы:

– совокупность структурных элементов системы и связей между ними;

– поведение элементов системы в процессе их взаимодействия;

– иерархию подсистем, объединяющих структурные элементы;

– архитектурный стиль (используемые методы и средства описания архитектуры, а также архитектурные образцы).

Архитектура ИС предусматривает различные представления, служащие разным целям:

– представлению функциональных возможностей системы;

– отображению логической организации системы;

– описанию физической структуры программных компонентов в среде реализации;

– отображению структуры потоков управления и аспектов параллельной работы;

– описанию физического размещения программных компонентов на базовой платформе.

Архитектурное представление – это упрощённое описание (абстракция) системы с конкретной точки зрения, охватывающее определённый круг интересов и опускающее объекты, несущественные с данной точки зрения. Архитектурные представления концентрируют внимание только на элементах, значимых с точки зрения архитектуры.

Архитектурно значимый элемент – это элемент, имеющий значительное влияние на структуру системы, её производительность, надёжность и возможность развития. Это элемент, важный для понимания системы. Например, в состав архитектурно значимых элементов объектно-ориентированной архитектуры входят основные классы предметной области, подсистемы и их интерфейсы, основные процессы и потоки управления [7].

Разработка модели архитектуры ИС промышленного характера на стадии, предшествующей её реализации или обновлению, в такой же мере необходима, как и наличие проекта для строительства большого здания.

2.2. СТРУКТУРНЫЕ МЕТОДЫ АНАЛИЗА И ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Методы структурного анализа и проектирования стремятся преодолеть сложность больших систем путём расчленения их на «чёрные ящики» и иерархической их организации. Выгода в использовании «чёрных

ящиков» заключается в том, что не требуется знать, как они работают, требуется знать только их входы и выходы, а также функции, которые они выполняют.

Таким образом, первым шагом упрощения сложной системы является её разбиение на «чёрные ящики», при этом такое разбиение должно удовлетворять следующим критериям:

– каждый «чёрный ящик» должен реализовывать единственную функцию системы;

– функция каждого «чёрного ящика» должна быть легко понимаема независимо от сложности её реализации;

– связь между «чёрными ящиками» должна вводиться только при наличии связи между соответствующими функциями системы;

– связи между «чёрными ящиками» должны быть простыми, насколько это возможно, для обеспечения независимости между ними.

Для понимания сложной системы недостаточно просто разбить её на отдельные части, ещё необходимо эти части организовать в иерархическую структуру.

Структурные методы широко используют визуальное моделирование, служащее для облегчения понимания сложных систем.

Для структурных методов характерно:

– разбиение системы на уровни абстракции с ограничением числа элементов на каждом из уровней (обычно от 3 до 7);

– ограниченный контекст, включающий лишь существенные на каждом уровне детали;

– использование строгих формальных правил записи;

– последовательное приближение к конечному результату.

В структурном анализе и проектировании используются различные модели, описывающие:

– функциональную структуру системы;

– последовательность выполняемых действий;

– передачу информации между функциональными процессами;

– отношения между данными.

Наиболее распространёнными моделями на сегодняшний день являются:

– функциональная модель SADT (Structured Analysis and Design Technique);

– модель моделирования процессов IDEF3;

– диаграммы потоков данных (DFD – Data Flow Diagrams);

– модель «сущность–связь» (ERM – Entity-Relationship Model).

2.2.1. МЕТОД ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ SADT

Метод SADT (Structured Analisys and Design Technique – технология структурного анализа и проектирования) разработан Дугласом Россом (SoftTech, Inc.) в 1969 г. для моделирования искусственных систем средней сложности. В 1993 году данный метод был утверждён в качестве федерального стандарта США IDEF0.

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает производимые объектом действия и связи между этими действиями.

При построении модели выполняются следующие правила:

– ограничение количества блоков на каждом уровне декомпозиции (3 – 6 блоков);

– связность диаграмм (номера блоков);

– уникальность меток и наименований (отсутствие повторяющихся имён);

– синтаксические правила для графики (блоков и дуг);

– разделение входов и управлений (правило определения роли данных);

– отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы являются главным компонентом модели, на них все функции и интерфейсы системы представлены как блоки и дуги соответственно. Место соединения дуги с блоком определяет тип интерфейса: управляющая информация входит в блок сверху, входная информация – с левой стороны блока, результаты (выход) – с правой стороны, а механизм – входит в блок снизу (рис. 2.1).

Одной из наиболее важных особенностей метода SADT является постепенное введение всё больших уровней детализации по мере создания диаграмм, отображающих модель [7].

Рис. 2.1. Функциональный блок и интерфейсные дуги

Построение SADT-модели заключается в выполнении следующих действий:

- сбор информации об объекте, определение его границ;
- определение цели и точки зрения модели;
- построение, обобщение и декомпозиция диаграмм;
- критическая оценка, рецензирование и комментирование.

Построение диаграмм начинается с представления всей системы в виде одного блока и дуг, изображающих интерфейсы с функциями вне системы. Затем этот блок детализируется на другой диаграмме с помощью нескольких блоков, соединённых дугами (рис. 2.2). Эти блоки определяют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых показана как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом в целях большей детализации.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые изображены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из диаграммы предыдущего уровня. На каждом шаге декомпозиции диаграмма предыдущего уровня называется родительской для более детальной диаграммы.

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из неё, потому что блок и диаграмма изображают одну и ту же часть системы (см. рис. 2.2).

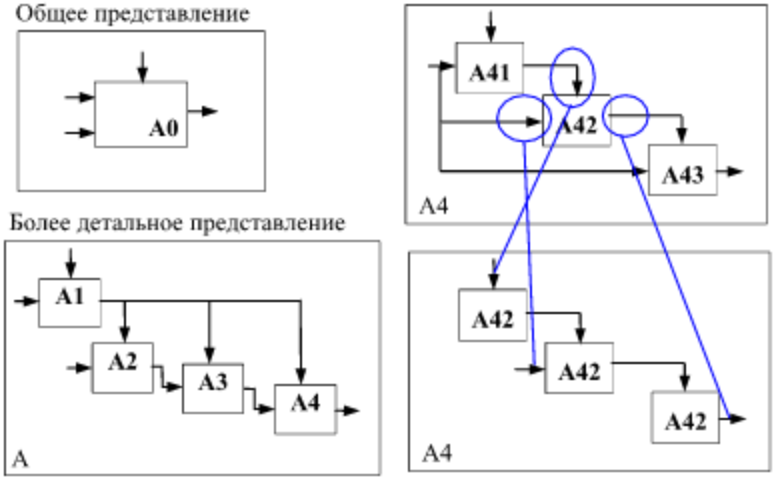

Рис. 2.2. Декомпозиция диаграмм SADT-модели

На SADT-диаграммах явно не указываются ни последовательность выполнения операций, ни время. Обратные связи, итерации, продолжающиеся процессы и перекрывающиеся по времени функции могут быть изображены с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д.

Механизмы показывают средства, с помощью которых осуществляется выполнение функций. Им может быть человек, компьютер или любое другое устройство, которое помогает выполнять данную функцию.

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть далее описан диаграммой нижнего уровня, которая в свою очередь может быть далее детализирована с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм.

Методология SADT может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы. Для уже существующих систем метод SADT может быть использован для анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.

2.2.2. МЕТОД МОДЕЛИРОВАНИЯ ПРОЦЕССОВ IDEF3

Метод моделирования IDEF3, являющийся частью семейства стандартов IDEF, был разработан в конце 1980-х гг. для закрытого проекта военно-воздушных сил США. Данный метод применяется, когда важно понять последовательность выполнения действий и взаимозависимости между ними. IDEF3 широко используется системными аналитиками как дополнение к методу функционального моделирования IDEF0. Основой модели IDEF3 служит сценарий процесса, который выделяет последовательность действий и подпроцессов анализируемой системы.

Как и в методе IDEF0, основной единицей модели IDEF3 является диаграмма. Другой важный компонент модели – действие («единица работы»). Диаграммы IDEF3 отображают действие в виде прямоугольника. Действия именуются с использованием глаголов или отглагольных существительных, каждому из действий присваивается уникальный идентификационный номер. В диаграммах IDEF3 номер действия обычно предваряется номером его родителя (рис. 2.3).

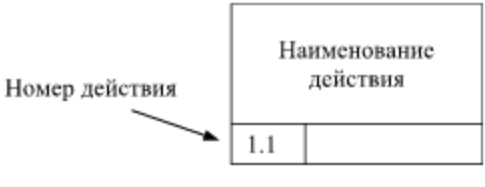

Рис. 2.3. Изображение и нумерация действия в диаграмме IDEF3

Существенные взаимоотношения между действиями изображаются с помощью различного типа связей.

Завершение одного действия может инициировать начало сразу нескольких других действий или, наоборот, определённое действие может требовать завершения нескольких других действий до начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используются для изображения ветвления процесса (рис. 2.4). Используют три типа соединений: И (&), ИСКЛЮЧАЮЩЕЕ ИЛИ (X) и ИЛИ (0).

На рисунке 2.4 после Действия_1 будут выполняться Действие_2, Действие_3 и Действие_4, после выполнения одного из них начнётся Действие_5.

В рассмотренном примере все действия выполнялись асинхронно, т.е. они не инициировались одновременно. Однако существуют случаи, когда время начала или окончания параллельно выполняемых действий должно быть одинаковым, т.е. действия должны выполняться синхронно. Для моделирования такого поведения системы используются различные виды синхронных соединений, которые обозначаются двумя двойными вертикальными линиями внутри прямоугольника (рис. 2.5).

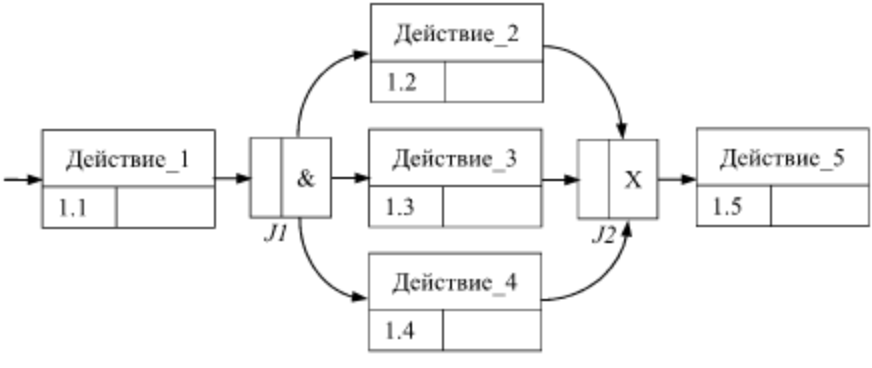

Рис. 2.4. Изображение типов соединений в диаграмме IDEF3
(*J1* – разворачивающие, *J2* – сворачивающие).

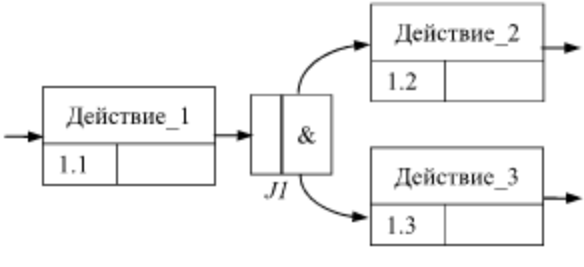

Рис. 2.5. Синхронное соединение

Начинающиеся синхронно действия не обязаны оканчиваться одновременно, также возможны ситуации синхронного окончания асинхронно начавшихся действий.

Все соединения на диаграммах должны быть парными, т.е. любое разворачивающее соединение имеет парное себе сворачивающее, при этом типы соединений не обязательно должны совпадать.

2.2.3. МОДЕЛИРОВАНИЕ ПОТОКОВ ДАННЫХ

Диаграммы потоков данных (Data Flow Diagrams – DFD) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Структуры потоков данных и определения их компонентов хранятся и анализируются в словаре данных. Каждая логическая функция (процесс) может быть детализирована с помощью DFD нижнего уровня; когда дальнейшая детализация перестаёт быть полезной, переходят к выражению логики функции с помощью спецификации процесса (мини-спецификации). Содержимое каждого хранилища также сохраняют в словаре данных, модель данных хранилища раскрывается с помощью ER-диаграмм.

В DFD не показываются процессы, которые управляют собственно потоком данных и не приводятся различия между допустимыми и недопустимыми путями.

Для построения DFD традиционно используются две различные нотации, соответствующие методам Йордона де Марко и Гейна–Сэрсона. Эти нотации незначительно отличаются друг от друга графическим изображением символов.

Модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от её ввода в систему до выдачи потребителю. Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам, которые в свою очередь преобразуют информацию, порождая новые потоки, и переносят её к другим процессам или подсистемам, накопителям данных или потребителям информации (внешним сущностям).

Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут уровень декомпозиции, на котором детализировать процессы далее не имеет смысла [7].

Основными компонентами диаграмм потоков данных являются:

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

Внешняя сущность (терминатор) представляет собой материальный объект или физическое лицо, представляющее собой источник или приёмник информации. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами анализируемой системы. Обозначается внешняя сущность квадратом (рис. 2.6), расположенным как бы над диаграммой и бросающим на неё тень. Её имя должно содержать существительное (например «Региональная ГНИ»). Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке данных.

Подсистема (система) на контекстной диаграмме изображается в виде прямоугольника, разделённого на три части. В верхнем поле указывается номер подсистемы (системы), который служит для её идентификации. В среднем поле вводится наименование подсистемы (системы) в виде предложения с подлежащим и соответствующими определениями и дополнениями. Нижнее поле служит для описания физической реализации.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определённым алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчётов; программа; аппаратно реализованное логическое устройство и т.д. На диаграмме процесс потоков данных изображается аналогично подсистеме (см. рис. 2.6).

Накопитель (хранилище) данных – это абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь. Физически реализоваться накопитель данных может в виде ящика в картотеке, таблицы, файла на

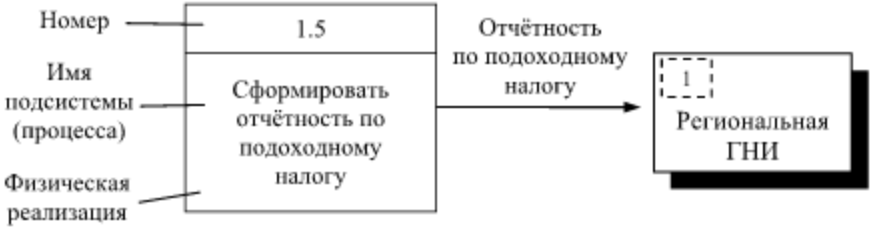

Рис. 2.6. Пример изображения потока данных

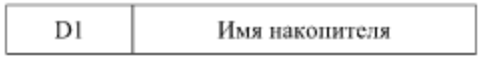

Рис. 2.7. Графическое изображение накопителя данных

магнитном носителе и т.д. На диаграмме потоков данных он идентифицируется буквой D и произвольным числом (рис. 2.7). Имя его выбирается из соображения наибольшей информативности для проектировщика.

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

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приёмнику. Реально поток данных может реализовываться разными способами, это может быть информация, передаваемая по кабелю между двумя устройствами; пересылаемые по почте письма; данные на флеш-накопителях, переносимые с одного компьютера на другой, и т.д.

На диаграмме поток данных изображается линией, оканчивающейся стрелкой, которая показывает направление потока. Каждый поток данных имеет имя, отражающее его содержание.

Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться в её источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним двунаправленным.

2.2.4. МОДЕЛИРОВАНИЕ ДАННЫХ

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

Наиболее распространённым средством моделирования данных (предметной области) является модель «сущность–связь» (ER).

С помощью ER-диаграмм определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). Во многих случаях информационная модель очень сложна и содержит множество объектов.

Сущность – это реальный или воображаемый объект, имеющий существенное значение для рассматриваемой предметной области.

Каждая сущность должна обладать уникальным именем, выраженным существительным в единственном числе (например, «Сотрудник»). В модели сущность изображается в виде прямоугольника с наименованием (рис. 2.8).

Конкретного представителя сущности называют ***экземпляром сущности***. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Например, экземпляром сущности «Сотрудник» может быть «Сотрудник Иванов».

Атрибут – это любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибуты представляют характеристики и свойства, ассоциированные с объектов.

Атрибут может быть обязательным (перед ними ставится знак «*») или необязательным (перед ними ставится 0). Обязательность означает, что атрибут не может принимать неопределённых значений.

Наименование атрибута должно быть выражено существительным в единственном числе (например, атрибутами сущности «Сотрудник» могут быть такие атрибуты, как «Табельный номер», «Фамилия», «Имя», «Отчество», «Должность», «Зарплата» и т.п.). Атрибуты изображаются в пределах прямоугольника, определяющего сущность (см. рис. 2.8).

Каждая сущность должна обладать уникальным идентификатором. ***Уникальным идентификатором*** называется неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из уникального идентификатора нарушает его уникальность.

Сущность может иметь несколько различных уникальных идентификаторов, они изображаются на диаграмме подчёркиванием или полужирным шрифтом (см. рис. 2.8).

Каждая сущность может обладать любым количеством связей с другими сущностями модели. ***Связь*** – это поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области.

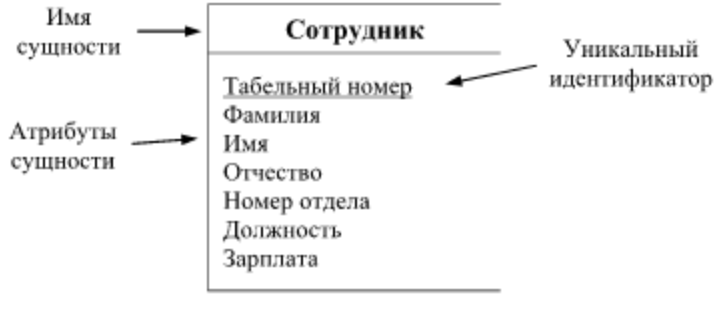

Рис. 2.8. Графическое представление сущности с атрибутами

Связям могут даваться имена, которые на диаграмме помещаются возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными.

Каждая связь имеет степень, которая определяет количество сущностей, участвующих в данной связи. Связь со степенью 2 называется бинарной, со степенью N – N-арной. Связь, в которой одна и та же сущность участвует в разных ролях, называется рекурсивной, или унарной.

Также каждая связь имеет две важные характеристики (на диаграмме обозначаются числами): класс принадлежности и мощность.

Класс принадлежности характеризует обязательность участия экземпляра сущности в связи. Он может принимать значение 0 (необязательное участие) или 1 (обязательное участие).

Мощностью связи называется максимальное число экземпляров сущности, которое может быть связано с одним экземпляром данной сущности. Она может быть равна 1, N (любое число) или может быть конкретным числом.

В зависимости от значения мощности связь может иметь один из следующих трёх типов:

- один-к-одному (обозначается 1:1);
- один-ко-многим (обозначается 1:n);
- многие-ко-многим (обозначается m:n).

Как и сущности, связи могут иметь атрибуты (рис. 2.9).

Обозначения на рис. 2.9 показывают, что каждый сотрудник обязательно работает в каком-либо отделе, а в некоторых отделах может и не быть сотрудников, каждый сотрудник может работать не более чем в одном отделе, а в каждом отделе может работать любое число сотрудников [12].

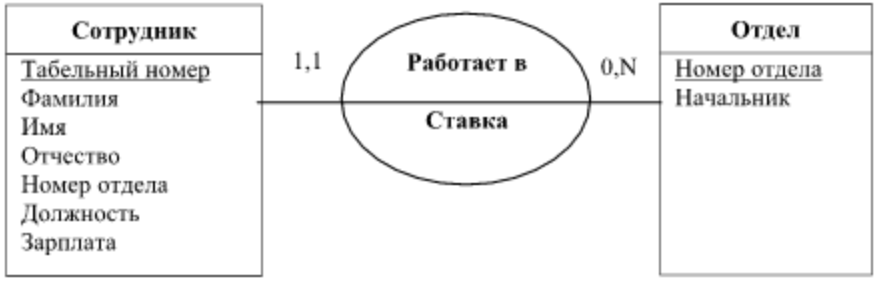

Рис. 2.9. Обозначение сущностей и связи

2.3. СРЕДСТВА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

2.3.1. РАЗВИТИЕ СРЕДСТВ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ СЛОЖНЫХ СИСТЕМ

Отдельные языки объектно-ориентированного моделирования начали появляться в середине 1970-х гг., когда различные исследователи и программисты предлагали свои подходы к объектно-ориентированному анализу и проектированию (Object-Oriented Analysis and Design – OOAD). В период 1989 – 1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50. При этом пользователи испытывали серьёзные затруднения при выборе языка моделирования, поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем.

К середине 1990-х гг. некоторые методы были существенно улучшены и приобрели самостоятельное значение при решении различных задач объектно-ориентированного анализа и проектирования. Наиболее известными в этот период становятся:

– метод Гради Буча (Grady Booch), получивший название объектно-ориентированного анализа и проектирования Буча;

– метод Джеймса Румбаха (James Rumbaugh), получивший название техники объектного моделирования (Object Modeling Technique – OMT);

– метод Айвара Джекобсона (Ivar Jacobson), получивший название объектно-ориентированной программной инженерии (Object-Oriented Software Engineering – OOSE).

Каждый из этих методов был ориентирован на поддержку отдельных этапов объектно-ориентированного анализа и проектирования. Например, метод OOSE содержал средства представления вариантов использования, которые имеют существенное значение на этапе анализа требований в процессе проектирования бизнес-приложений. Метод OMT наиболее подходил для анализа процессов обработки данных в информационных системах. Метод Буча нашёл широкое применение на этапах проектирования и разработки различных программных систем.

В 1994 году Гради Буч и Джеймс Румбах из компании Rational Software Corporation начали работу по унификации методов Буча и OMT. Несмотря на то что сами по себе эти методы были достаточно популярны, совместная работа была направлена на изучение всех известных объектно-ориентированных методов с целью объединения их достоинств. При этом Г. Буч и Дж. Румбах сосредоточили усилия на полной унификации результатов своей работы. Проект так называемого унифицированного метода (Unified Method) версии 0.8 был подготовлен и опубликован в ок-

тябре 1995 г. Осенью того же года к ним присоединился А. Джекобсон, главный технолог компании Objectory AB (Швеция), с целью интеграции своего метода OOSE с двумя предыдущими. Результатом деятельности этой группы специалистов стал язык UML (Unified Modeling Language, унифицированный язык моделирования).

В январе 1997 года был опубликован документ с описанием языка UML 1.0. Эта версия языка моделирования была достаточно хорошо определена, обеспечивала требуемую выразительность и мощность, предполагала решение широкого класса задач. В результате работы инициативной группы в составе OMG была предложена пересмотренная версия 1.1 языка UML. Основное внимание при разработке языка UML 1.1 было уделено достижению большей ясности семантики по сравнению с UML 1.0, а также учёту предложений новых партнёров. Эта версия языка была представлена на рассмотрение OMG, затем одобрена и принята в качестве стандарта OMG в ноябре 1997 г. Развитие языка продолжается, за последние годы увидели свет версии 1.3, 1.4, 1.5, а в 2003 г. вышел проект языка UML 2.0. Последняя на данный момент версия UML 2.4.1 опубликована в августе 2011 г.

Унифицированный язык моделирования стал промышленным стандартом для разработки и проектирования программного обеспечения. Благодаря UML разработчики получают мощный базис для успешного взаимодействия друг с другом и своими заказчиками, а также документирования разрабатываемого программного обеспечения.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределённых web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к её разработке и последующему развёртыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования [12].

2.3.2. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА UML

Язык UML позволяет записать детальный план, содержащий не только основные элементы ИС (системные функции и процессы), но и конкретные особенности её реализации (классы и иерархии классов). Основные понятия языка определяют средства структурной декомпозиции ИС. К ним относятся пакеты, подсистемы, модели и представления.

Пакеты в языке UML служат основным способом организации элементов модели ИС. Каждый пакет владеет всеми элементами, которые в него включены. Каждый элемент может принадлежать только одному пакету. Для визуализации пакетов существует специальная графическая нотация (рис. 2.10).

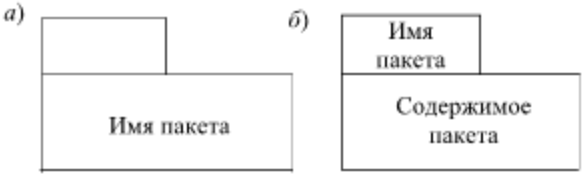

Рис. 2.10. Графическое изображение пакета

Внутри большого прямоугольника может записываться информация, относящаяся к данному пакету, а в случае её отсутствия – имя пакета, которое должно быть уникальным в пределах рассматриваемой модели (рис. 2.10, *а*). Если же такая информация имеется, то имя пакета указывается в верхнем поле (рис. 2.10, *б*).

Пакеты могут быть вложены друг в друга. Вложенный пакет называется подпакетом, поскольку все его элементы принадлежат более общему пакету. Вложенные пакеты могут образовывать пакетную иерархию. В UML отношение вложенности может быть изображено простым размещением одного пакета внутри другого (рис. 2.11, *а*).

Другим способом изображения отношения вложенности является представление дерева пакетов. В этом случае наиболее общий пакет (контейнер) изображается в верхней части рисунка, а его подпакеты уровнем ниже. Контейнер соединяется с подпакетами сплошной линией, на конце которой, примыкающем к контейнеру, изображается специальный символ, означающий, что подпакеты являются «собственностью» контейнера (рис. 2.11, *б*).

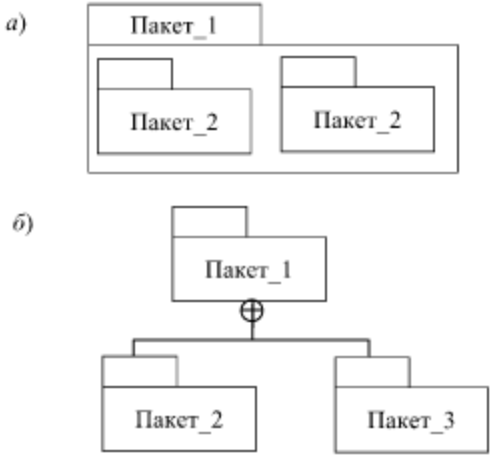

Рис. 2.11. Изображение вложенности пакетов друг в друга

Подсистема – вид пакета, описывающего определённую часть системы, выделенную в единое целое по реализационным или функциональным соображениям. Структуру подсистемы разделяют на две составляющие – декларативную и реализационную. Декларативная часть определяет внешнее поведение подсистемы и может включать в себя варианты использования и интерфейсы (спецификацию поведения), а реализационная описывает, каким образом реализуется декларативная часть.

Для графического представления подсистемы применяется обозначение, схожее с пакетом, но дополнительно разделённое на три секции (рис. 2.12). При этом в верхнем маленьком прямоугольнике находится символ, указывающий на подсистему.

Имя подсистемы записывается внутри большого прямоугольника, однако при наличии текста внутри него имя может быть записано рядом с обозначением «вилки». Операции подсистемы записываются в левой верхней секции, ниже указываются элементы спецификации, а справа от вертикальной линии – элементы реализации. При этом две последние секции помечаются соответствующими метками: «Элементы спецификации» и «Элементы реализации». Секция операций не помечается. Если в подсистеме отсутствуют те или иные секции, то они не отображаются на схеме.

Системой называют набор подсистем, организованных для достижения определённого результата и описываемых с помощью совокупности моделей. ***Модель*** является особым типом пакета, представляющим семантически замкнутую абстракцию системы. Она является полным и внутренне непротиворечивым упрощением реальной физической системы.

В UML для одной и той же физической системы могут быть определены различные модели, каждая из которых описывает систему с различных представлений. Каждая модель имеет свой собственный уровень абстракции и описывает ИС с определённого вида или представления.

Представление определяет способ видения системы, на основе которого создается её модель. Представление включает набор графических нотаций и их семантику. В UML ИС рассматривается с различных сторон с помощью моделей, многообразие которых отображается в форме диаграмм.

Рис. 2.12. Графическое изображение подсистемы

Объектное моделирование использует многообразие приёмов представления, отражающих процесс объектной декомпозиции с помощью логической (классы и объекты) и физической (компоненты и узлы) структур модели, а также их статические и поведенческие аспекты.

Диаграммы физического уровня предназначены для описания физической организации ИС. К ним относятся диаграммы компонентов и развёртывания. Диаграммы логического уровня разделяют на структурные, описывающие статические сущности (например, классы, объекты, компоненты) и динамические, описывающие поведенческие особенности ИС (например, прецеденты, состояния, деятельность).

2.3.3. КОМПОНЕНТЫ ЯЗЫКА UML

Язык UML включает набор графических элементов, используемых на диаграммах, и правила для объединения этих элементов.

Диаграммы используются для отображения различных представлений системы. Модель UML описывает, что должна делать система, но ничего не сообщает о том, как она будет реализована. Кратко рассмотрим наиболее общие диаграммы UML. Следует отметить, что возможно использовать гибридные диаграммы.

Диаграмма классов определяет типы классов системы и связи между ними. В языке UML класс представляется прямоугольником, разделённым на три области. Самая верхняя область содержит имя, в средней располагаются атрибуты, а в самой нижней – операции (рис. 2.13). Между классами могут быть связи, которые на диаграмме обозначаются линиями.

В UML имена классов зачастую включают несколько слов. При этом каждое слово в имени класса начинается с прописной буквы, а пробелы между словами отсутствуют (например, *ПерсональныйКомпьютер*). Имена атрибутов и операций строятся по тем же правилам, но первая буква в имени этих элементов обычно является строчной (например, *включитьПитание*). После имени операции следуют круглые скобки.

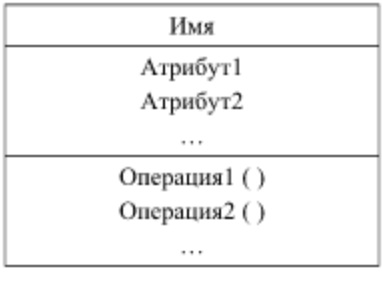

Рис. 2.13. Изображение класса в UML

Диаграммы классов представляют собой отправную точку процесса разработки ИС, а также помогают при анализе. Они позволяют аналитику общаться с клиентом в привычных для них терминах и стимулируют процесс выявления важных деталей в проблеме, которую требуется решить.

Диаграмма объектов. Объект представляет собой экземпляр класса – особую сущность, которая имеет заданные значения атрибутов и операций. Например, класс *ПерсональныйКомпьютер* может иметь атрибуты: *материнскаяПлата*, *процессор*, *оперативнаяПамять*, *видеокарта* и т.д. и операции: *включитьПитание*, *загрузитьОС*, *проверкаПароля* и т.д. Тогда объект *мойКомпютер*, например, может иметь следующие атрибуты: *материнскаяПлата – Gigabyte GA-M55plus*, *процессор – AMD Athlon 64 X2 3800 +* , *оперативнаяПамять – DDR2 DIMM Hynix 4 ГБ*, *видеокарта – ATI Radion 4880 1 ГБ*.

Объект в UML изображается прямоугольником, как и класс, но его имя подчёркнуто (рис. 2.14, *а*). Наименование экземпляра размещено слева от двоеточия, а наименование класса – с правой стороны. Имя объекта начинается со строчной буквы. Существуют также анонимные объекты (рис. 2.14, *б*), которые показывают, что объект принадлежит некоторому классу, но не имеет имени.

Диаграмма объектов фиксирует множество объектов и отношения между ними в определённый момент времени. При изображении диаграммы нужно помнить, что каждый объект представляет экземпляр соответствующего класса, а отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих отношений. При этом все связи изображаются сплошными линиями.

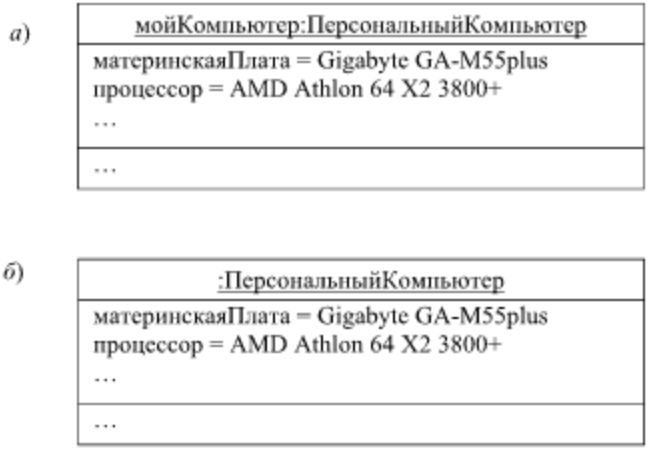

Рис. 2.14. Изображение объектов в UML

Диаграммы объектов применяются в основном для моделирования статического вида ИС с точки зрения проектирования или процессов. Также они могут включать пакеты и подсистемы, используемые для группирования элементов модели в более крупные блоки.

Диаграмма прецедентов. Прецедент – это описание поведения системы с точки зрения пользователя (рис. 2.15). Для разработчиков системы это полезный инструмент, предоставляющий надёжную методику формирования требований к ИС с точки зрения пользователя.

Небольшая простая фигурка, соответствующая пользователю персонального компьютера, называется исполнителем. Эллипс представляет прецедент. Исполнителем, инициирующим прецедент, может быть как человек, так и другая система.

Диаграмма состояний. В каждый момент времени объект находится в определённом состоянии, например, компьютер может находиться в состояниях: включённом, выключенном, загрузки операционной системы, ввода пароля, загрузки пользовательских параметров и т.п. (рис. 2.16). За время существования объект взаимодействует с другими объектами путём обмена сообщениями.

Рис. 2.15. Диаграмма прецедентов UML

Рис. 2.16. Диаграмма состояний UML

Диаграммы состояний позволяют моделировать жизненный цикл объекта с помощью состояний. Состояние определяет ситуацию, в течение которой объект удовлетворяет некоторому условию, выполняет определённые действия, либо ожидает наступления определённого события.

Символ вверху диаграммы представляет начальное состояние, а символ внизу соответствует конечному.

Переходы между состояниями не всегда линейны. Иногда переход диктуется некоторым условием.

Диаграмма последовательностей. Диаграммы классов и диаграммы объектов дают статическую информацию. Однако во время работы системы объекты взаимодействуют друг с другом, и это взаимодействие происходит во времени. Диаграмма последовательностей UML показывает временную динамику взаимодействия.

На диаграмме последовательности изображаются только объекты, непосредственно участвующие во взаимодействии, и не показываются возможные ассоциации с другими объектами (рис. 2.17). Имя объекта составное: указывается имя объекта и имя класса, которому он принадлежит, разделённые двоеточием. Запись подчёркивается, что является признаком объекта. Имя объекта может отсутствовать. В этом случае указывается только имя класса, а сам объект считается анонимным.

Диаграмма существует в двух измерениях: вертикальное измерение соответствует времени, горизонтальное представляет различные классы. Как правило, важна только последовательность событий, однако в приложениях реального времени ось времени может иметь реальную метрику.

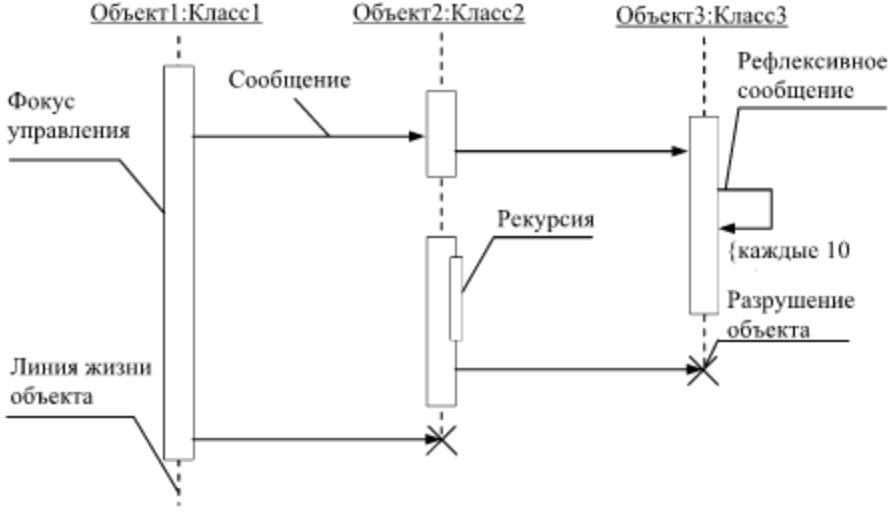

Рис. 2.17. Графические примитивы диаграммы последовательности

Таким образом, все объекты на диаграмме образуют порядок, определяемый степенью их активности при взаимодействии друг с другом. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (Объект1). Правее изображается другой объект, который непосредственно взаимодействует с первым.

Начальному моменту времени соответствует верхняя часть диаграммы. Взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения. Сообщения, расположенные на диаграмме выше, инициируются раньше тех, которые находятся ниже.

Линия жизни объекта изображается пунктирной вертикальной линией и служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, потенциально может участвовать во взаимодействиях.

Отдельные объекты, выполнив работу, могут быть уничтожены, чтобы освободить занимаемые ими системные ресурсы. Для таких объектов линия жизни обрывается в момент их уничтожения (Объект2 и Объект3).

Объекты могут находиться в активном состоянии, то есть выполнять определённые действия, или быть в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в UML используют фокус управления. Он изображается в виде вытянутого прямоугольника, верхняя сторона которого обозначает начало активности объекта, а нижняя – окончание. Прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всём её протяжении он является активным. Периоды активности объекта могут чередоваться с периодами ожидания. В этом случае у такого объекта имеется несколько фокусов управления (Объект2).

Объект может инициировать рекурсивное взаимодействие с самим собой. Для обозначения такого взаимодействия используется прямоугольник, присоединённый к правой стороне фокуса управления объектом (Объект2).

На диаграмме последовательности сообщения упорядочены по времени своего возникновения в моделируемой системе. Сообщения соединяют линии жизни или фокусы управления двух объектов. Неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами, т.е. за время передачи сообщения с соответствующими объектами не может произойти никаких изменений. В отдельных случаях объект может посылать сообщения самому себе, инициируя рефлексивные сообщения (Объект3). Такие сообщения изображаются прямоугольником со стрелкой, начало и конец которой совпадают.

В отдельных случаях выполнение действий на диаграмме последовательности может потребовать явной спецификации временных ограничений, накладываемых на интервал выполнения операций или передачу сообщений. В UML для записи временных ограничений используются фигурные скобки.

Следует заметить, что диаграммы последовательности позволяют отобразить не только взаимодействие отдельных объектов, но и описать работу всей информационной системы, вовлекая в рассмотрение внешних пользователей.

Диаграмма видов деятельности. Диаграмма видов деятельности UML очень похожа на блок схемы. В ней, как и в блок-схеме, изображается последовательность шагов (видов деятельности).

С помощью диаграммы видов деятельности описывается всё, что происходит во время какой-либо операции или процесса. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов.

Каждый вид деятельности изображается прямоугольником со скруглёнными углами (более узким и овальным, чем символ состояния). После завершения одного вида деятельности переход к следующему происходит автоматически. Переход от одного вида деятельности к другому изображается стрелкой. Как и на диаграмме состояний, на диаграмме видов деятельности имеется начальная точка, изображённая в виде закрашенного кружка, и конечная точка в виде «глазка» (рис. 2.18, *а*).

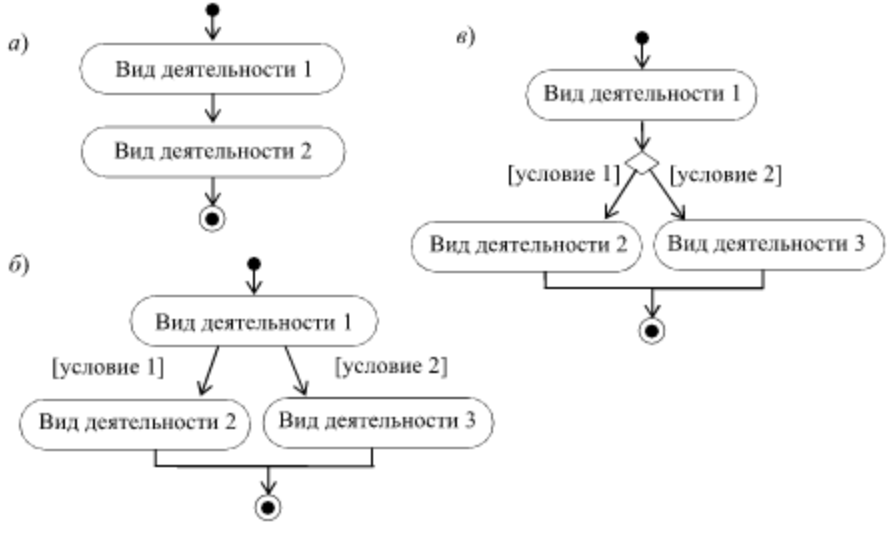

Рис. 2.18. Диаграмма видов деятельности UML

В последовательности видов деятельности практически всегда наступает этап принятия решения. Один набор условий выводит на один путь, следующий – на другой путь, причём эти пути исключают друг друга.

Точку принятия решений можно изобразить двумя способами. Первый способ – просто показать возможные пути развития событий после завершения этого вида деятельности (рис. 2.18, *б*). Во втором случае изображают переход к маленькому ромбику, который внешне похож на символ проверки (условия) в блок-схеме. Затем все возможные пути выходят из этого ромбика (рис. 2.18, *в*). В любом случае около пути указывается соответствующее условие в квадратных скобках.

При моделировании видов деятельности иногда очень важно показать два отдельных пути развития событий, которые идут параллельно друг другу, а затем сходятся. Для описания этого разбиения используется непрерывная жирная линия, перпендикулярная переходам, а пути изображаются выходящими из этой линии. Слияние путей представляется другой непрерывной жирной линией (рис. 2.19).

На каком-то этапе последовательности видов деятельности можно передать сигнал, при получении которого активизируется другой вид деятельности. Отправляемый сигнал изображают в виде выпуклого пятиугольника, а получаемый сигнал – вогнутого (рис. 2.20).

Диаграмма коммуникаций. Подобно диаграммам последовательностей, диаграммы коммуникации отражают взаимодействие объектов, но несколько иначе, чем на диаграммах последовательностей. На них изображаются объекты вместе с сообщениями, которыми они обмениваются.

Эти два типа диаграмм похожи. По сути, они семантически эквивалентны. Иными словами, в них представлена одна и та же информация. Диаграмму последовательностей можно преобразовать в диаграмму коммуникации, и наоборот.

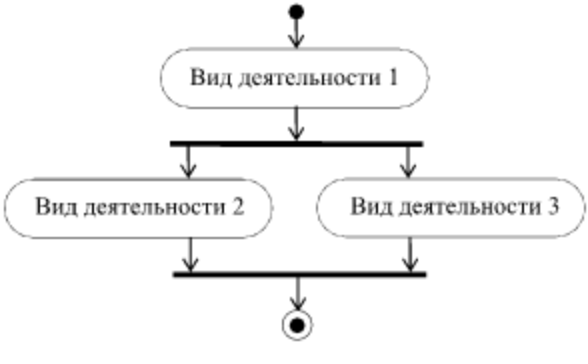

Рис. 2.19. Параллельные пути развития

Рис. 2.20. Отправка и получение сигнала

Полезно работать с обоими видами диаграмм. Диаграмма последовательностей акцентирует внимание на временном упорядочении взаимодействий. Диаграмма коммуникации ориентирована на состояние и общую организацию взаимодействующих объектов. Оба вида диаграмм отражают взаимодействие объектов, поэтому их относят к диаграммам взаимодействия.

Диаграмма коммуникации является расширением понятия объектных диаграмм. В дополнение к связям между объектами в диаграмму коммуникации включают сообщения, которые объекты передают друг другу. До сих пор этот момент не учитывался.

Чтобы изобразить обращение к объекту параллельно линии, которая соединяет объекты, рисуют стрелку. Направление стрелки указывает на объект, к которому обращаются. Метка возле стрелки служит для описания этого сообщения. В сообщении, как правило, даётся команда объекту-получателю выполнить одну из операций. Стрелки при этом выполняют ту же функцию, что и на диаграммах последовательностей.

Как уже упоминалось, любую диаграмму последовательностей можно преобразовать в диаграмму коммуникации, и наоборот. Следовательно, последовательную информацию можно описать и в диаграмме коммуникации. Для этого к метке сообщения нужно добавить номер, соответствующий номеру этого сообщения в последовательности сообщений. Номер от самого сообщения отделяется двоеточием (рис. 2.21).

Диаграмма компонентов. Программный компонент является модулем или частью системы. Поскольку он содержит реализацию одного или нескольких классов, то находится в компьютере, а не в воображении аналитиков. Компонент обеспечивает интерфейс с другими компонентами.

Рис. 2.21. Диаграмма коммуникации UML

В UML 1.x компонентами считались таблицы, файлы данных, исполняемые файлы, документы и динамически подключаемые библиотеки. В UML 2.0 все эти понятия называют общим именем – артефакт, под которым подразумевают фрагмент информации, используемый или генерируемый системой.

Работая с компонентами, приходится иметь дело с их интерфейсами. Интерфейс – это набор операций, который определяет поведение класса. Компонент обеспечивает интерфейсы доступа к нему для других программных компонентов. Для компонента, использующего доступ к интерфейсу другого компонента, этот интерфейс называется импортируемым.

В UML 1.x изображением компонента служил прямоугольник с двумя маленькими прямоугольниками на его левой стороне (рис. 2.22, *а*). В UML 2.0 появилось новое обозначение программного компонента – прямоугольник с ключевым словом *<<компонент>>* (рис. 2.22, *б*). Для соблюдения преемственности обозначений в UML 2.0 рекомендуется в верхнем правом углу нового обозначения добавлять пиктограмму компонента из UML 1.x (рис. 2.22, *в*).

Интерфейс можно представить двумя способами. В одном случае – это прямоугольник, содержащий информацию об интерфейсе (рис. 2.23, *а*). Он соединяется с компонентом пунктирной линией со стрелкой в виде полого треугольника. В другом случае – это маленький кружок, соединённый с компонентом сплошной линией (рис. 2.23, *б*).

Рис. 2.22. Обозначение компонентов в UML

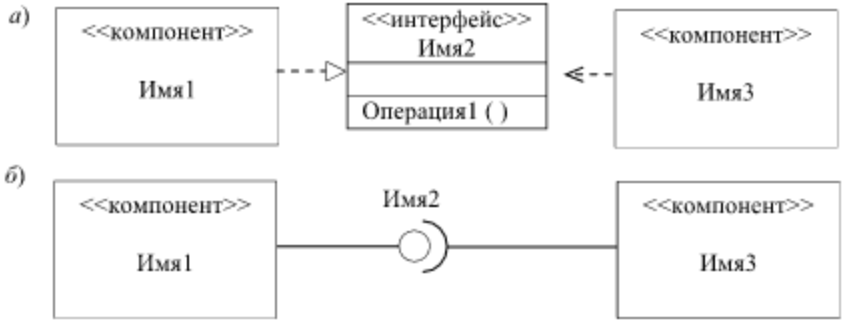

Рис. 2.23. Обозначение интерфейсов с реализацией и зависимостью

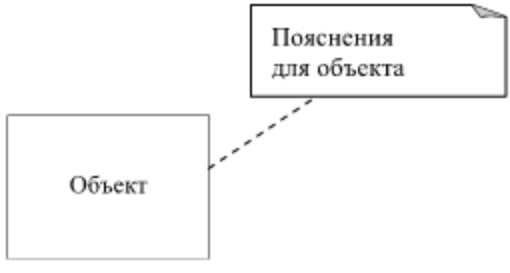

Рис. 2.24. Обозначение примечаний

В дополнение к реализации можно изобразить зависимость – связь между компонентом и интерфейсом, с помощью которого осуществляется доступ к другому компоненту (рис. 2.24).

Зависимость обозначается пунктирной линией со стрелкой. Реализацию и зависимость можно изобразить на одной диаграмме. Согласно использованной терминологии кружок обозначает экспортируемый интерфейс, а гнездо – импортируемый [12].

Примечания. Часто случается, что часть диаграммы недостаточно понятна. Неясно, почему он расположен именно там, или как с ней работать. В этом случае полезно использовать примечания. Они изображаются в виде прямоугольника с загнутым уголком. Внутри прямоугольника размещается пояснительный текст (см. рис. 2.24).

3. РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ

3.1. АРХИТЕКТУРА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

Приложения, работающие с базами данных, обычно состоят из интерфейса пользователя, компонентов, предоставляющих доступ к базе данных, и компонентов, соединяющих их друг с другом и с источником данных. Составляя эти компоненты в определённой последовательности, можно достаточно легко разработать приложение, взаимодействующее с базой данных. Общая схема приведена на рис. 3.1.

Как видно из рисунка, база данных представляет собой соединение пользовательского интерфейса и модуля данных. Модуль данных предназначен для хранения соответствующих компонентов. Одним из них является источник данных, предоставляющий данные другим частям приложения. Вторым компонентом является набор данных, содержащий в себе базу данных. Дополняет картину компонент, реализующий соединение с базой данных [8].

Общая структура приложения баз данных. Как было написано в вводной части, приложение обычно состоит из пользовательского интерфейса и средств доступа к данным. Интерфейс пользователя обычно разрабатывается из стандартных компонентов и располагается на базовой форме приложения.

Приложение может содержать произвольное число форм и использовать любую парадигму работы с множественными документами (MDI или SDI). Обычно одна форма отвечает за выполнение группы однородных операций, объединённых общим названием.

Визуальные компоненты отображения данных расположены на вкладке Data Controls. Они по большей части являются стандартными компонентами отображения данных с небольшими изменениями и дополнениями. В основе любого приложения баз данных лежат *наборы данных*, представляющие собой массивы записей, полученные из баз данных. Наборы данных хранятся в специальных компонентах, построенных на базе класса TDataSet. Для обеспечения связи набора данных с визуальными

Рис. 3.1. Обобщённая схема БД

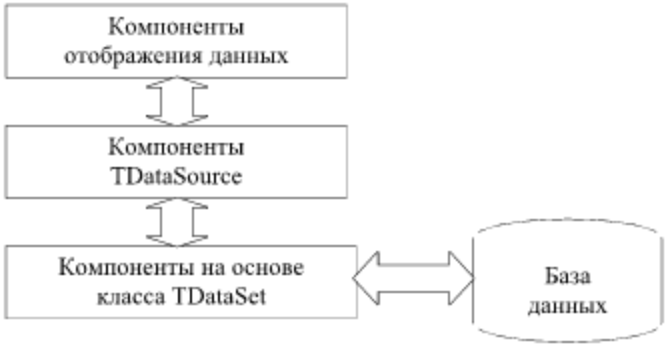

Рис. 3.2. Механизм доступа к данным

компонентами отображения данных предназначен специальный компонент – источник данных, реализуемый классом TDataSource. Компонент TDataSource обеспечивает передачу данных в визуальные компоненты, отслеживает синхронное изменение их состояния, если изменяется состояние связанного с ними набора данных, передачу изменённых данных обратно в набор. Этот компонент располагается на вкладке Data Access,

Базовый механизм доступа к данным основывается на трёх «китах»:

– о компонентах-потомках класса TDataSet (наборах данных);

– о компонентах TDataSource:

– о визуальных компонентах отображения данных, формирующих интерфейс пользователя.

Эта схема показана на рис. 3.2.

При открытии набора данных в него автоматически передаются записи из таблицы базы данных, курсор устанавливается на первую запись. При перемещении по набору данных значения записей автоматически обновляются (или не обновляются, в зависимости от расположения и типа курсора базы данных).

3.1.1. МОДУЛЬ ДАННЫХ

Модуль данных (Data Module) представляет собой невизуальный контейнер, в котором размещаются компоненты доступа к данным. В модуле данных можно размещать только невизуальные компоненты. Модуль данных доступен программисту на этапе разработки в виде формы, в которую он может положить компоненты и настроить их свойства. Модуль данных отличается от обычной формы, так как берёт своё начало от класса TComponent.

Для создания модуля данных можно воспользоваться репозиторием объектов, который активируется при выполнении команды меню New ► Data Module. Вкладка Diagram открывает окно формирования моделей

данных (диаграмм, схем). Модели данных позволяют наглядно отобразить связи, существующие между наборами данных. Это особенно удобно, если приходится работать с большим количеством таблиц.

На рисунке 3.3 показано «дерево объектов». Оно позволяет отображать в списке объекты, размещённые в модуле данных. Перетаскивая объекты из «дерева объектов» в рабочую область Diagram, можно легко настраивать их свойства, буквально соединяя объекты линиями связи. На рисунке 3.4 показана модель некой базы данных. Для того чтобы создать подобную модель, потребуется совсем немного времени. С помощью кнопки Property connector легко связать таблицы с компонентом-провайдером, а кнопка Master\Detail connector позволяет установить связь «один-ко-многим» в диалоговом окне Field Link Designer.

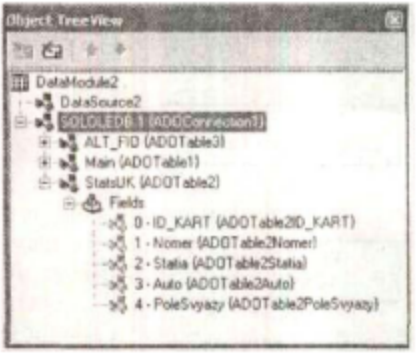

Рис. 3.3. Дерево объектов

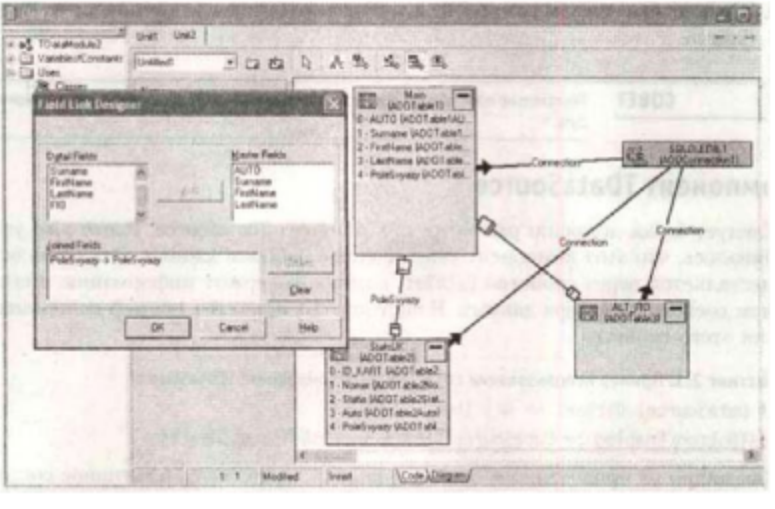

Рис. 3.4. Модель данных БД

Использование модуля данных при разработке приложения даёт программисту массу преимуществ. Например, при изменении какого-либо свойства объекта в модуле данных изменения автоматически отобразятся во всех связанных с ним компонентах. Так же немаловажным преимуществом является то, что все компоненты, обеспечивающие доступ к базе данных, собраны в одном месте. Это облегчает восприятие базы в целом.

3.1.2. ПОДКЛЮЧЕНИЕ ДАННЫХ

Компонент набора данных является основой приложения, взаимодействующего с базами данных. Он содержит набор данных из выбранных таблиц и позволяет эффективно управлять им. В процессе работы данный компонент использует функции соответствующей технологии через совокупность интерфейсов (методов, предоставляемых данной технологией). Рассмотрим основные шаги, которые потребуются для подключения набора данных и получения информации из него:

1. Поместить компонент в модуль набора данных и связать его с базой данных.
2. Подключить к компоненту таблицу, используя свойство TableName.
3. Активизировать связь, установить для компонента значение свойства Active как True.
4. Разместить на форме компонент TDataSource. Он обеспечивает взаимодействие визуальных компонентов отображения данных с набором данных.
5. Связать набор данных и компонент TDataSource.
6. Связать визуальные компоненты отображения данных с компонентом TDataSource.

Совет: *желательно присваивать объектам простые и осмысленные имена, отражающие их суть.*

Компонент TDataSource

Следует более детально рассмотреть компонент TDataSource. Ранее уже упоминалось, что этот компонент связывается с набором данных. Эта связь осуществляется через свойство DataSet, которое содержит информацию о текущем состоянии набора данных. В листинге 3.1 приведён пример использования этого свойства.

Листинг 3.1. Пример использования свойства State компонента TDataSource

```
if DataSourcel.Dataset <> nil then
PostButton.Enabled : = DataSourcel.State in [dsEdit. dslnsert]:
```

Как видно из приведённого примера, кнопка переводится в активное состояние в том случае, если связанный набор данных в текущий момент доступен для редактирования. У этого компонента существует набор свойств и методов, которые облегчают работу с ним.

Свойство AutoEdit автоматически переводит набор данных в состояние редактирования, если имеет значение True, когда связанный элемент ввода получает фокус.

3.1.3. СТАНДАРТНЫЕ КОМПОНЕНТЫ, СВЯЗЫВАЕМЫЕ С НАБОРОМ ДАННЫХ

Наиболее распространённым компонентом, без которого не обходится практически ни одна СУБД, является компонент, реализующий табличное представление данных TDBGrid.

Компонент TDBGrid используется для отображения содержимого набора данных в табличном виде, когда строки соответствуют записям набора данных, а столбцы – полям записи.

Объект DBGrid связывается с источником данных через свойство DataSource, которое в свою очередь ссылается на набор данных. В процессе выполнения приложения можно менять источники данных, используя одну таблицу для отображения данных от нескольких источников.

Для определения состава столбцов можно использовать редактор столбцов, реализованный диалоговым окном Column Editor. Он показан на рис. 3.5.

Если редактор столбцов не использовался, то используются поля, объявленные с помощью редактора полей набора данных. При этом все свойства столбцов и порядок их следования наследуются из редактора набора данных. В случае. когда производится динамическое формирование объектов TField, порядок следования полей и их характеристики соответствуют тем, что были заданы при определении структуры записи таблицы базы данных при создании.

Редактор столбцов вызывается двойным щелчком на компоненте. В диалоговом окне редактора можно установить различные свойства объектов-столбцов, такие как цвет фона, шрифт заголовка, основного поля таблицы и выборочного столбца.

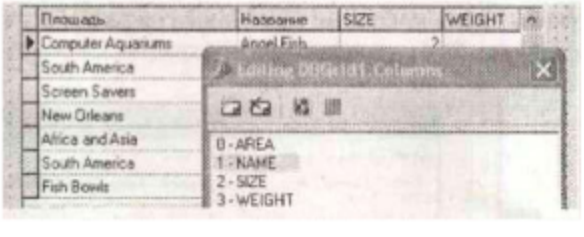

Рис. 3.5. Редактор столбцов

Обратиться к конкретному столбцу компонента TDBGrid можно как к элементу индексированного массива TDBGrid. Columns.Items[i].

Также для отображения и редактирования данных используются элементы TDBEdit. Они представляют собой модифицированные компоненты TEdit. Связь с источником данных осуществляется с помощью свойства DataSource. Поле, с которым связывается данный компонент, указывается в свойстве DataField.

Компонент TDBMemo представляет собой многострочный редактор. Обычно связывается с memo-полями. С его помощью часто сохраняют обширные текстовые массивы в соответствующих полях.

Компонент DBComboBox представляет собой список, связываемый с определённым полем набора данных. Значения списка хранятся в свойстве Items. Значения в список можно добавлять, удалять из него, сохранять и загружать с помощью методов Delete, Insert, LoadFromFile и SaveToFile.

Параметр Index указывает на позицию строки в списке, а параметр FileName содержит полный путь к файлу, откуда загружались значения списка. Данный компонент удобно использовать при организации пользовательских словарей.

Компонент TDBImage предназначен для отображения графической информации, хранящейся в связанном с ним BLOB-поле (binary large object). Связывается этот объект с набором данных через те же свойства, что и остальные компоненты отображения данных. Изображение, получаемое из набора данных, хранится в свойстве Picture.

Компонент TDBRichEdit представляет собой многострочный редактор с возможностью отображать форматированный текст. Данный компонент идеально подходит для отображения различных примечаний или хранения отчётов.

Компонент TDBChart служит для построения графиков на основе данных, получаемых из набора данных. Иногда при помощи всего нескольких щелчков мышью можно получать очень красивые объёмные; графики, отражающие различные зависимости.

Для того чтобы построить график, надо выполнить команду контекстного меню Edit Chart. В результате будет активировано диалоговое окно Editing DBChart, показанное на рис. 3.6. Для создания новой серии данных необходимо нажать кнопку Add. В результате будет активировано следующее диалоговое окно TeeChart Gallery, в котором можно будет выбрать тип графика. Перейдя на вкладку Series, можно получить доступ к настройкам данного графика. На этой странице расположены вкладки Format, General, Mark и Data Source. На вкладке Format можно задать различные параметры отображения графика. Вкладка General позволяет задать несколько свойств общего характера. На вкладке Mark можно определить режим подписи графика – легенду, размещение пояснений к нему и другие параметры. Вкладка Data Source, показанная на рис. 3.6, позволяет задать источник данных и определить поля, по которым будет строиться график.

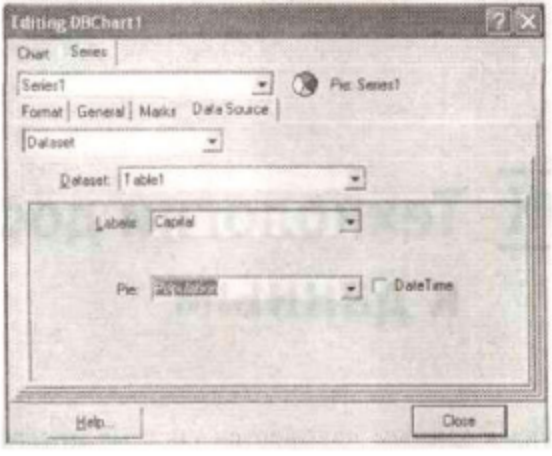

Рис. 3.6. Выбор источника данных

Из раскрывающегося списка можно выбрать тип источника данных строящегося графика. В данном примере стоит остановиться на значении Dataset. Далее из списка Dataset нужно выбрать конкретный набор данных. В списке Labels необходимо выбрать поле, значения которого будут подписываться под значениями графика. Поле, на основании которого будет построен график, выбирается в списке Pie из доступных полей.

3.2. ТЕХНОЛОГИИ ДОСТУПА К ДАННЫМ

Технологией доступа к данным называется система интерфейсов, обеспечивающая взаимодействие между приложением и базой данных. Во многих системах управления базами данных имеются библиотеки, содержащие интерфейсы прикладного программирования (application programming interface – API), представляющие собой функции, с помощью которых можно выполнять с данными те или иные действия.

Для того чтобы наиболее полно использовать возможности того или иного сервера баз данных, необходимо работать с ним напрямую, через API. Однако это означает полную зависимость приложения от того или иного сервера и сложность перехода на другую платформу, так как будет необходимо переписывать большое количество кода [8].

Этот вопрос призваны решить различные технологии доступа к данным. Они являются прослойкой между API конкретного сервера и приложением пользователя, предоставляя программисту простой унифицированный механизм работы с данными. На сегодняшний день существует множество технологий доступа к данным, таких как BDE, OLE, ODBC, ADO, и до сих пор разрабатываются новые, более надёжные, удобные в работе и более быстродействующие технологии.

3.2.1. ТЕХНОЛОГИЯ BDE

Фирма Borland разработала собственную технологию доступа к данным SQL Links, имеющую возможность взаимодействовать с ODBC через специальные «интерфейсы-мосты». Технология BDE является набором динамических библиотек, которые предоставляют интерфейсы, позволяющие передавать запросы на получение или модификацию данных из приложения в нужную базу данных и получать результат обработки. В процессе работы библиотеки используют вспомогательные файлы языковой поддержки и информацию о настройках среды.

Следует отметить, что Borland перестала развивать технологию BDE с 2002 г. Разрабатывать проекты с этой технологией возможно, но не рекомендуется.

3.2.2. СТАНДАРТ ODBC

ODBC – это стандарт, описывающий систему интерфейсов, с помощью которых прикладные программы могут обращаться к базам данных и обрабатывать их независимым от СУБД способом. ODBC предоставляет интерфейсы для доступа к реляционным базам данных и базам данных с табличной организацией. Широкое распространение стандарт получил благодаря поддержке Майкрософт.

На рисунке 3.7 схематично изображена архитектура ODBC.

Как видно из рисунка, приложение обращается к диспетчеру драйверов, а диспетчер в свою очередь обращается к источнику данных и производит с ними какие-либо действия. Источник данных – это база данных, приложение, операционная система или даже некая аппаратная платформа. Приложение инициирует запросы на установление соединения с источником данных на выполнение каких-либо действий с базой данных.

Рис. 3.7. Архитектура ODBC

Стандарт ODBC предоставляет разработчику набор интерфейсов для выполнения каждого из этих запросов и регламентирует стандартные коды ошибок, которые будут возвращены приложению в случае неудачного выполнения запроса.

Диспетчер драйверов (driver manager) служит связующим звеном между приложениями и драйверами СУБД. Когда приложение инициирует соединение с базой данных, диспетчер определяет тип СУБД и загружает соответствующий драйвер в память (или использует ранее загруженный). Диспетчер драйверов обрабатывает запросы на инициализацию соединения и контролирует формат запросов и порядок их поступления от приложения. Диспетчер драйвера является частью Windows.

Драйвер (driver) обрабатывает запросы, поступающие от приложения, преобразует их в набор команд API СУБД и, таким образом, производит какие-либо действия с базой данных. Драйвер отвечает за то, чтобы стандартные команды ODBC выполнялись корректно. В некоторых случаях источник данных не поддерживает некоторые функции, таким образом, их приходится выполнять драйверу. Если источник имеет полную поддержку SQL, то драйвер всего лишь передаёт запрос на обработку и получает результаты. На драйвере также лежит функция приведения кодов ошибок, поступающих от источника, к стандартным, определённым в ODBC.

Определяют два типа драйверов – одноуровневые и многоуровневые. Одноуровневые обрабатывают вызовы ODBC и операторы SQL. Многоуровневые обрабатывают только вызовы ODBC, оставляя СУБД осуществлять обработку SQL-запросов.

Уровень соответствия ODBC (ODBC conformance level) описывает то, какие возможности и функции доступны через API (интерфейс прикладных программ) драйвера. API драйвера содержит набор функций, которые может вызывать приложение для обращения к интерфейсам источника данных. Различают несколько уровней соответствия ODBC, обеспечивающих разные наборы возможностей:

- Базовый уровень (Core API);
- Первый уровень (Level 1 API);
- Второй уровень (Level 2 API).

Если какой-либо драйвер не поддерживает требуемый приложению уровень, то программист может реализовать нужные функции самостоятельно, обрабатывая данные, получаемые из источника соответствующим образом. Уровни соответствия SQL определяют, какие SQL-операторы, выражения и типы данных доступны драйверу на данном уровне. Стандартом определены три уровня соответствия SQL:

- Минимальный синтаксис (Minimum SQL grammar):
  - CREATE TABLE, DROP TABLE;
  - оператор SELECT (без вложенных подзапросов);

- INSERT, UPDATE, DELETE;
- простые выражения (сравнения, арифметические операции);
- типы данных CHAR, VARCHAR, LONGCHAR.

○ Базовый синтаксис (Core SQL grammar):
- минимальный синтаксис;
- ALTER TABLE, TREATE INDEX, DROP INDEX;
- CREATE VIEW, DROP VIEW;
- GRANT, REVOKE;
- полный синтаксис оператора SELECT;
- встроенные функции SUM, COUNT, MAX, MIN, AVG.

○ Расширенный синтаксис (Extended SQL grammar):
- базовый синтаксис;
- UPDATE и DELETE с использованием позиции курсора;
- скалярные функции SUBSTRING и ABS;
- переменные даты, времени и временная метка;
- пакетная обработка операторов SQL;
- хранимые процедуры.

Приложение может вызвать драйвер и определить, какой уровень соответствия SQL он поддерживает.

Драйверы ODBC могут поддерживать многопоточность (multithreaded driver), т.е. с одним драйвером могут одновременно работать несколько приложений в синхронном режиме, внося какие-либо изменения в источник данных. В случае если драйвер не является многопоточным, он работает только в асинхронном режиме.

Определение имён источников данных

Для того чтобы сделать источник данных доступным для приложения, необходимо указать соответствующие имена источников данных DSN (Data Source Name). Существует три типа пользовательских источников данных:

○ User DSN (пользовательский). Приложение будет работать только в том случае, если в системе зарегистрирован (в данный момент) именно тот пользователь, который настроил данный источник;

○ System DSN (системный). Драйвер этого типа виден всем приложениям, работающим на данной машине;

○ File DSN (файловый). Драйвер подобного типа может быть сделан общим ресурсом (совместное пользование) для всех пользователей, у которых установлен такой же драйвер и у которых есть соответствующие права доступа.

Для того чтобы создать источник данных, сначала нужно запустить утилиту ODBC Data Source Administrator при помощи команды меню Administrative Tools ► Data Sources (рис. 3.8).

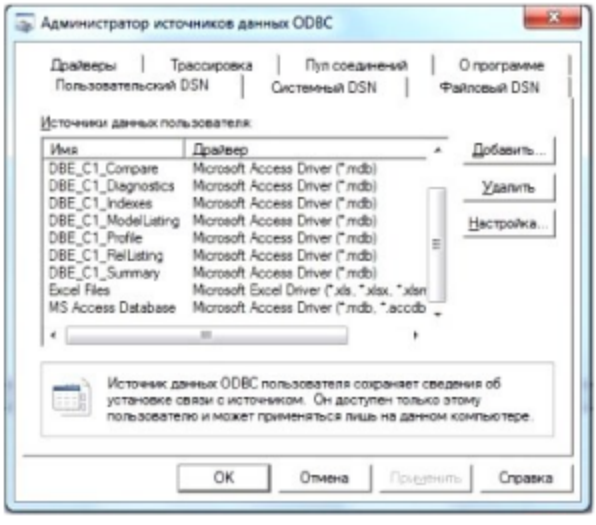

Рис. 3.8. Окно ODBC Data Source Administrator

Для примера можно создать соединение с файлом в формате Access. Нужно нажать кнопку Add, в результате чего будет активировано диалоговое окно, показанное на рис. 3.9. В нём надо выбрать драйвер Driver do Microsoft Access (*.mdb).

Выбор подтверждается нажатием кнопки Готово. Будет активировано диалоговое окно, в котором нужно задать имя источника, ввести путь к базе и настроить некоторые параметры, как показано на рис. 3.10.

Теперь необходимо указать используемую базу данных. Для этого нужно нажать кнопку Select. Будет отображено окно Select Database, в котором указывается путь к базе данных. С помощью кнопки Create можно создать собственную базу. Кнопка Repair позволяет восстановить повреждённый файл. Для того чтобы очистить файл от «мусора», используется кнопка Compact.

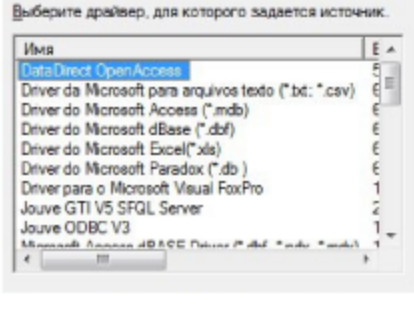

Рис. 3.9. Окно создания нового источника данных

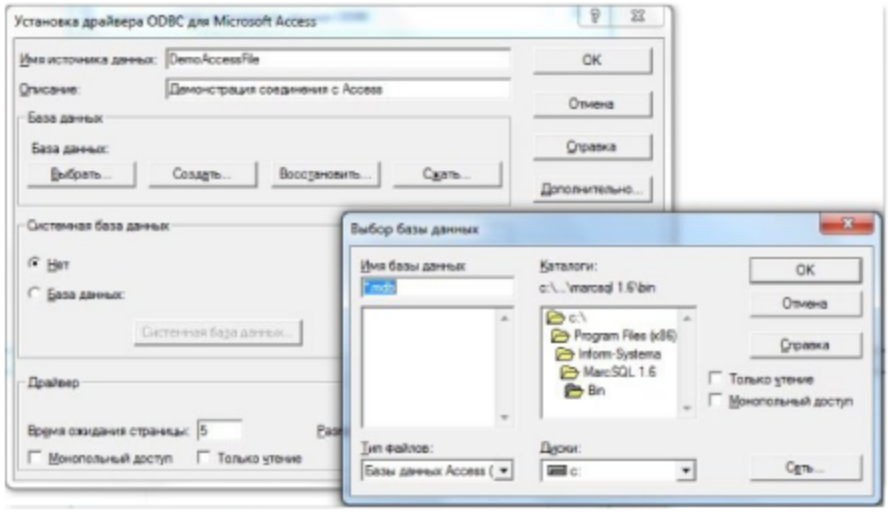

Рис. 3.10. Настройка источника данных

В поле Page Timeout задаётся интервал времени, в течение которого неиспользуемая страница данных будет содержаться в буфере. В свойстве Buffer Size указывается размер буфера, используя который, драйвер передаёт данные от приложения на диск и обратно. Размер буфера должен быть кратен 256. В окне Set Advanced Options можно дополнительно настроить соединение, задать логин и пароль, которые будут передаваться в базу по умолчанию, и иные специфичные настройки.

Группа органов управления Drivers позволяет получить информацию об установленных ODBC-драйверах. В этом разделе можно получить информацию о версии драйвера, но нельзя изменить его настройки. Раздел Tracing используется для настройки параметров отслеживания вызовов функций ODBC-драйверов. Менеджер драйвера может отслеживать несколько соединений или только одно. Для установки слежения за конкретным драйвером его надо выбрать в диалоговом окне Select DLL.

Вкладка Connection Pooling используется для того, чтобы указать, может или нет драйвер ODBC повторно создавать метки соединения для сервера базы данных. Когда соединение с источником данных освободится, менеджер драйверов поместит его в пул на определённый период. Когда поступит запрос на соединение с источником данных, менеджер драйверов назначит его из пула, не затрачивая ресурсов на создание. Таким образом, общая производительность возрастёт. Для указания необходимости помещения драйвера в пул установите значение свойства Connection Pooling ► PerfMon в True.

3.2.3. ТЕХНОЛОГИЯ OLE DB

OLE DB представляет собой низкоуровневый интерфейс, обеспечивающий доступ к различным источникам данных – реляционным и не реляционным, содержащим текст, графические и географические данные, к файлам электронной почты, содержимому файловых систем и создаваемым пользователями бизнес-объектам. OLE DB определяет набор интерфейсов компонентной объектной модели (Component model object, COM), включающих в себя службы различных систем управления базами данных для обеспечения универсального доступа к данным. С помощью этих интерфейсов программисты могут создавать дополнительные сервисы баз данных.

OLE-объекты являются COM-объектами и поддерживают все требуемые для таких объектов интерфейсы. По сути, OLE DB разбивает всю совокупность возможностей и функций СУБД на отдельные фрагменты – COM-объекты, выполняющие определённые функции. Некоторые объекты отвечают за выполнение запросов, другие – за обновление данных и т.д. Это свойство OLE DB позволяет преодолеть огромный недостаток ODBC. Чтобы драйвер ODBC считался законченным, производитель должен обеспечить в нём вызов всех интерфейсов, предоставляемых СУБД. В случае OLE DB производитель может выпустить драйвер с частично реализованной функциональностью, а позже добавлять в него новые интерфейсы.

Основные конструкции OLE DB

На верхнем уровне абстракции можно выделить три главных компонента:

- потребители;
- провайдеры данных;
- провайдеры сервисов.

Любое приложение, использующее интерфейсы OLE DB, является потребителем. В роли потребителя могут выступать прикладная программа базы данных, средство разработки, средство создания отчётов или же объектная модель ActiveX Data Object (ADO).

Провайдер данных (Data provider) представляет собой объект, «владеющий» данными, т.е. связанными с ним. Он находится между потребителем и массивом данных. В OLE DB все провайдеры представляют данные в виде виртуальных таблиц. Провайдер выполняет несколько задач:

- Принятие запросов на доступ к данным, поступающих от потребителя.
- Выполнение выборки и обновления данных.
- Передача результатов потребителю (данные или коды ошибок).

Провайдер сервисов (Service provider) реализует расширенные возможности, которые не поддерживаются обычными провайдерами данных,

Рис. 3.11. Компоненты OLE DB

и сам не «владеет» данными. Этот провайдер, к примеру, обеспечивает сортировку, фильтрацию, управление транзакциями, обработку SQL-запросов и т.д. Провайдер сервисов может работать с массивами данных напрямую либо через провайдер данных. На рисунке 3.11 представлена схема, отражающая суть сказанного.

Как видно из рисунка, потребитель может получать данные как непосредственно от провайдера данных, так и используя службы, которые предоставляет провайдер сервисов.

Ядро объектной модели OLE DB составляют четыре объекта:

- DataSource;
- Session;
- Command;
- Rowset.

Как видно из рисунка, объект DataSource, вызвав интерфейс IDBCreateSession, создаёт новую сессию. В свою очередь объект Session вызывает интерфейс IDBCreateCommand и создаёт объект Command, содержащий определённую команду. Далее объект Command вызывает интерфейс ICommand и создаёт объект Rowset, который содержит полученные результаты.

Объект DataSource используется потребителем данных OLE DB для связи с провайдером данных. Этот объект может быть создан различными способами: порождён от объекта Enumerator и связан с моникером (объект операционной системы Windows представляет собой именованную область памяти, котроую могут разделять несколько процессов) файла или с моникером источника данных. Также объект может быть создан с помощью вызова COM-функцин CoCreateInstance. Каждый провайдер OLE DB присваивает свой собственный идентификатор класса объекту, являющемуся источником данных. Объект DataSource содержит информацию о параметрах соединения, включая имя пользователя и пароль. Основная за-

дача объекта – предоставить информацию из массива данных. Он поддерживает несколько интерфейсов, необходимых для взаимодействия с SQL-сервером:

- Интерфейс IDBInitialize инициализирует и устанавливает среду данных и безопасности (аутентификация и другие действия).
- Интерфейс IDBCreateSession создаёт объект Session.
- Интерфейс IDBProperties позволяет получать информацию о возможностях провайдера и производить инициализацию необходимых свойств.

Объект Session содержит данные транзакций и генерирует наборы строк из источника данных, а также команды для запросов к источнику и манипулирования им. Если провайдер поддерживает команды, объект Session выступает в роли фабрики классов объекта Command. Объект Session тоже поддерживает несколько интерфейсов:

- Интерфейс IOpenRowset позволяет открывать набор строк из таблицы, индекса или диаграммы.
- Интерфейс IGetDataSource возвращает объект DataSource из объекта Session.
- Интерфейс IDBCreateCommand создаёт объект Command.

Объект Command служит для обработки команд, которые представляют собой строки, передаваемые от потребителя к провайдеру для исполнения. В большинстве случаев такая команда включает в себя SQL-оператор SELECT, но может содержать и другие операторы. Один сеанс (объект Session) может содержать несколько связанных с ним команд. Результатом выполнения команды является новый объект Rowset.

Объект Command поддерживает свой набор интерфейсов:

- Интерфейс ICommand используется для выполнения запросов или команд.
- Интерфейс ICommandText используется для определения текста запроса.
- Интерфейс ICommandProperties определяет требуемые свойства набора строк, возвращаемого по команде.
- Интерфейс ICommandWithParameters позволяет выполнять запросы с параметрами.

Объект Rowset позволяет провайдерам данных представлять данные из источников в табличном формате. Этот объект представляется как некоторое множество строк, в каждой из которых содержится один или несколько столбцов. Наиболее часто используемые интерфейсы объекта Rowset перечислены ниже:

- Интерфейс IRowset позволяет объекту сканировать ячейки.
- Интерфейс IAccessor присваивает столбцы наборам строк.
- Интерфейс IColumnsInfo позволяет получать информацию о столбцах набора строк.

- Интерфейс IRowsetInfo отвечает за получение информации о свойствах набора строк.
- Интерфейс IrowsetIndex требуется для индексированных наборов строк. Используется функциями, работающими с индексами.
- Интерфейс IConvertType производит проверку на возможность преобразования столбца набора строк одного типа к другому.

В состав OLE DB входят и другие объекты, помимо рассмотренных выше. Объект Enumerator используется для перечисления доступных объектов источников данных (провайдеров OLE DB), а также указания других перечислителей, имеющихся в системе.

Объект Transaction поддерживает транзакции с источниками данных. После включения в распределённую транзакцию используется интерфейс ITransaction для завершения или отмены транзакции.

Помимо кодов возврата и информации о состоянии, указывающей на успешный или же имеющий неблагоприятный исход вызов каждого метода OLE DB, провайдеры могут предоставлять расширенную информацию об ошибке с помощью объекта Error. Разработчику следует использовать ISupportErrorInfo для того, чтобы установить, может ли данный объект возвратить объекты ISupportErrorInfo и получить интерфейсы, которые возвращают эти объекты.

Стандартные провайдеры OLE DB

Некоторые производители СУБД поставляют со своими продуктами драйверы OLE DB для доступа к базам данных. Майкрософт поставляет стандартные провайдеры OLE DB в пакете Microsoft Data Access Components (MDAC).

Долгое время Microsoft рекомендовала использовать OLE DB взамен ODBC, но с анонсом Microsoft SQL Server 2014 было объявлено, что прекращается поддержка «родного» OLE DB для этого продукта и остаётся только поддержка ODBC.

3.2.4. ТЕХНОЛОГИЯ ADO

Технология Microsoft ActiveX Data Objects (ADO) представляет собой высокоуровневую объектную надстройку над OLE DB. Несмотря на то что OLE DB предоставляет полный набор интерфейсов для манипулирования данными, большинство разработчиков не нуждается в низкоуровневом контроле за процессом соединения с данными и управления ими, который предоставляет OLE DB. В то же время разработчики часто используют высокоуровневые языки, которые не поддерживают указатели на функции и другие механизмы C++. ADO может использоваться для работы с любыми провайдерами OLE DB. Схема приведена на рис. 3.12.

Рис. 3.12. Доступ к данным с помощью ADO

В качестве источников данных могут выступать различные хранилища информации, например таблицы, файлы и базы данных.

Основы ADO

Объектная модель ADO состоит из семи объектов, иерархия которых показана на рис. 3.13.

Объект Connection инкапсулирует в себе объекты OLE DB DataSource и Session. Он содержит единственную сессию с источником данных. Объект Connection определяет свойства соединения, определяет возможности локальных транзакций, предоставляет централизованный объект для получения информации об ошибках Error и указатели для использования схем запросов.

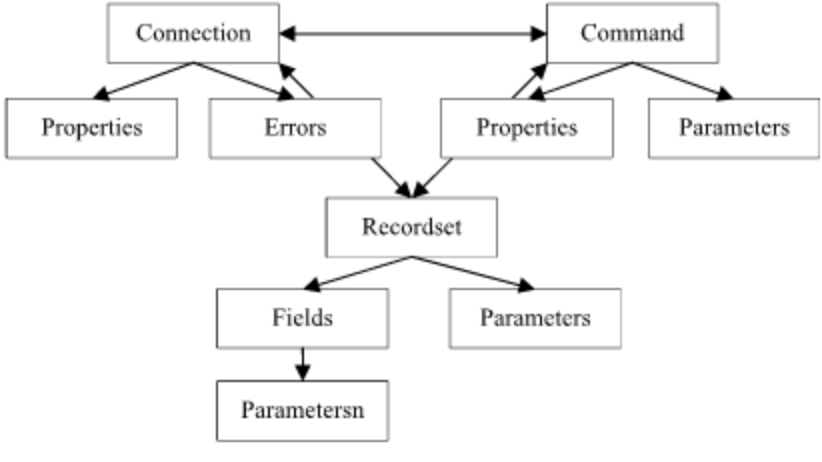

Рис. 3.13. Объектная модель ADO

Объект Command инкапсулирует одноимённый объект OLE DB Command. Объект используется для выполнения команд определения и манипуляции данными. Если в качестве источника данных выступает реляционная СУБД, объект может выполнить некоторые SQL-операторы. Объект Command позволяет определить параметры и установить порядок выполнения запросов. Коллекция объектов Parameter предоставляет доступ к параметрам.

Объект Recordset инкапсулирует функциональность объекта OLE DB Rowset. Объект Recordset является текущим интерфейсом доступа к данным, который может быть получен в результате обработки запроса или каким-либо другим способом. Объект позволяет контролировать используемый механизм блокировок, тип используемого курсора, число строк, возвращаемых в одном пакете, и т.д. Объект Recordset предоставляет доступ к коллекции объектов Field, которые содержат метаданные о свойствах столбцов набора данных, таких как имя, тип, длина и точность. Также содержатся текущие значения записей. Объект Recordset также используется для перемещения по набору данных и их модификации.

Каждый высокоуровневый объект ADO содержит коллекцию объектов Property. Объект Property позволяет ADO динамически публиковать возможности любого провайдера данных. Так как не все провайдеры поддерживают некоторые функции, очень важной особенностью объектной модели ADO является возможность предоставления динамического доступа к специфичным функциям.

Компоненты библиотеки VCL, предназначенные для работы с ADO, строятся на базе рассмотренных объектов.

Компонент TADOConnection

Компонент TADOConnection инкапсулирует объект ADO Connection. Данный компонент предназначен для соединения с хранилищами данных. С одним компонентом TADOConnection может быть связанно несколько компонентов TADOTable, TADOQuery.

Соединение с хранилищем данных открывается и закрывается при помощи свойства Connected или метода Open. Методу Open можно передать параметры UserID и Password, в которых хранятся логин и пароль. Закрыть соединение можно, вызвав метод Close.

Свойство ConnectOptions определяет тип соединения. Можно создавать синхронное и асинхронное соединения. По умолчанию соединение определяется как синхронное. В том случае, когда сервер работает довольно медленно, выбирается асинхронное соединение.

Свойство CursorLocation определяет порядок функционирования курсоров. Если для свойства задать значение clUseServer, то обработка строк будет производиться на сервере. Клиентское приложение будет по-

лучать лишь готовые результаты запросов. Если использовать значение clUseClient, то приложению пересылается весь набор данных, и курсор будет обрабатываться на клиенте. Однако использовать клиентский курсор для больших наборов данных невыгодно. Серверный курсор является чуть более медленным по сравнению с клиентским, но снимает обязанности по обработке данных с клиента и значительно снижает нагрузку на сеть.

Свойство IsolationLevel определяет уровень изоляции транзакции. Этот уровень определяет, как транзакции взаимодействуют с другими соединениями, одновременно обращающимися к таблице, и определяет область видимости транзакции.

Свойство KeepConnection определяет, может ли данное приложение поддерживать связь с базой данных, если нет открытых наборов данных. Когда свойство имеет значение True, соединение будет удерживаться в открытом состоянии. Для соединений с удалёнными СУБД или для приложений, которые часто открывают и закрывают наборы данных, установка значения свойства в True значительно уменьшает сетевой трафик и увеличивает скорость работы приложения, так как не требуется каждый раз проходить аутентификацию на сервере.

Для получения прямого доступа к объекту ошибок ADO следует обратиться к свойству Errors. Свойство DataSets содержит массив активных наборов данных, связанных с компонентом [9]. А свойство DataSetCount позволяет получить число активных наборов данных, связанных с компонентом. Пример его использования показан в листинге 3.2.

Листинг 3.2. Обращение к наборам данных по их индексам:

```
var
i: Integer;
begin
for I: = 0 to (ADQConnection1.DataSetCount – 1) do
ListBox1.Items.Add (ADOConnection1.DataSets[i].Name)
```

Свойство State позволяет узнать, в каком состоянии находится набор данных. А свойство ConnectionString содержит строку, в которой указывается информация, необходимая для установки соединения с источником данных.

До и после открытия и закрытия соединения возникают методы-обработчики событий AfterConnect, BeforeConnect, AfterDisconnect и BeforeDisconnect соответственно. Методы BegirTrans, ComitTrans и RollbackTrans инициируют транзакцию, подтверждают её и производят откат транзакции соответственно. Эти методы следует поместить в блоки секций try-finally и try-except.

Механизм соединения с хранилищем данных ADO

Перед созданием соединения необходимо определить его параметры. Для этого, как уже говорилось, предназначено свойство ConnectionString.

Набор параметров изменяется в зависимости от типа используемого провайдера и может настраиваться как вручную, так и с помощью редактора. Для того чтобы вызывать редактор соединений, необходимо дважды щёлкнуть на компоненте TADOConnection. В результате будет активировано диалоговое окно. В этом окне можно настроить соединение, используя поле Use Connection String, или загрузить параметры соединения из файла в разделе Use Data Link File.

Параметры соединения хранятся в файлах UDL, представляющих собой обычные текстовые файлы, содержащие параметры соединения.

Для того чтобы настроить соединение с данным провайдером, необходимо нажать на кнопку Build. Появится окно, в котором будет опубликован список доступных провайдеров.

На вкладке Provider можно выбрать подходящий провайдер данных OLE DB для конкретного источника данных. В списке провайдеров также присутствуют провайдеры, предназначенные для доступа к конкретным службам операционной системы. На вкладке Connection необходимо указать путь к базе данных или сервер. Вкладка Advanced предназначена для указания режима доступа, аналогично свойству Mode. Вкладка All предназначена для более «тонкой» настройки специфичных свойств провайдера. Для дальнейшей работы нужно выбрать провайдер Microsoft Jet 4.0 OLE DB Provider. Затем нужно перейти на вкладку Connection.

В появившемся окне необходимо указать путь к базе данных. В поле Select or enter a database name нужно указать путь к демонстрационной базе dbdemos.mdb. После указания пути к базе данных и задания остальных необходимых параметров нужно проверить созданное соединение с помощью кнопки Test Connection. Если параметры соединения указаны верно, появится сообщение Test connection succeeded. После закрытия этого окна в строке соединения будет отображена информация, с помощью которой провайдер сможет получить доступ к данным.

Компонент TADODataSet

Данный компонент представляет набор данных, получаемых из хранилища ADO. Компонент наследует свойства и методы класса TCustomADODataSet и добавляет несколько своих. Компонент имеет возможность получать результирующие наборы данных от одной или нескольких таблиц.

Используя свойство CommandText, можно указать текст команды, с помощью которой будут получены данные. Это может быть SQL-запрос, название таблицы или название хранимой процедуры. А в свойстве CommandType указывается тип исполняемой команды. Данный компо-

нент не может выполнять SQL-операторы, не возвращающие результирующие наборы (операторы DELETE, INSERT и UPDATE). Соединение с базой данных задаётся свойствами Connection или ConnectionString. Компонент связывается с компонентами отображения данных стандартным способом.

Компонент TADOTable

Компонент TADOTable используется для доступа к хранилищам данных ADO и представления информации из них в табличном виде. Компонент предоставляет прямой доступ к каждой записи и её полям, наследуя свойства и методы класса TCustomADODataSet. Компонент связывается с базой данных через свойства Connection или ConnectionString.

Имя таблицы указывается в свойстве TableNante. Свойство TableDirect указывает, каким образом набор данных связывается с хранилищем данных. Так как не все провайдеры поддерживают прямое соединение с набором данных, то в некоторых случаях для связи с хранилищем данных приходится использовать SQL-операторы. При установке свойству значения True компонент использует фоновые SQL-запросы для доступа к данным.

Используя свойство Readonly, можно установить ограничение «только для чтения» на данную таблицу, запретив, таким образом, возможность изменять данные. В свойстве MasterSource указывается компонент TDataSource, используемый для создания отношения ссылочной целостности Master-Detail. Метод GetIndexNames возвращает список индексов, доступных компоненту в качестве списка.

Компонент TADOQuery

Компонент TADOQuery позволяет выполнять SQL-запросы при работе с данными через ADO. Соединение с хранилищем данных осуществляется стандартным методом. Текст запроса содержится в свойстве SQL. В листинге 3.3 приведён пример типичного SQL-запроса, содержащегося в упомянутом свойстве.

Листинг 3.3. Выполнение SQL-запроса

```
with ADOQuery1 do begin
Close;
with SQL do begin
Clear;
Add ('SELECT EmpNo.LastName.FirstName.HireDate'):
Add ('FROM Employee');
end;
Open;
end.
```

Параметры запроса содержатся в свойстве Parameters. В случае если компонент возвращает набор данных, его следует открывать методом Open или присвоить свойству Active значение True. Если запрос не должен возвращать набор данных (операторы INSERT, UPDATE, DELETE и CREATE TABLE), то запрос следует выполнять вызовом метода ExecSQL. Метод возвращает число обработанных запросом записей [9].

Свойство RowsAffected содержит число записей, которые затронул последний выполнявшийся запрос.

Компонент TADOStoredProc

Компонент TADOStoredProc позволяет обращаться к хранимым процедурам, содержащимся в базах данных. Связь компонента с хранилищем данных осуществляется стандартно, через названные ранее свойства.

Имя хранимой процедуры определяется свойством ProcedureName. Во время разработки хранимую процедуру можно выбрать из списка уже существующих процедур.

Для определения входных и выходных параметров процедуры используется свойство Parameters. Через параметры хранимая процедура получает аргументы и возвращает результаты своей работы.

В случае если хранимая процедура будет вызываться множество раз с одними и теми же аргументами, рекомендуется заранее подготовить её к исполнению. СУБД разместит её в оперативной памяти. Для этого свойству Prepared следует присвоить значение True.

3.2.5. ТЕХНОЛОГИЯ DBEXPRESS

Технология dbExpress была разработана Borland для того, чтобы заменить устаревшую технологию BDE. Эта технология доступа к данным обеспечивает взаимодействие приложений с СУБД. Данная технология является кросс-платформенной, не зависит от конкретной СУБД и имеет открытые интерфейсы, предоставляющие методы для выполнения динамически формируемых запросов. dbExpress возвращает только однонаправленные курсоры, не поддерживает кеширование и, следовательно, не позволяет напрямую редактировать наборы данных.

Технология dbExpress представляет собой совокупность драйверов, компонентов, инкапсулирующих соединения, транзакции, запросы и наборы данных, а также интерфейсов, обеспечивающих универсальный доступ к функциям dbExpress. Драйверы доступа к СУБД реализованы в виде динамических библиотек. Данная особенность позволяет распространять приложения, использующие эту технологию доступа к данным, без лишних хлопот. На машине конечного пользователя обязательно должна быть установлена клиентская часть СУБД.

Драйверы для доступа к СУБД поставляются Borland и сторонними разработчиками.

Интерфейсы dbExpress

Базовые классы SQLDriver, SQLConnecti on, SQLComand и SQLCursor составляют ядро рассматриваемой технологии.

Интерфейс SQLDriver предоставляет методы, необходимые для обслуживания соединения с драйвером, и производит определённые действия, например загрузку поставщика клиента, инициализацию окружения, определение необходимых указателей. После отработки возвращается объект SQLConnecti on. Интерфейс поддерживает три метода. Методы setOption и getOption позволяют работать с параметрами драйвера. Метод getSQLConnection возвращает указатель на интерфейс ISQLConnection.

Интерфейс ISQLConnection устанавливает соединение с базой данных, возвращает экземпляры объекта SQLComand для выполнения запросов и хранимых процедур, возвращает экземпляры объекта SQLMetaData для получения метаданных и управляет транзакциями.

Метод Connect устанавливает соединение с базой данных. Его параметры pszServerName, pszUserNaire и pszPassword содержат название базы данных, имя пользователя и пароль.

Завершить текущее соединение с базой данных можно с помощью метода Disconnect. Метод getSQLCommand предоставляет интерфейс ISQLComand, который может быть использован при обработке SQL-запроса или хранимой процедуры. Данный метод следует вызывать только после того, как объект SQLConnection будет связан с базой данных.

Для получения информации о таблицах, хранимых процедурах, индексах, схемах и других объектах базы данных следует вызвать метод getSQLMetaData. Он возвращает интерфейс ISQLMetaData. Используя методы этого интерфейса, можно получить доступ к метаданным, перечисленным выше. Данный метод следует вызывать только после того, как объект SQLConnection будет связан с базой данных.

Используя методы setOption и getOption, можно управлять параметрами соединения и получать информацию о них. Для управления транзакциями предназначены методы beginTransaction, commit и rollback. Их названия достаточно прозрачно отражают функциональность методов. Параметр ulTransDesc содержит указатель на структуру TransactionDesc.

Метод getErrorMessage используется для получения информации о последней ошибке, произошедшей при выполнении какой-либо операции. Для определения длины строки, содержащей сообщение о последней произошедшей ошибке, следует вызвать метод getErrorMessageLen.

Интерфейс ISQLComand предоставляет методы для выполнения запросов и хранимых процедур. Он может быть использован для получения экземпляра интерфейса ISQLCursor после выполнения команды, возвращающей курсор. Он поддерживает выполнение «подготовленных» запросов.

Для установки свойств запроса можно использовать метод setOption. В его параметре eSOption указывается свойство команды, значение которого будет изменено. Параметр lValue содержит устанавливаемое значение.

Для получения информации о свойствах запроса следует использовать метод getOption. В его параметре eSOption указывается имя параметра, значение которого будет получено. Параметр plValue является указателем на переменную, в которую будет помещено полученное значение. Параметр iMaxLength определяет максимальный размер в байтах, который может быть возвращён в параметре plValue. Соответственно, в параметре Length возвращается длина значения, содержащегося в plValue.

В основе библиотеки VCL-компонентов, реализующих технологию dbExpress, лежат несколько описанных выше интерфейсов. Изложенная информация позволит лучше разобраться в механизме их работы.

Компонент TSQLConnection

Данный компонент используется для установления соединения с СУБД. С одним компонентом TSQLConnection может быть связано несколько компонентов набора данных через их свойство Connection. Компонент взаимодействует с драйвером и двумя файлами. Файл dbxdrivers.ini содержит список установленных драйверов и набор настроек для каждого драйвера. Файл dbxconnections.ini содержит базовый набор настроек.

Для установки соединения с базой данных следует для свойства Connected установить значение True. Также открыть или закрыть соединение можно с помощью методов Open и Close. При открытии и закрытии соединения с базой данных возникают события AfterConnect, AfterDisconnect, BeforeConnect и Be foreDisconnect. С помощью соответствующих методов можно реализовать различные проверки данных.

Свойство ConnectionName содержит имя настроенного ранее соединения. Его можно выбрать из списка. А свойство DriverName определяет драйвер dbExpress, используемый для взаимодейстлия с СУБД. Если необходимо получить имя динамической библиотеки драйвера dbExpress, следует обратить внимание на значение свойства LibraryName.

В свойстве Params содержится список параметров соединения. В параметрах можно указать имя пользователя и пароль, которые будут подставляться автоматически при установке соответствующего свойства.

Эти свойства получают свои значения автоматически при выборе конкретного соединения с базой данных в свойстве ConnectionName.

Свойство ConnectionState указывает текущее состояние соединения.

Если для каких-либо целей потребуется создать копию объекта SQLConnection, следует воспользоваться методом CloneConnection.

Свойство ActiveStatements позволяет узнать общее число запросов, выполняющихся на сервере в данный момент. А в свойстве MaxStmtsPerConn

указывается максимальное число одновременно выполняющихся запросов, установленное сервером. Если свойство имеет нулевое значение, то ограничение просто не установлено.

Обратиться к конкретному набору данных можно через его индекс, используя свойство DataSets. А метод GetTableNames позволяет получить список таблиц базы данных. Если его параметру SystemTables присвоить значение True, то метод также вернёт имена системных таблиц.

Свойство TableScope позволяет задать типы таблиц, которые попадут в список, полученный при вызове метода GetTableNames.

Свойство TableScope будет работать только в том случае, когда параметр SystemTables имеет значение False.

Для получения списка полей конкретной таблицы следует использовать метод GetFieldNames. В его параметре TableName указывается имя таблицы, имена полей которой необходимо получить.

Выполнение команды на сервере инициируется методом Execute. Параметр SQL содержит выполняемую команду, а в параметре Params указывается коллекция объектов типа TParams, содержащая параметры команды. Пример использования данного метода приведён в листинге 3.4.

Листинг 3.4. Пример использования метода Execute

```
procedure TForm1.InsertWithParamsButtonClick(Sender: TObject);
var
SQLstmt: String;
stmtParams: TParams;
begin
stmtParams: = TParams.Create;
try
SQLConnection1.Connected: = True;
stmtParams.CreateParam(ftString.'StateParam'. ptInput);
stmtParams.ParamByName('StateParam').AsString : = 'CA';
SQLstmt: = - 'INSERT INTO «Custom» ' +
' (CustNo.Company.State) ' +
'VALUES (7777. «Robin Dabank Consulting». :StateParam)';
SQLConnection1.Execute(SQLstmt.stmtParams);
finally
stmtParams.Free;
end;
end.
```

Начало, откат и фиксацию транзакции выполняют методы StartTransaction, Commit и Rollback соответственно. Параметры транзакции содержатся в записи TTransactionDesc.

В листинге 3.5 приведён пример работы с транзакцией. Тело транзакции помещено в блок try-except, что позволяет обработать возможную ошибку и произвести откат изменений.

Листинг 3.5. Пример работы с транзакциями

```
procedure TForml.TransferButtonClick(Sender: TObject);
var
Amt: Integer;
TD: TTransactionDesc;
begin
if not SQLConnection1.InTransaction then
begin
TD.TransactionID : = 1;
TD.IsolationLevel : = xilREADCOMMTTED;
SQLConnection1.StartTransaction(TD);
try
Amt : = StrToInt(AmtEdit.Text);
Debit.Params.ParamValues['Amount'] : = Amt;
Credit.Params.ParamValuest['Amount'] ; = Amt;
SQLConnectionl.Commit(TD): {on success, commit the changes};
except
SQLConnectionl.Rollback(TD): {on failure, undo the changes};
end;
end;
end.
```

Свойство MultipleTransactionsSupported указывает, поддерживает ли данный сервер одновременное выполнение нескольких транзакций. Если свойство имеет значение True, то сервер поддерживает множественные транзакции. В этом случае каждой транзакции, запущенной методом StartTransaction, необходимо присваивать уникальный идентификатор.

Свойство InTransaction позволяет выяснить, выполняется ли в данный момент транзакция. Если транзакция запущена, свойство имеет значение True.

Компонент TSQLDataSet

Компонент TSQLDataSet представляет собой однонаправленный набор данных, полученный с помощью технологии dbExpress. Он позволяет выполнять запросы и хранимые процедуры, а также получать данные из таблиц.

В свойстве CommandType указывается тип выполняемой команды, а в свойстве ConmandText – её текст либо имя таблицы или хранимой процедуры.

Для открытия или закрытия набора данных используются методы Open и Close. Впрочем, можно оперировать и свойством Active. Если запрос или хранимая процедура не возвращают набор данных, следует использовать метод ExecSQL. Его параметр ExecDirect указывает на необходимость подготовки команды перед выполнением, если она содержит параметры. Если команда не содержит параметров, параметру следует присвоить значение True.

Через свойство TransactionLevel можно определить уровень изоляции транзакции.

Соединение с компонентом TSQLConnection устанавливается через свойство SQLConnection. А свойство IndexDefs содержит описание всех индексов, используемых в результирующем наборе данных.

Компонент TSQLTable

Компонент TSQLTable инкапсулирует однонаправленный набор данных и используется для представления информации, получаемой из базы данных, в табличном виде. Компонент генерирует запросы на получение всех записей со всеми полями для выбранной таблицы.

Свойство IndexName содержит имя текущего индекса, по которому производится сортировка записей и по которому устанавливаются реляционные отношения между таблицами. Через свойство IndexFields можно обратиться к полям, входящим в данный индекс. Номер соответствующего поля указывается в параметре Index.

В свойстве IndexFieldNames перечислены поля, входящие в индекс, установленный текущим. Поля перечисляются через точку с запятой. Если необходимо узнать количество полей, входящих в текущий индекс, стоит обратиться к свойству IndexFieldCount. Метод GetIndexNames возвращает список индексов для рассматриваемой таблицы.

Для настройки отношений ссылочной целостности между таблицами используются свойства MasterFields и MasterSource.

Метод PrepareStatement генерирует SQL-запрос для получения данных от СУБД. А метод DeleteRecords применяется для удаления из таблицы всех записей.

Компонент TSQLQuery

Данный компонент используется для выполнения SQL-запросов на сервере. Он может возвращать результаты запроса в виде однонаправленного набора данных.

В свойстве SQL указывается текст SQL-запроса, который будет выполнен на сервере. А свойство Text представляет SQL-запрос в виде текстовой строки. Метод PrepareStatement вызывается для подготовки запро-

са к выполнению на сервере. После того как метод будет подготовлен, его можно выполнить с помощью метода ExecSQL. Этот метод обычно используется для тех SQL-запросов, которые не возвращают результатов.

Компонент TSimpleDataSet

Компонент используется для получения данных и обеспечивает их кеширование. Объект TSimpleDataSet является клиентским набором данных, использующим TSQLDataSet и TDataSetProvider для получения данных и передачи произведённых изменений на сервер. Он сочетает быстрый доступ к данным и лёгкость развёртывания однонаправленных наборов с возможностями редактирования и навигации, предоставляемыми клиентскими наборами данных. Таким образом, компонент предоставляет в распоряжение разработчика двунаправленный курсор, позволяет отображать данные в таблицах и редактировать их в режиме кеширования.

Соединение с источником данных осуществляется с помощью свойства Connection. Но можно использовать и дублирующее свойство ConnectionName, в котором указывается только наименование соединения.

Тип выполняемой команды указывается в свойстве CommandType, а её содержание задаётся в свойстве CommandText.

В свойстве FetchOnOemand можно указать возможность получения пакетов данных по запросу. По умолчанию свойство имеет значение True, т.е. клиентский набор данных получает дополнительные пакеты при перемещении курсора по набору данных. Если свойство имеет значение False, разработчик обязан явно производить получение новых данных с помощью метода GetNextPacket, который возвращает число переданных в пакете записей.

Компонент TSQLStoredProc

Компонент TSQLStoredProc используется для выполнения хранимых процедур на сервере и представления результатов их работы. Как и для других компонентов, соединение с базой данных указывается в свойстве SQLConnection. После соединения с базой данных можно выбрать выполняемую процедуру в свойстве StoredProcName. При разработке в инспекторе объектов для этого свойства отображается список хранимых процедур, расположенных на сервере. После выбора хранимой процедуры компонент TSQLStoredProc запрашивает список её параметров и использует полученную информацию для инициализации свойства Params. Некоторые СУБД не позволяют вернуть информацию о параметрах хранимой процедуры. В этом случае параметры необходимо определять самостоятельно.

Если хранимая процедура не возвращает набор данных, следует использовать метод ExecProc. Для выполнения процедуры, возвращающей значения, используется метод Open.

Компонент TSQLMonitor

Компонент TSQLMonitor перехватывает сообщения, пересылаемые между компонентом TSQLConnection и СУБД, и сохраняет их в списке. Его очень удобно использовать для отслеживания взаимодействий приложения с сервером.

Соединение с компонентом TSQLConnection устанавливается с помощью свойства SQLConnectior. В свойстве FileName указывается имя файла, в который сохраняется журнал сообщений.

Свойство TraceList содержит список сообщений, передаваемых от приложения к серверу и обратно.

В свойстве AutoSave можно указать режим автоматического сохранения сообщений в файл из списка TraceList, иначе данную возможность придётся реализовывать вручную.

В свойстве TraceCount указывается число сообщений, зарегистрированных на данный момент, а целочисленное свойство MaxTraceCount позволяет указать максимальное число сообщений, которое будет зарегистрировано. Если свойство имеет значение –1, то ограничения на число сообщений нет. Если указано нулевое значение, то TSQLMonitor не ведёт журнал сообщений.

3.2.6. ПРИМЕР РАБОТЫ С SQL SERVER 2008

В этом разделе будет приведён пример приложения, взаимодействующего с SQL Server 2008. Как обычно, следует создать новый проект и добавить в него модуль данных. В модуле данных потребуется установить компонент TSQLConnection. Двойной щелчок мышью на нём активирует окно редактора соединений. Нужно создать новое соединение, указать драйвер MSSQL и дать ему имя ConnectToNorthwind. Теперь необходимо настроить соединение. В свойстве HostName потребуется указать сетевое имя компьютера, которое можно получить после выполнения команды System Properties ► Computer Name ► Full Computer Name. Диалоговое окно System Properties можно также вызвать через контекстное меню MyComputer ► Properties.

В поле DataBase нужно выбрать базу данных Northwind. В поле User_Name необходимо ввести dbo, а в поле Password – набор символов, создающий пароль. В поле MSSQL Transisolation нужно выбрать из списка значение DirtyRead. В поле OS Authentication можно выбрать значение True. В этом случае SQL Server 2008 будет использовать средства аутентификации операционной системы, не используя собственных методов. Именно поэтому в качестве пароля для учётной записи dbo был использован ничего не значащий набор символов.

Также потребуется один компонент TSimpleDataSet, который нужно связать с TSQLConnection. Свойство DataSet компонента TSimpleDataSet

предоставляет доступ к внутреннему набору данных, получающему данные от сервера. В свойстве CommandType нужно выбрать значение ctTable, а в свойстве CommandText указать таблицу Customers. Осталось лишь активировать набор данных, присвоив свойству Active значение True. После этого можно активировать компонент TSimpleDataSet.

В модуле данных нужно разместить и настроить компонент TSQLMonitor. Также потребуется компонент TDataSource, который необходимо связать с компонентом TSimpleDataSet с помощью свойства DataSet.

На главной форме следует расположить компонент TDBGrid и пять кнопок. После связывания модуля данных с главной формой приложения можно связать TDBGrid с компонентом TDataSource. Кнопки Принять (PostBtn), Удалить (DeleteBtn), Отменить (CancelBtn), Сохранить (SaveBtn) и Обновить (RefreshBtn) обрабатываются кодом, приведённым в листинге 3.6. При нажатии на кнопку Принять изменения вносятся в кеш и могут быть отменены нажатием на кнопку Удалить. На рисунке 3.14 показан внешний вид основного окна программы.

Листинг 3.6. Код главной формы примера работы с компонентом TSimpleDataSet

```
procedure TMainForm.PostBtnClick(Sender: TObject);
begin
with ConnectionModule.SimpleDataSetl do begin
if State in [dslnsert.dsEdit] then
Post;
end;
end;
procedure TMainForm.DeleteBtnClick(Sender: TObject):
begin
with ConnectionModule.SimpleDataSetl do begin
if State = dsBrowse then
if MessageDlg('Вы уверены в том, что хотите удалить запись?'
mtConfirmation.[mbYes.mbNo], 0) = mrYes then
Delete;
end;
end;
procedure TMainForm.SaveBtnClick(Sender: TObject):
begin
with ConrectionModule.SimpleDataSetl do begin
if (ChangeCount > 0) then
ApplyUpdates(-l);
end;
end;
procedure TWainForm.CancelBtnClick(Sender: TObject);
```

```
begin
Connecti onModule.Si mpleDataSetl.CancelUpdates;
end;
procedure TMainForm.RefreshBtnClick(Sender: TObject);
begin
ConnectionModu1e.SimpleDataSetl.Refresh;
end;
```

Модуль данных содержит код обработчика ошибок, возникающих при сохранении данных на сервере вызовом метода ApplyUpdates. Код приведён в листинге 3.7.

Листинг 3.7. Код обработчика ошибок

```
procedure TConnectionModule.SimpleDataSetlReconcileError(
DataSet: TCustomClientDataSet: E: EReconcileError:
UpdateKind: TUpdateKind: var Action: TReconcileAction);
begin
Action: = HandleReconcileError(DataSet.UpdateKind. E);
end;
```

Можно попробовать запустить несколько копий приложения и посмотреть, как реализовано изменение одной записи несколькими пользователями [5].

Рис. 3.14. Пример работы с компонентом TSimpleDataSet

4. ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ И ЗАЩИТА ИНФОРМАЦИИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

4.1. ПРАВОВАЯ ОСНОВА ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

При проектировании информационных систем на всех стадиях и этапах работ необходимо учитывать вопросы, связанные с защитой информации, информационной безопасностью.

Становление информационного общества породило не только позитивные ожидания, связанные с положительными переменами в жизни социума, но, к сожалению, породило и серьёзные проблемы, в том числе связанные с информационной безопасностью. Решение этих проблем является весьма важным, так как от этого напрямую зависят национальные интересы в информационной сфере (включая соблюдение основных конституционных прав и свобод человека и гражданин), а также и существование самого информационного общества [11].

Необходимо отметить, что ведущую роль в становлении информационного общества играет государство, координируя усилия различных субъектов общества, законодательно обеспечивая условия для конкуренции в информационной индустрии, юридически защищая информационные права и свободы личности, при этом обеспечивая защищённость национальных интересов в информационной сфере.

За последние годы в Российской Федерации проведена существенная работа по созданию и совершенствованию обеспечения её информационной безопасности. Сформирована правовая основа обеспечения информационной безопасности. Приняты Закон Российской Федерации «О государственной тайне», Федеральные законы «Об информации, информационных технологиях и о защите информации», «О персональных данных», ряд других законов, ведётся работа по их реализации, подготовке законопроектов, регламентирующих общественные отношения в информационной сфере. Реализуется государственная программа Российской Федерации «Информационное общество (2011 – 2020 годы)».

В Российской Федерации правовую базу в области информационной безопасности составляют соответствующие международные договоры РФ; Конституция РФ; законы федерального уровня (включая федеральные конституционные законы, кодексы); указы Президента РФ; постановления Правительства РФ; нормативные правовые акты федеральных министерств и ведомств; нормативные правовые акты субъектов РФ, органов местного самоуправления) и т.д.

Среди нормативно-методических документов государственных органов РФ можно выделить Доктрину информационной безопасности РФ, руководящие документы ФСТЭК, приказы ФСБ. (Основные положения нормативных документов в области информационной безопасности более детально изучаются в рамках дисциплины «Информационное право»)

Кроме того, к нормативно-методическим относятся Стандарты информационной безопасности, из которых выделяют: международные стандарты; государственные (национальные) стандарты РФ; рекомендации по стандартизации; методические указания.

Успешному решению вопросов обеспечения информационной безопасности Российской Федерации способствуют государственная система защиты информации, система защиты государственной тайны, системы лицензирования деятельности в области защиты государственной тайны и системы сертификации средств защиты информации.

Вместе с тем стоит отметить, что состояние информационной безопасности Российской Федерации должно в полной мере соответствовать растущим потребностям общества и государства.

Современные условия политического и социально-экономического развития страны вызывают обострение противоречий между потребностями общества в расширении свободного обмена информацией и необходимостью сохранения отдельных регламентированных ограничений на её распространение.

В России, как и во всём мире, растёт популярность Интернета. При этом существенные проблемы информационной безопасности связаны и с тем, что преступники и злоумышленники всё чаще используют его в своих целях. Так, по сообщению секретаря Совбеза РФ Николя Патрушева, более 90 миллионов кибератак совершено на российские информационные ресурсы с 2010 г., из них 57 миллионов в 2014 г. [6]. Проблемы защищённости информационных ресурсов, обеспечения доступа к ним, а также защита от вредного и опасного содержимого, в том числе и в рамках обеспечения основных конституционных прав и свобод на информацию, безусловно, являются одними из наиболее важных.

Состояние защищённости национальных интересов в информационной сфере, определяющихся совокупностью сбалансированных интересов личности, общества и государства составляет понятие ***информационной безопасности Российской Федерации***, основные положения которой изложены в Доктрине информационной безопасности Российской Федерации, определяющей вектор многоплановых усилий государства в сфере обеспечения информационной безопасности [2].

Государственное регулирование в сфере применения информационных технологий предусматривает наряду с другими аспектами регулирование отношений, связанных с поиском, получением, передачей, произ-

водством и распространением информации с применением информационных технологий (информатизации), на основании принципов, установленных Федеральным законом от 27 июля 2006 г. № 149-ФЗ «Об информации информационных технологиях и о защите информации» (в редакции от 21.07.2014 г. № 242-ФЗ); развитие информационных систем различного назначения для обеспечения граждан (физических лиц), организаций, государственных органов и органов местного самоуправления информацией, а также обеспечение взаимодействия таких систем; создание условий для эффективного использования в Российской Федерации информационно-телекоммуникационных сетей, в том числе сети Интернет и иных подобных информационно-телекоммуникационных сетей [10].

При этом государственные органы, органы местного самоуправления в соответствии со своими полномочиями участвуют в разработке и реализации целевых программ применения информационных технологий; создают информационные системы и обеспечивают доступ к содержащейся в них информации на русском языке и государственном языке соответствующей республики в составе Российской Федерации.

Информационные системы (согласно ст. 13 Федерального закона «Об информации, информационных технологиях и о защите информации») включают в себя: государственные информационные системы (федеральные информационные системы и региональные информационные системы, созданные на основании соответственно федеральных законов, законов субъектов Российской Федерации, на основании правовых актов государственных органов); муниципальные информационные системы, созданные на основании решения органа местного самоуправления; иные информационные системы.

Оператором информационной системы (если иное не установлено федеральными законами) является собственник используемых для обработки содержащейся в базах данных информации технических средств, который правомерно пользуется такими базами данных, или лицо, с которым этот собственник заключил договор об эксплуатации информационной системы. В случаях и в порядке, установленных федеральными законами, оператор информационной системы должен обеспечить возможность размещения информации в сети Интернет в форме открытых данных.

Важно отметить, что права обладателя информации, содержащейся в базах данных информационной системы, подлежат охране независимо от авторских и иных прав на такие базы данных.

Установленные Федеральным законом «Об информации, информационных технологиях и о защите информации» требования к государственным информационным системам распространяются и на муниципаль-

ные информационные системы (если иное не предусмотрено законодательством Российской Федерации о местном самоуправлении).

В соответствии с техническими регламентами, нормативными правовыми актами государственных органов, нормативными правовыми актами органов местного самоуправления, принимающих решения о создании информационных систем могут устанавливаться особенности эксплуатации государственных информационных систем и муниципальных информационных систем.

Порядок создания и эксплуатации информационных систем, не являющихся государственными информационными системами или муниципальными информационными системами, определяется операторами таких информационных систем в соответствии с требованиями, установленными Федеральным законом «Об информации, информационных технологиях и о защите информации» или другими федеральными законами.

Защита информации согласно ст. 16 Федерального закона «Об информации, информационных технологиях и о защите информации» представляет собой принятие правовых, организационных и технических мер, направленных на:

– обеспечение защиты информации от неправомерного доступа, уничтожения, модифицирования, блокирования, копирования, предоставления, распространения, а также от иных неправомерных действий в отношении такой информации;

– соблюдение конфиденциальности информации ограниченного доступа;

– реализацию права на доступ к информации.

При создании информационных систем необходимо учитывать, что обладатель информации, оператор информационной системы в случаях, установленных законодательством Российской Федерации, обязаны обеспечить:

– предотвращение несанкционированного доступа к информации и(или) передачи её лицам, не имеющим права на доступ к информации;

– своевременное обнаружение фактов несанкционированного доступа к информации;

– предупреждение возможности неблагоприятных последствий нарушения порядка доступа к информации;

– недопущение воздействия на технические средства обработки информации, в результате которого нарушается их функционирование;

– возможность незамедлительного восстановления информации, модифицированной или уничтоженной вследствие несанкционированного доступа к ней;

– постоянный контроль за обеспечением уровня защищённости информации;

– нахождение на территории Российской Федерации баз данных информации, с использованием которых осуществляются сбор, запись, систематизация, накопление, хранение, уточнение (обновление, изменение), извлечение персональных данных граждан Российской Федерации (этот пункт введён Федеральным законом от 21 июля 2014 г. № 242-ФЗ – Собрание законодательства Российской Федерации, 2014, № 30, ст. 4243, вступает в силу с 1 сентября 2016 г.).

Необходимо отметить, что в соответствии с Федеральным законом «Об информации, информационных технологиях и о защите информации» требования о защите информации, содержащейся в государственных информационных системах, устанавливаются федеральным органом исполнительной власти в области обеспечения безопасности и федеральным органом исполнительной власти, уполномоченным в области противодействия техническим разведкам и технической защиты информации, в пределах их полномочий. При создании и эксплуатации государственных информационных систем используемые в целях защиты информации методы и способы её защиты должны соответствовать указанным требованиям.

Также необходимо учитывать, что федеральными законами могут быть установлены ограничения использования определённых средств защиты информации и осуществления отдельных видов деятельности в области защиты информации.

Нарушение указанных требований Федерального закона «Об информации, информационных технологиях и о защите информации» влечёт за собой дисциплинарную, гражданско-правовую, административную или уголовную ответственность в соответствии с законодательством Российской Федерации.

Системный подход к описанию информационной безопасности предлагает выделение следующих составляющих информационной безопасности [12]:

– законодательная, нормативно-правовая и научная база;

– структура и задачи органов (подразделений), обеспечивающих безопасность ИТ;

– организационно-технические и режимные меры и методы (политика информационной безопасности);

– программно-технические способы и средства обеспечения информационной безопасности.

Целью реализации информационной безопасности какого-либо объекта является построение Системы обеспечения информационной безопасности данного объекта.

4.2. СТАНДАРТЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ. ПОЛЕЗНЫЕ ССЫЛКИ

Ниже приведены полезные ссылки на интернет-ресурсы, содержащие правовую базу, аналитику, материалы по обеспечению информационной безопасности, национальные стандарты в области информационной безопасности.

http://www.gost.ru/wps/portal/pages.CatalogOfStandarts

РОССТАНДАРТ. Федеральное агентство по техническому регулированию и метрологии. Каталог национальных и международных стандартов с возможностью поиска по имеющейся базе.

http://www.fstec.ru

Официальный сайт Федеральной службы по техническому и экспортному контролю (ФСТЭК России), федерального органа исполнительной власти, осуществляющего межотраслевую координацию и функциональное регулирование деятельности по обеспечению защиты информации, содержащей сведения, составляющие государственную или служебную тайну, от её утечки по техническим каналам, от несанкционированного доступа к ней, от специальных воздействий на информацию и по противодействию техническим средствам разведки на территории Российской Федерации. Содержит нормативные правовые акты и информационные материалы.

http://fstec.ru/rss-lenta/113-tekhnicheskaya-zashchita-informatsii/dokumenty/gosudarstvennye-standarty/377-gosudarstvennye-standarty

Официальный сайт Федеральной службы по техническому и экспортному контролю (ФСТЭК России). Национальные стандарты. Стандарты в области информационной безопасности.

http://www.azi.ru

Сайт Межрегиональной общественной организации «Ассоциация защиты информации» (АЗИ), деятельность которой направлена на создание благоприятных условий для реализации потребностей граждан, бизнеса и органов государственной власти в продуктах и технологиях защиты информации. АЗИ активно взаимодействует с аппаратом Совета Безопасности РФ, ФСБ России, ФСТЭК России, другими министерствами и ведомствами, а также со многими финансово-экономическими структурами. Содержит нормативные правовые акты и информационные материалы.

http://leta.ru/library/search/

Правовая база, аналитика и исследования по теме защиты информации на официальном сайте LETA IT-company (компания LETA – лидер рынка информационной безопасности в России, занимает ведущие позиции в области защиты от утечек конфиденциальных данных (DLP), защиты персональных данных (ЗПДн); оказывает полный спектр услуг по информационной безопасности – от разработки стратегии ИБ, внедрения и сопровождения технических средств защиты информации до оказания услуг по расследованию и предотвращению компьютерных преступлений.).

http://www.infolaw.ru/

Электронная версия журнала «Информационное право». Представляет аналитические публикации по вопросам информационного права, в том числе защиты персональных данных, информационной безопасности.

http://www.itsec.ru/articles2/allpubliks

Электронный архив публикаций журнала «Информационная безопасность». Содержит информационно-аналитические материалы по вопросам защиты информации.

http://www.itsec.ru/rass.php

Архив электронной газеты «Информационная безопасность». Содержит публикации по вопросам информационной безопасности

http://www.secuteck.ru/articles2/allpubliks

Электронный архив публикаций журнала «Системы безопасности». Содержит информационно-аналитические материалы по вопросам защиты информации.

www.garant.ru

Интернет-версия системы «Гарант» (Правовая информационно-поисковая система).

www.consultant.ru

(Интернет-версия системы «КонсультантПлюс». (Правовая информационно-поисковая система).

ЗАКЛЮЧЕНИЕ

Информационные технологии – база для разработки информационных систем, которые являются главным информационным ресурсом, обеспечивающим эффективность бизнеса производственных систем.

Задачей повышения информатизации производственных систем является интеграция технологий с созданием автоматизированных комплексов, обеспечивающих управление производственными системами.

Проектирование информационных систем основано на учёте возможностей новых информационных технологий и потребностей обслуживания бизнес-процессов производственных систем.

Создание интегрированных автоматизированных систем управления включает задачи проектирования информационных систем на основе поэтапного реинжиниринга информационных процессов производственных систем.

Моделирование – это сложный путь в проектировании и разработке любого продукта, а систем в особенности, требующий от инженеров высокой квалификации, внимания, терпения, а от организации – значительных инвестиций в средства моделирования. Однако какими бы дорогостоящими ни были эти процессы, экономическая выгода от их использования очевидна: система без сбоев имеет наименьшие издержки. В настоящее время приобретает особое значение оценка защищённости систем, поскольку все системы стремятся объединить с глобальной сетью Интернет. Требования защищённости в некоторых отношениях подобны требованиям безопасности. В частности, они определяют нештатное поведение системы, а не её «рабочее» поведение. Однако, как правило, невозможно определить это поведение в виде простых ограничений, контролируемых системой.

Проблемы защищённости информационных ресурсов, обеспечения доступа к ним, а также защита от вредного и опасного содержимого, в том числе и в рамках обеспечения основных конституционных прав и свобод на информацию, безусловно, являются одними из наиболее важных.

Состояние защищённости национальных интересов в информационной сфере, определяющихся совокупностью сбалансированных интересов личности, общества и государства составляет понятие информационной безопасности Российской Федерации.

Современные информационные системы должны в полной мере отвечать требованиям соответствующих стандартов и нормативно-правовых актов в сфере информационной безопасности.

СПИСОК ЛИТЕРАТУРЫ

1. **Вендров, А. М.** Проектирование программного обеспечения экономических информационных систем : учебник / А. М. Вендров. – М. : Финансы и статистика, 2005. – 544 с.
2. **Доктрина** информационной безопасности Российской Федерации (утв. Президентом РФ 09.09.2000 № Пр-1895) [Электронный ресурс]. – Режим доступа : http://base.consultant.ru/. – Загл. с экрана.
3. **Домарев, В. В.** Безопасность информационных технологий. Системный подход / В. В. Домарев. – К. : ООО ТИД Диа Софт, 2004. – 992 с.
4. **Одинцов, И. О.** Профессиональное программирование. Системный подход / И. О. Одинцов. – СПб. : БХВ-Петербург, 2004. – 624 с.
5. **Парижский, С. М.** Delphi. Учимся на примерах / С. М. Парижский ; под ред. Ю. А. Шпака. – К. : МК-Пресс, 2005. – 216 с. : ил.
6. Патрушев: более 90 млн атак совершено на информресурсы РФ с 2010 года. Опубликовано на Интернет-портале «РИАНОВОСТИ» [Электронный ресурс]. – Режим доступа : http://ria.ru/defense_safety/20141001/1026452834.html /. – Загл. с экрана.
7. **Путилин, А. Б.** Компонентное моделирование и программирование на языке UML : практическое руководство по проектированию информационно-измерительных систем / А. Б. Путилин, Е. А. Юрагов. – М. : НТ Пресс, 2005. – 664 с.
8. **Сорокин, А. В.** Delphi. Разработка баз данных / А. В. Сорокин. – СПб. : Питер, 2005. – 477 с. : ил.
9. **Стивенс, Р.** Delphi. Готовые алгоритмы / Р. Стивенс ; пер. с англ. П. А. Мерешука. – 2-е изд., – М. : ДМК Пресс; СПб. : Питер, 2004. – 384 с. : ил.
10. **Об информации** информационных технологиях и о защите информации : федер. закон от 27 июля 2006 г. № 149-ФЗ (в ред. от 21.07.2014 г. № 242-ФЗ) [Электронный ресурс]. – Режим доступа : http://base.consultant.ru/. – Загл. с экрана.
11. **Шендрик, А. И.** Информационное общество и его культура: противоречия становления и развития. Опубликовано на Информационном гуманитарном портале «Знание. Понимание. Умение» 2014 [Электронный ресурс] / А. И. Шендрик. – Режим доступа : http://www.zpu-journal.ru/e-zpu/2010/4/Shendrik. – Загл. с экрана.
12. **Шмуллер, Д.** Освой самостоятельно UML за 24 часа / Д. Шмуллер. – М. : Издательский дом «Вильямс», 2005. – 416 с.

ОГЛАВЛЕНИЕ

Учебное электронное издание

ПЛАТЁНКИН Алексей Владимирович
РАК Игорь Петрович
ТЕРЕХОВ Алексей Васильевич
ЧЕРНЫШОВ Владимир Николаевич

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ. ПРОЕКТНЫЙ ПРАКТИКУМ

Учебное пособие

Редактор Л. В. К о м б а р о в а
Инженер по компьютерному макетированию М. Н. Р ы ж к о в а

ISBN 978-5-8265-1409-2

Подписано к изданию 13.05.2015.
Тираж 100 шт. Заказ № 232

Издательско-полиграфический центр
ФГБОУ ВПО «ТГТУ»

392000, г. Тамбов, ул. Советская, д. 106, к. 14.
Тел./факс (4752) 63-81-08, 63-81-33.
E-mail: izdatelstvo@admin.tstu.ru