Содержание:

ВВЕДЕНИЕ

Как и любая другая сложно организованная инфраструктура, программное обеспечение должно основываться на прочном фундаменте. Неверное определение ключевых принципов, некорректное определение общих вопросов или отсутствие возможности выявить отдаленные последствия основных решение способны вызвать критические нарушения в работе системы. Современные средства разработки и платформы ускоряют рутинные процессы создания приложений, но не лишают необходимости тщательного проектирования в строгом соответствии с требованиями и на основании конкретных сценариев. Неверно определенная архитектура приложения вызывает нестабильность в работе программного обеспечения, обуславливает невозможность поддержки существующих или перспективных бизнес-процессов, создает сложности при развертывании и администрировании в среде эксплуатации. В процессе выбора оптимального решения проектировщик системы обязан ориентироваться на потребности пользователя, возможности инфраструктуры и поставленные бизнес-задачи.

Востребованность информационных технологий в сфере бизнеса и в повседневной жизни способствует дальнейшему совершенствованию технологий, а также их усложнению в попытке решить поставленные задачи в максимально возможном объеме. Данные преобразования происходят как в области программного, так и аппаратного обеспечения.

Наступление эры персональных компьютеров сделало доступным применение вычислительных технологий не только крупном, но и среднем и малом бизнесе, отодвинув на второй план применение мейнфреймов, популярных на заре информационных технологий. Однако это не уменьшило необходимость обмена накопленными данными, их анализа, обеспечения совместной работы.

Архитектура «клиент-сервер» является одним из вариантов решения этих задач в современных условиях, чья успешность определила ее популярность во всех сферах деятельности. По сей день она остается динамично развивающейся технологией, с каждым днем предоставляющей все больше возможностей для

успешного решения бизнес-задач.

В связи с этим, изучение и анализ данной технологии представляются актуальными.

Целью данной работы является детальное рассмотрение вариантов архитектуры «клиент-сервер».

Обозначенная цель определяет следующие задачи курсовой работы:

1. Изучить главные характеристики технологии «клиент-сервер»;
2. Ознакомиться с основными моделями архитектуры «клиент-сервер»;

3.Детально рассмотреть варианты архитектуры «клиент-сервер», в зависимости от количества уровней, а также типов применяемого программного обеспечения.

ГЛАВА.1 ИСТОРИЯ РАЗВИТИЯ ТЕХНОЛОГИЙ ОБРАБОТКИ ИНФОРМАЦИИ.

1.1.Централизованная модель обработки информации.

Исторически первой наибольшее распространение получила централизованная модель. Ее расцвет связывают с появлением первых мэйнфреймов IBM System/360 в 1964 году и их последующих поколений.

Данная модель предусматривает использование одного или нескольких вычислительных устройств высокой производительности. Данные устройства обеспечивают выполнение большого количества различных приложений. При этом происходит совместное использование приложениями одних и тех же аппаратных ресурсов: оперативной памяти, дисковых и вычислительных систем. Все пользователи имеют непосредственное подключение к системе.

Для работы данных систем применяются специализированные операционные системы, такие как IBM Z/OS, Sun Solaris, HP-UX.

Централизованная модель обработки информации имеет ряд преимуществ, среди которых одним из наиболее важных является уменьшение затрат на передачу данных и сравнительно высокий уровень безопасности за счет нахождения приложений непосредственно рядом с данными. Также к преимуществом данной модели принято относить наличие интегрированных средств администрирования [11].

К недостаткам данных систем относят высокую стоимость оборудования, сложности масштабирования, трудности в настройках под меняющиеся потребности (пользователь зависит от администратора и лишен возможности настроить в полной мере рабочую среду под свои потребности). Компания Gartner предоставила прогноз, что последний мэйнфрейм будет выключен в 1993 году. Однако, несмотря на имеющиеся недостатки, системы централизованной обработки на основе мэйнфреймов продолжают использоваться и в наше время.

Несколько позже, примерно в 1980-х годах, появилась новая модель обработки информации, получившая название клиент-серверной.

1.2.«Клиент-серверная» модель обработки информации.

Клиент-сервер — форма вычислительной или сетевой архитектуры, подразумевающая распределение заданий или сетевой нагрузки между поставщиками услуг (серверами) и заказчиками (клиентами) [6].

Данная модель предусматривает распределение вычислительных ресурсов по большому количеству вычислительных систем, не имеющих совместно используемых ресурсов. При этом каждый отдельный узел обеспечивает функционирования одного или нескольких приложений, а их общее взаимодействие обеспечивается посредством сети. Данная модель возникла с целью получить возможность отказаться от применения мощных вычислительных систем, в связи с их высокой стоимостью и низким уровнем «гибкости».

В основе данной архитектуры лежит программное обеспечение, расположенное, как правило, на различных вычислительных системах и взаимодействующее через вычислительную сеть с помощью стандартизированных сетевых протоколов.

Серверное программное обеспечение выполняет функцию провайдера услуг и предоставляет свои ресурсы, как в виде вычислений, так и в виде данных и их обработки [1].

Высокие требования к серверному программному обеспечению обуславливают характеристики аппаратной платформы, подразумевающие в зависимости от предоставляемых возможностей, повышенную производительность, большой обрабатываемых объем данных, либо их сочетание.

В качестве клиента в наши дни используются, как правило, персональные компьютеры (так называемые «интеллектуальные терминалы», способные запускать собственное программное обеспечение, в отличие от классических терминалов, обеспечивающих только отображение предоставленной сервером информации). Подобная реализация архитектуры клиент-сервер позволяет обеспечить распределенную обработку информации, при которой часть работы выполняется на клиентской машине (например, отображение пользовательского интерфейса, заключительная обработка), а часть на сервере. Такая организация снижает нагрузку на серверное аппаратное обеспечение, позволяя увеличить производительность, либо количество клиентов одновременно работающих с данным сервером.

Поскольку в рамках аппаратного представления компьютеры классифицируются также на клиентов (вычислительные системы, запрашивающие ресурсы) и серверы (вычислительные системы, предоставляющие ресурсы), то одна и та же машина может выполнять функции сервера и клиента одновременно [4].

Применение технологии клиент-сервер допускает использование различных аппаратных платформ и операционных систем в пределах единого окружения. Целостность системы достигается путем применения стандартизированных сетевых протоколов и единых интерфейсов приложений.

Развитие парадигмы объектно-ориентированного программирования позволило унифицировать подход к созданию клиент-серверных приложений, разбив их на три основных компонента [8]:

- Компонент представления, обеспечивающий пользовательский интерфейс.

- Компонент прикладного назначения, предназначенный для решения конкретных задач в рамках приложения.
- Компонент управления, реализующий доступ к требуемым ресурсам (Рис.1).

Рисунок 1. Компоненты приложений.

Архитектура проектируемого приложения подразумевает объединение прикладных требований, с точки зрения решаемой задачи, и технических требований через поиск различных вариантов применения с последующей их реализацией в виде конечного программного обеспечения. Задача на этапе проектирования — выявление требований, влияющих на архитектуру приложения. Правильно построенная архитектура в значительной степени снижает бизнес-риски, связанные с разработкой и внедрением технического решения. Под хорошей архитектурой подразумевается наличие такого свойства, как гибкость, позволяющая адаптироваться к развитию технологий как в области аппаратного и программного обеспечения, так и пользовательских сценариев и требований [6].

На этапе проектирования следует учитывать эффект от принимаемых решений, неизбежно возникающие компромиссы между критериями качества (например между производительностью и безопасностью) и компромиссы, необходимые для решения системных, пользовательских и бизнес-задач. Необходимо принимать во внимание, что архитектура приложения должна [9]:

- раскрывать структуру системы, но скрывать детали реализации.
- Предоставлять возможность реализации всех вариантов использования и сценариев.
- Максимально возможно отвечать требованием всех заинтересованных сторон.
- Соответствовать требованиям как по качеству, так и по функциональности.

ГЛАВА.2 ДВУХУРОВНЕВЫЙ И МНОГОУРОВНЕВЫЙ ВАРИАНТЫ АРХИТЕКТУРЫ «КЛИЕНТ — СЕРВЕР»

2.1.Двухуровневый вариант архитектуры «клиент-сервер».

При анализе любой сети (включая одноранговые), построенной с применением современных сетевых технологий можно обнаружить компоненты клиент-серверного взаимодействия, как правило относящегося к двухзвенной архитектуре. Данное название (two-tier, 2-tier) подобная архитектура получила из-за особенностей реализации, при которой три базовых компонента распределяются между двумя узлами (клиентом и сервером).

Подобная реализация применяется в клиент-серверных системах, предусматривающих предоставление сервером ответа на клиентские запросы напрямую и в полном объеме, с задействованием исключительно собственных ресурсов сервера. Соответственно при этом не выполняются сторонние сетевые приложения и не происходит обращения к сторонним ресурсам для реализации некой части запроса [5].

В зависимости от расположения компонентов системы на стороне сервера или клиента выделяют несколько основных моделей их взаимодействия (в рамках двухзвенной организации архитектуры):

- сервер терминалов — распределенное представление данных
- файл сервер — доступ к удаленной базе данных и файловым ресурсам

- сервер БД — удаленное представление данных
- сервер приложений — удаленное приложение

Двухзвенная организация клиент-серверного приложения считается самой простой. При подобной реализации модели приложение состоит из сервера (иди группы идентичных серверов) и группы клиентов.

Существует два варианта реализации двухзвенного приложения: модель «толстого» клиента и модель тонкого клиента.

Модель «тонкого» клиента подразумевает работу компонента приложения и компонента управления на стороне сервера (Рис.2). На клиентское аппаратное обеспечение возлагается функция работы компонента представления.

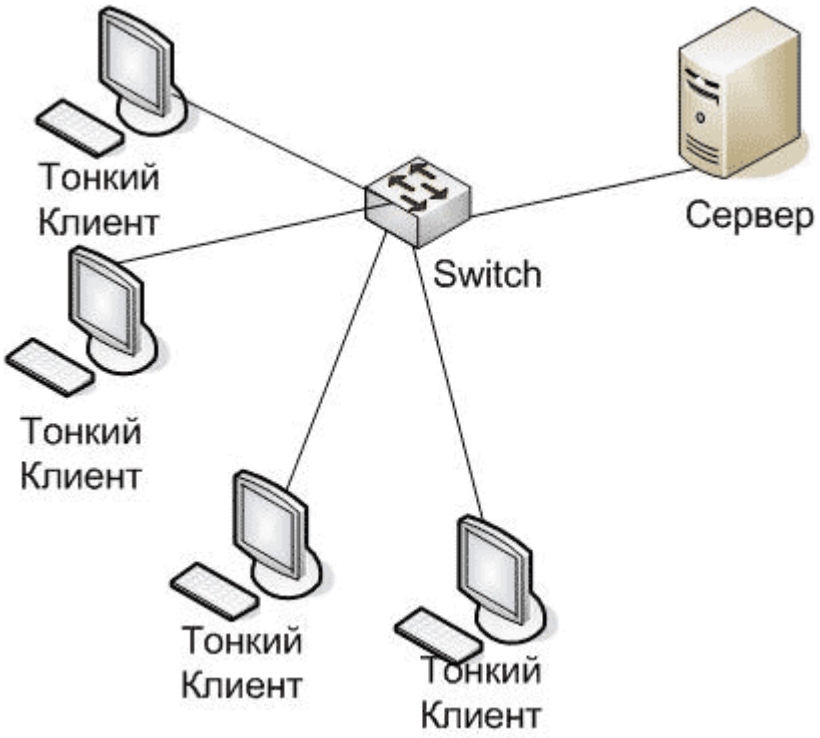


Рисунок 2. «Тонкий» клиент.

Реализация модели толстого клиента подразумевает выполнение компонента представления и прикладного компонента на стороне клиента. На сервер возлагается управление данными. [4]

В первоначальных реализациях двухзвенной архитектуры основная часть прикладных задач решалась на стороне клиента, в то время как на сервер возлагалась задача обработки SQL-запросов (модель «толстый клиент — тонкий сервер»). Впоследствии была создана архитектура, при которой на сервере

располагаются хранимые процедуры — откомпилированные программы, реализующие внутреннюю логику работы. Благодаря этом возникла возможность все большую часть задач выполнять на стороне сервера.

«Тонкий» клиент — это самый простой и очевидный способ реализации архитектуры «клиент-сервер», логически происходящий от централизованных систем обработки информации. При этом происходит лишь перенос компонента представления на клиентский компьютер, при этом на само программное приложение возлагаются функции сервера (т. е. оно реализует управленческий и прикладной компоненты). [8]

Тонкий клиент — это самый легкий способ перевода существующих централизованных систем на архитектуру «клиент — сервер». Достаточно перенести пользовательский интерфейс на компьютер пользователя, а само программное приложение будет осуществлять функции сервера (выполнять все процессы приложения и управлять данными).

Как и любая реализация, модель «тонкого клиента» обладает своими недостатками, наиболее ярким из которых является высокая нагрузка на сервер и сеть, вследствие того, что именно на серверной стороне производятся все вычисления, что требует интенсивного обмена данными между клиентским компьютером и сервером. [5]

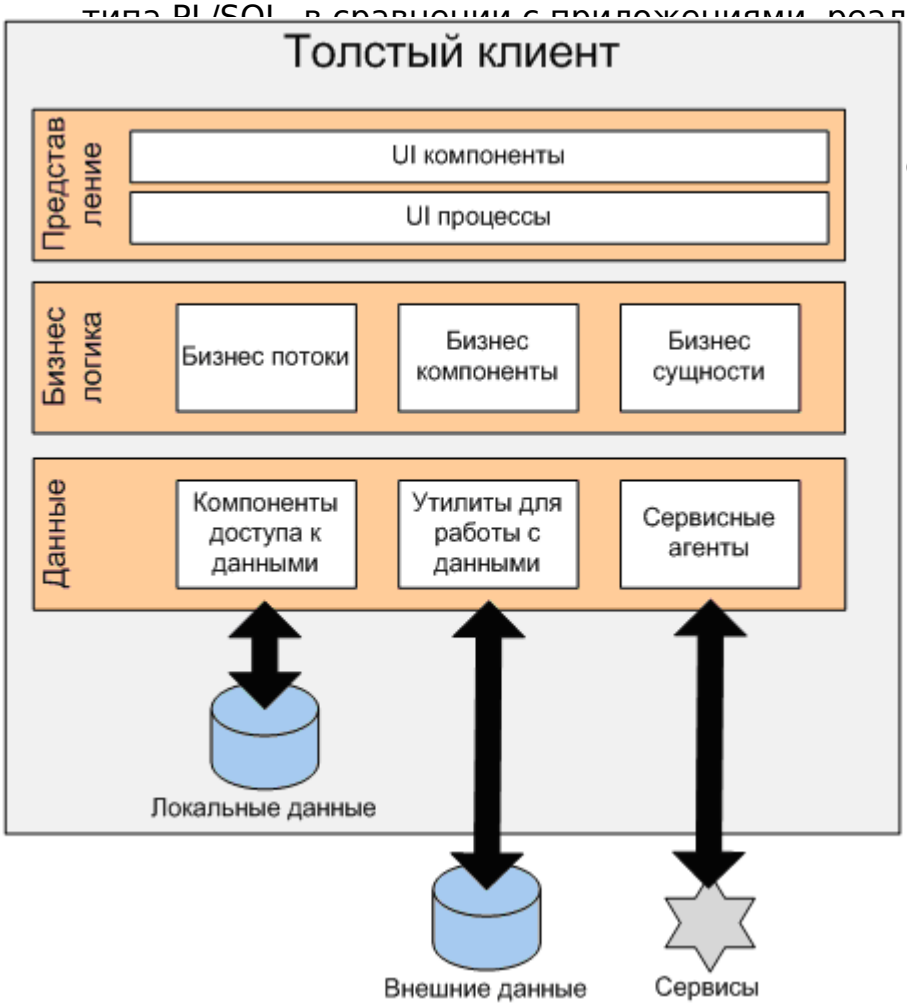
Наиболее рациональным представляется применение подобной модели организации в системах, наследуемых от центральной модели обработки информации, в которых разделение прикладного компонента и управления данными представляется нецелесообразным.

Кроме того, имеет смысл применять архитектуру «тонкого клиента» в приложениях, подразумевающих интенсивные вычисления и ограниченный объем управления данными, либо наоборот — приложения, подразумевающие обработку больших объемов данных, но не требующие интенсивных вычислений. При этом к аппаратному обеспечению будут предоставляться соответствующие специфические требования, подразумевающие либо высокую вычислительную мощность, либо наличие производительных систем хранения и передачи данных.

К недостаткам подобной реализации архитектуры также относят:

- Трудности реализации, обусловленные низким уровнем гибкости языков типа PL/SQL и отсутствием удобных средств отладки.

- Невысокий уровень производительности программ, написанных на языках типа PL/SQL, в сравнении с приложениями, реализованных на других языках.



ных с применением языков

угие платформы

Модель «толстого клиента»

позволяет использовать вычислительные мощности клиентских машин, что является актуальным в эпоху распространения персональных компьютеров (Рис.3).

Рисунок 3. «Толстый» клиент.

При этом на стороне клиента происходит выполнения и компонента представления, и компонента прикладного назначения. На стороне сервера реализуется работа компонента управления транзакциями баз данных.

Реализация приложения в форме модели «толстого» клиента позволяет более рационально использовать имеющиеся вычислительные ресурсы. При этом функции приложения распределены между разными компьютерами, что повышает сложность администрирования подобной системы. Рост количества компьютеров в

системе вызывает соответствующее повышение уровня сложности. В целом это повышает затраты на обслуживание системы, как временные, так и финансовые [7].

К другим недостаткам данной модели относят:

- Трудности в разграничении прав доступа, в связи с тем, что оно проводится по таблицам, а не действиям.
- Сложности обеспечения защиты данных, обусловленные сложностью формирования правильного распределения полномочий.
- Высокая нагрузка на коммуникационные каналы, обусловленная передачей необработанных данных.

Распространение языка программирования Java появление загружаемых «апплетов» предоставили возможность создавать клиент-серверные модели, представляющие собой нечто среднее между указанными моделями толстого и тонкого клиентов. Часть функций компонента прикладного назначения стало возможным загружать на клиентский компьютер в форме апплетов Java, что позволяет снижать нагрузку на сервер. При этом пользовательское окружение строилось на основе web-браузера, имеющего возможность запускать апплеты. Но такой подход имеет свои сложности, обусловленные, во многом, дополнительными трудностями администрирования и разработки приложений. Это связано с недостаточной стандартизированностью технологий, применяемых в браузерах [7]. Например, устаревшие версии браузеров, установленные на старых клиентских компьютерах, порой не имеют возможность исполнения апплетов Java.

Таким образом можно сказать, что двухуровневую модель организации клиент-серверного приложения целесообразно применять при обеспечении доступа к серверу приложений, а также при создании приложения по взаимодействию клиента с сервером баз данных [6]. Подобная архитектура может оказаться оптимальной в случае функциональной неизменности клиентской части приложения, при наличии эффективного системного управления.

Один из основных вопросов, встающих перед разработчиком двухуровневого приложения является расположение трех основных программных компонентов (представления, прикладного и управления) на двух аппаратных уровнях. Неизбежные компромиссы могут вызывать сложности с уровнем производительности и масштабируемости (при использовании архитектуры «тонкого клиента»), либо с администрированием системы (при использовании

«толстого клиента»). [7]

2.2.Трехуровневый вариант архитектуры «клиент — сервер».

В связи с наличием ряда серьезных проблем при реализации двухуровневой модели «клиент-сервер» было продолжено развитие данной архитектуры, благодаря чему была сформирована трехуровневый вариант.

Его активное развитие началось с середины 90-х годов. При этом информационная система по-прежнему представлена тремя компонентами (представления, прикладной и доступа к данным), но реализация данной модели представлена клиентским приложением («тонкий клиент»), взаимодействующим с сервером приложений, подключенным к серверу базы данных [10].

Первое звено данной модели представлено тонким клиентом, с работающим на нем компонентом представления. У него отсутствует прямая связь с базой данных, что повышает уровень безопасности. На данном уровне выполняются операции авторизации, валидации, шифрования, а также элементарные операции с данными. Вынесение прикладного компонента за пределы клиентского компьютера позволяет повысить масштабируемость информационной системы.

Основная часть бизнес-логики приложения располагается на уровне сервера приложений, представляющего второй уровень модели. Программное обеспечение данного звена выполняет задачи, требующие высокой вычислительной мощности, что позволяет снизить нагрузку на клиентские компьютеры, которые отправляют запросы не напрямую в базу данных, а к промежуточному слою. [9]

Сервер базы данных, представленный, как правило стандартной реляционной или объектно-ориентированной СУБД, является третьим уровнем модели.

Данный уровень также содержит хранимые процедуры, триггеры и схемы, представляющие приложение в терминах реляционной модели.

Рисунок 4. Трехуровневая модель архитектуры «клиент-сервер»

Основным отличием трехзвенной модели клиент-сервер является наличие промежуточного программного обеспечения (middleware), детальный разбор которого будет проведен в отдельной главе (Рис.4).

При использовании трехуровневой архитектуры снижается нагрузка на клиентское приложение, связанная с обработкой данных, его основной задачей становится исполнение компонента представления для информации, поступающей в обработанном виде с сервера приложения. Это снижает интенсивность обмена данными между клиентом и серверной частью, что позволяет разгружать каналы связи. При этом его реализация может быть выполнена на основе универсального браузера, с использованием унифицированных коммуникационных протоколов и общедоступных библиотек [8].

Интенсивный обмен данными, при использовании трехзвенной архитектуры, предполагается между сервером приложений и СУБД. Поскольку оба они могут располагаться в одном помещении, или быть развернутыми в виде логических модулей на одном физическом сервере это не вызывает перегрузки сети передачи информации.

При этом следует учитывать, что приложения, предъявляющие высокие требования с точки зрения безопасности, предусматривают выделение отдельного компьютера для выполнения функций сервера баз данных, который будет соединен с одним или несколькими серверами приложений, к которым подключены клиентские компьютеры.

Трехуровневую модель можно развернуть как в пределах корпоративной интранет-сети, так и распределенного интернет-приложения, в котором функции сервера приложений выполняет удаленный web-сервер.

2.3.Многоуровневый вариант архитектуры «клиент-сервер».

Дальнейшим развитием клиент-серверной архитектуры является многоуровневая модель, которая предусматривает использование нескольких серверов обработки данных.

Данная организация нацелена на дальнейшее разделение систем исполнения компонентов с целью более эффективного использования аппаратных ресурсов, снижения нагрузки на сеть передачи данных и повышения возможностей масштабирования. При этом следует учитывать, что трехуровневая модель является частным случаем многоуровневой, которую выделяют отдельно в связи с ее распространенностью и простотой организации.

Применение многоуровневой архитектуры целесообразно при использовании информации, хранящейся в нескольких источниках данных. При этом сервер, находящийся между серверами баз данных и сервером с реализованной бизнес-логикой выполняет задачу сбора разрозненных данных и их предоставления в целостном виде серверу приложений (Рис. 5) [9].

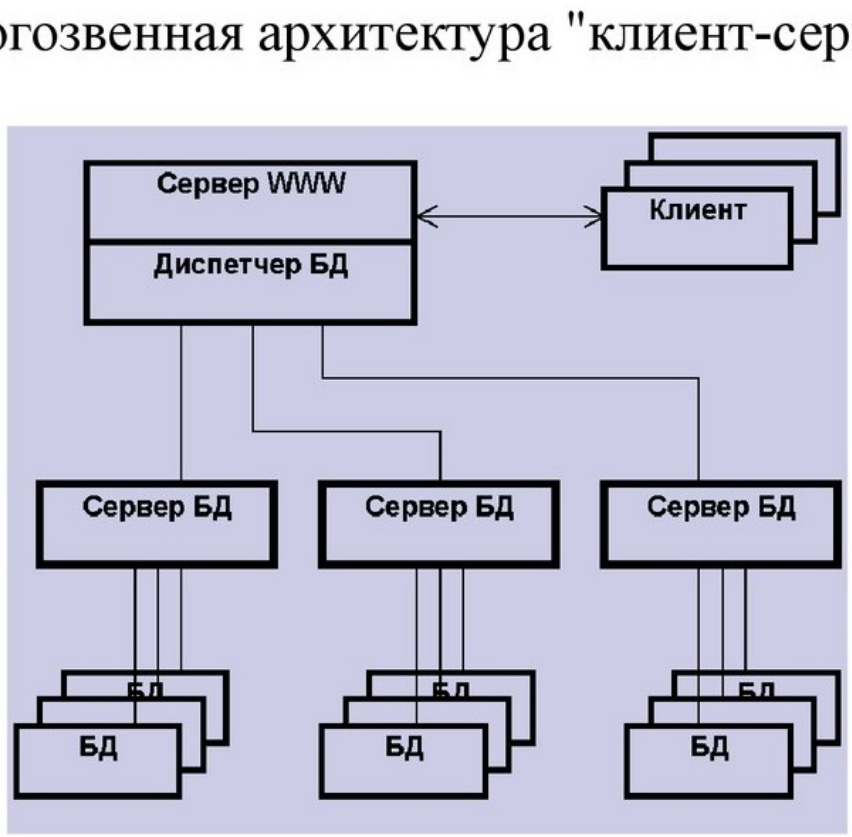


Рисунок 5. Многоуровневая архитектура «клиент-сервер».

При этом многоуровневое клиент-серверное приложение достаточно просто разворачивается на базе web-технологий. При этом в качестве клиентской части приложения используется универсальный браузер, а сервер приложений дополняется web-сервером. Вызов процедур сервера приложений создают с помощью технологий Java или Common Gateway Interface (CGI).

CGI – стандарт интерфейса, применяемый для обеспечения связи внешней программы с web-сервером. Программу, работающую по такому интерфейсу совместно с web-сервером, принято называть шлюзом.

Существует классификация, согласно которой в многоуровневой архитектуре выделяют пять уровней [9]:

- Представление
- Уровень представления
- Уровень логики
- Уровень данных
- Данные

Представлению соответствует та информация, которую непосредственно видит пользователь. К ней относятся изображения, сгенерированные html-страницы, таблицы стилей.

К уровню представления относятся элементы, реализующие общение с системой. Задачи приложения на этом уровне - отображение информации и интерпретация вводимых пользователем команд с их модификацией в соответствующие операции, представленные на уровне логики и данных.

Уровень логики реализует прикладные функции системы, необходимые для выполнения поставленной пользователем цели. К ним относятся вычисления на основании вводимых и хранимых данных, обеспечение проверки элементов данных, обработка команд, приходящих от уровня представления, и передача информации на уровень данных.

Уровень доступа к данным представлен компонентами, обеспечивающими взаимодействие со сторонними системами, выполняющими задачи, необходимые приложению [5].

Данные системы — данные которыми оперирует система, и которые, как правило, хранятся в базе данных.

Как и любая архитектура приложения, многоуровневая имеет смови преимущества и недостатки.

К достоинствами многоуровневой архитектуры относятся: [8]

- масштабируемость
- облегченная конфигурируемость вследствие изолированности уровней
- возможность быстро переконфигурировать систему при возникновении сбоев, а также при плановом обслуживании на одном из уровней
- повышенный уровень безопасности
- повышенный уровень надежности
- упрщение администрирования клиентского ПО
- сниженные требования к скорости передачи данных между сервером приложений и терминалами
- невысокие требования к вычислительным возможностям терминалов, что позволяет снизить их стоимость.

К недостаткам относится [7]:

- увеличение сложности серверной части приложения и, как следствие, рост расходов на администрирование и обслуживание
- увеличение сложности создания приложений
- высокие требования к производительности серверов приложений и сервера базы данных, соответственно, их высокая стоимость
- высокие требования к скорости канала между серверами приложений и сервером базы данных.

Несмотря на ряд недостатков, распределенные приложения реализованные в виде многоуровневой архитектуры завоевывают все большую популярность на рынке программного обеспечения в самых различных областях: от банковских систем до приложений в игровой сфере.

ГЛАВА.3 ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПРОМЕЖУТОЧНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1.Общая характеристика промежуточного программного обеспечения.

Компания International System Group дает следующее определение этому звену клиент-серверного приложения: это специальный уровень прикладной системы, который расположен между бизнес-приложением и коммуникационным уровнем и изолирует приложение от сетевых протоколов и деталей операционных систем [12].

Вычислительная среда клиент-серверного приложения может быть основана на различных операционных системах, аппаратных платформах, протоколах передачи данных, систем управления баз данных и средствах разработки.

Наличие общих прикладных интерфейсов промежуточного ПО обеспечивает взаимодействие между отдельными частями системы, не раскрывая подробности

реализации архитектуры. При этом изменения в системе, связанные, к примеру, с масштабируемостью, не требуют внесения изменений в приложение, при условии что API промежуточного ПО остается неизменным.

Данный уровень предоставляет возможность передачи и анализа разнородной информации. К примеру, формат представления данных, применяемый в мэйнфремах, отличается от того, что используется в Unix- или Windows- системах, но эта проблема нивелируется путем применения промежуточного программного обеспечения. Можно сказать, что данное звено играет роль «информационной шины», построенное над сетевым уровнем и предоставляющей доступ приложениям к разнородным ресурсам, а также обеспечивающей платформонезависимую связь отдельных прикладных компонентов [14].

Существующее промежуточное программное обеспечение можно классифицировать на две группы: реализующее доступ к базам данных и реализующее межпрограммное взаимодействие.

Вторая группа включает в себя:

- ПО, обеспечивающее вызов удаленных процедур (RPC)
- мониторы обработки транзакций
- средства интеграции распределенных объектов
- ПО, реализующее обработку сообщений

Поскольку множество прикладных задач имеет большое разнообразие, то создание универсального промежуточного программного обеспечения невозможно. Тем не менее на рынке наблюдается тенденция интеграции различных видов ПО внутри одного пакета, например брокеров объектных запросов и мониторов обработки транзакций.

Рассмотрим отдельные типы промежуточного ПО.

3.2.Системы доступа к БД.

Системы доступа к БД — один из наиболее распространенных типов ПО. Необходимость в подобных решениях возникает в разнородных системах, обеспечивающих одновременный доступ к различным источникам данных, включая

СУБД и хранилища данных от разных поставщиков [12]. При этом на сервере приложений развертывается SQL-шлюз, представляющий собой комплекс интерфейсов приложений, предоставляющих возможность построения унифицированных запросов к разнородным данным. Промежуточное ПО данного типа проводит анализ запроса с последующим его преобразованием в SQL-диалект требуемой СУБД. При этом используется синхронный механизм коммуникации, при котором исполнение приложения, осуществившего запрос, приостанавливается до получения данных. Подобный механизм коммуникации может создавать сложности с масштабированием приложения [14]. Применение подобного ПО востребовано в системах поддержки принятия решений, которые агрегируют данные из большого количества разнородных источников, но при этом не требуют управления оперативными транзакциями, например система, выстраивающая прогноз уровня продаж.

3.3.Вызов удаленных процедур.

Вызов удаленных процедур (remote procedure call) — программно обеспечение, реализующее организацию взаимодействия удаленных прикладных компонентов. Данный тип ПО организует синхронный тип коммуникации между прикладными модулями на клиентской и серверной сторонах. Чтобы обеспечить связь, вызов функции и передачу результата создаются специальные модули, называемые суррогатами (клиентский и серверный), к которым и происходит обращение. Суррогаты не содержат компонентов, реализующих бизнес-логику, и служат для обеспечения взаимодействия прикладных компонентов [12]. Любая функция, к которой может произойти обращение из удаленного клиента, обязана иметь реализацию подобного суррогатного процесса. При выполнении вызова клиентской программой удаленной процедуры он, включая параметры, поступает к клиентскокму суррогату. Тот выполняет преобразование вызова в форму сетевого сообщения и передает на сторону серверного суррогата. Тот производит распаковку полученных данных и их передачу прикладному серверу, выполняя затем обратную операцию с полученными результатами. Таким образом происходит изоляция прикладных модулей как на стороне сервера, так и клиента от уровня сетевых коммуникаций.

Таким образом происходит реализация принципов структурного программирования в распределенной среде. Клиентская часть приложения обращается к процессу-

суррогату как к действительному серверному приложению, причем этот вызов абсолютно идентичен вызову локальной функции. При этом за вызовом удаленной процедуры следует передача управления той же процедуре, и, как следствие, приостановка выполнения клиентского приложения на период реализации вызова.

Данный механизм имеет свои недостатки, в частности клиентская часть привязана к конкретным серверным суррогатам с этапа компиляции приложения, и не имеет возможности изменения в период выполнения [14].

Подобных недостатков лишены более новые решения, вроде ПО, ориентированного на обработку сообщений предоставляющих возможность динамического выбора сервера, или мониторов обработки транзакций, имеющих поддержку оптимального распределения нагрузки между серверами и средства восстановления.

В основе ПО вызова удаленных процедур лежит язык описания интерфейсов (interface definition language), посредством которого создаются интерфейсы, определяющие контрактные отношения между клиентской и серверной сторонами. Интерфейс включает определение имени функции, а также описание параметров вызова и результатов его обработки. Подобный инструмент обеспечивает независимость механизма вызова удаленных процедур от конкретных языков программирования: при вызове удаленной процедуры на стороне клиента могут применять свои языковые конструкции, которые модифицируются IDL-компилятором в собственный формат. Соответственно, на стороне сервера происходят преобразования в форму языка программирования, с помощью которого реализована серверная часть приложения [12].

В основе некоторой части программного обеспечения этого типа лежит стандарт Open Group DCT RPC.

DCE (Distributed Computer Environment) относится к категории свободно распространяемого программного обеспечения и представлено в виде нескольких реализаций, предназначенных под конкретные операционные системы. Кроме основного механизма обеспечения взаимодействия компонентов распределенного приложения, DCE включает в себя реализацию некоторых востребованных в подобной среде служб, таких как распределенная файловая система, служба каталогов, средства обеспечения безопасности (Рис.6).

DCE имеет свои ограничения, в частности синхронные механизм коммуникации обуславливает сложности масштабируемости системы и снижают производительность, потому на сегодняшний день востребованы конкурирующие

технологии промежуточного ПО, такие как архитектура распределенных объектов CORBA, или ПО, ориентированное на передачу сообщений. При этом DCE может служить дополнительным компонентом новейшего ПО этой сферы [14].

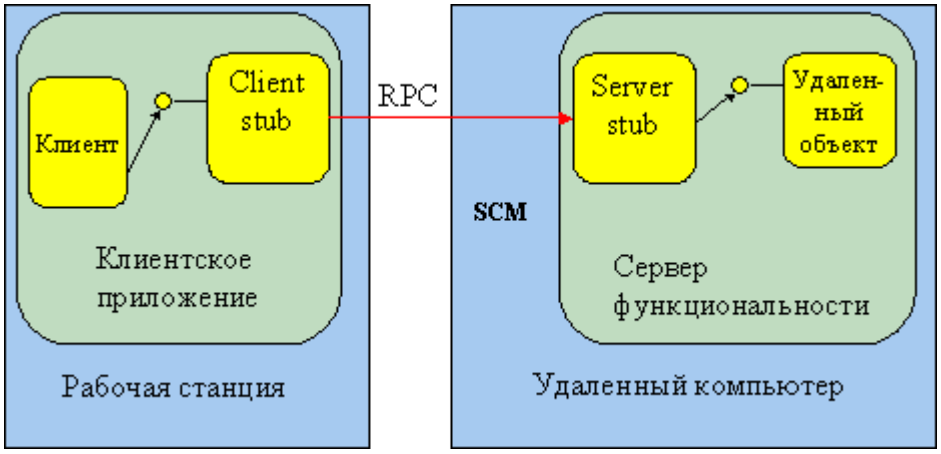


Рисунок 6. Служба вызова удаленных процедур

Кроме того на рынке существуют реализации вызова удаленных процедур на основе асинхронного механизма коммуникации. Подобная реализация не останавливает исполнение клиентского процесса в период выполнения запроса, что позволяет сократить объем поддерживаемой информации о коммуникации между клиентом и сервером, как следствие это обеспечивает лучшую масштабируемость подобных систем.

3.4.Мониторы обработки транзакций.

Мониторы обработки транзакций — тип промежуточного программного обеспечения, который широко применялся на мэйнфреймах для развертывания банковских и страховых систем, и использовал специфическое окружение, предназначенное для этих машин [13]. В 90-х годах прошлого столетия появились реализации данного ПО в среде Unix и Windows (Рис.7).

Основной задачей этого ПО в период появления было снижения числа соединений терминалов с базами данных. При этом, обращение клиентского приложения к серверу базы данных вызывает запуск отдельного процесса. Мониторы обработки транзакций брали на себя посреднический функции между терминалом и базой данных, становясь таким образом концентратором соединений. Со временем мониторы обработки транзакций расширили свою функциональность, став одним

из наиболее сложноорганизованных приложений промежуточного программного обеспечения.

Как следует из названия, их основное предназначение — автоматизированная поддержка приложений, созданных в форме последовательности транзакций.

Каждая транзакция представляет собой некий комплекс обращений к базе данных (или иному ресурсу) и некоторых действий над ней, который характеризуется наличием четырех свойств: атомарность, согласованность, изолированность и долговременность (ACID: Atomicity, Consistency, Isolation, Durability) [12].

- Атомарность — операции транзакции представлены минимальным неделимым блоком, у которого определены начало и конец. Данный блок либо выполняется в полном объеме, либо не исполняется вовсе. В случае возникновения сбоя в процессе исполнения транзакции, выполняется откат к исходному состоянию.
- Согласованность — после исполнения блока транзакции все используемые ресурсы имеют согласованное состояние.
- Изолированность — механизм монитора обработки транзакций реализован таким образом, что при одновременном доступе транзакций, поступающих от различных приложений к разделяемым ресурсам не происходит их взаимного влияния.
- Долговременность — изменения ресурсов, произошедшие в ходе исполнения транзакции являются долговременными.

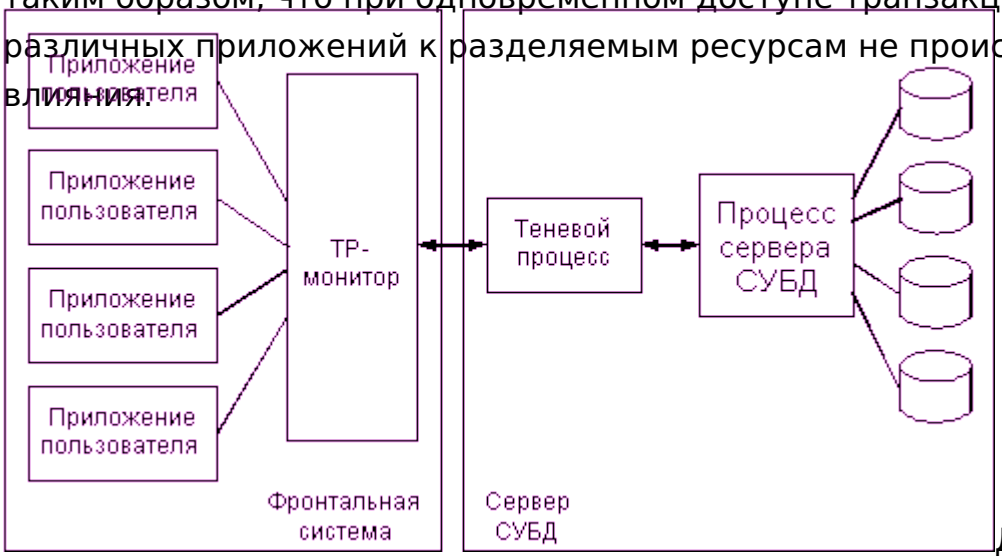


Рисунок 7. Монитор обработки транзакций.

Те системы, в которых не реализован механизм монитора обработки транзакций, выполнение принципа ACID возлагается на серверы распределенной базы данных, использующей протокол 2PC.

Данный протокол определяет двухфазный процесс, при котором в первой фазе все задействованные ресурсы опрашиваются о готовности к выполнению запрошенных действий. В случае. Если от каждого из серверов получен положительный ответ, происходит исполнение транзакции. В случае возникновения сбоя на любом из уровней, выполняется откат всей транзакции с возвращением системы в исходное состояние.

Но исполнение протокола P2C может быть гарантированно исполнено в системе с распределенными базами данных лишь при условии, что все источники данных относятся к одному поставщику. Поэтому в масштабных распределенных средах, включающих тысячи терминалов и работающих с множеством разнородных источников данных возникает необходимость в использовании мониторов обработки транзакций [14]. Они обеспечивают эффективную координацию при работе с разнородными данными от различных поставщиков за счет применения транзакционной архитектуры XA, являющейся стандартом для этого типа промежуточного программного обеспечения. Данная архитектура предоставляет интерфейс для обеспечения взаимодействия монитора транзакций с менеджеров ресурсов, таким как СУБД.

Спецификация XA входит в общий стандарт распределенной обработки транзакций (DTP - distributed transaction processing), разработанного группой X/Open [13].

Данный стандарт обеспечивает совместную работу с ресурсами множества клиентских программ, сохраняя согласованность системы.

На уровне прикладных программ взаимодействие с монитором транзакций выполняется путем применения интерфейса TX, обеспечивающего реализацию семантики транзакций, разделяя запросы на логические составляющие.

Кроме указанного, современные мониторы обработки транзакций берут на себя функции планирования, распределения ресурсов и определения приоритетов нескольких приложений одновременно, за счет чего достигается снижение вычислительной нагрузки и сокращается время отклика системы. При этом обеспечивается более эффективное использование многопоточности.

Мониторы транзакций предоставляют мультиплексирование запросов от нескольких терминалов к ресурсам. Для большинства приложений непосредственный доступ клиентского приложения к базе данных составляет лишь некоторую часть от общего времени соединения. С применением мониторов транзакций становится возможным обслуживание 100 клиентских терминалов за счет 10 активных соединений с сервером, на котором развернуты требуемые ресурсы. Это позволяет преодолевать одно из наиболее «узких мест» в обеспечении производительности и масштабируемости распредленных приложений — необходимости поддержки индивидуального соединения с ресурсом для каждого из клиентских приложений. Часть современных мониторов транзакций содержат функционал, позволяющий оптимизировать нагрузку на серверы и обеспечивающий восстановление после сбоя.

Резюмируя, можно сказать, что мониторы обработки транзакций являются эффективным механизмом, позволяющим реализовать распределенное приложение со сложной бизнес-логикой.

3.5.Интеграция распределенных объектов.

Интеграция распределенных объектов — механизм, реализуемый промежуточным программным обеспечением, позволяющий интегрировать объекты, расположенные на удаленных платформах [12].

В результате развития объектно ориентированной-парадигмы возникло два стандарта, имеющих отношение к распределенным приложениям — общая архитектура брокеров объектных запросов CORBA (Common Object Request Broker Architecture), разработанная группой OMG и COM/DCOM (Common Object Model/Distributed COM), созданный компанией Microsoft.

Данные стандарты определяют принципы вызова удаленных процедур на объектные распределенные приложения и обеспечивают прозрачность реализации и физического размещения серверного объекта для клиентской части приложения. Кроме того они предоставляют возможность взаимодействия объектов, созданных на различных объектно-ориентированных языках, не раскрывая при этом детали сетевого взаимодействия.

В DCOM взаимодействие удаленных объектов основано на спецификации DCE RPC, в то время как CORBA содержит представление брокера объектных запросов (ORB), в котором реализован синхронный механизм, имеющий сходство с механизмом вызова удаленных процедур (RPC).

ORB реализует отправку объектных запросов, обеспечивает поиск требуемых сервисов и возврат результата. Одним из базовых компонентов стандарта CORBA является язык описания интерфейсов IDL, который обеспечивает «контрактные» отношения между терминалом и сервером и реализуется независимость от конкретного объектно-ориентированного языка [14].

В CORBA IDL реализованы основные принципы объектно-ориентированной парадигмы: инкапсуляция, полиморфизм, наследование. DCOM также предполагает использование собственной версии IDL разработанной Microsoft, но в рамках этой модели он выполняет вспомогательную функцию и применяется для упрощения описания объектов. Реальная интеграция, при применении данной технологии, происходит на уровне бинарных кодов, а не абстрактных интерфейсов, что составляет одно из основных отличий между двумя стандартами.

В соответствие с объектно-ориентированной парадигмой обе модели предоставляют возможность динамического связывания удаленных объектов, при этом клиентское приложение может выполнить обращение к серверному объекту на этапе исполнения, не обладая информацией о данном объекте на этапе компиляции). С этой целью в CORBA реализован интерфейс динамического вызова DII, в модели COM существует механизм OLE Automation [13]. При необходимости предоставляется информация о доступных объектах сервера из хранилища метаданных об объектах: Type Library при использовании стандарта COM и Interface Repositary в случае применения CORBA. Данный механизм обладает особой ценностью при реализации больших распределенных приложений, поскольку предоставляет возможность изменять функциональность серверов без существенной модификации клиентской части приложения.

Одно из основных преимуществ CORBA заключается в кроссплатформенности данного решения и поддержке гетерогенных окружений, в то время как DCOM является разработкой одной компании, и оптимизировано под использование в системах построенных на ОС Windows. Так же следует отметить более широкую поддержку CORBA в средствах разработки от различных поставщиков.

3.6.Промежуточное ПО, ориентированное на обработку сообщений.

Промежуточное ПО, ориентированное на обработку сообщений — сравнительно новая технология, при использовании которой между приложениями происходит обмен байтовыми строками, называемыми сообщениями, путем обращения к API-интерфейсу Message Oriented Middleware (MOM) [12]. Это позволяет изолировать приложения от операционной системы и сетевых протоколов. Данный тип промежуточного программного обеспечения поддерживает не только клиент-серверную модель, но и приносит элементы равноправного взаимодействия (модель peer-to-peer) между отдельными компонентами архитектуры.

Данная модель реализует как синхронную, так и асинхронную связь на основе сетевых протоколов как с установлением соединения, так и без него, что обеспечивает высокий уровень гибкости и адаптируемости системы под меняющиеся условия.

Все примеры данной архитектуры принято классифицировать на три типа, в зависимости от выбранного типа обмена сообщениями:

- очереди сообщений,
- подписка/публикация,
- передача сообщений

Для реализации взаимодействия приложений системы с передачей сообщений устанавливается логическое соединение между отдельными программными компонентами. Подобное решение плохо подходит для слабо связанных программ.

Модели иного типа представляет асинхронный механизм очереди сообщений (Рис.8). Подобная реализация не требует поддержки непосредственного соединения одного компонента с другим, при этом осуществляет гарантированную доставку сообщения, даже при условии, что конечное приложение на данный момент недоступно. Приложение-отправитель передает сообщение механизму очереди и продолжает свое выполнение. Сообщение помещается при этом в промежуточное хранилище, расположенное на диске или в оперативной памяти, откуда передается приложению получателю немедленно, или спустя какое-то время, когда получатель будет доступен.

Рисунок 8. Механизм очереди сообщений.

Соответственно, модули, применяющие подобную организацию связи, могут работать независимо друг от друга, не нуждаясь во временной синхронизации, что является подходящим решением для приложений, ориентированных на мобильных пользователей, а также для поддержки приложений, развернутых в среде с медленными либо ненадежными соединениями [12].

Значительная часть подобных систем включают в себя менеджер очереди, который предназначен для управления локальными очередями, обеспечивает гарантированную передачу сообщения приложению-адресату, и путем коммуникации с менеджерами других узлов контролирует маршрут передачи сообщения по сети, выбирая иной путь при необходимости.

Очереди сообщений могут быть долговременного типа. В этом случае, при возникновении сбоя в работе менеджера сообщения восстанавливаются после его перезапуска. Подобный тип менеджера предпочтителен для систем с критичными данными, например банковских распределенных приложений [14].

Кроме того системы на основе очередей сообщений можно разделить на следующие три типа:

- обеспечивающие надежную доставку сообщений — подобная система гарантирует, ни одно сообщение не будет потеряно в процессе функционирования
- обеспечивающие гарантированную доставку сообщений: сообщение доставляется приложению-адресату немедленно, или через промежуток времени, не превышающий заданное значение в случае недоступности каналов передачи сообщений
- обеспечивающие застрахованную доставку сообщений — при этом каждое сообщение доставляется только один раз

В основе систем на базе очереди сообщений лежит новая парадигма создания приложений — программы, управляемые событиями. Возникновения события в одном приложении в форме сообщения вызывает определенное действие в другом. Такая модель взаимодействия наиболее приближена к реальной бизнес-логике. По этой причине именно системы на базе очередей сообщений представляют собой большинство среди всех систем, ориентированных на обработку сообщений.

Последняя модель, представленная среди систем данного типа определяется как модель подписки/публикации. При данном варианте архитектуры одно приложение публикует сообщение в сети, а другое оформляет подписку на интересующую его информацию. При подобной организации взаимодействия различные приложения абсолютно независимы и в общем случае не обладают информацией о взаимном существовании, физическом расположении и состоянии друг друга. Это позволяет обеспечить динамическую конфигурацию системы с возможностью произвольного добавления и изменения клиентов и серверов при прерывании работы системы и позволяет добиться полной изоляции прикладных компонентов от любых деталей реализации других модулей системы.

Для обеспечения стандартизации в области промежуточного программного обеспечения ориентированного на обработку сообщений с целью эффективного взаимодействия решений от разных поставщиков в 1994 году был создан консорциум Message-Oriented Middleware Association (MOMA), в кторый вошли такие компании, как IBM и Sun Microsystems. Позже она была переименована в International Middleware Association, которая занимается общим развитием систем промежуточного слоя [12].

ЗАКЛЮЧЕНИЕ

В данной работе было дано определение понятия «архитектура «клиент-сервер», рассмотрены факторы, способствующие ее возникновению, история развития, приведены ее основные характеристики и компоненты. Были изучены типы серверов и **о**сновные клиент-серверные технологии.

Это позволило рассмотреть различные виды архитектуры «клиент-сервер», сравнить их между собой, определив достоинства и недостатки. Были рассмотрены различные варианты архитектуры, в зависимости от количества уровней, а также

используемого промежуточного программного обеспечения.

Цели и задачи курсовой работы считаю выполненными.

Подводя итоги, можно сделать вывод, применение многоуровневой модели архитектуры «клиент-сервер» является наиболее рациональным при решении большинства бизнес-задач. Свойства системы, ее преимущества и ограничения во много определяются не только количеством звеньев, но и используемым типом промежуточного программного обеспечения. Его дальнейшее совершенствование во многом определит последующее развитие всей технологии и позволит более точно соответствовать заявленным требованиям.

В целом, технология «клиент-сервер» является перспективной, поскольку позволяет обеспечить совместную работу множества клиентских приложений с использованием разнородных распределенных данных, что является востребованным решением для множества современных бизнес-задач.

БИБЛИОГРАФИЯ

1. Баканов В.М. Программное обеспечение компьютерных сетей и информационных систем.— М., 2003.
2. Дуглас Камер Сети TCP/IP, том 1. Принципы, протоколы и структура. М.: Вильямс, 2003. — 851 с.
3. Когаловский М.Р. Энциклопедия технологий баз данных.—М.:Финансы и статистика, 2005. — 800 с.
4. Программное обеспечение компьютерных сетей: Учебное пособие / О. В. Исаченко. — М.: ИНФРА-М, 2012. — 117 с.
5. Руденков Н.А., Долинер Л.И. Основы сетевых технологий: Учебник для вузов. Екатеринбург: Изд-во Уральского Федерального ун-та, 2011 – 300 с.
6. Руководство Microsoft по проектированию архитектуры приложений. 2-е издание. — 2009. 529 с.
7. Соммервилл Иан. Инженерия программного обеспечения 6-е изд., пер. с англ. — М.: Вильямс, 2002. — 624 с.
8. Таненбаум Э., М. ван Стеен. Распределенные системы. Принципы и парадигмы/ — СПб.: Питер, 2003. — 877 с.
9. Фаулер, Мартин. Архитектура корпоративных программных приложений.: Пер. с англ. — М.: Издательский дом "Вильямс", 2006 — 544 с.

10. Хьюз Камерон. Параллельное и распределенное программирование на C++ Издательский дом ISBN, 2004.
11. Эбберс М. Введение в современные мейнфреймы: основы z/OS., пер. с англ. - М.: 2007
12. Дубова Наталья. Все про промежуточное ПО. // Открытые системы. СУБД. - 1999. - NN 07-08.
13. Касаткин Александр. Средства Middleware и их классификация. // PCWeek, 1999. - N19.
14. Эйнджел Джонатан. Промежуточное программное обеспечение. // Журнал сетевых решений. - 1999. - N11