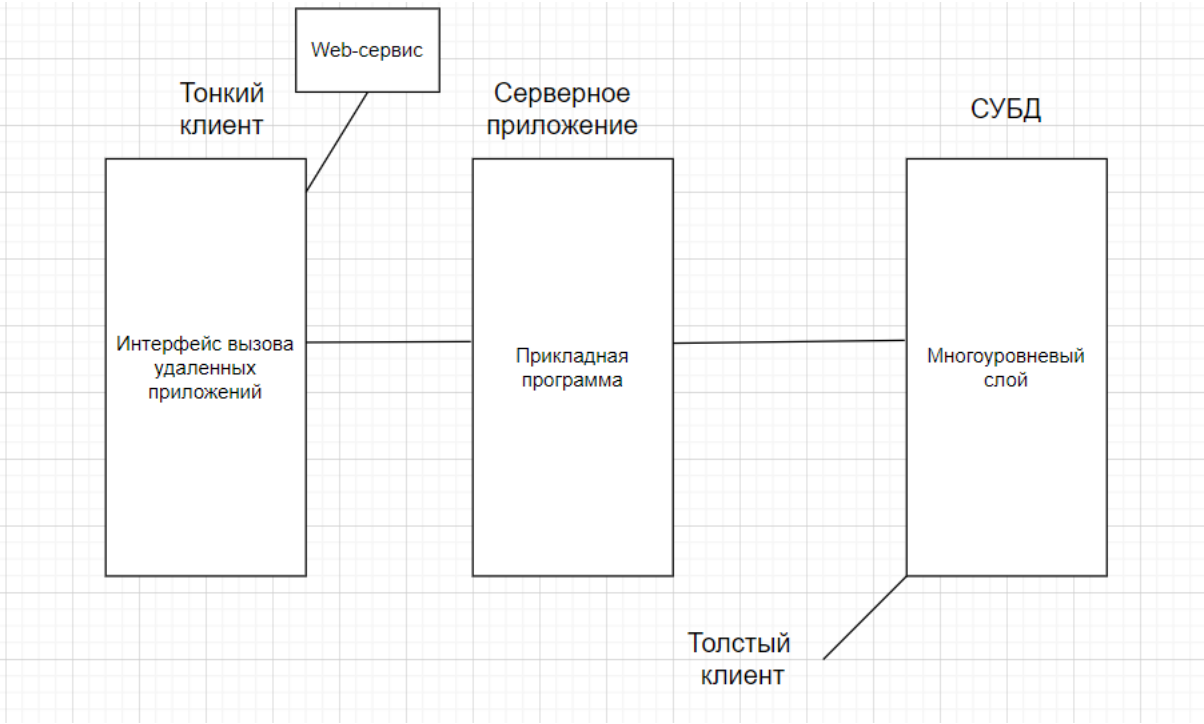
# xpМногоуровневая архитектура доступа к данным

Обычно для небольших организаций разработчики применяют двухуровневую архитектуру клиент-сервер, когда с рабочих станций осуществляется удаленный доступ к БД. В самых простых, примитивных системах даже не используются возможности, предоставленные СУБД, такие как триггеры и хранимые процедуры. Хотя такие системы и именуются клиент-серверными, они имеют весьма мало общего с истинными распределенными приложениями. Более того, идеология "толстого" клиента принуждает к установке на рабочее место весьма производительных компьютеров, потому что на них производятся все основные вычисления, и обмен данными с удаленными серверами БД производится сквозь толстый многоуровневый слой драйверов, которые должны быть установлены на персональной рабочей станции и модернизированы их разработчиками для каждого рабочего места.

Таким образом, если пропускная способность сети недостаточно велика или недостаточно эффективно организован поток прохождения транзакций, то при этом быстродействующие процессоры клиентских машин совершенно бездействуют, а в обратном случае, наоборот, сервер БД сильно загружен и не успевает отвечать каждому из многочисленных клиентов.

Практика показывает, при числе одновременно работающих клиентов более 30 необходимо переходить на трехзвенную архитектуру. В трехзвенной архитектуре всю логику работы с сервером БД можно возложить на специальный сервер приложений. Разделенное на отдельные фрагменты приложение уменьшает нагрузку и на машину-клиента, и на сервер БД, перенося соответствующие операции на специализированный сервер приложений.

Серверная часть приложений лучше защищена, а сами приложения могут либо непосредственно адресоваться к другим серверным приложениям, либо масштабировать запросы к ним.




## Клиентская часть

Прикладная программа доступна с любого компьютера, на котором установлен браузер. Пользователю нет необходимости изучать интерфейс прикладной программы, потому что он всегда преобразуется к стандарту HTML-страницы. Это помогает снизить затраты на обучение. Кроме того, пользователи совершенно не знают особенности платформы и операционной системы, поскольку они имеют дела только с браузером.

## Серверная часть

Это приложение доступно любому пользователю сети Internet/Intranet, имеющего право обращаться к ним. Поскольку все операции по сопровождению и совершенствованию системы производятся на сервере, отпадает необходимость сопровождать и модернизировать части приложений, находящихся на машинах-клиентах. Такая координация легко обеспечивает работу очень большого числа пользователей.

Клиентские приложения обращаются не непосредственно к серверной СУБД путем вызова функций клиентских API, а к серверу приложений, являющимся для них источником данных. При этом собственно клиентская часть серверной СУБД на рабочей станции, где используется клиентское приложение, присутствовать не обязана. Вместо них используется одна единственная динамически загружаемая библиотека dbclient.ddl.

Таким образом, созданная информационная система становится трехзвенной, а сервер приложений является средний звеном в цепи тонкий клиент - сервер приложений - сервер БД

Данная система может быть реализована с помощью набора компонентов и классов Delphi, обеспечивающих создание серверов приложений и клиентских прикладных программ.

## Технология MIDAS

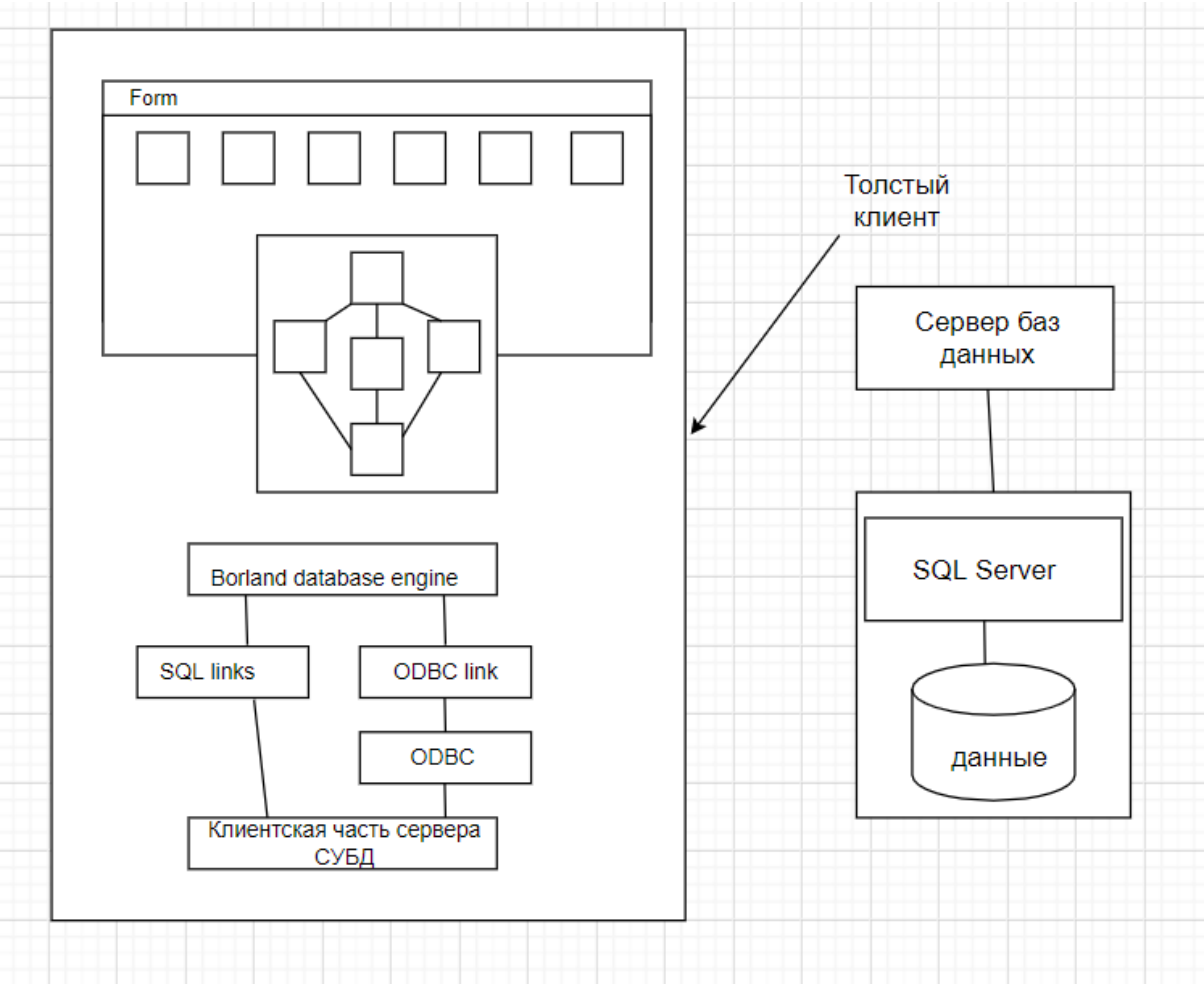
Эта технология предназначена для эксплуатации созданных с помощью C++ Builder и Delphi удаленных серверов авторизации, предоставляющих своим контроллерам доступ к данным серверных СУБД.

В комплект поставки Delphi входит Midas Development KIT, представляющий собой ограниченную версию MIDAS, с помощью которой можно разработать такой сервер приложений(но не для коммерческой эксплуатации).

Подавляющее большинство клиентских приложений для доступа к источникам данных используют вызовы функций прикладных программных клиентских частей соответствующих серверных СУБД. В приложениях, написанных с помощью средств разработки Borland эти вызовы осуществляются посредством использования библиотеки Borland Database Engine(BDE).

Клиентское приложение требует наличия на рабочей станции конечного пользователя клиентской части серверной СУБД и присутствия в оперативной памяти набора динамически загружаемых библиотек как из клиентской части, так и из BDE.

Если используется ODBC, то на рабочей станции нужен ODBC драйвер или ODBC администратор.

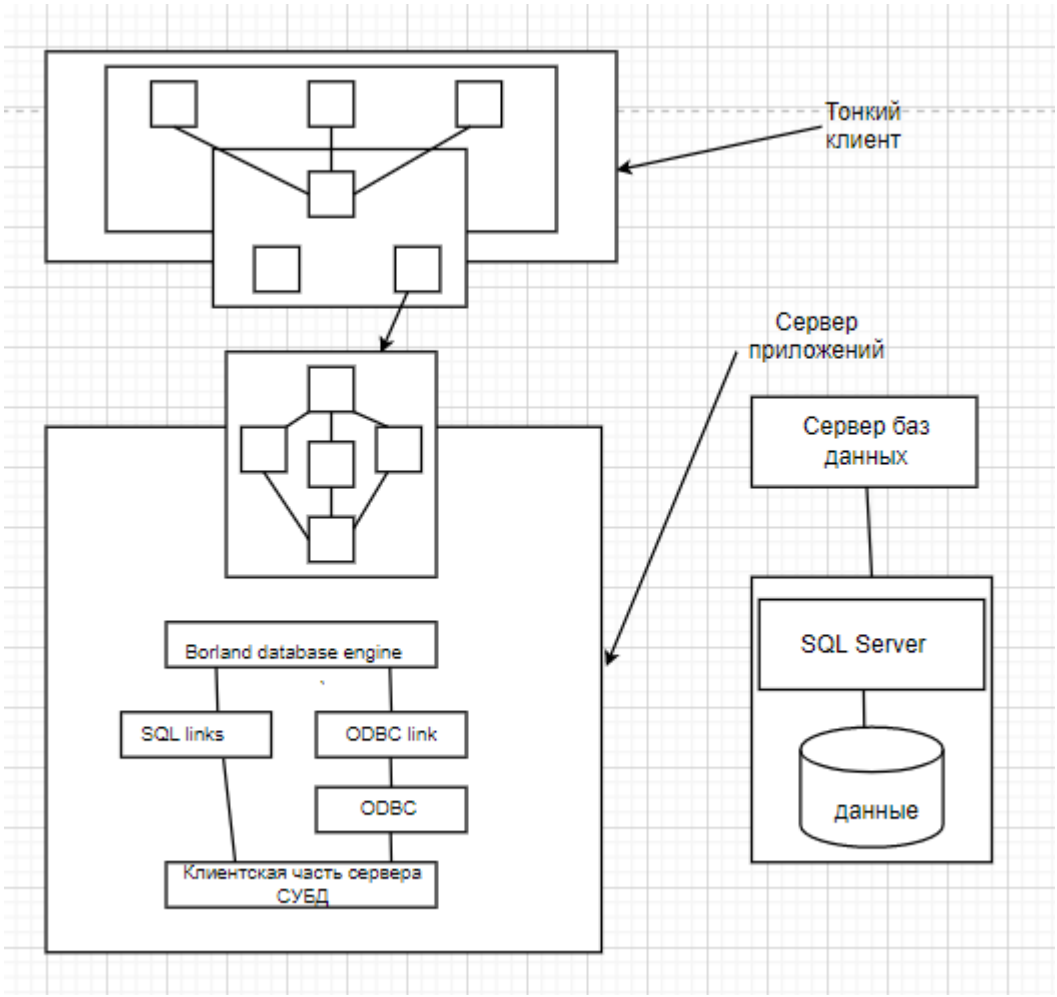



Классические двухуровневые информационные системы

## Тонкие клиенты

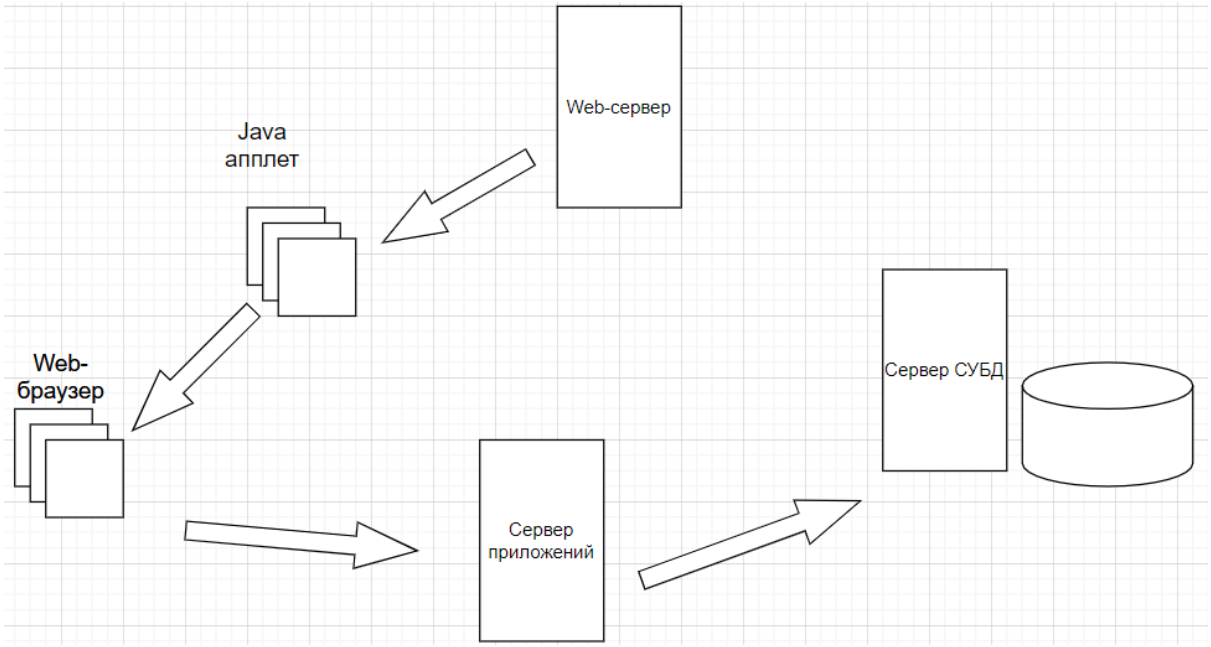
Обычное клиентское приложение делится на две части. Первая часть, содержащая так называемый презентационный уровень работы с данными(пользовательский интерфейс), называется обычно "тонким клиентом". Она не содержит в своем составе ни BDE, ни иных библиотек, ни ODBC, ни клиентской части серверной СУБД. Вторая часть реализует функциональность, связанную с доступом к данным(а нередко и какую нибудь иную функциональность, например статическую обработку этих данных или генерацию отчетов) и, как правило, обладает минимальным пользовательским интерфейсом. Эта часть называется обычно сервером приложений(или сервером доступа к данным - Data Access Server) и является клиентом серверной СУБД.

"Тонкие" клиентские приложения при получении данных обращаются не непосредственно к серверной СУБД с помощью вызова функций клиентских API, а к серверу приложений, являющихся для них источником данных. При получении такого запроса сервер приложений обращается к серверу баз данных, клиентом которого он является, со своим собственным запросом. Получив от сервера результат выполнения собственного запроса, сервер приложений передает данные клиенту.




Каким образом на практике реализуется данная технология? Все способы ее реализации базируются на идее осуществления вызовов удаленных процедур путем маршаллинга между объектами клиента и внутри сервера. В частности, реализация удаленного доступа может использовать технологию и стандарты PCE, CORBA и др. На платформе Windows - это DCOM и OLE Enterprize. В этом случае сервера доступа к данным реализуется как удаленный сервер автоматизации по запросу клиентов внутри своего адресного пространства.

## Использование активных форм для создание тонкого клиента




Как уже было сказано, наличие "тонких" клиентов не только снижает требования к ресурсам рабочей станции, но и упрощает конфигурацию программного обеспечения для них. Следствием этого является возможность осуществлять поставку таких приложений не путем создания отдельного дистрибутива, а через Internet, используя Web-сервер в качестве источника очередной версии приложения и Web-браузер в качестве средства для его установки. При этом присутствует возможность выполнения такого приложения непосредственно в браузере, используя тег <ответ>

В принципе возможны ситуации, когда сервер приложений, Web-сервер и пользователь находятся в трех разных городах.

## WCF(Windows Communication Foundation)

Начиная с версии .NET 3.0 был представлен API -интерфейс, специально предназначенный для построения распределенных систем - WCF.

В отличие от других распределенных API интерфейсов(DCOM) WCF предлагает единую унифицированную и распределенную объектную модель для программирования, которая может использоваться для взаимосвязи с моделями ранее реализованных распределенных технологий.

API интерфейсов распределенных систем:

**DCOM**

До появления платформы .NET основным API интерфейсом в MS была распределенная модель компонентных объектов - DCOM.

Недостатки: 1.API используется исключительно на Windows.

2. Сама модель DCOM не представляла собой оснований для поддержки полноценных изменений, включая различные ОС или распределенных данных между архитектурами.

3. DCOM была наилучшим образом приспособлена для разработки внутренних приложений поскольку передача типов COM за пределами клиентов влекла за собой дополнительные сложности.

DCOM устарел.

**Веб-служба XML**

В отличие от традиционных браузерных веб-приложений веб-служба обеспечивает способ предоставления функциональности удаленных компонентов через стандарты веб-приложений(HTTP и XML).

**WCF** - это инструментальный набор распределенных вычислений, который интегрирует все ранее независимые технологии распределенной обработки в единый API интерфейс.

Например, при создании внутреннего приложения, где все подключенные машины работают под управлением Windows можно использовать протокол TCP. Те же самые службы также могут быть представлены с применением протоколов HTTP и SOAP, чтобы позволить внешним клиентам пользоваться их функциональностью.