**Конспект лекции 4**

**Основы построения программно-информационных систем с веб-архитектурой.**

**Архитектура информационных систем**

Современные программные приложения и информационные системы достигли такого уровня развития, что термин "архитектура" в применении к ним уже давно не удивляет. Грамотно построить информационную систему, эффективно и надежно функционирующую не проще, чем сконструировать и возвести современное многофункциональное здание .

Когда речь заходит об "архитектуре информационной системы", обычно не возникает недостатка в определениях. Есть даже Web-сайты, которые собирают такие определения

Под архитектурой программных систем будем понимать совокупность решений относительно:

* организации программной системы;
* выбора структурных элементов, составляющих систему и их интерфейсов;
* поведения этих элементов во взаимодействии с другими элементами;
* объединение этих элементов в подсистемы;
* архитектурного стиля, определяющего логическую и физическую организацию системы: статические и динамические элементы, их интерфейсы и способы их объединения.

Архитектура программной системы охватывает не только ее структурные и поведенческие аспекты, но и правила ее использования и интеграции с другими системами, функциональность, производительность, гибкость, надежность, возможность повторного применения, полноту, экономические и технологические ограничения, а также вопрос пользовательского интерфейса.

По мере развития программных систем все большее значение приобретает их интеграция друг с другом с целью построения единого информационного пространства предприятия. Как можно видеть из вышеприведенных определений интеграция является важнейшим элементом архитектуры.

Для того чтобы построить правильную и надежную архитектуру и грамотно спроектировать интеграцию программных систем необходимо четко следовать современным стандартам в этих областях. Без этого велика вероятность создать архитектуру, которая неспособна развиваться и удовлетворять растущим потребностям пользователей ИТ. В качестве законодателей стандартов в этой области выступают такие международные организации как SEI (Software Engineering Institute), WWW (консорциум World Wide Web), OMG (Object Management Group), организация разработчиков Java – JCP (Java Community Process), IEEE (Institute of Electrical and Electronics Engineers) и другие.

Рассмотрим классификацию программных систем по их архитектуре:

* Централизованная архитектура;
* Архитектура "файл-сервер";
* Двухзвенная архитектура "клиент-сервер";
* Многозвенная архитектура "клиент-сервер";
* Архитектура распределенных систем;
* Архитектура Веб-приложений;
* Сервис-ориентированная архитектура.

Следует заметить, что, как и любая классификация, данная классификация архитектур информационных систем не является абсолютно жесткой. В архитектуре любой конкретной информационной системы часто можно найти влияния нескольких общих архитектурных решений.

Далее подробно рассмотрим особенности каждой архитектуры.

***Централизованная архитектура***

Централизованная архитектура вычислительных систем была распространена в 70-х и 80-х годах и реализовывалась на базе мейнфреймов (например, IBM-360/370 или их отечественных аналогов серии ЕС ЭВМ), либо на базе мини-ЭВМ (например, PDP-11 или их отечественного аналога СМ-4) . Характерная особенность такой архитектуры – полная "неинтеллектуальность" терминалов. Их работой управляет хост-ЭВМ.

Достоинства такой архитектуры:

* пользователи совместно используют дорогие ресурсы ЭВМ и дорогие периферийные устройства;
* централизация ресурсов и оборудования облегчает обслуживание и эксплуатацию вычислительной системы;
* отсутствует необходимость администрирования рабочих мест пользователей;

Главным недостатком для пользователя является то, что он полностью зависит от администратора хост-ЭВМ. Пользователь не может настроить рабочую среду под свои потребности – все используемое программное обеспечение является коллективным.

Использование такой архитектуры является оправданным, если хост-ЭВМ очень дорогая, например, супер-ЭВМ.

Классическое представление централизованной архитектуры показано на рисунке.

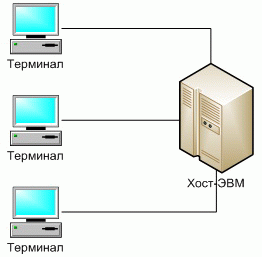


Рис .: Классическое представление централизованной архитектуры

Центральная ЭВМ должна иметь большую память и высокую производительность, чтобы обеспечивать комфортную работу большого числа пользователей.

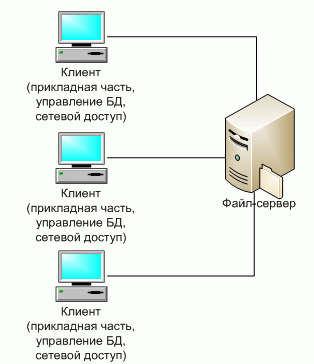
Все приложения, работающие в такой архитектуре, полностью находятся в основной памяти хост-ЭВМ. Терминалы являются лишь устройствами ввода-вывода и таким образом в минимальной степени поддерживают интерфейс пользователя.

***Архитектура "файл-сервер"***

**Файл-серверные приложения** – приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных

* Функции сервера: хранения данных и кода программы.
* Функции клиента: обработка данных происходит исключительно на стороне клиента.

Классическое представление информационной системы в архитектуре "файл-сервер" представлено на рисунке



**Рис.** Классическое представление архитектуры "файл-сервер"

Организация информационных систем на основе использования выделенных файл-серверов все еще является распространенной в связи с наличием большого количества персональных компьютеров разного уровня развитости и сравнительной дешевизны связывания PC в локальные сети

Конечно, основным достоинством данной архитектуры является простота организации. Проектировщики и разработчики информационной системы находятся в привычных и комфортных условиях IBM PC в среде MS-DOS, Windows или какого-либо облегченного варианта Windows Server. Имеются удобные и развитые средства разработки графического пользовательского интерфейса, простые в использовании средства разработки систем баз данных и/или СУБД.

Достоинства такой архитектуры :

* многопользовательский режим работы с данными;
* удобство централизованного управления доступом;
* низкая стоимость разработки;
* высокая скорость разработки;
* невысокая стоимость обновления и изменения ПО.

Недостатки :

* проблемы многопользовательской работы с данными: последовательный доступ, отсутствие гарантии целостности;
* низкая производительность (зависит от производительности сети, сервера, клиента);
* плохая возможность подключения новых клиентов;
* ненадежность системы.

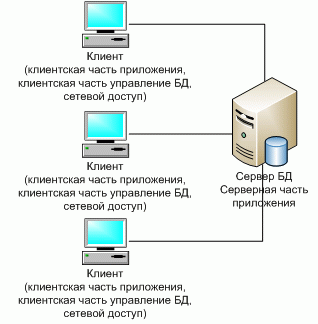
Простое, работающее с небольшими объемами информации и рассчитанное на применение в однопользовательском режиме, файл-серверное приложение можно спроектировать, разработать и отладить очень быстро [[14](http://www.intuit.ru/studies/courses/611/467/literature#literature.4.14)]. Очень часто для небольшой компании для ведения, например, кадрового учета достаточно иметь изолированную систему, работающую на отдельно стоящем PC. Однако, в уже ненамного более сложных случаях (например, при организации информационной системы поддержки проекта, выполняемого группой) файл-серверные архитектуры становятся недостаточными.

***Архитектура "клиент-сервер"***

**Клиент-сервер** (Client-server) – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами .Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture). Под клиент-серверным приложением в этом случае понимается информационная система, основанная на использовании серверов баз данных.

Схематически такую архитектуру можно представить, как показано на рисунке:



**Рис.** Классическое представление архитектуры "клиент-сервер"

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных, которая, фактически, является индивидуальным представителем СУБД для приложения.

Заметим, что интерфейс между клиентской частью приложения и клиентской частью сервера баз данных, как правило, основан на использовании языка SQL. Поэтому такие функции, как, например, предварительная обработка форм, предназначенных для запросов к базе данных, или формирование результирующих отчетов выполняются в коде приложения.

Наконец, клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

Посмотрим теперь, что же происходит на стороне сервера баз данных. В продуктах практически всех компаний сервер получает от клиента текст оператора на языке SQL.

* Сервер производит компиляцию полученного оператора.
* Далее (если компиляция завершилась успешно) происходит выполнение оператора.

Разработчики и пользователи информационных систем, основанных на архитектуре "клиент-сервер", часто бывают неудовлетворены постоянно существующими сетевыми накладными расходами, которые следуют из потребности обращаться от клиента к серверу с каждым очередным запросом. На практике распространена ситуация, когда для эффективной работы отдельной клиентской составляющей информационной системы в действительности требуется только небольшая часть общей базы данных. Это приводит к идее поддержки локального кэша общей базы данных на стороне каждого клиента.

Фактически, концепция локального кэширования базы данных является частным случаем концепции реплицированных баз данных. Как и в общем случае, для поддержки локального кэша базы данных программное обеспечение рабочих станций должно содержать компонент управления базами данных – упрощенный вариант сервера баз данных, который, например, может не обеспечивать многопользовательский режим доступа. Отдельной проблемой является обеспечение согласованности (когерентности) кэшей и общей базы данных. Здесь возможны различные решения – от автоматической поддержки согласованности за счет средств базового программного обеспечения управления базами данных до полного перекладывания этой задачи на прикладной уровень.

Преимуществами данной архитектуры являются

* возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;
* все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
* поддержка многопользовательской работы;
* гарантия целостности данных.

Недостатки:

* неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
* администрирование данной системы требует квалифицированного профессионала;
* высокая стоимость оборудования;
* бизнес логика приложений осталась в клиентском ПО.

При проектировании информационной системы, основанной на архитектуре "клиент-сервер", большее внимание следует обращать на грамотность общих решений. Технические средства пилотной версии могут быть минимальными (например, в качестве аппаратной основы сервера баз данных может использоваться одна из рабочих станций). После создания пилотной версии нужно провести дополнительную исследовательскую работу, чтобы выяснить узкие места системы. Только после этого необходимо принимать решение о выборе аппаратуры сервера, которая будет использоваться на практике.

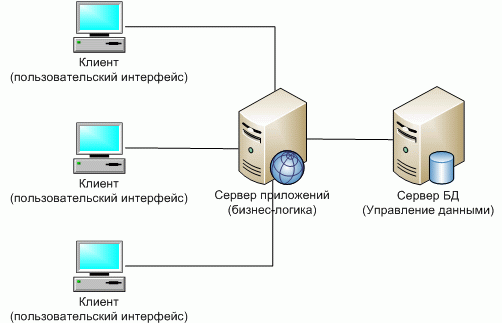
Увеличение масштабов информационной системы не порождает принципиальных проблем. Обычным решением является замена аппаратуры сервера (и, может быть, аппаратуры рабочих станций, если требуется переход к локальному кэшированию баз данных). В любом случае практически не затрагивается прикладная часть информационной системы.

Также данный вид архитектуры называют архитектурой с "толстым" клиентом.

**Многоуровневая архитектура клиент-сервер** (Multitier architecture) – разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Среди многоуровневой архитектуры клиент-сервер наиболее распространена трехуровневая архитектура (трехзвенная архитектура, three-tier), предполагающая наличие следующих компонентов приложения: клиентское приложение (обычно говорят "тонкий клиент" или терминал), подключенное к серверу приложений, который в свою очередь подключен к серверу базы данных.

Схематически такую архитектуру можно представить, как показано на рисунке



**Рис.** Представление многоуровневой архитектуры "клиент-сервер"

* Терминал – это интерфейсный (обычно графический) компонент, который представляет первый уровень, собственно приложение для конечного пользователя. Первый уровень не должен иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надежности). На первый уровень может быть вынесена и обычно выносится простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.
* Сервер приложений располагается на втором уровне. На втором уровне сосредоточена большая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы, а также погруженные в третий уровень хранимые процедуры и триггеры.
* Сервер базы данных обеспечивает хранение данных и выносится на третий уровень. Обычно это стандартная реляционная или объектно-ориентированная СУБД. Если третий уровень представляет собой базу данных вместе с хранимыми процедурами, триггерами и схемой, описывающей приложение в терминах реляционной модели, то второй уровень строится как программный интерфейс, связывающий клиентские компоненты с прикладной логикой базы данных.

В простейшей конфигурации физически сервер приложений может быть совмещен с сервером базы данных на одном компьютере, к которому по сети подключается один или несколько терминалов.

В "правильной" (с точки зрения безопасности, надежности, масштабирования) конфигурации сервер базы данных находится на выделенном компьютере (или кластере), к которому по сети подключены один или несколько серверов приложений, к которым, в свою очередь, по сети подключаются терминалы.

Плюсами данной архитектуры являются:

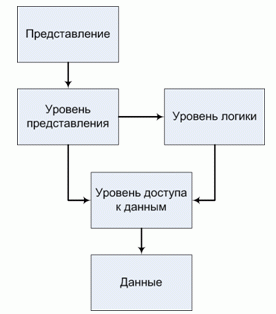
* клиентское ПО не нуждается в администрировании;
* масштабируемость;
* конфигурируемость – изолированность уровней друг от друга позволяет быстро и простыми средствами переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней;
* высокая безопасность;
* высокая 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.

Минусы:

* 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.

Некоторые авторы (например, Мартин Фаулер) представляют многозвенную архитектуру (трехзвенную) в виде пяти уровней:

1. Представление;
2. Уровень представления;
3. Уровень логики;
4. Уровень данных;
5. Данные.



**Рис..** Пять уровней многозвенной архитектуры "клиент-сервер"

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.

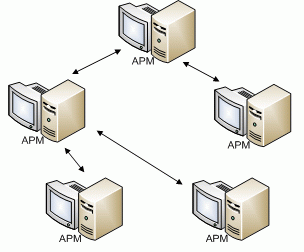
Уровень доступа к данным – это подмножество функций, обеспечивающих взаимодействие со сторонними системами, которые выполняют задания в интересах приложения.

Данные системы обычно хранятся в базе данных.

5.1.6. Архитектура распределенных систем

Такой тип систем является более сложным с точки зрения организации системы. Суть **распределенной** **системы** заключается в том, чтобы хранить локальные копии важных данных.

Схематически такую архитектуру можно представить, как показано на рисунке



**Рис.** Архитектура распределенных систем

Более 95 % данных, используемых в управлении предприятием, могут быть размещены на одном персональном компьютере, обеспечив возможность его независимой работы .Поток исправлений и дополнений, создаваемый на этом компьютере, ничтожен по сравнению с объемом данных, используемых при этом. Поэтому если хранить непрерывно используемые данные на самих компьютерах, и организовать обмен между ними исправлениями и дополнениями к хранящимся данным, то суммарный передаваемый трафик резко снизится. Это позволяет понизить требования к каналам связи между компьютерами и чаще использовать асинхронную связь, и благодаря этому создавать надежно функционирующие распределенные информационные системы, использующие для связи отдельных элементов неустойчивую связь типа Интернета, мобильную связь, коммерческие спутниковые каналы. А минимизация трафика между элементами сделает вполне доступной стоимость эксплуатации такой связи. Конечно, реализация такой системы не элементарна, и требует решения ряда проблем, одна из которых своевременная синхронизация данных.

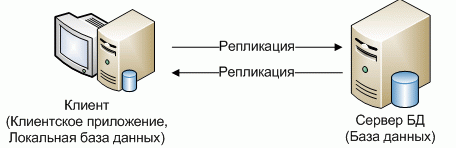
Каждый АРМ независим, содержит только ту информацию, с которой должен работать, а актуальность данных во всей системе обеспечивается благодаря непрерывному обмену сообщениями с другими АРМами. Обмен сообщениями между АРМами может быть реализован различными способами, от отправки данных по электронной почте до передачи данных по сетям.

Еще одним из преимуществ такой схемы эксплуатации и архитектуры системы, является обеспечение возможности персональной ответственности за сохранность данных. Так как данные, доступные на конкретном рабочем месте, находятся только на этом компьютере, при использовании средств шифрования и личных аппаратных ключей исключается доступ к данным посторонних, в том числе и IT администраторов.

Такая архитектура системы также позволяет организовать распределенные вычисления между клиентскими машинами. Например, расчет какой-либо задачи, требующей больших вычислений, можно распределить между соседними АРМами благодаря тому, что они, как правило, обладают одной информацией в своих БД и, таким образом, добиться максимальной производительности системы.

**Распределенные системы с репликацией**

Данными между различными рабочими станциями и централизованным хранилищем данных, передаются репликацией . При вводе информации на рабочих станциях – данные также записываются в локальную базу данных, а лишь затем синхронизируются.



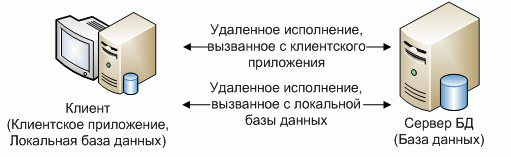
**Рис.** Архитектура распределенных систем с репликацией

***Распределенные системы с элементами удаленного исполнения***

Существуют определенные особенности, которые невозможно качественно реализовать на обычной распределенной системе репликативного типа. К этим особенностям можно отнести :

* использование данных из сущностей, которые хранятся на удаленном сервере (узле);
* использование данных из сущностей, хранящихся на разных серверах (узлах) частично;
* использование обособленного функционала, на выделенном сервере (узле).

У каждого из описанных типов используется общий принцип: программа клиент или обращается к выделенному (удаленному) серверу непосредственно или обращается к локальной базе, которая инкапсулирует в себе обращение к удаленному серверу:



**Рис..** Архитектура распределенных систем с удаленным исполнением

***Архитектура Веб-приложений***

Обычно Веб-приложения создаются как приложения в архитектуре "клиент-сервер", но серверная часть имеет различные архитектурные решения .

Изначально World Wide Web (WWW) представлялась ее создателям как "пространство для обмена информацией, в котором люди и компьютеры могут общаться между собой". Поэтому первые Веб-приложения представляли собой примитивные файловые серверы, которые возвращали статические HTML-страницы запросившим их клиентам. Таким образом, Веб начиналась как документо-ориентированная.

Следующим этапом развития Веб стало появление понятия приложений, которые базировались на таких интерфейсах, как CGI (или FastCGI), а в дальнейшем – на ISAPI. Common Gateway Interface (CGI) – это стандартный интерфейс работы с серверами, позволяющий выполнять серверные приложения, вызываемые через URL. Входной информацией для таких приложений служило содержимое HTTP-заголовка (и тело запроса при использовании протокола POST). CGI-приложения генерировали HTML-код, который возвращался браузеру. Основной проблемой CGI-приложений было то, что при каждом клиентском запросе сервер выполнял CGI-программу в реальном времени, загружая ее в отдельное адресное пространство.

Появление Internet Server API (ISAPI) позволило не только решить проблемы производительности, которые возникали с CGI-приложениями, но и предоставить в распоряжение разработчиков более богатый программный интерфейс. ISAPI DLL можно было ассоциировать с расширениями имен файлов через специальную мета-базу. Эти два механизма (CGI и ISAPI) послужили основой создания первого типа Веб-приложений, в которых, в зависимости от каких-либо клиентских действий, выполнялся серверный код. Таким образом, стала возможной динамическая генерация содержимого Веб-страниц и наполнение Веб перестало быть чисто статическим.

Интерфейс ISAPI – это особенность Microsoft Internet Information Server. ISAPI-приложения представляют собой динамические загружаемые библиотеки (DLL), которые выполняются в адресном пространстве Веб-сервера. У других Веб-серверов через некоторое время также появилась возможность выполнять приложения, реализованные в виде библиотек. В случае Веб-серверов Netscape этот программный интерфейс назывался NSAPI (Netscape Server API). У довольно популярного Веб-сервера Apache также имеется возможность выполнять Веб-приложения, реализованные в виде библиотек; такие библиотеки называются Apache DSO (Dynamic Shared Objects).

Естественно, что при использовании как CGI-, так и ISAPI-приложений разработчики в основном решали одни и те же задачи, поэтому естественным шагом стало появление нового, высокоуровневого интерфейса, который упростил задачи генерации HTML-кода, позволил обращаться к компонентам и использовать базы данных. Таким интерфейсом стала объектная модель Active Server Pages (ASP), построенная на основе ISAPI-фильтра.

Основной идеей ASP с точки зрения создания интерфейса приложения является то, что на Веб-странице присутствуют фрагменты кода, который интерпретируется Веб-сервером и вместо которого пользователь получает результат выполнения этих фрагментов кода.

Вскоре после появления ASP были созданы и другие технологии, реализующие идею размещения внутри Веб-страницы кода, выполняемого Веб-сервером. Наиболее известная из них на сегодняшний день – технология JSP (Java Server Pages), основной идеей которой является однократная компиляция Java-кода (сервлета) при первом обращении к нему, выполнение методов этого сервлета и помещение результатов выполнения этих методов в набор данных, отправляемых в браузер.

Новейшая версия технологии Active Server Pages – ASP .NET, являющаяся ключевой в архитектуре Microsoft .NET Framework. С помощью ASP .NET можно создавать Веб-приложения и Веб-сервисы, которые не только позволяют реализовать динамическую генерацию HTML-страниц, но и интегрируются с серверными компонентами и могут использоваться для решения широкого круга бизнес-задач, возникающих перед разработчиками современных Веб-приложений.

В общем случае клиентом Веб-сервера может быть не только персональный компьютер, оснащенный обычным Веб-браузером. Одновременно с широким распространением мобильных устройств появилась и проблема предоставления Веб-серверами данных, которые могут быть интерпретированы этими устройствами. Поскольку мобильные устройства обладают характеристиками, отличными от характеристик персональных компьютеров (ограниченным размером экрана, малым объемом памяти, а нередко и невозможностью отобразить что-либо, кроме нескольких строк черно-белого текста), для них существуют и другие протоколы передачи данных (WAP – Wireless Access Protocol) и соответствующие языки разметки (WML – Wireless Markup Language, СHTML – Compact HTML и т.п.). При этом возникает задача передачи данных на мобильное устройство в соответствующем формате (и для этой цели существуют специальные сайты), либо, что представляется более удобным, происходит опознание типа устройства в момент его обращения к серверу и преобразование исходного документа (например, в формате XML) в формат, требующийся данному мобильному устройству (например, с помощью XSLT-преобразования).

Другим способом поддержки различных типов клиентов является создание "разумных" серверных компонентов, которые способны генерировать различный код в зависимости от типа клиента. Такой подход, в частности, реализован в Microsoft ASP .NET.

Другим направлением развития клиентских частей Веб-приложений стало размещение некоторой части логики приложения (такой как проверка корректности вводимых данных) в самом Веб-браузере. В частности, современные Веб-браузеры способны интерпретировать скриптовые языки (VBScript, JavaScript), код на которых, как и ASP-код, внедряется в Веб-страницу, но интерпретируется не Веб-сервером, а браузером и соответственно выполняется на клиентском устройстве. Кроме того, современные браузеры способны отображать и выполнять Java-аплеты – специальные Java-приложения, которые пользователь получает в составе Веб-страницы, а некоторые из браузеров могут также служить контейнерами для элементов управления ActiveX – выполняющихся в адресном пространстве браузера специальных COM-серверов, также получаемых в составе Веб-страницы. И в Java-аплетах, и в элементах управления ActiveX можно реализовать практически любую функциональность.

Отметим, что с ростом объема используемых данных и числа посетителей Веб-сайтов возрастают и требования к надежности, производительности и масштабируемости Веб-приложений. Следующим этапом эволюции подобных приложений стало отделение бизнес-логики, реализованной в Веб-приложении, а нередко и сервисов обработки данных и реализации транзакций от его интерфейса. В этом случае в самом Веб-приложении обычно остается так называемая презентационная часть, а бизнес-логика, обработка данных и реализация транзакций переносятся в сервер приложений в виде бизнес-объектов. В зависимости от типа сервера приложений подобные бизнес-объекты могут быть выполняющимися самостоятельно COM-серверами, CORBA-серверами, а также объектами COM+, выполняющимися с помощью служб компонентов Windows 2000, или объектами EJB (Enterprise Java Beans), исполняемыми сервером приложений, поддерживающим спецификаци ю J2EE (Java 2 Enterprise Edition). В качестве механизма доступа к данным подобные объекты могут использовать OLE DB, ODBC, JDBC (это зависит от того, как реализован бизнес-объект).

Нередко подобные бизнес-объекты предоставляют доступ к данным корпоративных информационных систем либо реализуют какую-либо часть их функциональности. Нередко они позволяют, например, интегрировать Веб-сайт с CRM-системами (Customer Relationship Management) или с ERP-системами (Enterprise Resource Planning), сохраняя в корпоративных системах сведения о посетителях сайта и предоставляя потенциальным клиентам сведения об имеющейся продукции для осуществления заказов.

Поскольку современный Интернет – это не столько средство демонстрации присутствия компании на рынке или инструмент маркетинга, сколько инструмент ведения бизнеса, достаточно важными становятся задачи реализации организации через Интернет таких взаимоотношений с клиентами, как продажа товаров и услуг. И здесь довольно важными становятся решения для электронной коммерции типа "предприятие-клиент" (B2C – business-to-consumer). Не менее важными становятся и задачи интеграции Веб-приложений c данными и приложениями партнеров с целью реализации схемы "предприятие-предприятие" (B2B – business-to-business), позволяющей заключать торговые сделки между предприятиями, обмениваться каталогами товаров, проводить аукционы, создавать электронные торговые площадки.

Отметим, что, будучи составной частью подобного решения, Веб-сервер должен уметь не только выполнять приложения и взаимодействовать с сервером приложений, но и использовать сервисы интеграции, сервисы управления приложениями и данными, а также сервисы для разработчиков.

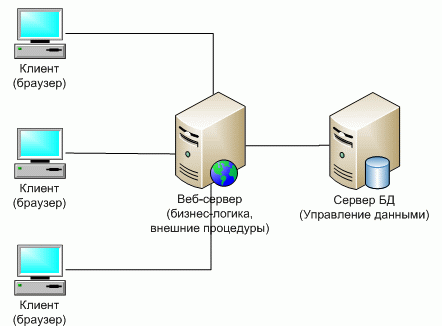
Следующим шагом эволюции Веб-приложений, помимо доступа к корпоративным данным и данным партнеров, стало получение доступа к корпоративным приложениям. Для решения этой задачи интеграции Веб-приложений с внутренними информационными системами предприятий и с приложениями, обеспечивающими взаимодействие с клиентами и партнерами, используются специальные решения, называемые корпоративными порталами.

Нередко частью портального решения являются средства управления информационным наполнением Веб-сайта – ведь объем данных, доступных пользователям с помощью сайтов крупных компаний и порталов, сейчас таков, что управление этими данными "вручную" не представляется возможным.

Обобщая вышесказанное можно выделить основные особенности веб-архитектуры :

* отсутствие необходимости использовать дополнительное ПО на стороне клиента – это позволяет автоматически реализовать клиентскую часть на всех платформах;
* возможность подключения практически неограниченного количества клиентов;
* благодаря единственному месту хранения данных и наличия системы управления базами данных обеспечиваются минимальные требования для поддержания целостности данных;
* доступность при работоспособности сервера и каналов связи;
* недоступность при отсутствии работоспособности сервера или каналов связи;
* достаточно низкая скорость Веб сервера и каналов передачи данных;
* относительно объема данных – архитектура Веб систем не имеет существенных ограничений.

Схематически такую архитектуру (в трехзвенном варианте) можно представить, как показано на рисунке.



**Рис.** Архитектура Веб-приложений

***Сервис-ориентированная архитектура***

Решение многих описанных выше задач, возникающих при создании современных Веб-приложений, теперь начинает возлагаться на Веб-сервисы – не зависящие от платформы, объектной модели и клиента программные компоненты, которые можно вызывать из клиентских Веб-приложений (а также из самих Веб-сервисов) через основанный на протоколе HTTP и языке XML протокол SOAP [[20](http://www.intuit.ru/studies/courses/611/467/literature#literature.4.20)]. Для описания Веб-сервисов используется XML-подобный язык WSDL, а для организации реестров Веб-сервисов, в которых разработчики и компании могут искать необходимые им сервисы, а также публиковать данные о своих сервисах – интерфейс UDDI.

Поддержка Веб-сервисов стала одним из главных стратегических направлений для многих компаний, специализирующихся на выпуске серверов приложений, систем управления базами данных и средств разработки приложений.

Сервис-ориентированная архитектура **(SOA, service-oriented architecture)** – модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами .

OASIS (Организация по распространению открытых стандартов структурированной информации) определяет SOA следующим образом (OASIS Reference Model for Service Oriented Architecture V 1.0): **Сервисно-ориентированная архитектура** – это парадигма организации и использования распределенных информационных ресурсов таких как: приложения и данные, находящихся в сфере ответственности разных владельцев, для достижения желаемых результатов потребителем, которым может быть: конечный пользователь или другое приложение.

В основе SOA лежат принципы многократного использования функциональных элементов ИТ, ликвидации дублирования функциональности в ПО, унификации типовых операционных процессов, обеспечения перевода операционной модели компании на централизованные процессы и функциональную организацию на основе промышленной платформы интеграции.

Компоненты программы могут быть распределены по разным узлам сети, и предлагаются как независимые, слабо связанные, заменяемые сервисы-приложения. Программные комплексы, разработанные в соответствии с SOA, часто реализуются как набор веб-сервисов, интегрированных при помощи известных стандартных протоколов (SOAP, WSDL, и т. п.)

Интерфейс компонентов SОА-программы предоставляет инкапсуляцию деталей реализации конкретного компонента (ОС, платформы, языка программирования, вендора, и т. п.) от остальных компонентов. Таким образом, SOA предоставляет гибкий и элегантный способ комбинирования и многократного использования компонентов для построения сложных распределенных программных комплексов.

SOA хорошо зарекомендовала себя для построения крупных корпоративных программных приложений. Целый ряд разработчиков и интеграторов предлагают инструменты и решения на основе SOA (например, платформы IBM WebSphere, Oracle/BEA Aqualogic, Microsoft Windows Communication Foundation, SAP NetWeaver, ИВК Юпитер, TIBCO, Diasoft).

Основными целями применения SOA для крупных информационных систем, уровня предприятия, и выше являются :

* сокращение издержек при разработке приложений, за счет упорядочивания процесса разработки;
* расширение повторного использования кода;
* независимость от используемых платформ, инструментов, языков разработки;
* повышение масштабируемости создаваемых систем;
* улучшение управляемости создаваемых систем.

Принципы SOA:

* архитектура, как таковая, не привязана к какой-то определенной технологии;
* независимость организации системы от используемой вычислительной платформы (платформ);
* независимость организации системы от применяемых языков программирования;
* использование сервисов, независимых от конкретных приложений, с единообразными интерфейсами доступа к ним;
* организация сервисов как слабосвязанных компонентов для построения систем.

Архитектура не привязана к какой-то определенной технологии. Она может быть реализована с использованием широкого спектра технологий, включая такие технологии как REST, RPC, DCOM, CORBA или веб-сервисы. SOA может быть реализована, используя один из этих протоколов и, например, может использовать, дополнительно, механизм файловой системы, для обмена данными.

Главное, что отличает SOA, это использование независимых сервисов, с четко определенными интерфейсами, которые, для выполнения своих задач, могут быть вызваны неким стандартным способом, при условии, что сервисы заранее ничего не знают о приложении, которое их вызовет, а приложение не знает, каким образом сервисы выполняют свою задачу.

SOA также может рассматриваться как стиль архитектуры информационных систем, который позволяет создавать приложения, построенные путем комбинации слабосвязанных и взаимодействующих сервисов. Эти сервисы взаимодействуют на основе какого-либо строго определенного платформо-независимого и языково-независимого интерфейса (например, WSDL). Определение интерфейса скрывает языково-зависимую реализацию сервиса.

Таким образом, системы, основанные на SOA, могут быть независимы от технологий разработки и платформ (таких как Java, .NET и т. д.). К примеру, сервисы, написанные на C#, работающие на платформах .Net и сервисы на Java, работающие на платформах Java EE, могут быть с одинаковым успехом вызваны общим составным приложением. Приложения, работающие на одних платформах, могут вызывать сервисы, работающие на других платформах, что облегчает повторное использование компонентов.

SOA может поддерживать интеграцию и консолидацию операций в составе сложных систем, однако SOA не определяет и не предоставляет методологий или фреймворков для документирования сервисов.

Языки высокого уровня, такие как BPEL, или спецификации, такие как WS-CDL и WS-Coordination, расширяют концепцию сервиса, предоставляя метод оркестрации, для объединения мелких сервисов в более обширные бизнес-сервисы, которые, в свою очередь, могут быть включены в состав технологических процессов и бизнес-процессов, реализованных в виде составных приложений или порталов.

5.1.9. Ключевые термины

Архитектура, Централизованная архитектура, Архитектура "файл-сервер", Двухзвенная архитектура "клиент-сервер", , Терминал, Сервер приложений, Сервер базы данных, Архитектура распределенных систем, Архитектура Веб-приложений, Сервис-ориентированная архитектура.

Архитектура облачных приложений.

В настоящее время технологии "облачных" вычислений приобретают все большую популярность, а концепция Cloud Computing является одной из самых модных тенденций развития информационных технологий. По оценкам Gartner, "облака" — один из главных приоритетов бизнеса на 2010 год. Крупнейшие мировые ИТ вендоры (Microsoft, Amazon, Google и прочие) так или иначе внедряют сервисы "облачных" вычислений.

**Сегодня под облачными вычислениями обычно понимают возможность получения необходимых вычислительных мощностей по запросу из сети**, причем пользователю не важны детали реализации этого механизма и он получает из этого "облака" все необходимое. Ярким примером могут служить поисковые системы, интерфейс которых очень прост, но в то же время они предоставляют пользователям огромные вычислительные ресурсы для поиска нужной информации. Сегодня крупные вычислительные центры не только позволяют хранить и обрабатывать внутри себя определенные данные, но и дают возможности для создания собственных виртуальных дата-центров, позволяя молодым компаниям не тратить силы на создание всей инфраструктуры с нуля. На сегодняшний день существует множество определений "облачных вычислений". Зачастую они расходятся в своем значении и акцентах. Рассмотрим некоторые из этих определений для того чтобы понять что такое "облачные" вычисления с разных точек зрения.

**Облачные вычисления представляют собой динамически масштабируемый способ доступа к внешним вычислительным ресурсам в виде сервиса, предоставляемого посредством Интернета, при этом пользователю не требуется никаких особых знаний об инфраструктуре "облака" или навыков управления этой "облачной" технологией**.

**Cloud computing – это программно-аппаратное обеспечение, доступное пользователю через Интернет или локальную сеть в виде сервиса, позволяющего использовать удобный интерфейс для удаленного доступа к выделенным ресурсам (вычислительным ресурсам, программам и данным)**. Компьютер пользователя выступает при этом рядовым терминалом, подключенным к Сети. Компьютеры, осуществляющие cloud computing, называются "вычислительным облаком". При этом нагрузка между компьютерами, входящими в "вычислительное облако", распределяется автоматически.

**Облачные вычисления - это новый подход, позволяющий снизить сложность ИТ-систем, благодаря применению широкого ряда эффективных технологий, управляемых самостоятельно и доступных по требованию в рамках виртуальной инфраструктуры, а также потребляемых в качестве сервисов**. Переходя на частные облака, заказчики могут получить множество преимуществ, среди которых снижение затрат на ИТ, повышение качества предоставления сервиса и динамичности бизнеса".

**"Облако" является новой бизнес-моделью для предоставления и получения информационных услуг**.Эта модель обещает снизить оперативные и капитальные затраты. Она позволяет ИТ департаментам сосредоточиться на стратегических проектах, а не на рутинных задачах управления собственным центром обработки данных.

Облачные вычисления – это не только технологическая инновация в ИТ, но и способ создания новых бизнес-моделей, когда у небольших производителей ИТ-продуктов, в том числе и в регионах, появляется возможность быстрого предложения рынку своих услуг и мало затратного способа воплощения своих бизнес-идей. Поддержка облачных вычислений в сочетании с инвестициями в молодые компании создают быстро развивающуюся экосистему инновационных производств.

Облачные вычисления являются рыночным ответом на систематическую специализацию и усиление роли аутсорсинга в ИТ. По сути, переход к облачным вычислениям означает аутсорсинг традиционных процессов управления ИТ-инфраструктурой профессиональными внешними поставщиками. Большинство современных поставщиков решений сферы облачных вычислений предоставляет возможность не только использовать существующие облачные платформы, но и создавать собственные, отвечающие технологическим и юридическим требованиям заказчиков.

"Облачные вычисления" работают следующим образом: вместо приобретения, установки и управления собственными серверами для запуска приложений, происходит аренда сервера у Microsoft, Amazon, Google или другой компании. Далее пользователь управляет своими арендованными серверами через Интернет, оплачивая при этом только фактическое их использование для обработки и хранения данных. Вычислительные облака состоят из тысяч серверов, размещенных в датацентрах, обеспечивающих работу десятков тысяч приложений, которые одновременно используют миллионы пользователей. Непременным условием эффективного управления такой крупномасштабной инфраструктурой является максимально полная автоматизация. Кроме того, для обеспечения различным видам пользователей - облачным операторам, сервис-провайдерам, посредникам, ИТ-администраторам, пользователям приложений - защищенного доступа к вычислительным ресурсам облачная инфраструктура должна предусматривать возможность самоуправления и делегирования полномочий.

Концепция "облачных" вычислений появилась не на пустом месте, а явилась результатом эволюционного развития информационных технологий за последние несколько десятилетий и ответом на вызовы современного бизнеса. Аналитики Гартнер групп (Gartner Group) называют "Облачные" вычисления — самой перспективной стратегической технологией будущего, прогнозируя перемещение большей части информационных технологий в "облака" в течение 5–7 лет. По их оценкам, к 2015 году объём рынка облачных вычислений достигнет 200 миллиардов долларов.

В России технологии "облачных" вычислений делают лишь первые шаги. Несмотря на существующие предложения со стороны крупнейших международных корпораций Microsoft, IBM, Intel, NEC, а также ряда отечественных ИТ-поставщиков спрос на облачные сервисы в России пока невелик. Однако, по прогнозу аналитической компании IDC, за ближайшие 5 лет рынок облачных услуг в России вырастет более чем на 500% и составит 113 миллионов долларов.

Перспективы "облачных" вычислений неизбежны, поэтому знание об этих технологиях необходимо любому специалисту, который связывает свою текущую или будущую деятельность с современными информационными технологиями.