ОП

АКАДЕМИЯ
СОВРЕМЕННЫХ
ИНФОКОММУНИКАЦИОННЫХ
ТЕХНОЛОГИЙ

ОП Основы построения современных Инфокоммуникационных Систем

ОП.14 Архитектура информационных систем

ОП.14

(количество частей – 1, число страниц - 21)

Введение

Развитие компьютерных технологий всегда шло в двух основных направлениях: с одной стороны – вычисления, а с другой - накопление и обработка информации. Развитие вычислительной компоненты стимулировалось, главным образом, необходимостью проведения огромных объемов расчетов для создания ядерного оружия и ракетной техники.

Однако почти сразу на появление компьютеров отреагировал бизнес. В бизнесе не требуются большие объемы расчетов. Но в наиболее широко распространенных видах бизнеса (банковском, гостиничном, страховом, биржевом) основной проблемой всегда являлись объемы информации, которую необходимо собирать, надежно хранить и оперативно обрабатывать. Эти процессы осуществляются с помощью *информационных систем* (ИС), определение которых мы дадим ниже.

Следует заметить, что в области информационных технологий многие определения, термины, классификации носят достаточно условный характер. Это связано, как с отсутствием единого центра их стандартизации, так и с быстрым развитием самих технологий. Поэтому все попытки систематизировать, классифицировать, определить используемые понятия в той или иной степени носят условный и отчасти субъективный характер, что должно быть учтено при изучении представленного ниже материала.

Данное пособие состоит из двух информативных частей. В первой (раздел 2) представлены общие понятия ИС, их функции, задачи, принципы проектирования, а также основы функционирования баз данных, без чего невозможно понимание второй части пособия. Вторая часть (раздел 3) посвящена собственно архитектуре ИС.

1. Общее представление об информационной системе

1.1. Определение информационной системы

Совокупность определенным образом организованной информации на какую-то тему называется *базой данных* (БД), а тематика БД - ее *предметной областью*. Федеральный закон РФ от 27 июля 2006 года № 149-ФЗ «Об информации, информационных технологиях и о защите информации» определяет *информационную систему (ИС)*, как «совокупность содержащейся в БД информации и обеспечивающих ее обработку информационных технологий и технических средств».

Данное определение ИС далеко не единственное. Так международный стандарт ISO/IEC 2382-1 дает следующее определение: «Информационная система — система обработки информации, включающая связанные с ней ресурсы, такие как людские, технические и финансовые, предназначенная для обеспечения информацией и распространения информации». Российский ГОСТ РВ 51987 определяет ИС, как «автоматизированную систему, результатом функционирования которой является представление выходной информации для последующего использования».

Далее мы будем трактовать ИС в узком смысле слова, как совокупность БД и комплекса аппаратно-программных средств хранения, изменения, поиска

информации, а также взаимодействия с пользователем, что, в принципе, не противоречит ни одному из приведенных выше определений.

1.2. Специфика ИС

В зависимости от конкретной области применения ИС могут очень сильно различаться по своим функциям, архитектуре, реализации. Однако можно выделить, по крайней мере, два свойства, которые являются общими для всех ИС.

Во-первых, поскольку любая ИС предназначена для сбора, хранения и обработки информации, *в ее основе лежит среда хранения и доступа к данным*. Среда должна обеспечивать уровень надежности хранения и эффективность доступа, которые соответствуют области применения информационной системы.

Во-вторых, поскольку ИС ориентируются на конечного пользователя, который может не являться (и, как правило, не является) специалистом в области программирования и компьютерной техники, *ИС обязана обладать простым, удобным, легко осваиваемым интерфейсом*. Интерфейс ИС должен предоставить конечному пользователю все необходимые для его работы функции, но в то же время не дать ему возможность выполнять какие-либо лишние действия.

1.3. Задачи и функции ИС

Конкретные задачи, которые должны решаться ИС, зависят от той прикладной области, для которой предназначена система, но можно выделить некоторое количество задач, не зависящих от специфики прикладной области. Эти задачи связаны с общими чертами ИС.

Основные функции ИС:

- сбор, хранение и формальная обработка больших объемов информации;
- ведение совокупности данных сложной структуры;
- логическая и содержательная обработка информации в процессе решения функциональных задач;
- выдача информации в форме, удобной для принятия решений.

Функции ИС реализуются двумя *классами* функциональных задач: информационными и технологическими. *Информационные задачи* ИС обеспечивают переработку и представление информации, непосредственно используемой в процессах управления или принятия решений человеком. *Технологические задачи* связаны с актуализацией базы данных, поддержанием ее в целостном состоянии, эксплуатацией и настройкой информационной системы.

К ИС обычно предъявляются следующие технические требования:

- способность к изменениям и настройке на новые функциональные области;
- реакция системы на запросы пользователей в требуемый период времени;
- возможность расширения приложений и включения новых;
- технологичность эксплуатации и сопровождения системы;
- надежность функционирования;
- эффективность использования вычислительных ресурсов.

1.4. Функционально-блочный метод разработки ИС

Один из возможных путей построения ИС – использование так называемого *функционально-блочного подхода*, при котором для каждого приложения разрабатываются либо независимые функциональные продукты, либо используются готовые системы или пакеты, объединяемые с помощью специальных интерфейсных модулей. Эти модули отображают сложные структуры данных при переходе от одного функционального блока к другому и обеспечивают синхронизацию разнородных систем для обеспечения целостности данных. Разработка такого интерфейса — трудоемкая задача, а производительность работы функционально-блочной системы невелика в силу необходимости выполнения сложных преобразований данных.

При использовании функционально-блочного метода единая интегрированная модель предметной области для всех приложений не строится, и, соответственно отсутствует единая модель программного обеспечения (ПО). Прикладные программы, реализующие функциональные требования каждого блока, разрабатываются в отрыве друг от друга и часто в разное время.

Использование разнородных программных средств, построенных на основе различных методологических установок, не обеспечивает необходимого концептуального единства создаваемой информационной системы, что всегда отрицательно сказывается на ее основных характеристиках, в первую очередь на надежности, производительности, целостности, технологичности, простоте эксплуатации.

1.5. Базы данных в ИС

Альтернативой функционально-блочному методу реализации ИС служит подход, получивший в настоящее время широкое распространение. Он основан на применении интегрированного системного программного обеспечения, важнейшую компоненту в котором составляют *системы управления базами данных (СУБД)*. Именно через СУБД осуществляется взаимодействие пользователя с БД.

Основной функцией любой СУБД является поддержка независимости, целостности и непротиворечивости данных в условиях коллективного использования. Под *независимостью данных* понимается способность СУБД создавать различные представления об одних и тех же хранимых данных, остающиеся инвариантными к изменениям среды функционирования БД. При этом БД обеспечивает коллективное использование информации и необходимые условия для естественной эволюции существующих приложений ИС без их разрушения.

Благодаря концепции БД обеспечивается независимость описания предметной области и задач приложений от структур данных и методов их обработки, программ — от логической структуры базы данных, логической структуры данных — от методов их физической организации.

В информационных системах, использующих БД, можно:

- сделать программы ввода, модификации и поиска данных независимыми от программ содержательной обработки приложений;
- минимизировать объем хранимых данных путем исключения их дублирования;
- избежать противоречий в хранимых данных;
- обеспечить сохранность и целостность информации:

- многократно использовать одни и те же данные различными прикладными программами;
- обеспечить гибкость и адаптивность структуры данных к изменяющимся информационным потребностям пользователей;
- поддерживать адекватность базы данных моделируемой ПО;
- обеспечить защиту данных от несанкционированного доступа.

Концепция БД позволяет создавать интегрированные ИС, обеспечивающие:

- поддержание сложных и разнообразных структур объектов предметной области;
- содержание в своем составе большого числа типов данных, значительные объемы фактографической или текстовой информации;
- высокую достоверность обработки и хранения больших объемов данных.

1.6. Целостность данных и классическая транзакция

Крупные (корпоративные) ИС обязательно должны допускать возможность работы с нескольких рабочих мест. Причем, пользователи должны иметь возможность одновременно изменять содержимое БД (вводить, обновлять, удалять данные) и производить выборку из нее. Такая коллективная работа должна производиться согласованно, причем, согласованность действий должна обеспечиваться автоматически. Согласованность действий подразумевает, например, что оператор, формирующий отчеты, не сможет воспользоваться данными, которые начал, но еще не закончил формировать другой оператор. Точно также оператор, желающий обновить или удалить данные, не сможет выполнить операцию до тех пор, пока не закончится аналогичная операция над теми же данными, которую ранее начал, но еще не закончил другой оператор. При поддержке согласованности действий все результаты, получаемые от ИС, будут достоверны и непротиворечивы.

Требование согласованности действий привело к введению понятия *классической транзакции*, под которой понимается последовательность операций изменения БД и/или выборки из БД, воспринимаемая СУБД как единое целостное действие. Это означает, что при успешном завершении транзакции СУБД гарантирует наличие в БД результатов всех операций изменения, произведенных при выполнении транзакции. Условием успешного завершения транзакции является то, что БД находится в целостном состоянии. Если это условие не выполняется, то СУБД производит *полный откат транзакции*, т.е. ликвидацию в БД результатов всех операций изменения, произведенных при выполнении транзакции. Таким образом, БД будет находиться в целостном состоянии при начале любой транзакции и останется в целостном состоянии после успешного завершения любой транзакции. Понятие транзакции поддерживают все развитые СУБД.

СУБД может очень просто обеспечить выполнение транзакции, выполняя их последовательно при "параллельно" работающих операторах, даже если они вовсе не конфликтуют по данным. Но развитые СУБД стремятся максимально перемешивать запросы и операторы изменения БД, поступающие от разных транзакций, с тем лишь условием, что конечный результат выполнения всего набора транзакций будет эквивалентен результату их некоторого последовательного выполнения. Такая политика СУБД называется политикой *полной сериализации*

смеси транзакций.

Понятие транзакции актуально не только для многопользовательских СУБД. Оно применимо и к персональным СУБД, с которыми в любой момент времени работает только один пользователь. Во-первых, нарушение целостности БД может произойти при окончании транзакции. Во-вторых, может произойти аварийное выключение питания, в результате чего теряется содержимое основной памяти, в буферах которой могли находиться измененные, но еще не записанные во внешнюю память блоки БД. И, в-третьих, возможна авария внешнего носителя БД. Во всех трех случаях необходим откат транзакции. Традиционное решение - переписать на исправный внешний носитель архивную копию БД, после чего повторить операции всех транзакций, которые были выполнены после архивации, а затем выполнить откат всех транзакций, не закончившихся к моменту аварии.

В корпоративных ИС по естественным причинам часто возникает потребность в хранении данных в разных хранилищах. Например, иногда бывает удобно хранить некоторую часть информации как можно ближе к тем рабочим местам, в которых она чаще используется. Такие БД называются *распределенными* (они будут подробно рассмотрены в разделе 2.7). В этом случае при построении ИС приходится решать задачу согласованного управления распределенными информационными ресурсами (иногда применяя методы репликации данных). При однородном построении распределенной БД (на основе однотипных серверов баз данных) эту задачу обычно удается решить на уровне СУБД (большинство производителей развитых СУБД поддерживает средства управления распределенными базами данных). Если же система разнородна (т.е. для управления отдельными частями распределенной БД используются разные серверы), то приходится прибегать к использованию вспомогательных инструментальных средств интеграции разнородных БД.

Модели данных могут строиться по-разному. Если БД строится по принципу взаимосвязанных таблиц, она называется *реляционной*. Если один тип объекта является главным, а все нижележащие – подчиненными, то *иерархической*. Если же любой тип данных одновременно может быть и главным, и подчиненным, то такая база называется *сетевой*. Наиболее часто используются реляционные базы.

При разработке ИС важная роль отводится пользовательскому интерфейсу. Задача эргономичности интерфейса не формализуется. Построение интерфейса - это творческая задача, при решении которой нужно учитывать требования технической эстетики и эргономики, а также принимать во внимание особенности конкретной области применения ИС. Пользователи часто судят о качестве системы, исходя из качества ее интерфейса. От него же зависит и эффективность использования ИС в целом.

1.7. Этапы и стадии построения ИС

Самой первой проблемой построения ИС является проблема проектирования. Нельзя начинать техническую разработку, не имея тщательно проработанного проекта. Хотя строгая терминология в проектировании ИС до конца не разработана, можно выделить *три этапа проектирования: логический, физический, и проектирование интерфейсов*. Логический этап проектирования, свою очередь, делится на несколько стадий.

Первой стадией логического проектирования является анализ требований корпорации. Для этого на основе экспертных запросов необходимо выявить все актуальные и потенциальные потребности корпорации, которые должны удовлетворяться проектируемой ИС, понять какие потоки данных существуют внутри

корпорации, оценить объемы информации, которые должны поддерживаться и обрабатываться информационной системой.

Вторая стадия проектирования - выработка концептуальной схемы БД, которая будет лежать в основе ИС. Должна быть выбрана система нотаций, в которой будет представляться концептуальная схема. Концептуальное представление базы данных должно сохраняться как часть документации информационной системы на все время ее существования и будет использоваться при ее сопровождении и развитии.

Поскольку большинство БД является реляционными, *то третья стадия проектирования* состоит в выборе определений схемы реляционной базы данных в терминах языка SQL - универсального компьютерного языка, применяемого для создания, модификации и управления данными в реляционных БД.

На четвертой стадии должен быть осуществлен выбор архитектуры системы. В частности, очень важно решить, какой будет БД - централизованной или распределенной. Если принимается решение о распределенном характере БД, то необходимо произвести соответствующую декомпозицию набора определений схемы БД. Решение об архитектуре ИС возможно и до выработки общей реляционной схемы БД. Тогда декомпозиция производится на уровне концептуальной схемы, а затем для каждой отдельной части концептуальной схемы создается реляционная схема в терминах языка SQL.

Наиболее просто вопрос декомпозиции решается тогда, когда образующиеся разделы БД логически автономны. В терминах концептуальной схемы это означает, что между разделенными сущностями отсутствуют прямые или *транзитивные (через другие сущности) связи*. В терминах реляционной схемы: ни в одном разделе не присутствует таблица, ссылающаяся на таблицу, которая располагается в другом разделе. Если требование логической автономности компонентов распределенной БД выполнено, то дальнейшее проектирование можно производить для каждого компонента независимо. Если же разделы БД логически неавтономны, то придется учитывать конкретные возможности используемого серверного продукта, на чем мы далее останавливаться не будем и перейдем к физическому проектированию базы данных и проектированию интерфейсов. Эти две стадии могут выполняться параллельно.

Физическое проектирование включает две основных стадии, первая из которых, как правило, не зависит от особенностей выбранного серверного SQL-ориентированного продукта, а вторая зависит, причем критически. На первой стадии этого этапа определяется набор так называемых *индексов*. Индекс содержит в себе уникальные идентификаторы записей и дополнительную информацию об организации данных. Поэтому, если при выполнении запроса сервер или локальная СУБД обращается для отбора записи к индексу, то это занимает значительно меньше времени, т. к. понятно, что идентификатор гораздо меньше самой записи. Кроме этого, индекс "знает", как организованы данные и может ускорять обработку за счет группирования записей по сходным значениям параметров.

Для того, чтобы правильно выбрать набор индексов, необходимо еще раз тщательно проанализировать требования корпорации и оценить, какие запросы будут выполняться наиболее часто. Это непростая задача, но нужно учитывать, что создание индекса на большой заполненной таблице (когда информационная система уже функционирует) требует значительного времени.

Вторая стадия состоит в определении областей внешней памяти, в которых будут храниться фрагменты БД. По этому поводу вообще невозможно дать какие-

либо общие рекомендации, поскольку вопрос непосредственного размещения данных во внешней памяти является специфическим для каждой конкретной СУБД.

Параллельно с физическим проектированием БД ИС может проводиться проектирование и разработка интерфейса системы и ее обрабатывающей части. К этому моменту уже обычно известны требования, предъявляемые к интерфейсу, и функции самой ИС. На этой стадии производится выбор инструментальных средств, которые позволят достаточно быстро произвести эффективную реализацию.

В наше время нет необходимости программировать интерфейсные функции вручную. Целесообразнее использовать имеющиеся полуфабрикаты, определяемые вкусом и привычками разработчика, а также общей ориентацией проекта.

Что же касается обрабатывающих частей ИС, то все зависит от того, что они должны делать. Имеется масса примеров ИС, в которых вся обработка данных состоит в преобразовании их форматов при вводе и выводе, формировании отчетов и т.д. Такие функции можно фактически не программировать, поскольку существуют стандартные процедуры их автоматического генерирования по соответствующим описаниям. Но если требуется более сложная обработка данных, то в любом случае потребуется явное программирование, и в этом случае уже не столь важно, на каком языке программирование будет вестись.

1.8. Требования к техническим средствам ИС

Естественно, что требования к техническим средствам определяются требованиями к ИС в целом. Какие бы информационные возможности не требовались сотрудникам корпорации, окончательное решение всегда принимается ее руководством, которое корректирует требования к ИС и формирует окончательное представление об аппаратной среде. Конечно, нельзя не учитывать и влияние технических специалистов на мнение руководителя. Чем грамотнее составляется обоснование на приобретение технических средств, тем более обоснованным оказывается решение руководителя. Многое определяется возможными денежными затратами. Но следует отметить, что попытка изначально построить открытую систему потребует минимальных затрат на начальном этапе становления корпоративной ИС.

Выбор технических средств для построения корпоративной ИС - это непростая и неоднозначно решаемая задача, включающая технические, политические и даже эмоциональные аспекты. Она требует одинаковой тщательности выбора общей аппаратной архитектуры системы и конфигурации каждого из ее компонентов.

2. Классификация архитектур информационных систем

2.1. Общие положения

В первом разделе были кратко рассмотрены основные требования, которым должна удовлетворять ИС, и решаемые ею задачи. При этом подчеркивалось, что требования, предъявляемые к конкретным ИС, и спектр решаемых ими задач во многом зависит от целей, для достижения которых система разрабатывается. Соответственно, и архитектурные решения при проектировании и разработке ИС могут быть различными.

Уже отмечалось, что мировая терминология в области информационных систем вообще, и русскоязычная, в частности, окончательно не устоялась. Это в полной мере относится и к архитектуре ИС. Практически каждый год возникают новые технологии и архитектурные решения, для которых в маркетинговых целях

придумываются оригинальные названия, далеко не всегда точно отражающие смысл технологии и/или архитектуры.

Как и любая классификация, классификация архитектур ИС, изложенная в данном курсе, не является абсолютно жесткой. В архитектуре любой конкретной ИС часто можно найти элементы нескольких «классических» архитектур. А в различных литературных источниках можно найти различные принципы классификации архитектур ИС. Тем не менее, при архитектурном проектировании представляется необходимым иметь хотя бы частично формализованный архитектурный базис.

Ниже под *архитектурой ИС* мы будем понимать концепцию, определяющую модель, структуру, выполняемые функции и взаимосвязь компонентов ИС.

Для классификации архитектур ИС удобно разделить ее компоненты по выполняемым функциям на три слоя (иногда говорят о трех уровнях, звеньях – различных переводах английского слова «tier») - слои представления, бизнес логики и доступа к данным (рис. 1).

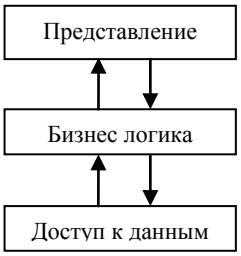

Рис. 1. Компоненты ИС

Слой представления – это все, что связано с взаимодействием пользователя и рабочей станции: нажатие кнопок, движение мыши, вывод на монитор изображений и результатов поиска и т.д.

Слой бизнес логики (приложений) – это реакция приложений на действия пользователя или на внутренние события, правила обработки данных. К нему относятся, например, формулы расчёта выплат по ссудам (в финансовых ИС), автоматизированная отправка сообщений руководителю проекта по окончании выполнения заданий подчиненными (в ИС управления проектами), отказ от отеля при отмене рейса авиакомпанией (в ИС туристического бизнеса) и т. д.

Слой доступа к данным – это все, что относится к данным, связанных с решаемой приложением прикладной задачей: их хранение, выборка, модификация и удаление.

2.2. Файл-серверная архитектура

Организация ИС на основе использования выделенных файл-серверов все еще достаточно распространена в связи с большим количеством разнотипных персональных компьютеров и сравнительной дешевизны их связывания в локальные сети. Такая организация является привлекательной, прежде всего, для не очень опытных в области системного программирования разработчиков ИС. Это объясняется тем, что при опоре на файл-серверные архитектуры сохраняется автономность прикладного (и большей части системного) программного обеспечения, работающего на каждом персональном компьютере сети. Своим появлением файл-серверные приложения обязаны появлению локальных вычислительных сетей (ЛВС), по которым передавались файлы. Сначала в сети все

компьютеры были равноправны (т.н. *одноранговые сети*). Потом возникла естественная идея хранения всех общедоступных файлов на выделенном компьютере в сети, который и получил название *файл-сервера*.

В файл-серверной архитектуре компоненты ИС, выполняемые на разных персональных компьютерах, взаимодействуют только за счет наличия общего хранилища файлов, которое размещается на файл-сервере. В классическом случае на каждом персональном компьютере дублируются не только прикладные программы, но и средства управления базами данных. Таким образом, файл-сервер представляет собой просто расширение дисковой памяти, доступное всем персональным компьютерам ИС (рис.2)

Достоинства файл-серверной архитектуры:

1. Простота организации. Проектировщики и разработчики информационной системы могут работать на привычных им IBM PC в среде, например, MS-DOS или Windows. Имеются удобные и развитые средства разработки графического пользовательского интерфейса, простые в использовании средства разработки систем баз данных и/или СУБД. Отсюда непосредственно следует и низкая стоимость разработки.

2. Возможность работы одновременно нескольких пользователей (многопользовательский режим работы с данными). Однако (см. ниже) из-за необходимости передавать по сети файлы целиком для последующей их обработки на клиентском месте число пользователей обычно ограничено пропускной способностью сети.

3. Удобство централизованного управления доступом - свойство, непосредственно связанное с простотой организации.

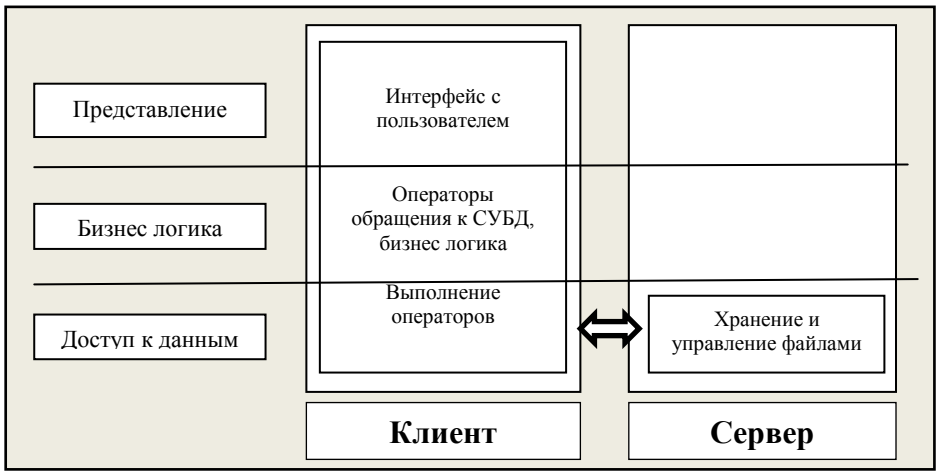

Рис. 2. ИС в архитектуре "файл-сервер"

Но простота организации порождает и *недостатки данной архитектуры:*

1. Перегрузка трафика, обусловленная тем, что для выборки полезных данных необходимо просмотреть на стороне клиента весь соответствующий файл целиком. В этой архитектуре мы имеем "толстого" клиента и очень "тонкий" сервер в том смысле, что почти вся работа выполняется на стороне клиента, а от сервера требуется только достаточная емкость дисковой памяти.

2. Децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным. Такое решение снижает надежность приложения.

3. Слабые возможности расширения.

Из сказанного следует, что простое, работающее с небольшими объемами информации и рассчитанное на применение в однопользовательском режиме, файл-серверное приложение можно спроектировать, разработать и отладить очень быстро. Оно целесообразно для использования в небольшой компании, где для ведения, например, кадрового учета достаточно иметь изолированную систему, работающую на одном выделенном персональном компьютере. Хотя и в этом случае требуется большая аккуратность конечных пользователей для надежного хранения и поддержания целостного состояния данных. Однако, в уже ненамного более сложных случаях (например, при организации информационной системы поддержки проекта, выполняемого группой) использование файл-серверной архитектуры становится нерациональным.

2.3. Клиент-серверная двухслойная архитектура

Под клиент-серверной архитектурой ИС понимают такое ее построение, при котором на выделенном сервере хранится не только сама БД, но и некоторая часть приложений, обеспечивающих взаимодействие с клиентом. При двухслойной реализации эта архитектура представлена на Рис. 3. В этом случае:

— на стороне клиента выполняется часть приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты и выполняющие другие специфичные для приложения функции.

— клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления БД, которая, фактически, является индивидуальным представителем СУБД для приложения.

Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных (физической схемы данных). В результате клиентские программы манипулируют данными на уровне слоя бизнес-логики (рис. 3).

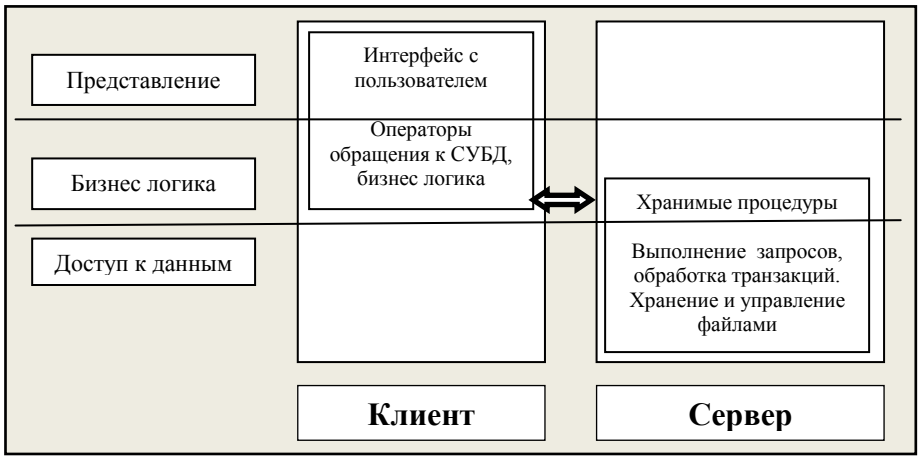

Рис. 3. ИС в архитектуре "клиент-сервер", двухслойная архитектура

Использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных. Такие ИС независимы от аппаратной (а часто и программной) части сервера БД и поддерживают графический интерфейс пользователя на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений в этом случае существенно сокращаются. Клиент становится «тоньше», но не намного, а сервер соответственно немного «толстеет».

Основные *особенности* данной архитектуры:

— клиентская программа работает с данными через запросы к серверному ПО.

— базовые функции приложения разделены между клиентом и сервером.

Достоинства:

1. Полная поддержка многопользовательской работы.

2. Гарантия целостности данных.

Недостатки:

1. Бизнес логика приложений осталась в клиентском ПО. При любом изменении алгоритмов, надо обновлять пользовательское ПО на каждом клиенте.

2. Достаточно высокие требования к пропускной способности коммуникационных каналов с сервером.

3. Слабая защита данных от взлома, в особенности от недобросовестных пользователей системы.

4. Высокая сложность администрирования и настройки рабочих мест пользователей системы.

5. Необходимость использования мощных персональных компьютеров на клиентских местах.

6. Высокая сложность разработки системы из-за необходимости выполнять бизнес-логику и обеспечивать пользовательский интерфейс в одной программе.

Нетрудно заметить, что большинство недостатков классической 2-х слойной архитектуры «клиент-сервер» проистекают от использования клиентской станции в качестве исполнителя бизнес логики ИС. Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении. При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть.

2.4. Клиент-серверная архитектура, переходная к трехслойной (2.5 слоя)

Использование хранимых процедур и вычисление данных на стороне сервера сокращают трафик, увеличивают безопасность. Однако на клиентской рабочей станции все равно приходится реализовать часть бизнес логики, поскольку не удается написать всю бизнес-логику приложения на не предназначенных для этого встроенных языках СУБД. Поскольку, с одной стороны, часть бизнес-функций все-

таки удается перенести на сервер, но с другой стороны, физически такие системы состоят из двух компонентов, эту архитектуру часто называют *2.5-слойный клиент-сервер (рис. 4).*

В отличие от 2-слойной архитектуры 2.5-слойная *обладает следующими достоинствами:*

1. Она обычно не требует наличия высокоскоростных каналов связи между клиентской и серверной частями системы, так как по сети передаются уже готовые результаты работы с данными - почти полностью эта работа производится на стороне сервера.

2. Существенно улучшается защита информации, поскольку пользователям даются права на доступ к функциям системы, а не на доступ к ее данным.

Однако вместе с преимуществами данного подхода архитектура 2.5-слоя перенимает и все его *недостатки*, а именно:

1. Ограниченную масштабируемость.

2. Зависимость от программной платформы.

3. Ограниченное использование сетевых вычислительных ресурсов.

4. Поскольку программы для серверной части системы пишутся на встроенных в СУБД языках описания хранимых процедур, предназначенных для валидации данных (подтверждения соответствия полученных данных запросу) и построения несложных отчетов, а вовсе не для написания ИС масштаба предприятия, снижается быстродействие системы, повышается трудоемкость создания модификаций ИС. В результате повышается стоимость аппаратных средств, необходимых для ее функционирования.

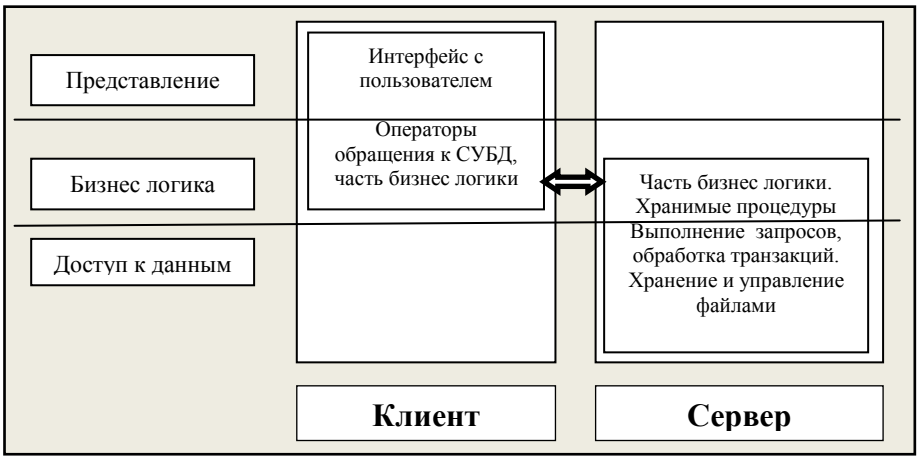

Рис. 4. ИС в архитектуре "клиент-сервер", архитектура в 2,5 слоя

2.5. Клиент-серверная трехслойная архитектура

Для того, чтобы сделать клиентов истинно "тонкими" и для повышения общей эффективности системы, применяются трехслойные архитектуры "клиент-сервер". В этой архитектуре, кроме клиентской части системы и сервера базы данных, вводится промежуточный сервер приложений (рис. 5).

На стороне клиента выполняются только интерфейсные действия (очень «тонкий» клиент), а вся логика обработки информации поддерживается в сервере

приложений.

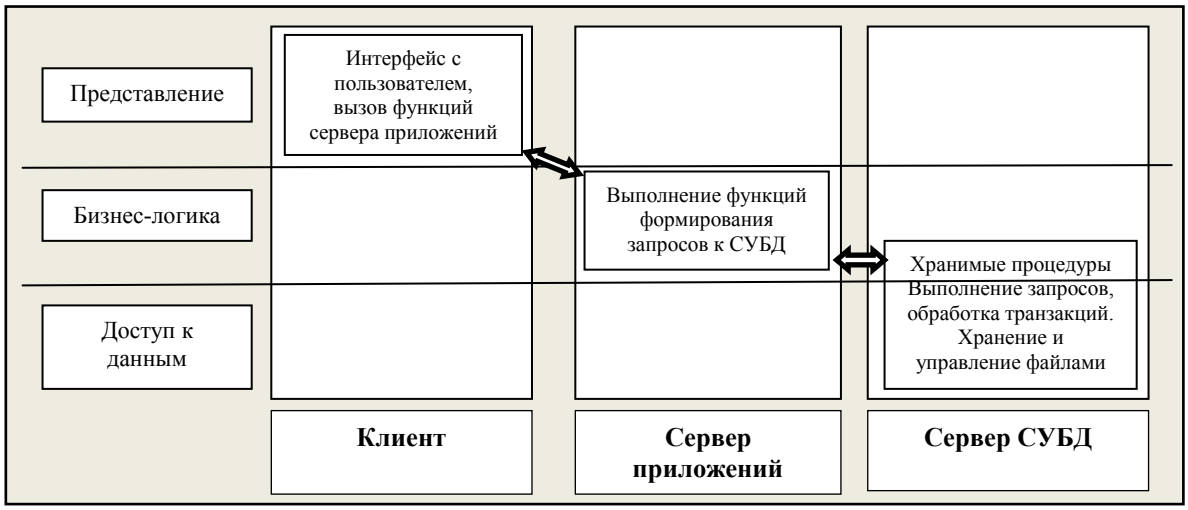

Рис. 5. ИС в архитектуре "клиент-сервер", трехслойная архитектура

Такая организация системы весьма напоминает организацию первых больших вычислительных машин (мэйнфреймов) с той лишь разницей, что роль мэйнфрейма выполняют сервера приложений и БД, а на пользовательском месте стоит не терминал, а персональный компьютер, обеспечивающий графический интерфейс пользователя. Но надо учесть, что возврат организации ИС по типу мэйнрейма произошел на ином технологическом уровне и сохранил только внешние черты этой организации. В качестве клиентских интерфейсных программ в настоящее время очень эффективно применяются стандартные интернет-браузеры. Обязательным стало использование СУБД со всеми их преимуществами. Программы для серверной части пишутся, в основном, на специализированных языках, пользуясь механизмом хранимых процедур сервера БД.

Отметим, важную особенность данной архитектуры: каждый из слоев ИС реализуется на своих аппаратных средствах: слой представления – на рабочих станциях с «тонким» клиентом, слой бизнес логики – на сервере приложений и слой доступа к данным – на сервере БД.

Интерфейс между клиентской частью приложения и клиентской частью сервера БД, как правило, основан на использовании языка SQL. Поэтому такие функции, как, например, предварительная обработка форм, предназначенных для запросов к БД, или формирование результирующих отчетов выполняются в клиентской части приложения. Кроме того, клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу БД, передавая ему текст оператора языка SQL. На стороне сервера БД в продуктах практически всех компаний происходит прием от клиента текста оператора на языке SQL.

Из сказанного следует, что в клиент-серверной трехслойной архитектуре клиенты могут быть минимально "тонкими", а сервер остается достаточно "толстым", чтобы удовлетворить потребности всех клиентов.

Вместе с тем, присущая архитектуре "клиент-сервер" необходимость

обращения от клиента к серверу при каждом запросе не всегда удобна. На практике часто встречаются ситуации, когда для эффективной работы отдельной клиентской составляющей ИС в действительности достаточна лишь небольшая часть общей БД. Это приводит к идее поддержки локального ***кэша*** общей БД на стороне каждого клиента, где под кэшем понимается промежуточная буферная память с быстрым доступом, содержащая информацию, запрашиваемую с наибольшей вероятностью.

Фактически, концепция локального кэширования БД является частным случаем концепции реплицированных БД. Для поддержки локального кэша БД программное обеспечение рабочих станций должно содержать компонент управления базами данных - упрощенный вариант сервера баз данных. Отдельной проблемой является обеспечение согласованности (когерентности) кэшей и общей БД. Здесь возможны различные решения - от автоматической поддержки согласованности за счет средств базового программного обеспечения управления БД до полного перекладывания этой задачи на прикладной уровень. При организации локальных кэшей клиенты становятся более «толстыми», но сервер «тоньше» не делается.

Подводя итог сказанному, отметим достоинства и недостатки трехслойной архитектуры клиент-сервер.

Достоинства:

1. Решение проблемы, связанной с переустановкой клиентских частей программ на многих, в том числе, удаленных, компьютерах при модернизации системы за счет «тонкого» клиента. Использование «тонкого» клиента повышает также информационную безопасность и надежность ИС.

2. Наилучшая среди всех архитектур масштабируемость, как горизонтальная, так и вертикальная; горизонтальная масштабируемость обусловлена тем, что одна и та же система может работать как на одном отдельно стоящем компьютере, выполняя на нем программы СУБД, сервера приложений и клиентской части, так и в сети, состоящей из сотен и тысяч машин. Вертикальная масштабируемость, т.е. расширение функциональных возможностей ИС обусловлена тем, что она не требует замены сервера приложений на новый при добавлении новой функции.

3. Между клиентской программой и сервером приложения передается лишь минимально необходимый поток данных - аргументы вызываемых функций и возвращаемые от них значения, что является теоретическим пределом эффективности использования линий связи.

4. Сервер приложения ИС может быть запущен в одном или нескольких экземплярах на одном или нескольких компьютерах, что позволяет использовать вычислительные мощности организации столь эффективно и безопасно, как этого пожелает администратор ИС.

5. Дешевый трафик между сервером приложений и СУБД; трафик между сервером приложений и СУБД может быть большим, однако это всегда трафик локальной сети, пропускная способность которой достаточно велика и дешева; при запуске сервера приложений и СУБД на одной машине сетевой трафик сводится к нулю.

6. Снижение нагрузки на сервер БД по сравнению с 2.5-слойной схемой, а значит и повышение скорости работы системы в целом.

Недостатки:

1. Архитектура "клиент-сервер" требует, как правило, более мощных и, следовательно, дорогих аппаратных средств, чем архитектура "файл-сервер".

2. Выше расходы на администрирование и обслуживание серверной части.

2.6. Архитектуры на основе технологии Intranet

Возникновение и внедрение в широкую практику глобальной сети Internet естественным образом повлияло на технологию создания корпоративных ИС, породив направление, известное теперь под названием Intranet. *Информационная Intranet-система* - это корпоративная система, в которой используются методы и технологии Internet. Такая система может быть локальной, изолированной от остального мира Internet, или опираться на виртуальную корпоративную подсеть Internet. В последнем случае особое значение приобретают средства защиты корпоративной информации от несанкционированного доступа, которые выходят за рамки данного курса.

Хотя в общем случае в Intranet-системе могут использоваться различные службы Internet (e-mail, ftp, WWW и т.д.), на практике чаще всего используется гипермедийная служба WWW (World Wide Web). Для этого имеются две основные причины. Во-первых, с использованием языка гипермедийной разметки документов HTML можно сравнительно просто разработать удобную для использования информационную структуру, которая в дальнейшем будет обслуживаться одним из готовых Web-серверов. Во-вторых, наличие нескольких готовых к использованию клиентских частей - браузеров, избавляет от необходимости создавать собственные интерфейсы с пользователями, предоставляя им удобные и развитые механизмы доступа к информации. В ряде случаев такая организация корпоративной ИС (рис. 6) оказывается достаточной для удовлетворения потребностей компании.

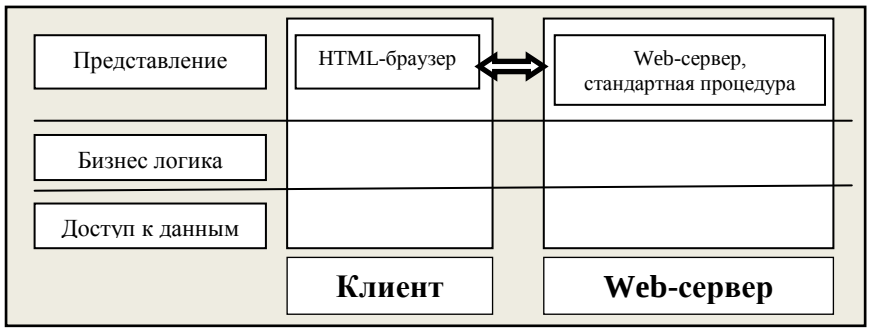

Рис. 6. Простая архитектура Intranet-системы

Однако, при всех своих преимуществах (простота организации, удобство использования, стандартность интерфейсов и т.д.) эта схема обладает сильными ограничениями. Прежде всего, как видно из

Рис. 6, в ИС отсутствует прикладная обработка данных. Пользователь может только просмотреть информацию, поддерживаемую Web-сервером. Гипертекстовые структуры трудно модифицируются. Для того, чтобы изменить наполнение Web-сервера, необходимо приостановить работу системы, внести изменения в HTML-описания и только затем продолжить нормальное функционирование. И, наконец, далеко не всегда достаточен поиск информации в стиле просмотра гипертекста. Часто необходимы БД и соответствующие средства выборки данных.

Что касается логики приложения, то при использовании Web-технологии существует возможность ее реализации на стороне Web-сервера. Для этого могут использоваться два подхода - CGI (Common Gateway Interface) и API (Application Programming Interface). Оба подхода основываются на наличии в языке HTML специальных конструкций, информирующих клиента-браузера, что ему следует послать Web-серверу специальное сообщение, при получении которого сервер должен вызвать соответствующую внешнюю процедуру, получить ее результаты и вернуть их клиенту в стандартном формате (рис. 7).

Не останавливаясь подробно на различиях подходов CGI и API, отметим, что подход CGI является более надежным (внешняя программа выполняется в отдельном адресном пространстве), но менее эффективным, чем подход API (в этом случае внешние процедуры компонуются совместно со стандартной частью Web-сервера).

Аналогичная техника широко используется для обеспечения унифицированного доступа к БД, которые также могут быть включены в Intranet-системы. Язык HTML позволяет вставлять в гипертекстовые документы формы. Когда браузер натыкается на форму, он предлагает пользователю заполнить ее, а затем посылает серверу сообщение, содержащее введенные параметры. Как правило, к форме приписывается некоторая внешняя процедура сервера.

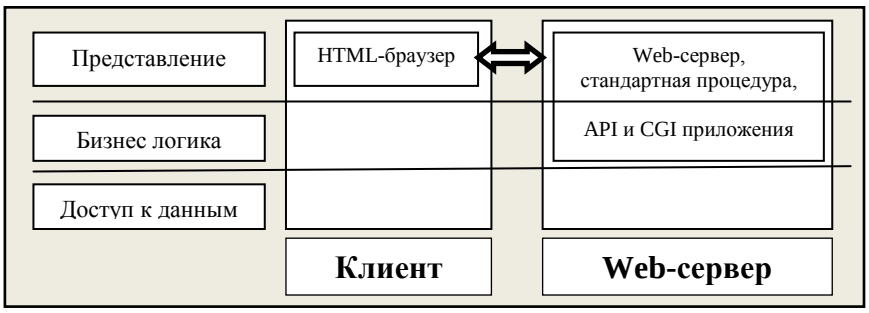

Рис. 7. Архитектура Intranet-системы, включающая логику приложений

При получении сообщения от клиента сервер вызывает эту внешнюю процедуру с передачей параметров пользователя. Понятно, что такая внешняя процедура может, в частности, играть роль шлюза между Web-сервером и сервером БД. В этом случае параметры должны специфицировать запрос пользователя к БД. В результате получается конфигурация ИС, схематически изображенная на рис. 8.

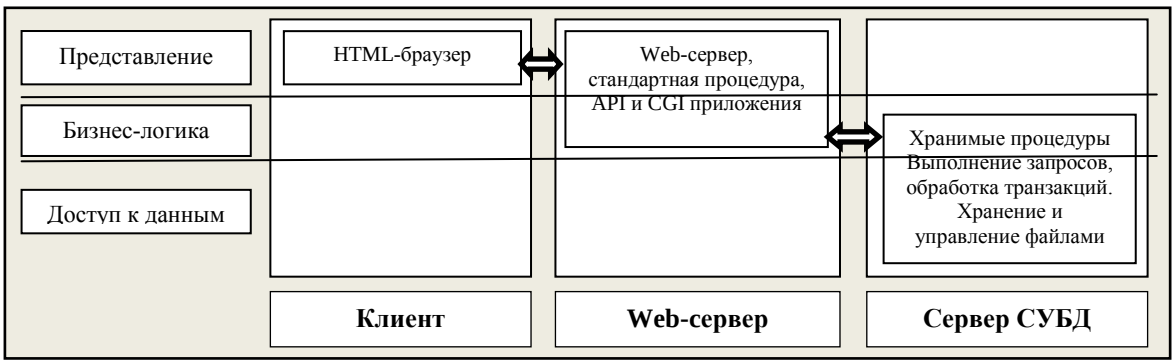

Рис. 8. Архитектура Intranet-системы, включающая логику приложений и сервер БД

На принципах использования внешних процедур основывается также возможность модификации документов, поддерживаемых Web-сервером, а также

создание временных "виртуальных" HTML-страниц.

При создании приложений на основе Intranet-технологий используется язык Java - интерпретируемый объектно-ориентированный язык программирования, созданный на основе языка Си++. Специальные прикладные программы, написанные на языке Java (апплеты), выполняются в веб-браузере с использованием виртуальной Java машины (Java Virtual Mashin) и играют роль шлюза к серверу БД. Апплеты могут быть также привязаны в HTML-документу. В этом случае они поступают на сторону клиента вместе с документом и выполняются либо автоматически, либо по явному указанию. При применении подобной техники доступа к БД схема организации Intranet-системы становится такой, как показано на рис. 9.

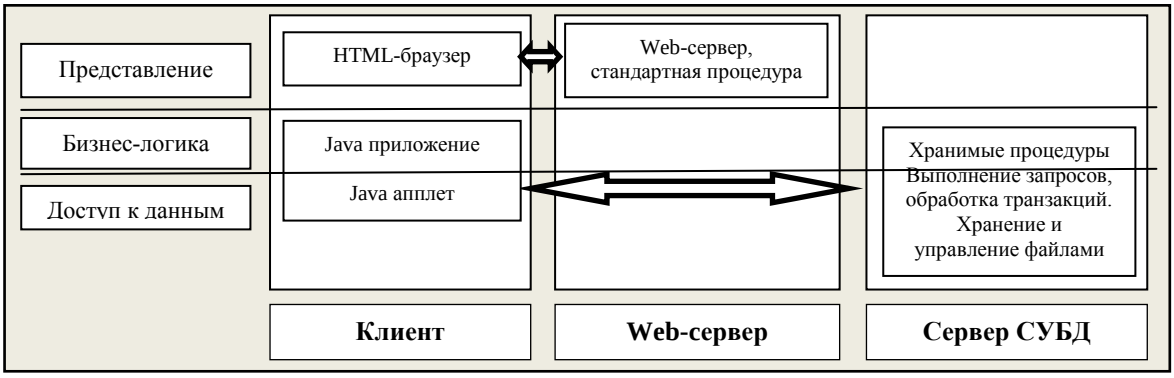

Рис. 9. Доступ к базе данных со стороны клиента Intranet-системы

Intranet является удобным и мощным средством разработки и использования ИС. Единственным относительным *недостатком* такого подхода можно считать постоянное изменение (в сторону совершенствования) самой технологии и естественное отсутствие стандартов. С другой стороны, если ИС создана с использованием текущего уровня технологии и удовлетворяет потребности корпорации, то нет необходимости что-то менять в системе по причине появления более совершенных технологий.

2.7. Распределенные информационные системы

Во Введении отмечалось, что в области информационных технологий часто используется неустановившаяся терминология, а классификации далеко не всегда однозначны. Это в полной мере относится и к распределенным ИС. В литературе можно найти различные определения распределенных систем, причем большинство из них нельзя считать удовлетворительным, и каждое из них не согласуется с остальными. Крайней точкой зрения можно считать позицию, когда распределенной ИС называют любую ИС, построенную более, чем на одном аппаратном средстве. С этой точки зрения двухслойную архитектуру «клиент-сервер» уже следует считать распределенной даже при наличии одного сервера БД.

Более правильным представляется определение распределенной ИС, как набора независимых ИС со своими базами данных, которые представляются пользователям единой объединенной системой. При этом выделяется два момента:

— все вычислительные машины работают автономно;

— пользователи воспринимают распределенную ИС, как единую систему.

Из последнего условия следует, что БД распределенной ИС функционируют и обмениваются данными по единым, централизованно определенным правилам

Распределенные ИС характеризуются следующими особенностями:

— от пользователей скрыты различия между компьютерами и способы связи между ними;

— пользователи и приложения единообразно работают в распределенных системах, независимо от того, где и когда происходит их взаимодействие.

Важное достоинство распределенных ИС - практически неограниченные возможности горизонтального масштабирования. Это является прямым следствием их построения из независимых ИС.

Хотя распределенные системы обычно существуют достаточно долго, отдельные их части могут временно выходить из строя. Пользователи и приложения не должны уведомляться о том, что отдельные части системы заменены, отремонтированы или в систему добавлены новые части для поддержки дополнительных пользователей. Для того чтобы поддержать представление системы в едином виде, организация распределенных систем часто включает в себя дополнительный слой программного обеспечения, находящийся между слоем интерфейса с пользователем и слоем управления данными.

Этот промежуточный слой (middleware) выполняет функции управления транзакциями и коммуникациями, транспортировки запросов, управления именами и множество других. Это важный компонент распределенных систем. Существует фундаментальное различие между технологией клиент-сервер и трехслойной технологией распределенной ИС. В первом случае клиент явным образом запрашивает данные, зная структуру БД. Клиент передает СУБД запрос, в ответ получает данные, т.е. имеет место жесткая связь типа "точка-точка", для реализации которой все СУБД используют SQL-канал. В трехслойной распределенной ИС клиент явно запрашивает один из сервисов (предоставляемых прикладным компонентом), передавая ему некоторое сообщение и получая ответ также в виде сообщения. Клиент направляет запрос в информационную шину, ничего не зная о месте расположения сервиса. Имеет место так называемая поставка функций клиенту. При этом важно, что для клиента БД закрыта слоем сервисов. Более того, он вообще ничего не знает о ее существовании, так как все операции над БД выполняются внутри сервисов.

Таким образом, в первом случае мы имеем жесткую схему связи "точка-точка" с передачей открытых запросов и данных, исключающую возможность модификации и работающую только в синхронном режиме "запрос-ответ". Во втором случае определен гибкий механизм передачи сообщений между клиентами и серверами, позволяющий организовывать взаимодействие между ними многочисленными способами.

Существует еще один важный аспект использования распределенных ИС. В разделах 2.5, 2.6 говорилось о достоинствах тонких клиентов. Но на практике в качестве клиентских рабочих станций сегодня часто используются достаточно мощные компьютеры. Современный ноутбук обладает мощностью, которой несколько лет назад обладал сервер, полностью удовлетворявший запросы крупной корпорации. Использование такого ноутбука в качестве тонкого клиента нерационально. Целесообразнее хранить часто используемые данные на самих компьютерах, организовав обмен между ними исправлениями и дополнениями к

хранящимся данным (аналог кэш-памяти, рассмотренной в разделе 3.5). При этом суммарный трафик резко снижается, что позволяет понизить требования и к каналам связи между компьютерами. Благодаря этому можно создавать надежно функционирующие распределенные ИС, использующие не очень надежные каналы связи - Интернет, мобильную телефонную сеть, коммерческие спутниковые каналы. Причем, минимизация трафика между элементами делает вполне доступной стоимость эксплуатации такой связи. Реализация такой системы требует, разумеется, решения ряда проблем, одна из которых своевременная синхронизация данных.

В разделе 2.4 отмечалось одно из достоинств архитектуры 2,5 слоя. Поскольку пользователям предоставлялись права доступа к функциям системы, а не к данным, обеспечивался высокий уровень защиты информации. Распределенная ИС также обеспечивает повышенный уровень защиты информации, но по иной причине. Дело в том, что данные, хранящиеся на конкретной рабочей станции, при использовании средств шифрования, паролей и личных аппаратных ключей доступны только ее пользователю. Таким образом, исключается доступ к данным посторонних, в том числе и администраторов системы.

Распределенная архитектура ИС также позволяет распределить между равноправными клиентскими рабочими станциями не только БД, но и сами вычисления, если их объемы превышают возможности одной рабочей станции. Такое распределение особенно эффективно в том случае, если задача, требующая больших вычислений, распределяется между рабочими станциями, обладающими одной и той же информацией в своих БД, что позволяет добиться максимальной производительности ИС в целом.

Таким образом, построенные на основе распределенной архитектуры ИС для предприятий малого и среднего бизнеса будут обеспечивать надежность эксплуатации, безопасность информации и высокую скорость вычислений, что в первую очередь и требуется от любой корпоративной ИС.

Заключение

Технология ИС, как и информационные технологии вообще развиваются чрезвычайно быстро. Усложняется структура их организации, совершенствуются аппаратные средства, появляются принципиально новые возможности, как, например это произошло с появлением технологии Internet/Intranet. Поэтому некоторые понятия в этой динамичной сфере устаревают быстрее, чем их удается классифицировать или стандартизовать. Тем не менее, такие попытки при всей их условности и несовершенстве должны регулярно предприниматься, хотя бы для того, чтобы упростить изучение ИС. Одна из таких попыток и предпринята в данном пособии.

Литература.

1. Кузнецов С. Д. Проектирование и разработка корпоративных информационных систем.: М. Центр Информационных Технологий, 1998.

2. Коннолли, Т., Бегг, К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. — М.: Издательский дом «Вильяме», 2003.

3. Дейт К. Дж. Введение в системы баз данных.: Пер. с англ. – 6-е изд. – Киев: Диалектика, 1998.

4. Попов И.Г., Мамонов С.Г. Информационные системы. М.: Инфра, 2007.

5. Петров В.Н. Информационные системы.: Питер, 2008.

6. Докучаев В.А., Беленькая М.Н., Яковенко Н.В. Основы сетевых технологий. Вводный курс. Часть 1, http://www.acikt.ru/component/jshopping/osnovy-postroeniya-sovremennykh-infokommunikatsionnykh-sistem/vvodnyj-kurs-osnovy-setevykh-tekhnologij.html?Itemid=101

7. Беленькая М.Н., Докучаев В.А., Малиновский С.Т., Яковенко Н.В. Основы сетевых технологий и высокоскоростной передачи данных. Вводный курс. Часть 2, http://www.acikt.ru/component/jshopping/osnovy-postroeniya-sovremennykh-infokommunikatsionnykh-sistem/osnovy-setevykh-tekhnologij-i-vysokoskorostnoj-peredachi-dannykh.html?Itemid=101

Автор: д.ф-м.н., профессор Кальфа А.А.