Содержание:

1. Введение.

Область использования информационных систем каждый день расширяется, и усложняются. Кое-какие системы развиваются так, что приобретают мировой масштаб, и от их правильного и безотказного функционирования зависит работа огромного количества людей. Вместе со сложностью и масштабностью систем растут и требования к ним. Данная система не является чем-то новым или недавно придуманным, два-три десятилетия назад при разработке информационных систем популярной была модель "хост-компьютер + терминалы", реализованная на базе мэйнфреймов (например, IBM-360/370 либо же российских аналогов - компьютеров серии ЕС ЭВМ), или на базе так называемых мини-ЭВМ (например, PDP-11, ещё имевших отечественный аналог - СМ-4). . Эта модель обладала бесспорными по тем временам достоинствами. Во-первых, пользователи такой системы использовали различные ресурсы хост компьютера и дорогостоящие для тех времен периферийные устройства (принтеры, графопостроители, дисковые накопители и т.д.). Также проектирование и разработка таких систем вызывает высокую сложность, а методы и средства, отличаются от принятых при разработке "монолитных" систем. Свойственной особенностью подобных систем была абсолютная "неинтеллектуальность" терминалов, применяемых в качестве рабочих мест - их работой управлял все тот же хост-компьютер. Используемое программное обеспечение работало только с "локальными" ресурсами компьютера. Правда, часть этих ресурсов была уже "псевдолокальной", например, файлы на сетевом диске. Файл обрабатывается самим узлом. И через какое-то время было ясно, что классические методы не работают. Начавшийся высокий рост индустрии персональных компьютеров вначале мало что изменил в принципе построения программных систем - большинство программ по-прежнему работали с локальными ресурсами. С увеличением объема перерабатываемых данных, и с возрастанием их стоимости стало понятно, что обрабатывать их на клиентских машинах больше нельзя. Любая ошибка на них ,а с количеством клиентов, растет вероятность ошибки, приводит к потере данных, либо к их блокировкам во время работы, а следовательно приводит к снижению всей производительности системы. Следующим главным шагом стало глобальное распространение технологии клиент-серверной обработки. Это были "двухролевые" системы: клиент работал

отображением пользовательского интерфейса и выполнение кода приложения, а работа сервера поручалась СУБД. В применении к примеру с файлом переход к клиент-серверной архитектуре может быть изображен следующим образом: вместо того, чтобы читать файл целиком и обрабатывать его, клиент отправляет серверу запрос, в котором указывает, как файл должен быть обработан. Глобальный переход на технологию "клиент-сервер" помог решить много старых проблем, но, как это часто случается, создал много новых. Часто перенос части задач на сервер, может повлиять в негативную сторону на общей производительности системы, и наоборот, перенос части нагрузки на клиента может привести к потере централизации. Часто встречающаяся трудность была и всё ещё остается определение границ функционала между клиентом и сервером. По мере роста популярности систем "клиент-сервер" вместе с ней набирала силу технология объектно-ориентированного программирования, которая предлагала перейти к системной архитектуре с тремя слоями: слой представления отводится пользовательскому интерфейсу, слой предметной области предназначен для описания основных функций приложения, необходимых для достижения поставленной перед ним цели, а третий слой представляет источник данных. С появлением Web технологий пользователи были заинтересованы в системе "клиентсервер", где в роли клиента выступал бы Web-браузер. В наше время ещё считают, что бум технологий, связанных с клиент-серверной архитектурой, все еще продолжается – большинство работающих в настоящее время информационных систем выполнено в этой технологии. Но актуальными являются направления - трехслойные и многослойные, а также децентрализованные приложения. Очень важна правильная и четкая организация информационных бизнесрешений это главный фактор успеха любой компании, особенно важным это является для предприятий среднего и малого бизнеса, которым крайне важна система, которая способна предоставить весь объем бизнес-логики для решения задач компании. В то же время, такие системы для компаний со средним и малым масштабом сетей часто попадают под критерий —цена - качество, то есть должны обладать максимальной производительностью и надежностью при доступной цене. Опыт последних лет разработки программного обеспечения (ПО) показывает, что архитектура информационной системы должна выбираться с учетом нужд бизнеса, а не личных пристрастий разработчиков. Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture).

1. Архитектура "клиент-сервер".

Архитектура информационной системы - концепция, характеризующая модель, структуру, производимые функции и связь компонентов информационной системы. Клиент-сервер (Client-server) — вычислительная или сетевая архитектура, в которой задания либо сетевая нагрузка распределены среди сервером и клиентами. Сервер — это программа, исполняющая запросы отправленные со стороны клиентов на получение ресурсов определенного вида. Клиент — это программа, использующая услугу, представляемую программой сервера.

Зачастую пользователи называют клиентом или сервером компьютер, на котором работает какая-либо из этих программ. По сути, клиент и сервер — это роли, исполняемые программами. Клиенты и сервера физически имеют возомжность находиться на одном компьютере. Одна и та же программа способна быть и клиентом, и сервером одновременно.

Клиент-серверная система характеризуется присутствием двух взаимодействующих друг с другом независимых процессов - клиента и сервера, которые, имеют возможность работать на различных компьютерах, обмениваясь данными по сети. По такой подобной схеме могут быть построены системы обработки данных на базе СУБД, почтовые и другие системы. Файл-серверная система так же пользуется технологией клиент-сервер, но с точки зрения архитектуры прикладных программ важным является то, какого рода ресурсы предоставляет клиентам сервер. В файл-серверной системе данные сохраняются на файловом сервере (например, Novell NetWare или Windows NT Server), а их обработка исполняется на рабочих станциях, на коих, работает одна из "настольных СУБД" - Access, FoxPro, Paradox и т.п. Приложение на рабочей станции делает следующие задачи как - составление пользовательского интерфейса, логическая обработка данных и конкретное манипулирование данными. Файловый сервер выполняет задачи самого невысокого уровня - открытие, закрытие и изменение файлов, но не базы данных. База данных находится только в памяти рабочей станции. И так добивается конкретного управления данными несколькими автономными и несогласованными между собой процессами. Не считая того, что для всякой обработки все данные нужно передать по сети с сервера на рабочую станцию. Понятие архитектуры клиент-сервер в системах управления предприятием связано с разделением всякой прикладной программы на три ведущих компонента или же слоя. Этими тремя компонентами считаются:

- компонент представления (визуализации) данных;
- компонент прикладной логики;
- компонент управления базой данных.

2.Варианты архитектуры "клиент-сервер".

Двухуровневая клиент-серверная архитектура.

Данная архитектура возымела высокое распространение в начале 1990-х годов по причине подъема рынка персональных компьютеров и снижения спроса на мэйнфреймы. Сначало системы базировались на традиционной двухуровневой архитектуре. Клиент-серверным приложением данном случаи называют информационную систему, основанная на применении серверов баз данных. Компанией Gartner Group, была разработана двухзвенная модель взаимодействия клиент-сервер, двухзвенным она была названа по причинен того, что три элемента приложения по разному распределяются между двумя узлами. Первой моделью распределенного представления данных, которая реализовывалась на универсальных вычислительных машинах с подключенными к ней неинтеллектуальными терминалами. Управление данными и взаимодействие с пользователем систематизировались в единой программе, на терминал загружался только интерфейс пользователя, сформированный на центральном компьютере. В следствии с ростом популярности персональных компьютеров (ПК) и локальных сетей, были разработаны модели доступа к удаленной базе данных. Базовой архитектурой в начале для сети персональных компьютеров была файл-сервер. В этой архитектуре один компьютер является файловым сервером, а на клиентах выполняются приложения, в которых совмещены элементы представления, СУБД и прикладная программа. Протокол обмена представляет собой коллекцию вызовов не высокого уровня файловой системы. Данная архитектура возымела следующие недостатки - большой сетевой трафик и отсутствие единого доступа к ресурсам. Появление использование другой реализации модели доступа к удаленной базе данных произошло благодаря созданию первых специализированных серверов баз данных появилась возможность другой реализации. Это позволило ядру СУБД функционировать на сервере, а общение с протоколом обмена при помощи языка SQL, и следовательно уменьшить нагрузку на сеть и унифицировать интерфейс "клиент-сервер". Но, это не позволило заметно уменьшить сетевой трафик, и по-прежнему невозможно грамотное администрирование, причина тому совмещение

различных функций в одной программе. Для решения этих недостатков разработали концепцию активного сервера, который использовал принцип хранимых процедур. Это позволило часть прикладного компонента перенести на сервер (модель распределенного приложения). Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер.

Преимущества такого подхода:

- возможно централизованное администрирование прикладных

функций,

- значительно снижается сетевой трафик (т.к. не идет передача на сервер, а вызывается сохраненная процедур).

Недостаток – малое количество средств разработки хранимых процедур.

На практике обычно используется смешанный подход:

- простейшие прикладные функции выполняются сохраненными

процедурами на сервере;

- более сложные прикладные функции реализуются на клиенте

непосредственно в прикладной программе;

Код приложений должен выполняться на стороне клиента, которое включает в себя компоненты для поддержки интерфейса с пользователями. Часть работающая на стороне клиента работает с той частью клиентского ПО, которая является посредником СУБД для программ. Работа между клиентом и сервером баз данных, использует язык запросов SQL. Поэтому такие функции, например, предварительная обработка форм, предназначенных для запросов к базе данных, или формирование результирующих отчетов выполняются в коде приложения. В программах практически всех компаний сервер получает от клиента запрос на языке SQL. Сервер производит компиляцию запросов. Далее при успешной компиляции происходит выполнение команды. Разработчики и пользователи информационных систем, основанных на архитектуре "клиент-сервер", часто недовольны из-за постоянных сетевых расходов, которые следуют из потребности обращаться от клиента к серверу с каждым очередным запросом. На практике

часто встречаемая ситуация, когда для эффективной работы отдельной клиентской составляющей необходимо только небольшая часть общей базы данных. Это повлекло к необходимости поддержки локального кэша всей базы данных на стороне каждого клиента. Система локального кэширования базы данных представляет собой частный случай идеи реплицированных баз данных. Программное обеспечения рабочей станции обязано иметь элементы управления базами данных, иначе говоря, более простой вариант сервера базы данных. Отдельной проблемой является обеспечение согласованности (когерентности) кэшей и общей базы данных. Есть несколько вариантов решения – автоматическая поддержка договоренности за счет средств основного ПО управления базами данных до полного перевода этой задачи на прикладной уровень.

Преимуществами данной архитектуры являются:

- осуществимо распределение функций вычислительной системы между несколькими самостоятельными ПК;
- все данные сохраняются на сервере, который, должен быть, защищен серьезней чем клиентская часть, к тому же на сервере обеспечивается контроль прав доступа к данным;
- поддержка многопользовательской работы;
- обеспеченная целостность данных.

Недостатками данной архитектуры являются:

- неработоспособность сервера может сделать неработоспособной всю

вычислительную сеть;

- администрирование данной системы требует квалифицированного

профессионала;

- высокая стоимость оборудования;
- бизнес логика приложений осталась в клиентском ПО.

Когда идет проектирование информационной системы на архитектуре клиент-сервер, главное обратить внимание на сколько грамотны решения. Оборудование опытной версии может быть минимальным. После создания опытной версии нужно исследование работы, чтобы найти узкие места системы. Только после этого необходимо принимать решение о выборе аппаратуры сервера, которая будет использоваться на практике. Увеличение размеров информационной системы не

сопровождается крупными проблемами. Часто встречаемым решением - обычная замена аппаратуры сервера и почти не затрагивается прикладная часть системы. Этот вид архитектуры называется архитектура с "толстым" клиентом. Особенностью является размещение логики представления данных и бизнес-логика на клиенте, который работает с логикой хранения и накопления данных на сервере, используя язык запросов SQL. Но при установке "толстого клиента", требуется большое количество специальных библиотек и специфичных настроек окружения, на большое число клиентских компьютеров с различными операционными системами, как правило, вызывает массу проблем. Для решения этих трудностей была создана альтернатива - двухзвенная архитектура "с тонким клиентом". При правильной реализации программы-клиента лишь отображает графический интерфейс пользователя (GUI) и отправляет и принимает запросы, а вся бизнес-логика выполняется сервером. Идеальным клиентом является интернет-браузер, который предустановлен изначально в операционной системе пользовательского компьютера и не требует специальной настройки. К сожалению, эта схема тоже имеет свои недостатки, такая как серверу иногда приходится брать на себя несвойственные для него функции реализации бизнес логики приложения.

Multitier architecture - вид клиент-серверной архитектуры, где на несколько серверов вынесены функция обработки данных. Что позволяет разделить функции хранения, обработки и представления данных для достижения большей эффективности использования возможностей системы. Среди многоуровневой архитектуры клиент-сервер наиболее распространена трехуровневая архитектура, оно предполагает наличие таких компонентов приложения: клиентское приложение, подключенное к серверу приложений, который в свою очередь подключен к серверу базы данных.

Терминал – это интерфейсный компонент, который представляет первый уровень, собственно приложение для конечного пользователя. По требованиям первый уровень не должен иметь прямых связей с 22 базой, быть нагруженным основной бизнес-логикой и хранить состояние приложения. На первый уровень выносится простая бизнес-логика такие как: проверка вводимых значений на допустимость и соответствие формату, несложные операции с данными, уже загруженными на терминал, интерфейс авторизации, алгоритмы шифрования. Сервер приложений находится на 2-ом уровне. На 2-ом уровне сконцентрирована значительная часть бизнес-логики. За его пределами остаются элементы, экспортируемые на терминалы, а кроме того погруженные в третий уровень хранимые процедуры и триггеры. Сервер базы данных обеспечивает хранение данных и выносится на 3-ий

уровень. Как правило это обычная объектно-ориентированная СУБД. Если 3-ий уровень представляет собой базу данных совместно с сохранёнными процедурами, триггерами и схемами, описывающие приложение в виде реляционной модели, то 2-ой уровень строится как программный интерфейс, связывающий клиентские элементы с прикладной логикой базы данных. В простейшей конфигурации, сервер приложений и сервер базы данных могут находиться на одной машине с подключенными к ней терминалами. Согласно требованиям безопасности, надежности, масштабирования конфигурации сервер должен находиться на отдельном компьютере, с подключенными к нему серверами приложений, к которым, в свою очередь, по сети подключаются терминалы.

Плюсами данной архитектуры являются:

- Клиентское ПО не нуждается в администрировании;
- Масштабируемость;
- Конфигурируемость – изолированность уровней друг от друга позволяет быстро и простыми средствами переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней;
- Высокая безопасность;
- Высокая надежность;
- Низкие требования к скорости канала (сети) между терминалами и

сервером приложений;

- Низкие требования к производительности и техническим характеристикам терминалов, как следствие снижение их стоимости.

Минусами данной архитектуры являются:

- Растет сложность серверной части и, как следствие, затраты на

администрирование и обслуживание;

- Более высокая сложность создания приложений;
- Сложнее в разворачивании и администрировании;
- Высокие требования к производительности серверов приложений и сервера базы данных, а, значит, и высокая стоимость серверного оборудования;
- Высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.

В рамках технологии "клиент-сервер" было, положено начало развития корпоративного программного обеспечения в многозвенной архитектуре. Совместно с клиентской частью приложения в них появились серверы приложений. Особенности:

- Программа-клиент реализует GUI (graphical user interface), передает запросы серверу приложений и принимает от него ответ,
- Сервер приложений создает бизнес-логику и обращается с запросами к серверу "3-его уровня",
- Сервер 3-его уровня обслуживает запросы сервера приложений.

Программа-клиент, таким образом, может быть "тонкой". Данная архитектура имеет следующие преимущества:

- Внесение изменений на каждом из звеньев можно производить независимо;
- Звенья не обмениваются между собой большими объемами информации, что позволило снизить нагрузку на сеть;
- Обеспечивается масштабирование и простая модернизация оборудования и программного обеспечения, поддерживающего каждое из звеньев, в том числе обновление серверного парка и терминального оборудования, СУБД и т.д.;
- приложения могут разрабатываться на языках 3-его или 4-ого поколения например: Java, C/C++.

Следующий шаг – это увеличение количества звеньев, и возрастает не только благодаря разбиению при "утоньшении" каждого звена, но и вся бизнес-модель разрабатывается как многозвенная. Современные корпоративные программные системы это сложный комплекс взаимодействующих между собой на разных уровнях компонентов, каждые из которых могут играть роль клиента для одних компонентов и сервера для других. Часто встречаемой проблемой систем, основанных на двухзвенной и/или на многозвенной архитектуре, является требование к мобильности в как можно более широком классе аппаратно-программных сред. Но даже выбрав только UNIX-ориентированные локальные сети, то всё равно применяется разная аппаратура и протоколы связи. Попытки создания межплатформенных систем, приводит к их высокой загруженности сетевыми деталями в ущерб функциональности. Еще более затрудненный момент этой проблемы связан с возможностью использования разных представлений данных в разных узлах неоднородной локальной сети. В разных компьютерах может существовать различная адресация, представление чисел, кодировка символов и т.д. Это особенно важно для серверов высокого уровня, телекоммуникационных,

вычислительных, баз данных. Общим решением проблемы мобильности такого рода систем является использование технологий, реализующие протоколы удаленного вызова процедур (RPC - Remote Procedure Call) стандартным и платформо-независимым способом. При использовании таких технологий обращение к сервису в удаленном узле выглядит как обычный вызов процедуры. Средства RPC, в которых, естественно, содержится вся информация о специфике аппаратуры локальной сети и сетевых протоколов, переводит вызов в последовательность сетевых взаимодействий. Тем самым, специфика сетевой среды и протоколов скрыта от прикладного программиста. При вызове удаленной процедуры, программы RPC производят изменение форматов данных с клиентской стороны в собственные форматы, и затем преобразование в форматы данных сервера. При отправлении ответных параметров производятся обратные преобразования. Благодаря этому системы основанные на стандартном пакете RPC, могут быть легко перенесены в любую открытую среду. К представлению относится вся информация, непосредственно отображаемая пользователю: сгенерированные html-страницы, таблицы стилей, изображения. Уровень представления охватывает все, что имеет отношение к общению пользователя с системой. К главным функциям слоя представления относятся отображение информации и интерпретация вводимых пользователем команд с преобразованием их в соответствующие операции в контексте логики и данных. Уровень логики содержит основные функции системы, предназначенные для достижения поставленной перед ним цели. К таким функциям относятся вычисления на основе вводимых и хранимых данных, проверка всех элементов данных и обработка команд, поступающих от слоя представления, а также передача информации уровню данных. Уровень доступа к данным – это подмножество функций, обеспечивающих взаимодействие со сторонними системами, которые выполняют задания в интересах приложения. Данные системы обычно хранятся в базе данных.

3.Модели клиент-сервер.

Существует, не меньше, 3-х вариантов модели клиент-сервер:

1. доступ к удаленным данным (RDA-модель);

2. сервер базы данных (DBS-модель);

3. сервера приложений (AS-модель).

RDA-модель и DBS-модель представляют собой двухзвенную архитектуру и поэтому не подходят в качестве базовой модели распределенной системы. AS-модель уже трехзвенная. Преимущество у этой модели в независимости интерфейса пользователя от компонентов обработки данных. Трехзвенной считают из-за наличия следующих черт:

- компонент интерфейса с пользователем;
- программное обеспечение промежуточного слоя (middleware);
- компонент управления данными.

Middleware - это ключевой элемент трехзвенных распределенных систем. Он исполняет операции такие как - управление транзакцией и коммуникацией, транспортировка запросов и прочие функции. Присутствует базовое отличие среди технологий вида "сервер запросов — клиент запросов" и трехзвенными технологиями. В первоначальном случае клиент очевидным способом запрашивает данные, понимая структуру базы данных. Например, клиент отправляет на сервер СУБД, SQL-запрос, а в обратно получает данные. Создается жесткая связь типов, и для их создания все СУБД используют закрытый SQL-канал. Он строится двумя процессами: SQL/Net на компьютере-клиенте и SQL/Net на компьютере-сервере и порождается по инициативе клиента оператором connect. Канал называют закрытым из-за невозможности, написать программу, способная зашифровать SQL-запросы по особенному алгоритму или будет иначе вмешиваться в процесс передачи данных между клиентским и серверным приложением. В случае трехзвенной схемы клиент явно запрашивает один из сервисов, например, передавая ему некоторое сообщение, и получает ответ также в виде сообщения. Клиент направляет запрос во внешнюю среду, ничего не зная о месте расположения сервиса. Имеет место так называемая "поставка функций" клиенту. Для клиента сама база данных видна исключительно посредством набора сервисов. Более того, он вообще ничего не знает о ее существовании, т. к. все операции над базой данных выполняются внутри сервисов. Таким образом, речь идет о двух принципиально разных подходах к построению информационных систем клиент-сервер. Двухзвенная архитектура на сегодняшний день может считаться достаточно устаревшей и, в связи с развитием распределенных информационных систем, постепенно отходит на второй план. И если для быстрого создания несложных приложений с небольшим числом пользователей этот метод подходит как нельзя лучше, то при построении корпоративных распределенных информационных систем он абсолютно непригоден в силу вышеперечисленных причин. Термин "клиент-сервер" означает такую архитектуру программного

комплекса, в которой его функциональные части взаимодействуют по схеме "запрос-ответ". Если рассматривать взаимодействующие части этого комплекса, то часть называемая клиент исполняет активную функцию, иначе говоря инициирует запросы, а серверная часть пассивную функцию отвечает на них. По ходу развития системы роли могут изменяться, например, некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому. Любая информационная система должна иметь, следующие основные части - модули хранения данных, обработки и интерфейса. Каждая из этих частей может быть создана или изменена независимо друг от друга. Например, можно изменить графический интерфейс пользователя, не изменяя программы хранения и обработки данных. В обычной клиент-серверной архитектуре основные части приложения распределяют по двум физическим модулям. Как правило ПО хранения данных находится на сервере, интерфейс пользователя уже устанавливают на стороне клиента, а обработку данных разделяют между клиентом и сервером. В данном случаи и состоит главный недостаток двухуровневой архитектуры, с которого вытекают ряд неприятных отличительных черт, очень усложняющих разработку клиент-серверных систем. А именно, при разбиении алгоритмов обработки данных необходимо синхронизировать поведение обеих частей системы. Все разработчики обязаны обладать полной информацией о последних изменениях, внесенных в систему, и понимать эти изменения. Это требование вызывает большие сложности при разработке клиент-серверных систем, установке и сопровождении, поскольку необходимо тратить значительные усилия на координацию действий разных групп специалистов. При разработке часто появляются противоречия, что тормозит развитие системы и заставляет изменять готовые элементы. Чтобы избежать несогласованности частей архитектуры, то пытаются выполнить обработку данных на стороне клиента, либо на сервере. Каждый подход имеет свои недостатки. В первом способе нерационально используется сеть, по причине передачи необработанных, иначе говоря, избыточных данных. К тому же сложнее поддерживать системы и ее изменение, иначе могут возникнуть ошибки или несогласованность данных из-за того, что замена алгоритма вычислений или исправление ошибки требует одновременной полной замены всех интерфейсных программ. Если же вся обработка информации выполняется на сервере (когда такое вообще возможно), то возникает проблема описания встроенных процедур и их отладки. Дело в том, что язык описания встроенных процедур обычно является декларативным и, следовательно, в принципе не допускает пошаговой отладки. Кроме того, систему с обработкой информации на сервере абсолютно невозможно

перенести на другую платформу, что является серьезным недостатком.

4.Толстый и тонкий клиенты.

Как значится в словаре Free Online Dictionary of Computing, тонкий клиент - это клиентское устройство (или программа), передающее большую часть исполняемых им функций серверу. Толстый клиент определить намного проще - это все клиенты, не являющиеся тонкими. Тонкий клиент (thin client) — терминал сети без собственных запоминающих устройств, которого вычислительная мощность и объем памяти ограничивается нуждами пользователя. Все программы и приложения, становятся доступны для пользователей при выполнении регистрации на сервере. Тонким клиентом называю компьютер, имеющий малую мощность железной начинки и позволяющий пользователю осуществлять ввод и вывод данных при помощи вычислительной мощности на более мощном компьютере или сервере, он осуществляет связь при помощи каналов средней пропускной способности, так же к данному клиенту возможно подключение устройств ввода/вывода данных такие как - сканеры, мониторы, принтеры и проекторы. Клиент называется тонким, если он не имеет совсем или имеет лишь малую часть бизнес-логики, т. е. отображает только графический интерфейс пользователя. Толстым клиентом называются клиенты, имеющие основную часть бизнес-логики. Лучший и часто встречаемый пример тонкого клиента является Web-браузер, он настолько универсальный, что способен работать с полностью разным прикладным программам, о которых не имеет информации, и, тем не менее обеспечивает пользователя удобным интерфейсом. Упрощение технологии обслуживания рабочих мест используя соответствующие сервисные инструменты, позволяет администратору системы единовременно обслуживать любое количество устройств. Так же возможность контроля за действиями пользователя обеспеченное отсутствию накопителей на рабочем месте клиента, пользователь не может вносить свои изменения в конфигурацию программного обеспечения или устанавливать свои программы. Мобильность пользователей, не привязанный к конкретному рабочему месту, может на своё усмотрение перемещаться в пределах локальной сети, применяя устройства дистанционного доступа. Тонкий клиент позволяет сэкономить на эксплуатации при не слишком большом различии в стоимости оборудования. Технология «тонкий клиент-сервер» основан на 3-х составляющих:

1) стопроцентное выполнение прикладных задач на терминальном сервере,

2) многопользовательская операционная система,

3) технология распределенного отображения пользовательского интерфейса приложений.

Пользователи имеют возможность одновременно заходить в систему и выполнять приложения на сервере в разных, защищенных друг от друга сессиях сервера. В системе с использованием тонкого клиента по сети или коммутируемой телефонной линии на сервер передаются сигналы, отражающие нажатие на ту или иную клавишу либо то или иное движение мыши. А сервер производит соответствующие действия и формирует изменения экрана пользователя и передаѐт эти изменения тонкому клиенту. Тонкий клиент получает от сервера изменѐнные образы экрана и отображает их на дисплее (в современных системах может передаваться не весь изменѐнный экран, а лишь части изображения с соответствующими командами, на основании которых программное обеспечение тонкого клиента формирует изменѐнную картинку). В роли клиента может выступать любой ПК, но, поскольку на нем почти не выполняются операции по обработке данных, в качестве тонких клиентов можно применять и недорогие терминалы, имеющие низкую производительность, не содержащие компоненты с движущимися частями, оснащенные, как правило, устройствами с весьма ограниченным объемом ОЗУ. При работе в терминальной системе все прикладные программы, данные и параметры настроек хранятся на сервере. Это дает много преимуществ в плане начального развертывания рабочих мест одно из них — это отсутствие необходимости устанавливать ПО на каждом терминале, что дает больше удобства проведения резервного копирования данных, восстановления сессий после сбоев. Технология тонкого клиента имеет преимущество - она ориентирована на дистанционный доступ для сотрудников. Это позволяет работать вне офиса или рабочего места. При необходимости повысить вычислительную мощность всей системы достигается заменой всего лишь одного устройства – терминального сервера, все рабочие места автоматически переходят на более высокий уровень производительности без необходимости замены каких-либо устройств. Толстый или Rich-клиент - это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства:

- Толстый клиент обладает широким функционалом в отличие от тонкого.
- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Имеет возможность подключения к банкам без использования сети Интернет.
- высокое быстродействие.

Недостатки:

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.

Большинство современных средств быстрой разработки приложений (RAD), которые работают с различными базами данных, реализует стратегию: "толстый" клиент обеспечивает интерфейс с сервером базы данных через встроенный SQL. Такой вариант реализации системы с "толстым" клиентом, кроме перечисленных выше недостатков, обычно обеспечивает недопустимо низкий уровень безопасности. Например, в банковских системах приходится всем операционистам давать права на запись в основную таблицу учетной системы. Кроме того, данную систему почти невозможно перевести на Web-технологию, так как для доступа к серверу базы данных используется специализированное клиентское ПО. Итак, рассмотренные выше модели имеют следующие недостатки.

"Толстый" клиент:

- сложность администрирования;
- усложняется обновление ПО, поскольку его замену нужно производить одновременно на всей системе;
- усложняется распределение прав, так как разграничение доступа происходит по таблицам;
- сильно нагружается сеть вследствие передачи по ней необработанных данных;
- слабая защита данных, из-за фактора сложности правильного распределения полномочий.

"Толстый" сервер:

- усложняется создание, так как языки типа PL/SQL не пригодны для разработки подобного Программного обеспечения и нет хороших средств отладки;
- производительность программ, написанных на языках типа PL/SQL, значительно ниже, чем созданных на других языках, что имеет важное значение для сложных систем;
- программы, написанные на СУБД-языках, обычно работают недостаточно надежно;
- ошибка в них может привести к выходу из строя всего сервера баз данных;
- получившиеся таким образом программы полностью непереносимы на другие системы и платформы.

Для решения данных проблем используются многоуровневые архитектуры клиент-сервер. Рассмотрим следующие компоненты:

- презентационная логика (Presentation Layer - PL);
- бизнес-логика (Business Layer - BL);
- логика доступа к ресурсам (Access Layer - AL).

Таким образом, можно придти к нескольким моделям клиент-серверного

взаимодействия 1):"Толстый" клиент. Наиболее часто встречающийся вариант реализации архитектуры клиент-сервер в уже внедренных и активно используемых системах. Такая модель подразумевает объединение в клиентском приложении как PL, так и BL. Серверная часть, при описанном подходе, представляет собой сервер баз данных 2)., реализующий AL. К описанной модели часто применяют аббревиатуру RDA - Remote Data Access.

"Тонкий" клиент. Модель 3), начинающая активно использоваться в корпоративной среде в связи с распространением Internet-технологий и, в первую очередь, Web-браузеров. В этом случае клиентское приложение обеспечивает реализацию PL, а сервер объединяет BL и AL. Сервер бизнес-логики. Модель с физически выделенным в отдельное приложение блоком BL.

1).Хотя, рассматриваемые в этой части варианты разделения функциональности между клиентом и сервером являются "классическими", далее будет использоваться не только устоявшаяся традиционная, но и более

новая терминология, возникшая вследствие распространения в корпоративных средах Internet/intranet-технологий и стандартов. 2). Хотя в качестве серверной части, в общем случае, выступает менеджер многопользовательского доступа к

информационным ресурсам, в этой статье

будет сохраняться ориентация на серверы баз данных, как оконечное серверное звено. 3). Модели, основанные на Internet-технологиях и применяемые для построения внутрикорпоративных систем получили название intranet. Хотя intranet-системами сегодня называют все, что так или иначе использует стек протоколов TCP/IP, с ними скорее следует связать использование Webбраузеров в качестве клиентских приложений. При этом важно отметить тот факт, что браузер не обязательно является HTML-"окном", но, в не меньшей степени, представляет собой универсальную среду загрузки объектных приложений/компонент -Java или ActiveX. Описанные три модели организации клиент-серверных систем в определенной степени являются ориентирами в задании жесткости связей между различными функциональными компонентами, чем строго описываемыми программами в реальных проектах. Жесткость связей в схеме взаимодействия компонент системы часто определяется отсутствием (или наличием) транспортного или сетевого уровня (Transport Layer - TL), обеспечивающего обмен информацией между различными компонентами.

5.Серверы приложений

Посмотрим на то, что же происходит в реальной жизни. С точки зрения применения описанных моделей, при проектировании прикладных систем разработчик часто сталкивается с правилом 20/80. Суть этого правила заключается в том, что 80% пользователей обращаются к 20% функциональности, заложенной в систему, но оставшиеся 20% задействуют основную бизнес-логику - 80%. В первую группу пользователей попадают операторы информационных систем, занимающиеся ввод и редактирование информации, а также рядовые сотрудники и менеджеры, обращающиеся к поисковым и справочным механизмам. Во вторую группу пользователей попадают эксперты, аналитики и менеджеры управляющего звена, которым требуются как специфические возможности отбора информации, так и развитые средства ее анализа и представления. С точки зрения реализации моделей необходимо обеспечить прозрачность взаимодействия между различными компонентами системы, а, следовательно, обратиться к существующим стандартам такого взаимодействия. Любая прикладная система, вне зависимости от выбранной модели взаимодействия, требует такой инструментарий, который смог бы существенно ускорить сам процесс создания системы и, одновременно с этим, обеспечить прозрачность и наращиваемость кода. На фоне разработки и внедрения

систем корпоративного масштаба явно присутствует тенденция использования объектно-ориентированных компонентных средств разработки. Соответственно, полноценное применение объектов в распределенной клиент-серверной среде требует и распределенного объектно-ориентированного взаимодействия, то есть возможности обращения к удаленным объектам. Таким образом, мы приходим к анализу существующих распределенных объектных моделей. На настоящий момент наибольшей проработанностью отличаются COM/DCOM/ ActiveX и CORBA/DCE/Java. Если в первом случае требуемые механизмы поддержки модели являются неотъемлемой частью операционной платформы Win32 (Windows 95/NT/CE), то во втором случае предусмотрена действительная кроссплатформенность (например, везде, где есть виртуальная машина Java). Если попытаться объективно оценить (хотя любая такая попытка во многом субъективна) перспективы применения этих моделей, то для этого необходимо понять требования к операционным платформам, выдвигаемые различными функциональными компонентами системы. При построении реальных систем корпоративного масштаба уже мало обходиться их разделением на три базовых фрагмента PL, BL, AL. Так как бизнес-логика является блоком, наиболее емким и специфичным для каждого проекта, именно ее приходится разделять на более мелкие составляющие. Такими составляющими могут быть, например, функциональные компоненты обработки транзакций (Transaction Process Monitoring), обеспечения безопасности (Security) при наличии разграничения прав доступа и выходе в Internet (Fire-wall), публикование информации в Internet (Web-access), подготовки отчетов (Reporting), отбора и анализа данных в процессе принятия решений (Decision Support), асинхронного уведомления о событиях (Event Alerts), тиражирования данных (Replication), почтового обмена (Mailing) и др. Вследствие наличия такого огромного количества функций, закладываемых в блоки поддержки бизнес-логики, появляется понятие сервера приложений (Application Server - AS). Причем, сервер приложений не просто является некоим единым универсальным средним BL-звеном между клиентской и серверной частью системы, но AS существует во множественном варианте, как частично изолированные приложения, выполняющие специальные функции, обладающие открытыми интерфейсами управления и поддерживающие стандарты объектного взаимодействия. Проникновение информационных технологий в сферу бизнеса в качестве неотъемлемого условия успешного управления приводит к тому, что системы корпоративных масштабов требуют сочетания различных клиентсерверных моделей в зависимости от задач, решаемых на различных конкретных направлениях деятельности предприятия. Вспомнив, снова, о правиле

20/80 можно придти к выводу, что наиболее оптимальным выбором, с точки зрения управляемости и надежности системы, является сочетание различных моделей взаимодействия клиентской и серверной части. По сути, мы приходим даже не к трехуровневой, а многоуровневой (N-tier) модели, объединяющей различных по "толщине" клиентов, серверы баз данных и множество специализированных серверов приложений, взаимодействующих на базе открытых объектных стандартов. Существенным облегчением в реализации многоуровневых гетерогенных систем является активная работа ряда производителей программного обеспечения, направленная на создание переходного ПО. В отличие от продуктов middleware, обеспечивающих верхний транспортный уровень (универсальные интерфейсы доступа к данным ODBC, JDBC, BDE; Message Oriented Middleware - MOM; Object Request Broker - ORB;), переходное ПО отвечает за трансляцию вызовов в рамках одного стандарта обмена в вызовы другого - мосты ODBC/JDBC и BDE/ODBC, COM/CORBA, Java/ActiveX и т.п. Причем, различные стандарты взаимодействия могут применяться в различных связках узлов системы, а мосты встраиваться в любой узел или выделяться в своеобразные серверы приложений, с физическим выделением в узлах сети. Двигаясь между клиентами слева-направо на диаграмме, мы можем наблюдать переход между различными моделями распределенных вычислений - через intranet к Internet.

6.Клиент-серверные вычисления.

В 1970-х и 1980-х годов была эпоха централизованных вычислений на мэйнфреймах IBM занимающих более 70% в компьютерном бизнесе мира. Бизнес транзакции, деятельности и базы данных, запросы и техническое обслуживание - все исполнялось на мэйнфреймах IBM. Этап перехода к клиент-серверным вычислениям представляет совершенно новую концепцию и технологию реорганизации всего делового мира. Вычислительные парадигмы 1990-х годов называли «волной будущего». Машина-клиент обычно управляет интерфейсами процессов, таких как графический интерфейс (графический пользовательский интерфейс), отправкой запросов на сервер программ, проверкой данных, введенных пользователем, а также управляет местными ресурсами, а пользователь взаимодействует с такими, как монитор, клавиатура, рабочие станции, процессора и других периферийных устройств. С другой стороны, сервер выполняет запрос клиента, с помощью службы. После того как сервер получает запросы от клиентов, он выполняет поиск базы данных, обновления и управляет целостностью данных и отправляет ответы на запросы клиентов. Цель клиент-серверных вычислений - позволить каждой

сетевой рабочей станции (клиента) и принимающей (сервера) быть доступными, по мере необходимости приложения, а так же обеспечивать доступ к существующему программному обеспечение и аппаратным компонентам от различных поставщиков для совместной работы. Когда эти два условия соединены, становятся очевидными преимущества клиент-серверной архитектуры, такие как экономия средств, повышение производительности, гибкости и использования ресурсов. Клиент-серверные вычисления состоят из трех компонентов, клиентского процесса, запрашивающего обслуживание и серверного процесса предоставления запрашиваемых услуг, с Middleware между ними для их взаимодействия. Клиент

Машина клиент обычно управляет пользовательским интерфейсом частей приложения, проверкой данных, введенных пользователем, отправкой запросов на сервер программы. Кроме того, клиентский процесс также управляет местными ресурсами, что позволяет пользователю взаимодействовать с монитором, клавиатурой, рабочими станциями, процессорами и другими периферийными устройствами.

Машина Сервер выполняет служебные запросы клиента. После того как сервер получает запросы от клиентов, он производит поиск базы данных, обновления, управляет целостностью данных и отправляет ответы на запросы клиентов. Серверных процесс может работать на другой машине в сети; тогда сервер используется как файловая система услуг и приложений сервисов. Или в некоторых случаях, другой рабочий стол машины обеспечивает применение услуг. Сервер выступает в качестве программного обеспечения двигателя, который управляет общим ресурсам, таким как базы данных, принтеры, линии связи, или процессоров высокой мощности. Основная цель серверного процесса - выполнение фоновых задач, которые являются общими для приложений. Простейшая форма серверов - это дисковый сервер и файл-сервер. Если клиент передает запросы на файл или группы файлов по сети на файловый сервер, эта форма обслуживания данных требует большой пропускной способности и может замедлить сеть с большим количеством пользователей. Более продвинутые формы серверов - это серверы баз данных, сервер транзакций и серверов приложений.

Middleware позволяет приложениям прозрачно контактировать с другими программами или процессами независимо от местоположения. Ключевым элементом Middleware является NOS (Network Operating System), которая предоставляет такие услуги, как маршрутизация, распределение, обмен сообщениями и управления сервисной сети. NOS полагается на коммуникацию протоколов предоставления конкретных услуг. Прежде чем пользователь может

получить доступ к услугам сети, клиент-серверный протокол требует установку физического соединения и выбор транспортных протоколов. Клиент-серверный протокол диктует, каким образом клиенты запрашивают информацию и услуги от сервера, а также как сервер отвечает на эту просьбу.

Для того чтобы связать клиентов и серверов вместе и в полной мере использовать ресурсы, содержащиеся в каждой машине, мы должны разработать сетевую систему, которая на это способна. Сети должны быть прозрачны для пользователей. Сети и приложения должны работать вместе так же хорошо, как если бы они работали на одном компьютере. Кроме того, сеть должна обеспечить возможности самовосстановления, чтобы можно было перенаправить сетевой трафик вокруг испорченного кабеля и быть достаточно гибкой, чтобы реагировать на бизнес-изменения в окружающей среде. Для простоты используются локальные сети LAN. Сейчас существует три различные LAN топологии: звезда, кольцо и шина и, по крайней мере, пять конкурирующих стандартов для передач, и два стандарта для информации, необходимой для управления сетью. Локальные сети стали настолько сложными, что они требуют собственную операционную систему. Сеть продолжает быть одним из наименее изученных и наиболее важным из компонентов в информационной структуре организации. Большинство организаций, приверженных клиент-серверной архитектуре согласны, что связь локальных сетей не место, чтобы сэкономить деньги. Мы не должны пытаться связать несовместимые локальные сети с различными платформами. Программное обеспечение, аппаратное обеспечение и операционная система, используемая в сети, должны быть тщательно протестированы.

Открытые системы и стандарты

Одна из наиболее важных особенностей клиент-серверной архитектуры - это открытость системы. Открытой системе соответствует широкий набор формальных стандартов и поддержка платформы от различных производителей. Открытые системы требует принятия стандартов в рамках всей организации. Открытую систему делает успешной открытые стандарты системы, которые должны быть приемлемыми для обеих сторон: пользователей и производителей систем и будут приняты на всех уровнях организации. Для того чтобы все эти компоненты работали вместе как сложная

система, мы должны придерживаться нескольких видов стандартов. Имеются

ввиду стандарты в четырех областях клиент-серверных вычислений, таких как платформы (программного и аппаратного обеспечения), сети, Middleware и приложения. Спецификации стандартов должны быть разработаны на основе консенсуса и быть общедоступными Стандарты должны быть всеобъемлющими и последовательными и определять интерфейсы, услуги и поддержку форматов для достижения совместимости. В настоящее время есть несколько консорциумов, работающих в разработке стандартов для открытых систем.

- -OSF (Open Software Foundation) - некоммерческий консорциум компьютерных производителей, разработчиков программного обеспечения и поставщиков чипов для разработок, основанных на стандартах программного обеспечения.
- UNIX-International - консорциум производителей компьютеров, разработчиков программного обеспечения, целью которых является создание UNIX и связанных с ними стандартов разработки и лицензирования продуктов UNIX.
- -OMG - международные организации системы поставщиков, разработчиков программного обеспечения и пользователей, сторонники развертывания объекта управления технологиями в разработке программного обеспечения. Применяя общую основу для всех объектно-ориентированных приложений, организации смогут управлять гетерогенными средами.
- CORBA (Common Object Request Broker Architecture), разработанная OMG, DEC, NCR, HP и SUN является новым механизмом, который позволяет объектам (приложениям) называть друг друга по сети.
- -SQL Access Group - это промышленный консорциум работает над определением и осуществления технических условий для гетерогенного доступа SQL данных с использованием принятых международных стандартов.

2.4. Модель клиент - сервер в Интернете

Взаимодействие клиента и сервера в Интернете осуществляется с помощью запросов, посылаемых клиентом серверу, и ответов сервера на запрос клиента:

Суть распределенных систем - связь между процессами, реализующими не только взаимодействие компьютеров, но и частей (уровней) приложений. Взаимодействие частей приложений реализуется с помощью протоколов, описывающих состав и формат данных, пересылаемых соответствующими частями клиентских и серверных приложений друг другу для решения поставленной задачи. В Интернете разбиение приложений на части осуществляется на базе стека протоколов TCP/IP: В

этой модели разработчики имеют большую свободу в определении того, какие части клиент-серверного приложения будут на клиентском компьютере и какие на сервере. Логика пользовательского интерфейса существовала почти исключительно на сервере. Мы можем вновь приступить к использованию более эффективных и хорошо структурированных клиентсерверных моделей. Есть, конечно, еще технические вопросы, но мы в состоянии лучше строить истинные клиент-серверные приложений в настоящее время.

Клиент-серверную модель можно разделить на три части :

- User Interface - Пользовательский интерфейс
- Business or Application Logic - Бизнес и логика приложения
- Data Management - Управление данными

Традиционное развитие веб-приложений распространило реализации пользовательского интерфейса в сети, причем большая часть логики интерфейса пользователя и код выполняется на сервере (тонкий клиент, толстый сервер).

Этот метод имеет несколько ключевых проблем:

- Неудовлетворительное распределение обработки - с большим числом клиентов, делает все обработки на сервере неэффективно.
- Высокая латентность ответа пользователя - традиционные вебприложения не реагируют достаточно быстро. Высокое качество взаимодействия с пользователем является очень чувствительным к задержкам, и очень быстрая реакция имеет важное значение.
- Трудная модель программирования – программирование пользовательского интерфейса через клиент-сервер достаточно сложно.
- Если правила доступа распространяются через пользовательский код интерфейса, то при росте кода пользовательского интерфейса возникают новые векторы атаки.
- Сложное управления на серверах.
- Offline трудности. Код пользовательского интерфейса должен выполняться на клиенте и в автономном ситуациях.
- Снижение возможностей для взаимодействия. Когда клиент-сервер состоит из передачи внутренней части пользовательского интерфейса в браузере, бывает очень трудно понять эту связь и использовать ее для других приложений.

Необходимо решить, какой код должен работать на клиенте, а какой на сервере. Коду пользовательского интерфейса лучше работать на браузере, а бизнес-логике

и управлению данными лучше работать на стороне сервера. Хороший дизайн включает в себя создание объектов, которые инкапсулируют большую часть своего поведения и минимальной площади поверхности. Он должен быть интуитивно понятным и легко взаимодействовать с хорошо разработанным интерфейсом объекта. Кроме того, клиент-серверное взаимодействие должно быть построено на хорошо продуманном интерфейсе. Проектирование модульных удаленных интерфейсов часто называют сервис-ориентированной архитектурой (SOA). Клиент-серверная реализация высокого качества должна иметь простой интерфейс между клиентом и сервером. На стороне клиента должен инкапсулировать презентации и код взаимодействия с пользователем. Код на стороне сервера должен инкапсулировать правила поведения и взаимодействия данных. Веб-приложения должны быть разделены на два основных элемента, пользовательский интерфейс и веб-сервис.

Преимущества чистой модели клиент-сервер:

- Масштабируемость. Чем больше клиентов, которые используют приложения, тем больше клиентских машин, которые доступны, в то время как сервер остается постоянным.
- Немедленный ответ пользователя. Клиентский код может немедленно реагировать на действия пользователя, а не ждать для передачи данных по сети.

Организованная модель программирования. Такая модель обеспечивает более чистый подход к безопасности. Когда все запросы идут через код пользовательского интерфейса, данные могут передаваться через различные интерфейсы до проверки безопасности. Это может сделать анализ безопасности более сложным. С другой стороны, с понятным интерфейсом вебслужбы есть четко определенные шлюз безопасности для работы

и анализа безопасности. Клиентская часть управления - сохранение информации о состоянии сессии на клиенте уменьшает нагрузку на сервер. Это также позволяет клиентам использовать более «спокойное» взаимодействие, которое может еще больше повысить масштабируемость и возможности кэширования. Автономные приложения. Если большая часть кода для приложений уже построена, чтобы работать на клиенте, создание автономной версии приложения почти наверняка будет легче. Взаимодействие. К структурированным данным с минимальными API-интерфейсами для взаимодействия намного проще подключать дополнительных потребителей и производителей и взаимодействовать с существующими

системами. 3.4. 3-уровневая архитектура.

В настоящее время клиент-серверная архитектура имеет гибкую модульную архитектуру. Она может быть изменена или дополнена. Различные подходы могут быть скомбинированы в различных комбинаторных последовательностях, удовлетворяющих практически любые вычислительным потребностям. Поскольку Интернет становится важным фактором в вычислительных средах клиент-серверных приложений, работающих через Интернет станет важным нового типа распределенных вычислений. Интернет расширит охват и мощь клиент-серверной архитектуры. С помощью общепринятых стандартов, это позволит облегчить и расширить клиент-серверную архитектуру как внутри, так и между компаниями. Так же из-за интернета будут происходить изменения в языках программирования к технологии распределенных объектов. Клиент-серверная архитектура по-прежнему остается единственной и лучшей архитектурой с точки зрения использования Интернета и других новых технологий. Но, независимо от развитий других архитектурных подходов, клиент-серверная архитектура, вероятно, останется основой для большинства вычислительных событий в течение следующего десятилетия.

3.5. Прошлое и будущее клиентов.

Конечно же, с появлением большого количества людей, соединенных посредством компьютерной сети, сразу пошло бурное развитие клиентсерверных приложений. Но сервер в этом случае используется крайне экономно – это средство для хранения данных и выполнения несложных операций. Основную вычислительную роль берет на себя именно клиент, присоединенных к этой сети. Довольно удачная архитектура дает возможность использовать максимально ресурсы компьютера, богатый пользовательский интерфейс, важные данные пользователя хранятся на компьютере дома, а не сервере неизвестно какой страны. Проблем несколько – заставить человека установить какое-либо приложение можно лишь действительно предоставив ему его как средство решения проблемы. А как же быть тому количеству предпринимателей, которые любыми способами хотят привлечь человека своим выгодным товаром? Пока что клиент-серверная архитектура не сильно им в этом помогает. Да и приложения пишутся для определенной платформы и версии операционной системы, что напрочь лишает взаимодействия пользователей разных операционных систем. Но прогресс не стоит на месте, и в 1989 году была предложена концепция всемирной паутины. А уже через четыре

года появляется первый браузер Mosaic. Сразу же в мире приложений начинает вырисоваться два класса приложений – тонкие клиенты, примером которого может служить тот же браузер Mosaic и толстые клиенты, которые по прежнему берут на себя значительную часть обработки информации и используют сервер для хранения и взаимодействия. Клиентские приложени я не собирается сдавать позиций, потому что все, что может сейчас браузер - это отображать информацию. Динамика отсутствует. Одно лишь значительное преимущество – сайты становятся визитной картой, имеющие свой оригинальный дизайн, расположение меню, цветовую гамму. Для клиентских приложений это не было распространено – все создавалось на основе стандартных вариантов, стандартное место расположения меню, стандартные цвета и шрифты. С одной стороны простота и функциональность, с другой стороны – красота и изящество. Конечно же, красота и изящество не могла не найти своих сторонников. Также пользователи разных операционных систем могли спокойно просматривать сайты. Возможны некоторые несоответствия в отображении информации, но все же лучше чем ничего. С появлением JavaScript в 1996 году html-страницы получают небывалую динамику и немного начинают походить на обычные клиентские приложения. Конечно же, до полноценных клиентских приложений им, скорее всего, не дойти никогда, но страницы оживают. Живые страницы делают переворот в интернете, и идет бурное совершенствование скриптовой технологии, что приводит к рождению в 2005 году AJAX, а именно идеи асинхронного обращения к серверу для получения лишь необходимой части страницы, а не всей страницы. Инновационные решения, основанные на AJAX, типа карт Windows Live Local, приблизили веб-приложения к уровню удобства обычных клиентских программ. Вот тут веб-страницу можно было уже спутать с обычным клиентским приложением. Но выглядеть как в клиентском приложении, и работать как клиентское приложение – это разные вещи. Все же доступ к ресурсам ограничен, соединение между клиентом и сервером одноразовое – уже просмотренные страницы при следующем обращении необходимо загружать опять. Но это проблемы, которые могут быть решены с появлением нового http-протокола. В свое время, толстые клиенты, работающие на компьютере пользователя, при грандиозном развитии веб-технологий становятся в прямом смысле толстыми – они сложны для клиента при обновлении версии, они прихотливы к семействам операционных систем, плохо взаимодействуют с веб-серверами, потому что зачастую построены с использованием клиентсерверной структуры. Здесь надо было срочно искать способы исправить ситуацию. Корпорация Microsoft не задержалась с предложением и выпустила на рынок программную технологию Microsoft .NET Framework, призванную объединить

множество различных служб, написанных на разных языках, для общей совместимости. Эта виртуальная машина может быть установлена на разных семействах Windows, а также на других операционных системах, что позволяет использовать любой из языков NET-семейства для написания работоспособных приложений для всех операционных систем, на которых установлен framework. Одна из проблем, которая так долго преследовала клиентские приложения, частично была решена. Итак, гонка клиентских и веб-приложений находится на той стадии,

когда веб просто физически не может проникнуть глубже в ресурсы пользователя, чтобы увеличить быстродействие. Возможности в реализации отдельных техник все же еще не доступны по сравнению с клиентскими приложениями и есть проблема постоянного обращения к серверу, потому что все еще мы работаем с обычным html-кодом. Клиентские же приложение со своим богатством и простотой реализации сложнейших техник веба слишком неохотно потребляются пользователями, привыкшими к этому времени с помощью браузера решать самые сложные свои задачи. К тому же, клиентские приложения по прежнему привязаны к определенным протоколам передачи данных для обмена информацией. А это влечет за собой дублирование определенных сервисов. Так мы плавно перешли от истории к дням сегодняшним и видим, что бесспорного лидера нет, и каждая из технологий имеет свои достоинства и недостатки. После долгих блужданий возле клиентского компьютера интернетгиганты все же готовы смирится с тем, что для увеличения быстродействия отдельных сервисов, для дальнейшего усложнения систем придется рассчитывать на мощности своих серверов, а не пытаться максимально глубоко влезть в ресурсы пользователей. В связи с этим последнее время очень интенсивно пошло развитие сервисов, построенных для использования облачных вычислений (cloud computing) – технология обработки данных, в которой программное обеспечение предоставляется пользователю как Интернет-сервис. При этом пользователь не заботится об архитектуре облака, а лишь получает необходимые ему мощности от целых кластеров. Такие сервисы на данный момент уже предоставляют Microsoft, Amazon (Elastic Compute Cloud). Тем самым вся необходимая пользователю функциональность перемещается на сервера тех фирм, которые ее предоставляют. Доступ осуществляет через браузер, а, значит, отсутствует привязанность к разным семействам операционных систем. Ярким примером может служить: Gmail - почтовый клиент Google, предоставляющий богатый инструментарий для работы с почтой прямо из браузера. Тем же временем продолжается совершенствование способов приблизить веб в клиентским приложениям. В 2006 году корпорация Microsoft выпустила плагин к IE – Silverlight,

который позволяет запускать приложения, содержащие анимацию, векторную графику и аудио-видео ролики, что характерно для RIA (Rich Internet application). Софтверные же компании имеют другую позицию – а именно видят будущее в smart client-ах – локальных приложениях, которые всецело ориентированы на потребление всевозможных сервисов из вне. Smart Client — это легко устанавливаемое и управляемое клиентское приложение, предоставляющее пользователю адаптивный, отзывчивый и богатый пользовательский интерфейс, полностью использующее возможности локальных ресурсов компьютера и интеллектуально управляющее взаимодействием с распределенными источниками данных. Ключевыми особенностями, отличающими Smart Client, являются: Богатый пользовательский интерфейс. Чтобы называться «умным», клиентское приложение должно иметь удобный пользовательский интерфейс, подстраиваясь под нужды пользователя, допуская персонализацию и предоставляя все современные способы управления (drag’n’drop, контекстные меню, дочерние окна, нотификации и т. д.) Простая установка, не требующая участия пользователя. Приложение должно предлагать пользователю автоматическую установку, не требующую перезагрузки, долгого ожидания или большого объема закачиваемых файлов. Автоматическая установка обновлений. Появление новых версий приложения должно автоматически проверяться, их установка так же должна происходить в автоматическом режиме. Возможность работы при отсутствии соединения с сервером. Если приложение в своей работе взаимодействует с удаленными источниками данных, оно также должно работать и предоставлять максимум возможной функциональности и при «отсоединенной» (оффлайн) работе. Примерами существующих смарт-клиентов могут быть: IssueVision - help desk management application TaskVision – клиентское приложение, которое позволяет подключенным пользователям создавать задачи, проекты и распределять их между другими пользователями. Взаимодействие между пользователями

построено с использованием веб-сервисов.

Поскольку обмен структурированными данными между клиентом с

сервисом производится с помощью стандартного языка XML – то

приложение может взаимодействовать с большинством существующих

сервисов, не зависимо от языка реализации. Однако даже с этими решениями

у «smart» клиентов в случае прерывания связи с Internet только один выбор

— отключаться, поэтому для устранения этого неудобства в Microsoft предложили технологию Live Mesh, позволяющую локально запускать Webприложения.

Звучит немного парадоксально – имеется в виду, что приложение может работать с данными и при следующем подключении уже синхронизировать их с сервером.

Такая возможность (работать оффлайн) также будет включена в последний Silverlight, что позволит даже с веб-страницами работать в оффлайн режиме.

Заключение

В ближайшее время, как и последние много лет, основной средой обмена информацией останется интернет. Судя по тенденциям, клиентские и веб приложения будут развиваться параллельно, только немного другим путем – теперь это будут не монолитные порталы, написанные одной командой и использующие ресурсы одной эко-системы. Это будут наряженные èлки – один костяк и множество подключенных сервисов, возможно даже разработанные разными фирмами. Это приведет к тому, что основное внимание и львиная доля времени будет расходоваться на разработку сложных сервисов, но потом они легко будут подключаться к всевозможным порталам, приложениях и другим сервисам. Тем самым в скором будущем мы будем находиться не только во всемирной паутине, но еще и каждая ниточка этой паутины будет состоять из такого же сложного смешения

различных сервисов, потребляемых различными устройствами. Но это огромное разнообразие сервисов будет полезно после окончательного внедрения нового протокола IPV6, что позволит подключать к интернету даже микроволновые печи, холодильники и т.д. Именно управление таким огромным количеством устройств в сети приведет к созданию множества сервисов и порталов, которые в онлайн режиме помогут управлять вашими электроприборами.

Используемая литература

Г.М.Лодыженский Системы баз данных. Коротко о главном. СУБД N 1, 2, 3, 4 1995.

Д.Васкевич Стратегии клиент/сервер. Диалектика, Киев, 1997.

Бирн Джеффри Л. Microsoft SQL Server: Руководство администратора / Пер. с англ. – М.: ЛОРИ, 1998.

Бучек Г. ASP .NET. – СПб.: Питер, 2002.

Галисеев Г.В. Программирование в среде Delphi 7: Самоучитель / Г.В. Галисеев. – М.: Вильямс, 2003.

Макдональд М. ASP.NET / М. Макдональд. – СПб.: БХВ-Петербург, 2003.