РАЗРАБОТКА ПРИЛОЖЕНИЙ ТИПА КЛИЕНТ-СЕРВЕР С ИСПОЛЬЗОВАНИЕМ UML

Бурба В.К., ДонНТУ, каф. ЭВМ
Руководитель: Святный В.А., проф. каф. ЭВМ
burbaviktor@front.ru

Abstract

Burba V.K. Development of the client-server applications using UML.

Recent successes in the field of development of hardware maintenance and a communication facility have led to avalanche growth of number of the parallel, distributed and real time systems. Today the majority of commercial, industrial, military, medical and consumer products are supplied with microprocessors. Such systems meet in microwaves and videorecorders, in phones and TVs, in cars and planes, in submarines and spacecrafts, in automatic devices after sale of the aerated drinks and cash dispenses, in systems of diagnostics of patients and control systems of manufacture, in controllers of robots and control systems of lifts, in control systems of city and air transport, in e-mail and commerce, in "intellectual" transport and information highways. The list can be continued indefinitely. All this parallel systems, and many of them are besides distributed. Such changes, in turn, promoted change of the requirements shown to the software. Reaction to these changes became a wide circulation of object-oriented methods of designing. Object-oriented concepts are especially important for the analysis and designing of the software as they concern fundamental questions of an adaptability and development. The unified language of modelling UML offers the standardized notation for the description of object-oriented models.

Введение

Недавние успехи в области разработки аппаратного обеспечения и средств связи привели к лавинообразному росту числа параллельных и распределенных систем, а также систем реального времени. Значительное понижение цен на микропроцессоры и полупроводниковые микросхемы и столь же существенное увеличение производительности микропроцессоров, наблюдаемые на протяжении нескольких лет, сделали рентабельными также распределенные системы и системы реального времени на базе микрокомпьютеров. Сегодня большинство коммерческих, промышленных, военных, медицинских и потребительских продуктов снабжаются микропроцессорами и целиком либо в значительной части управляются программами. Такие системы встречаются в микроволновых печах и видеомагнитофонах, в телефонах и телевизорах, в автомобилях и самолетах, в подводных лодках и космических кораблях, в автоматах по продаже газированных напитков и банкоматах, в системах диагностики пациентов и системах управления производством, в контроллерах роботов и системах управления лифтами, в системах управления городским и воздушным транспортом, в электронной почте и коммерции, в “интеллектуальных” транспортных и информационных магистралях. Список можно продолжать до бесконечности. Все это параллельные системы, а многие из них являются к тому же распределенными. Такие изменения, в свою очередь, способствовали изменению требований, предъявляемых к программному обеспечению. Реакцией на эти изменения стало широкое распространение объектно-ориентированных методов проектирования. Объектно-ориентированные концепции особенно важны для анализа и проектирования программного обеспечения, поскольку они касаются фундаментальных вопросов адаптируемости и развития.

Унифицированный язык моделирования UML предлагает стандартизованную нотацию для описания объектно-ориентированных моделей.

Технологии клиент-серверных систем

На заре развития вычислительной техники компьютерные приложения являлись в основном пакетными заданиями. Все программы были последовательными и работали в автономном режиме. Теперь же, когда имеется огромное число интерактивных программ и отчетливо прослеживается тенденция перехода к распределенным микрокомпьютерным системам, многие приложения по природе своей параллельны. Растет количество систем, в том числе распределенных, в которых роль параллелизма очень велика. Для параллельного приложения характерно наличие многих задач, выполняемых одновременно. При этом порядок следования внешних событий часто непредсказуем, и периоды обработки таких событий могут перекрываться. В параллельном приложении, как правило, есть несколько активных объектов – каждый со своим потоком управления. В этом случае поддерживается асинхронная передача сообщений: активный объект-отправитель может послать асинхронное сообщение активному объекту-получателю, а затем продолжить выполнение вне зависимости от того, дошла ли информация до адресата. Если объект-получатель в момент прихода сообщения занят выполнением другой работы, то сообщение ставится в очередь. Концепция параллельных задач, называемых также параллельными процессами, служит основой проектирования параллельных приложений. Параллельное приложение состоит из множества задач, выполняемых одновременно. Концепции проектирования параллельных задач применимы также к распределенным приложениям. Основная сложность заключается в том, чтобы разбить приложение на параллельно выполняемые задачи и предоставить средства обмена сообщениями и синхронизации этих задач между собой. Для распределенных приложений нужно позаботиться и о других характеристиках. Распределенное приложение – это параллельное приложение, которое исполняется в распределенной среде, состоящей из нескольких географически разнесенных узлов (рис.1.)

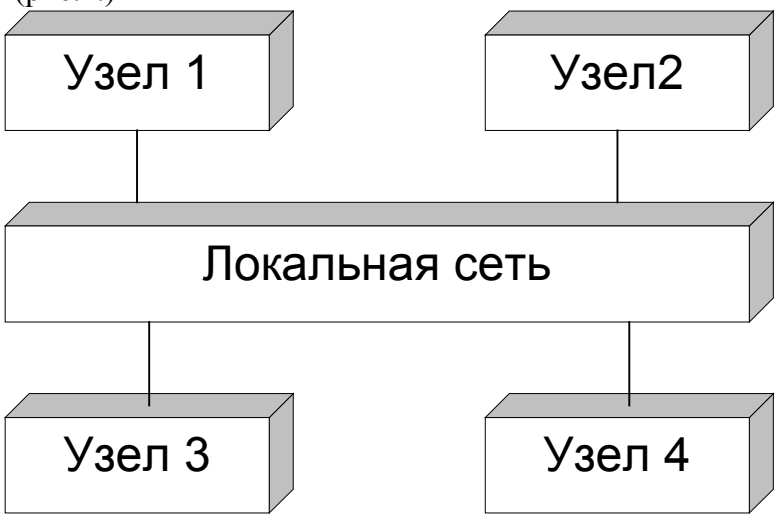

Рис. 1. Распределенная среда обработки

На рис.1. показана типичная распределенная среда, где есть несколько узлов, связанных между собой сетью. Каждый узел – это компьютер с собственной локальной памятью. Кроме того, в каждом узле имеется сетевая карта. Важной особенностью распределенной среды является то, что у узлов нет общей памяти, то есть

распределенное приложение состоит из параллельных процессов, работающих в разных узлах. В то же время, каждый процесс может иметь несколько потоков, исполняемых в том же узле. Так как разделяемой памяти нет, то процессы в разных узлах должны обмениваться информацией, посылая сообщения по сети.

Распределенная обработка имеет следующие преимущества:

- повышенная доступность. Если некоторые узлы временно недоступны, то операция выполняется в редуцированной конфигурации;
- гибкая конфигурация. Одно и то же приложение допустимо сконфигурировать различными способами, разместив его на подходящем числе узлов;
- более локализованное и управление и администрирование. Распределенную подсистему, выполняемую на своем собственном узле, можно спроектировать так, что она будет автономной, то есть практически независимой от других подсистем, работающих на других узлах;
- постепенное расширение системы. Если нагрузка сильно возрастает, систему легко расширить за счет добавления новых узлов;
- уменьшение затрат. Зачастую распределенное решение оказывается дешевле централизованного, особенно если принять во внимание стремительно уменьшающуюся стоимость и возрастающую производительность микрокомпьютеров;
- балансирование нагрузки. В некоторых приложениях общая нагрузка на систему может быть распределена между различными узлами;
- уменьшение времени отклика. Запросы пользователей локальных систем обрабатываются быстрее.

Типичным примером распределенных приложений являются приложения типа клиент-сервер.

Клиент-серверная система логически состоит из двух компонентов:

- клиент, который запрашивает некоторый сервисы;
- сервер, который предоставляет запрашиваемые сервисы.

Клиент-серверная система – это распределенное приложение, в котором клиент и сервер географически удалены друг от друга (рис.2). Сеть, соединяющая клиентов с серверами, может быть как локальной так и глобальной. Клиент посылает серверу запрос по сети, а сервер выполняет этот запрос и возвращает клиенту результаты.

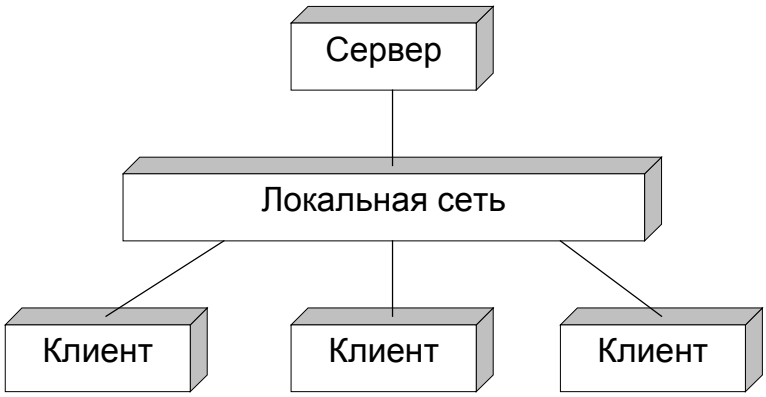

Рис.2. Базовая конфигурация системы клиент-сервер

Обычно клиенты и серверы работают на разных машинах. Они могут быть реализованы на разных платформах, под разными операционными системами и в различных сетях. Клиент – это, как правило, настольный персональный компьютер или рабочая станция. Часто он поддерживает графический интерфейс пользователя (GUI). У сервера обычно имеется большой объем памяти и дисков, мощный процессор и средства повышения надежности. Кроме управления данными, он предоставляет услуги прикладного характера. В простейшей системе клиент-сервер имеется один сервер и несколько клиентов. Типичным примером является приложение, которое обслуживает банкоматы. Банкоматы размещены на территории города и обмениваются данными с центральным банковским сервером.

В более сложной системе может присутствовать несколько серверов. Клиент может обращаться к различным серверам, а сами сервера могут обмениваться данными друг с другом (рис.3.).

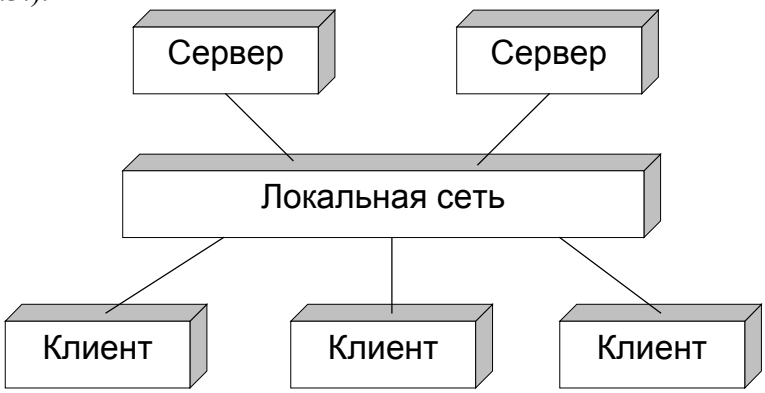

Рис.3. Конфигурация для распределенной обработки.

В качестве многосерверного приложения можно привести межбанковскую систему, где любой банкомат может обмениваться данными с любым банковским сервером, входящим в систему. В многоуровневых клиент-серверных системах сервер иногда может выступать в роли клиента другого сервера (рис.4.).

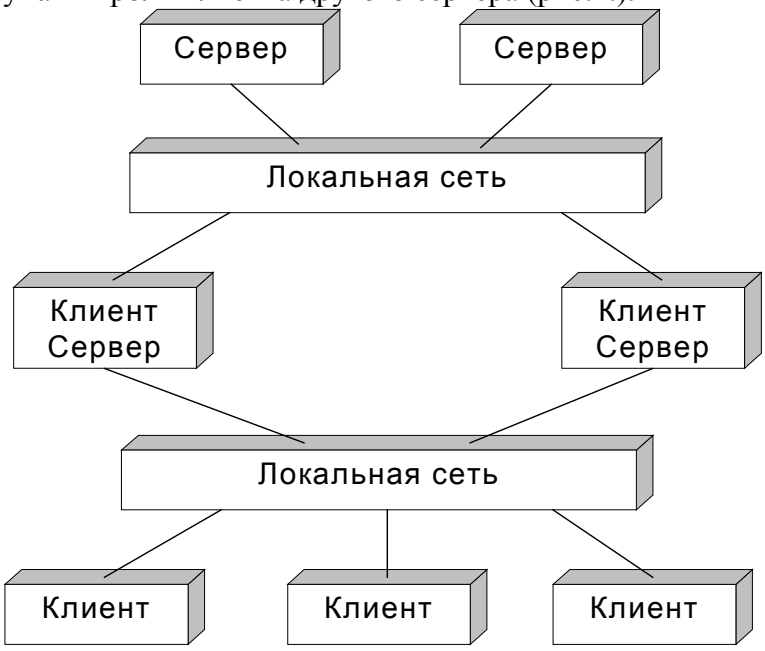

Рис.4. Конфигурация многоуровневой распределенной системы

В распределенном приложении кроме обмена информацией между клиентом и сервером может также присутствовать обмен информацией между равноправными узлами на основе асинхронного обмена сообщениями.

Рост числа клиент-серверных систем вызван некоторыми тенденциями в производстве оборудования, в частности увеличением мощности процессоров настольных персональных компьютеров, снижением стоимости микросхем и ростом объема как оперативной, так и дисковой памяти. Кроме того, повышается быстродействие вычислительных сетей, стремительно развивается Internet. Что касается программного обеспечения, широкое распространение получили реляционные базы данных, предоставляющие распределенный доступ к информации, графические интерфейсы и многозадачные приложения на платформе Windows, а также технологии программного обеспечения промежуточного слоя. На рис.5. приведена конфигурация системы клиент-сервер. В одном узле развернуто клиентское приложение с графическим интерфейсом пользователя (GUI). Оно работает под управлением стандартной операционной системы типа Windows 98 и пользуется стандартным коммуникационным программным обеспечением TCP/IP. Сверху операционной системы и коммуникационного программного обеспечения находиться программный слой, образующий программное обеспечение промежуточного слоя. В другом узле развернуто серверное приложение, пользующееся сервисами, которые предоставляет аналогичное программное обеспечение промежуточного слоя, размещенное поверх операционной системы UNIX или Windows NT. Для долговременного хранения информации применяется файловая система или СУБД.

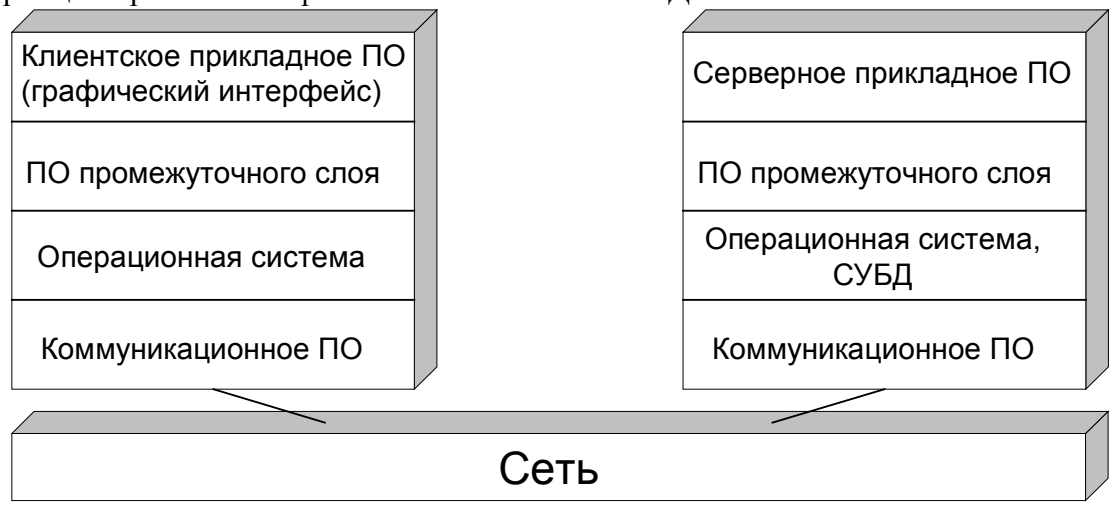

Рис.5. Технология клиент-сервер

Стандартом сетевых коммуникаций между открытыми системами является эталонная многоуровневая архитектура взаимодействия открытых систем (ISO OSI)(рис 6). В модели ISO семь уровней, каждый из которых отвечает за определенный аспект сетевых коммуникаций и предоставляет интерфейс в виде набора операций уровню, расположенному непосредственно над ним. Для каждого уровня в узле-отправителе есть эквивалентный уровень в узле-получателе.

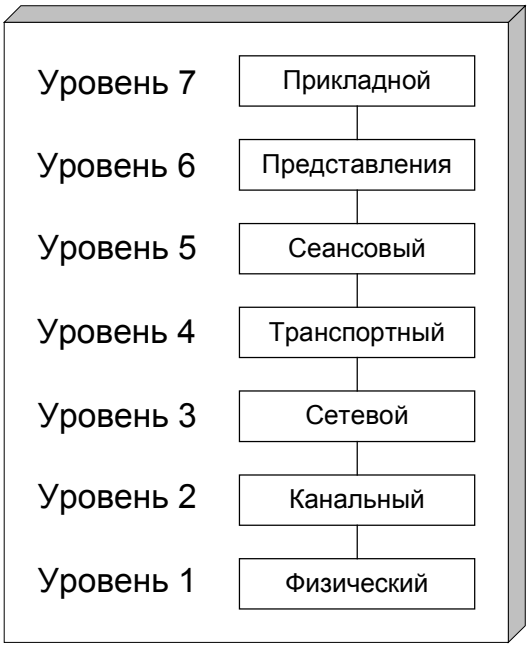

Рис. 6. Семиуровневая эталонная модель ISO

В сети Internet наиболее широкое распространение получил набор протоколов TCP/IP. Этот стек состоит из пяти уровней, показанных на рис.7. Уровни физический и интерфейсный соответствуют модели ISO. Физический уровень имеет дело с базовым сетевым оборудованием. Интерфейсный уровень определяет, как данные группируются во фреймы и как такие фреймы передаются по сети. На межсетевом уровне определяется формат пакетов данных, передаваемых через Internet, и механизмы прохождения пакетов через цепочку маршрутизаторов от отправителя к получателю. Узел маршрутизатора на рис.8. представляет собой шлюз, соединяющий локальную сеть с глобальной.

Транспортный уровень собирает пакеты в том порядке, в котором они были посланы, и формирует из них сообщение. TCP (Transmission Control Protocol) – это протокол транспортного уровня, работающий совместно с протоколом IP (Internet Protocol) межсетевого уровня. Уровень IP предоставляет ненадежный сервис отправки диаграмм, TCP же на его основе предоставляет надежный сервис. Он организует виртуальное соединение между приложениями двух узлов. Для транспорта сообщений TCP пользуется протоколом IP. Уровень 5 называется прикладным, на нем реализованы различные сетевые приложения, например передача файлов (FTP), электронная почта и WWW.

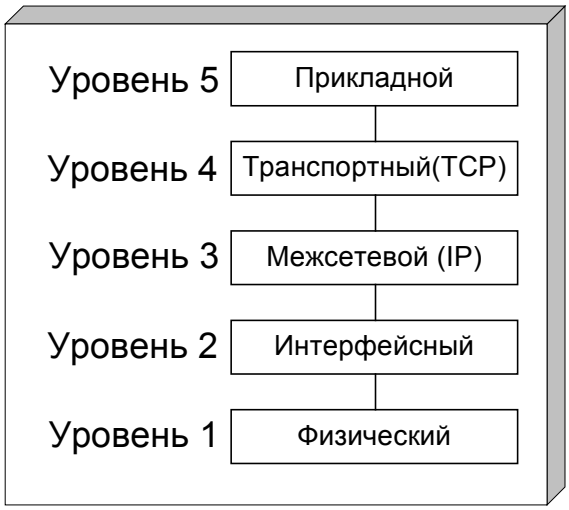

Рис. 7. Уровни модели TCP/IP

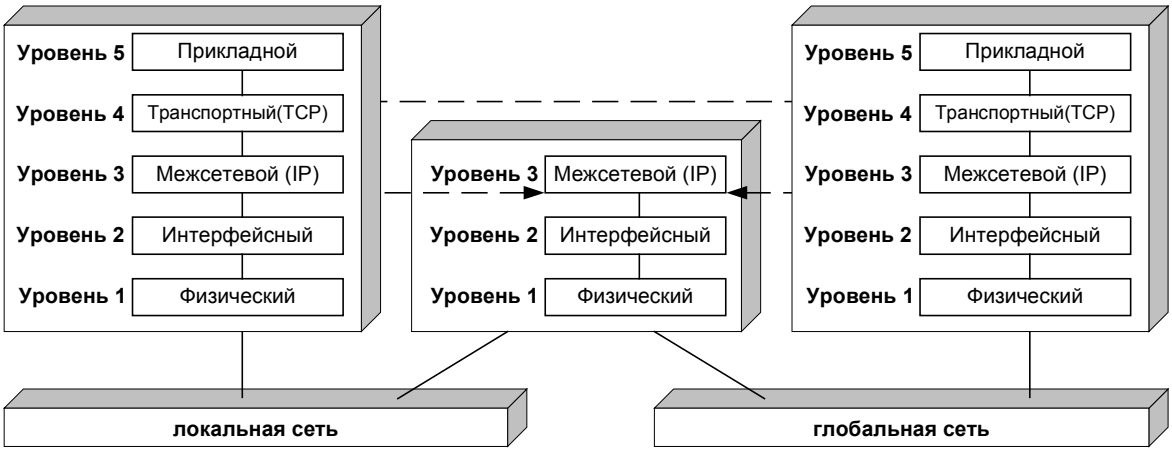

Рис.8. Обмен данными через сеть Internet по протоколам TCP/IP

Программное обеспечение промежуточного слоя

Распределенным системам часто приходится работать в таких средах, когда в разных узлах установлено различное оборудование и операционные системы. Примером может служить ситуация, когда клиенту под управлением Windows нужно общаться с сервером под управлением системы Unix. Программное обеспечение промежуточного слоя – это слой программного обеспечения, располагаемый поверх операционной системы с целью создания однородной платформы, на которой могут функционировать распределенные приложения. Ранней формой программного обеспечения промежуточного слоя был механизм вызова удаленных процедур RPC. Другие примеры – это технология вызова удаленных методов RMI в Java, а также технологии COM и CORBA. Предоставляя единообразный метод взаимодействия объектов, технологии программного обеспечения промежуточного слоя, такие как CORBA, COM и Java Beans, поощряют повторное использование компонентов. Коммуникации в модели клиент-сервер часто основаны на вызове удаленных процедур. При таком подходе процедуры находятся в адресном пространстве сервера и дистанционно запрашиваются клиентами. Сервер получает от клиента запрос, активизирует нужную процедуру и возвращает ответ. При объектном подходе к организации клиент-серверных приложений объекты получают глобальные имена и

могут вызываться непосредственно на сервере. Существует два подхода к распределенным вычислениям: модель распределенных объектов и модель мобильного кода. В первом случае объекты размещаются на сервере и вызываются дистанционно, как в Java RMI и CORBA. Во втором случае требуемые объекты перемещаются с сервера на клиент. Примером такого метода могут служить Java-апплеты.

При удаленном вызове процедур клиент в одном узле запрашивает удаленную процедуру сервера, находящегося в другом узле. Вызов удаленной процедуры аналогичен вызову локальной процедуры, поэтому то, что сервер находится далеко, скрыто от клиента. Процедура, необходимая клиенту (клиентская заглушка) принимает запрос и произвольные параметры, упаковывает их в сообщение и отправляет сообщение серверу. Удаленная серверная заглушка распаковывает сообщение и вызывает нужную процедуру, передавая ей параметры. Когда серверная процедура заканчивает обработку запроса, она возвращает результаты серверной заглушке, которая упаковывает их в ответное сообщение и отправляет клиентской заглушке. Клиентская заглушка извлекает результаты из сообщения и возвращает их клиенту в виде выходных параметров. Таким образом, роль клиентской и серверной заглушек сводится к тому, чтобы представить вызовы удаленных процедур так, как если бы они были локальными. На рис.9 изображен объект, обращающийся к локальной процедуре другого объекта. На рис.10. представлено распределенное решение той же задачи,

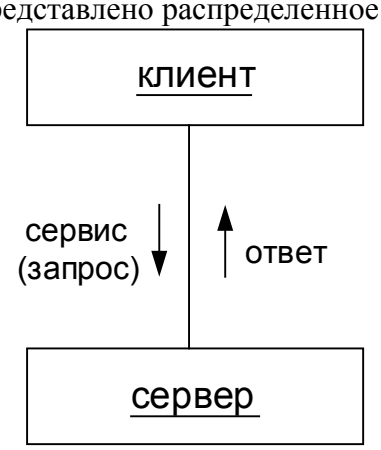

Рис.9. Вызов локальных процедур

когда объект на клиентском узле вызывает удаленную процедуру, принадлежащую объекту на удаленном серверном узле. Локально вызывается клиентская заглушка, которая упаковывает имя и параметры процедуры в сообщение, отправляемое по сети. Интерфейсный уровень удаленного узла получает сообщение и передает его серверной заглушке, которая распаковывает сообщение и вызывает указанную процедуру серверного объекта. После завершения процедуры серверная заглушка упаковывает ответ и посылает его по сети. Затем клиентская заглушка распаковывает сообщение и передает его клиентскому объекту.

При программировании в среде Java возможно использование технологии программного обеспечения промежуточного слоя RMI, которая позволяет распределенным Java-объектам общаться друг с другом. В этом случае вместо отправки сообщения некоторой процедуре как в RPC клиентский объект посылает сообщение другому объекту и вызывает метод этого объекта (функцию или процедуру). Роль клиентской заглушки из RPC играет клиентский заместитель (рис. 11). Заместитель предоставляет клиентскому объекту тот же интерфейс, что и серверный объект, и скрывает от клиента все детали коммуникации. Соответственно серверный заместитель выполняет функции серверной заглушки, пряча детали коммуникации от серверного

объекта. Серверный заместитель вызывает метод объекта. Если серверного объекта не существует, заместитель создает его. Разные серверные объекты могут предоставлять один и тот же интерфейс, а значит, обслуживать данный запрос клиента. Серверный объект, который будет обрабатывать запрос, выбирается во время выполнения.

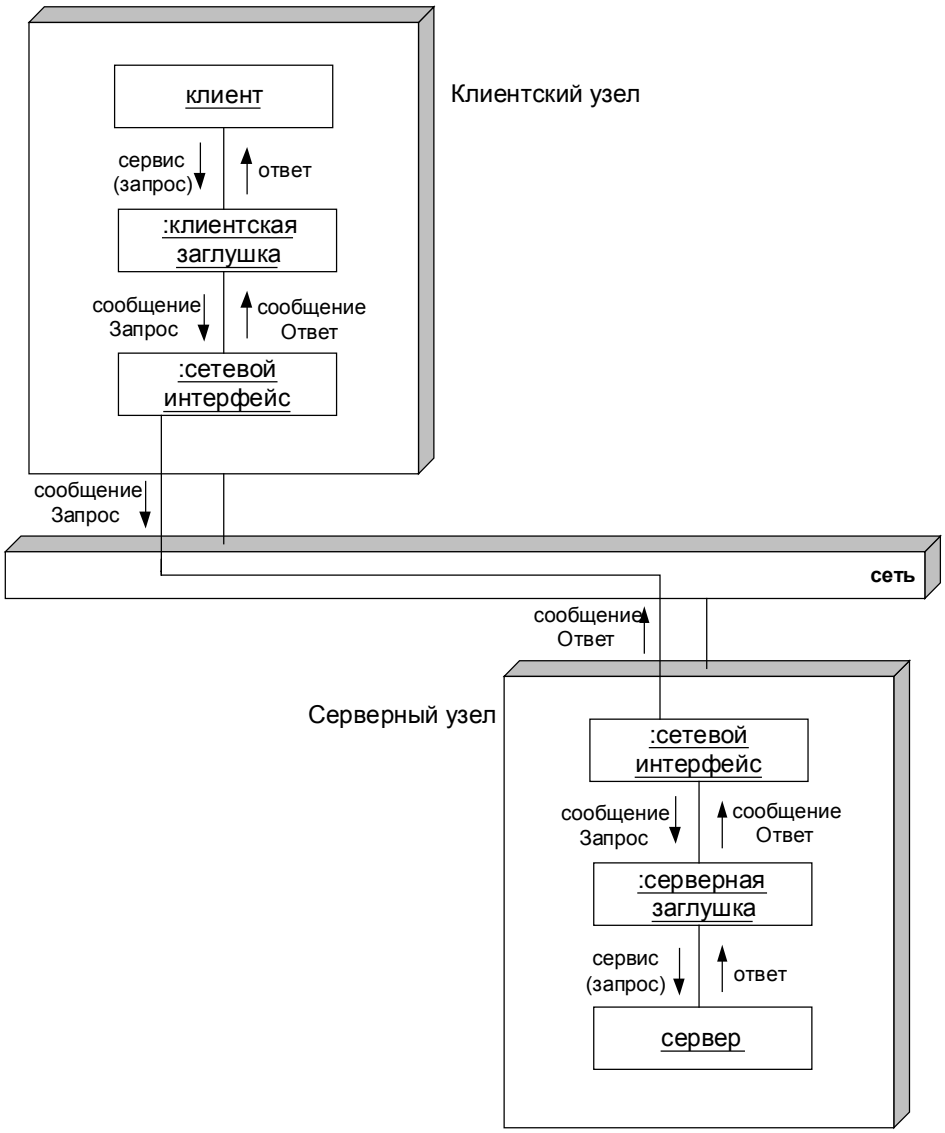

Рис.10. Вызов удаленных процедур (RPC)

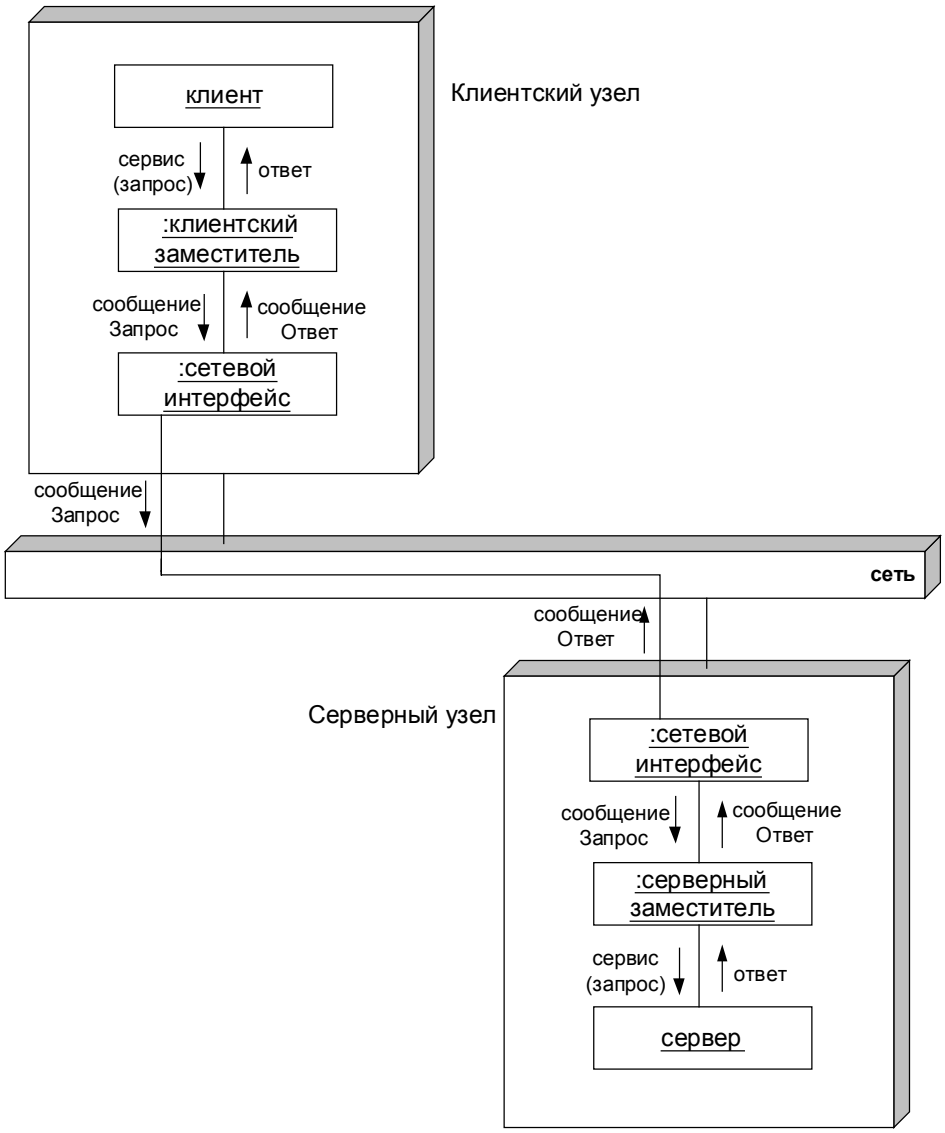

Рис.11. Вызов удаленных методов (RMI)

CORBA (Common Object Request Broker Architecture) - единая архитектура брокера объектных запросов – это стандарт открытых систем, разработанный группой Object Management Group (OMG), который обеспечивает взаимодействие между объектами на различных платформах под различными операционными системами. Брокер объектных запросов (ORB) выполняет функции программного обеспечения промежуточного слоя, поддерживающего отношения вида клиент-сервер между распределенными объектами. Серверные объекты предоставляют сервисы, которые клиенты могут запрашивать с помощью ORB. В общем случае клиенты и серверы – просто роли объектов. Таким образом, объект способен выступать в роли клиента в отношениях с одним объектом и в роли сервера в отношениях с другим. С помощью ORB клиентский объект в состоянии вызывать операции серверного объекта, не зная, где тот находится, на какой платформе (аппаратной и программной) исполняется, какие коммуникационные протоколы нужны для связи с ним и на каком языке он написан. Клиент, имеющий ссылку на серверный объект, может вызывать любые сервисы этого объекта через ORB.

ORB обладает следующими достоинствами:

- прозрачностью местоположения. ORB определяет местоположение объекта по ссылке на него;
- прозрачностью реализации. Клиент не должен знать, как реализован серверный объект, на каком языке он написан, на какой аппаратной и программной платформе исполняется;
- прозрачностью состояния выполнения объекта. Если серверный объект неактивен, то ORB активизирует его перед доставкой запроса;
- прозрачностью коммуникационного механизма. Клиент не знает, какой протокол будет использовать ORB.

Интерфейс серверного объекта описывается на языке определения интерфейсов (Interface Definition Language - IDL), не зависящем от конкретных языков программирования. Интерфейс определяется независимо от реализации. Спецификации, записанные на IDL, затем переводятся на язык реализации. Реализация объекта кодируется сразу на целевом языке. Группа OMG разработала стандартные отображения IDL на различные языки программирования, такие как C, C++, Java. Компиляторы IDL генерируют клиентские заглушки и серверные каркасы, аналогичные клиентским и серверным заместителям. Клиентская заглушка создает запрос и передает его от имени клиента. Серверный каркас получает запрос и доставляет его объекту, реализация которого должна удовлетворять требованиям CORBA. Функциональность, обеспечиваемая заглушками и каркасами в CORBA, аналогична механизму заглушек, применяемому в RPC. Программная заглушка выполняет упаковку параметров примерно так же, как при вызовах удаленных процедур. Высокоуровневые типы данных перед отправкой необходимо упаковать в такие структуры, которые можно передать по сети, например в поток байтов. Принимающий каркас должен распаковать входящее сообщение и вызвать запрошенную операцию серверного объекта.

В случае статического вызова заглушки и каркасы заранее скомпонованы с исполняемыми файлами. Статические интерфейсы определены в IDL-описаниях, созданных разработчиком. Эти описания транслируются в код заглушек, каркасов и заголовочных файлов, как определено в правилах отображения на конкретный язык. Этот подход представлен на рис.12. Большую гибкость обеспечивает динамическое связывание. В этом случае клиентский объект во время выполнения решает, с каким серверным объектом он будет общаться. Сервер регистрирует IDL-описание в архиве интерфейсов, который можно опрашивать во время выполнения. Клиент пользуется интерфейсом динамического вызова (рис.12), который представляет собой обобщенную заглушку, не зависящую от IDL-описания интерфейса вызываемого объекта. Такой подход позволяет клиенту обращаться к объекту во время выполнения, ничего не зная о его интерфейсе на стадии компиляции. CORBA поддерживает также интерфейс динамического каркаса на стороне сервера, то есть механизм динамического связывания для серверов, которые должны обрабатывать входящие запросы на обслуживание от объектов, не имеющих скомпилированных из IDL-описаний каркасов. Это полезно для коммуникации с такими внешними объектами, как шлюзы и браузеры.

Брокер объектных запросов позволяет клиенту прозрачно вызывать операцию серверного объекта, предоставляя службу имен. Когда создается объект CORBA, ему присваивается уникальная объектная ссылка. Служба имен CORBA дает ссылку на поименованный объект, а клиент затем вызывает операцию этого объекта.

Другой сервис, предоставляемый CORBA, - это трейдинг. Он позволяет получить ссылку на объект путем сопоставления характеристик известных объектов с характеристиками, посланными клиентом.

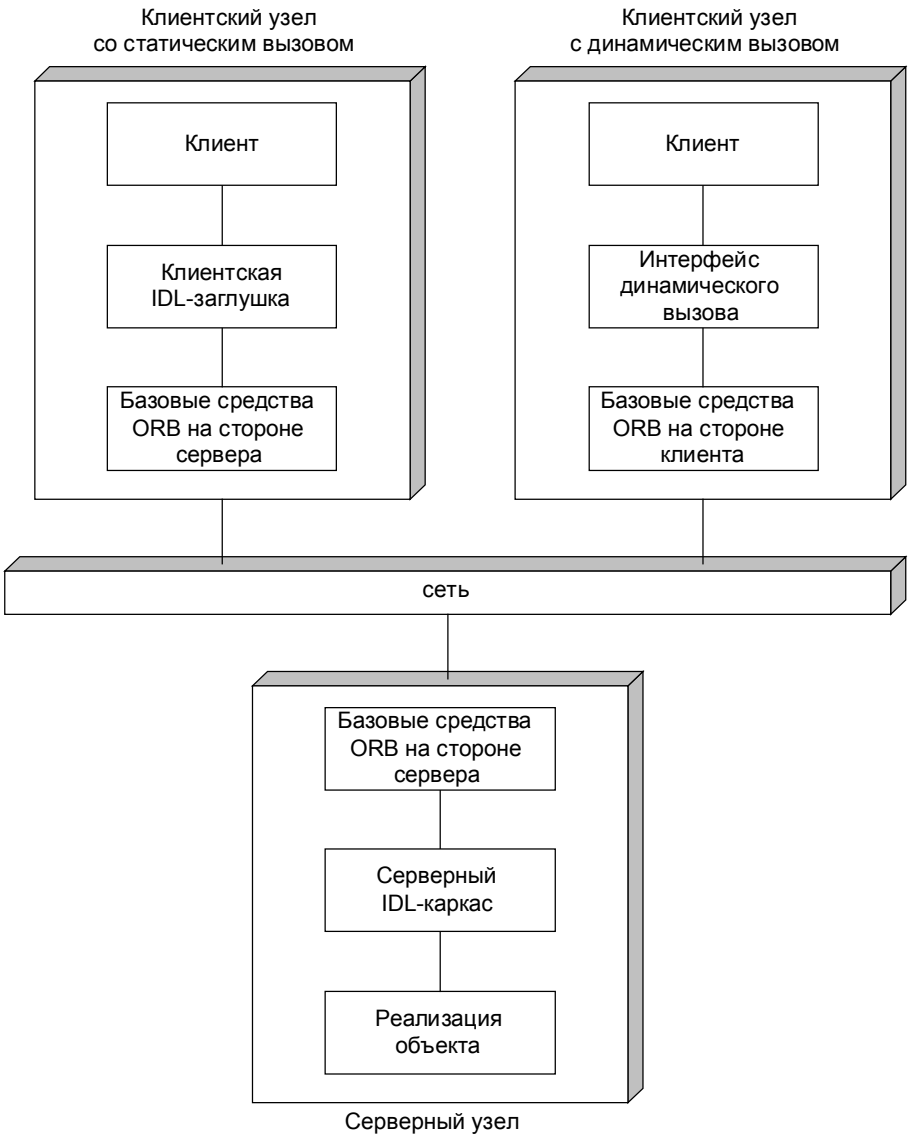

Рис.12. Архитектура CORBA

Помимо CORBA, существуют и другие распределенные технологии, например ActiveX и COM компании Microsoft и JavaBeans и Jini от компании Sun.

DCOM – это предложенная Microsoft распределенная объектная технология, построенная на основе архитектуры COM (Component Object Model – компонентная модель объектов). COM предоставляет каркас для взаимодействия приложений в среде Windows. DCOM позволяет клиенту общаться с компонентом, находящимся в удаленном узле, перехватывая вызовы клиента и переадресуя их серверу. И COM, и CORBA включают язык IDL, но CORBA задумана как стандарт, тогда как COM – патентованная технология, работающая только на платформе Windows.

Компоненты ActiveX – это исполняемые программы, которые согласуются со стандартом Microsoft COM и функционируют на платформе Windows. Их можно загрузить и выполнить внутри COM-совместимых контейнеров. Примером такого контейнера служит Web-браузер Internet Explorer.

JavaBeans предоставляет собой компонентную технологию на базе языка Java, предназначенную для специализированных приложений. JavaBeans – это компоненты пользовательского интерфейса на стороне клиента, а Enterprise JavaBeans – это компоненты на стороне сервера. Bean – компонент, состоящий из набора классов и

ресурсов. Из bean-объектов удобно собирать приложения с помощью специального инструментального средства. Во время сборки разрешается наблюдать за поведением объекта и адаптировать его для конкретных нужд. Bean – объекты, могут генерировать или обрабатывать входящие события. Инструмент сборки способен определять, какие события генерирует и получает объект, а также связывать объекты-отправители с объектами-получателями. Адаптированные и связанные bean-объекты допустимо сохранять для последующего использования.

Jini (Java Intelligent Network Infrastructure – сетевая интеллектуальная инфраструктура Java) – это технология соединения для встроенных систем и сетевых приложений, цель которых упростить взаимодействие компьютеров и других устройств. Jini предназначена для сотовых телефонов, цифровых камер, и других портативных устройств. Она использует технологию Java, а устройства соединяются посредством Java RMI.

Jini предоставляет службу поиска, выступающую в роли брокера между сервис-провайдерами и клиентами. В состав данной технологии входят также протоколы для обнаружения, присоединения и поиска ресурсов. Сервис-провайдер, например цифровой видеомагнитофон, регистрируется в службе имен Jini. Поэтому новый провайдер должен сначала динамически найти службу поиска, а затем зарегистрироваться в ней. Для каждого сервиса, который должен предоставлять провайдер, он должен загрузить Java-объект, обеспечивающий интерфейс к данному сервису.

Клиент Jini, например цифровая видеокамера, отыскивает службу имен, пользуясь протоколом обнаружения. Затем с помощью этой службы клиент находит нужный сервис, например сервис записи, предоставляемый видеомагнитофоном. После этого загружает из службы поиска Java-объект, который позволит ему напрямую взаимодействовать с устройством.

Система обработки транзакций

Приложения для обработки транзакций относятся к классу критических для бизнеса. Сюда входят системы ввода заказов, резервирование авиабилетов и кассовые терминалы. Транзакционное приложение занимается обновлением информации в базе данных. Главной составляющей нагрузки на приложение являются запросы на обновление информации. Некоторые транзакционные приложения должны работать постоянно, в других допустимы короткие перерывы.

Транзакция – это запрос клиента к серверу, состоящий из двух или более операций, которые образуют единую логическую функцию, причем она должна быть выполнена полностью, либо не выполнена совсем. Транзакции создаются клиентом и посылаются серверу для обработки. В классической конфигурации клиента и сервера обработка целиком возлагается на сервер. В распределенных приложениях сервер может распределить одну или несколько операций между другими серверами.

Например, при электронном переводе денежных средств, транзакция считается завершенной, если успешно произошли все операции. Если какую-то операцию осуществить не удается, транзакцию следует отменить. Это означает, что результаты выполнения отдельных составляющих операций необходимо аннулировать, чтобы система пришла в такое состояние, как будто данная транзакция и не начиналась.

У транзакций выделяются следующие свойства:

- атомарность. Транзакция – это неделимая единица работы. Она либо выполняется полностью, либо не выполняется вовсе;
- непротиворечивость. После завершения транзакции система должна оказаться в непротиворечивом состоянии;

- изолированность. На поведение транзакции не должны оказывать влияние другие транзакции;
- устойчивость. Изменения сохраняются после завершения транзакции, даже если за ним последует сбой системы.

Монитор обработки транзакций координирует поток информации между различными клиентами, инициирующими запросы, и приложением обработки транзакций, которое отвечает на эти запросы.

В клиент-серверной среде, которая реализована на различных платформах и под управлением различных операционных систем, мониторы выполняют следующие действия:

- посылают и принимают сообщения от клиентов и серверов;
- управляют потоком транзакций;
- распределяют нагрузку между серверами;
- поддерживают глобальные транзакции, которые относятся к нескольким распределенным базам данных, в частности гарантируют резервное копирование и восстановление глобальных транзакций;
- реализуют интерфейс с менеджерами ресурсов, например с операционной системой и СУБД;
- предоставляют средства для администрирования системы.

Современные мониторы поддерживают трехуровневую архитектуру клиент-сервер:

- функциональность клиента. Представление информации и взаимодействие с пользователем. Например, клиент может находиться на персональном компьютере.
- функциональность сервера приложений, поддерживающего бизнес-логику. Клиент общается с сервером приложений посредством сообщений;
- управление данными. В частности, реляционная база данных под управлением СУБД может быть распределена между несколькими узлами.

Изоляция клиента от сервера приложений позволяет раздельно проектировать и разрабатывать пользовательский интерфейс и бизнес-логику.

Проектирование архитектуры клиент-серверного приложения с использованием UML

Одна из важнейших целей, стоящих перед архитектурой клиент-серверного приложения, – спроектировать параллельную, основанную на обмене сообщениями программу, обеспечив при этом высокую степень конфигурируемости. Например, должна быть возможность разместить каждую подсистему в отдельном физическом узле. Распределенное приложение, состоящее из нескольких подсистем, допустимо сконфигурировать так, чтобы оно исполнялось в различных физических узлах. Три главных шага проектирования клиент-серверных приложений:

- Декомпозиция системы. Необходимо разбить приложение на отдельные подсистемы, которые потенциально могут исполняться в разных узлах. Из-за разнесенности, единственный возможный способ общения между подсистемами– это обмен сообщениями. Также необходимо

определить интерфейсы подсистем. Для начального выявления подсистем применяются критерии разбиения, цель которых проверить, что система действительно будет состоять из конфигурируемых компонентов, которые можно эффективно отобразить на физические узлы.

- Декомпозиция подсистем. Необходимо разбить подсистемы на параллельные задачи и скрывающие информацию объекты. Так как подсистема может исполняться только в одном узле, для нее применяются методы проектирования параллельных приложений.
- Конфигурация системы. После того как клиент-серверное приложение спроектировано, можно определить и сконфигурировать его экземпляры. На этой стадии необходимо указать экземпляры компонентов и связи между ними, а также спроецировать их на аппаратную конфигурацию, состоящую из нескольких физических узлов.

Примером клиент-серверной архитектуры служит банковская система (рис. 13). Например, в банке может быть единственное приложение, которое обслуживает множество территориально разнесенных банкоматов, общающихся с центральным банковским сервером. В более сложных клиент-серверных архитектурах могут присутствовать брокеры объектов, которые регистрирует сервисы, предоставляемые различными серверами. Клиент запрашивает сервисы у брокера. Такая организация обеспечивает независимость от географического местонахождения и платформы. Другой вариант клиент-серверной архитектуры – это использование программных агентов, выполняющих функции посредников между клиентами и серверами.

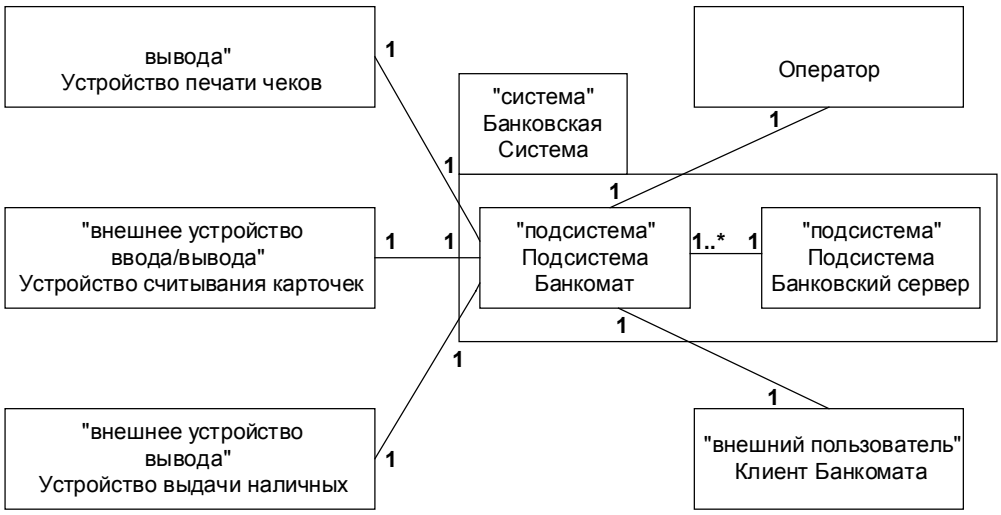

Рис.13. Программная архитектура клиент-сервер: банковская система.

Заключение

В данной работе представлен обзор технологий параллельной и распределенной обработки, требующихся для создания клиент-серверных приложений. Также приведено описание технологии клиент-сервер, которая применяется для создания сред распределенной обработки. В данной работе приведен обзор сетевых протоколов ISO и TCP/IP, а также технологий программного обеспечения промежуточного слоя, таких как RPC, RMI, CORBA, COM и систем обработки транзакций. Объектно-ориентировнные концепции особенно важны для анализа и проектирования программного обеспечения, поскольку они касаются фундаментальных вопросов. Язык моделирования UML (Unified Modeling Language) предлагает стандартизованную

нотацию для описания объектно-ориентированных моделей. Поэтому использование этого языка для анализа, проектирования и описания любых приложений, в том числе и клиент-серверных систем, является необходимым и единственно правильным решением, так как дает возможность разносторонне описать систему на любой стадии ее разработки.

Литература

1. Грейди Буч, Джеймс Рамбо, Айвар Джекобсон “UML руководство пользователя”, ДМК Пресс, 432с., 2004г.
2. Tanenbaum A.S. “Computer Networks”, Prentice Hall, 1996
3. Wendy Boggs, Michael Boggs “Mastering UML with Rational Rose”, SYBEX, 580p., 1999
4. Мартин Фаулер, Кендалл Скотт “UML. Основы”, СПб: Символ-Плюс, 192с., 2002