**Національний технічний університет України**

**“Київський Політехнічний Інститут”**

**ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ**

**КАФЕДРА ТЕХНІЧНОЇ КІБЕРНЕТИКИ**

---

ЕЛЕКТРОННИЙ КОНСПЕКТ ЛЕКЦІЙ

З ДИСЦИПЛІНИ

# “РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕРЕЖЕВИХ ПРОТОКОЛІВ”

НАПРЯМ - 050103 «Програмна інженерія»

**Киев - 2016 г.**

## Оглавление

# 1 Открытые системы и модель OSI

## 1.1 Протоколы, интерфейсы, стеки протоколов

Компьютерные сети, как правило, состоят из различного оборудования разных производителей, и без принятия всеми производителями общепринятых правил построения ПК и сетевого оборудования, обеспечить нормальное функционирование сетей было бы невозможно.
То есть для обеспечения нормального взаимодействия этого оборудования в сетях необходим единый унифицированный стандарт, который определял бы алгоритм передачи информации в сетях. В современных вычислительных сетях роль такого стандарта выполняют сетевые протоколы.
В связи с тем, что описать единым протоколом взаимодействия между устройствами в сети не представляется возможным, то необходимо разделить процесс сетевого взаимодействия на ряд концептуальных уровней (модулей) и определить функции для каждого модуля и порядок их взаимодействия, применив метод декомпозиции.
Используется многоуровневый подход метода декомпозиции, в соответствии с которым множество модулей решающих частные задачи упорядочивают по уровням образующим иерархию, процесс сетевого взаимодействия можем представить в виде иерархически организованного множества модулей.

Многоуровневое представление средств сетевого взаимодействия имеет свою специфику, связанную с тем, что в процессе обмена сообщениями участвуют две стороны, то есть необходимо организовать согласованную работу двух иерархий, работающих на разных компьютерах.

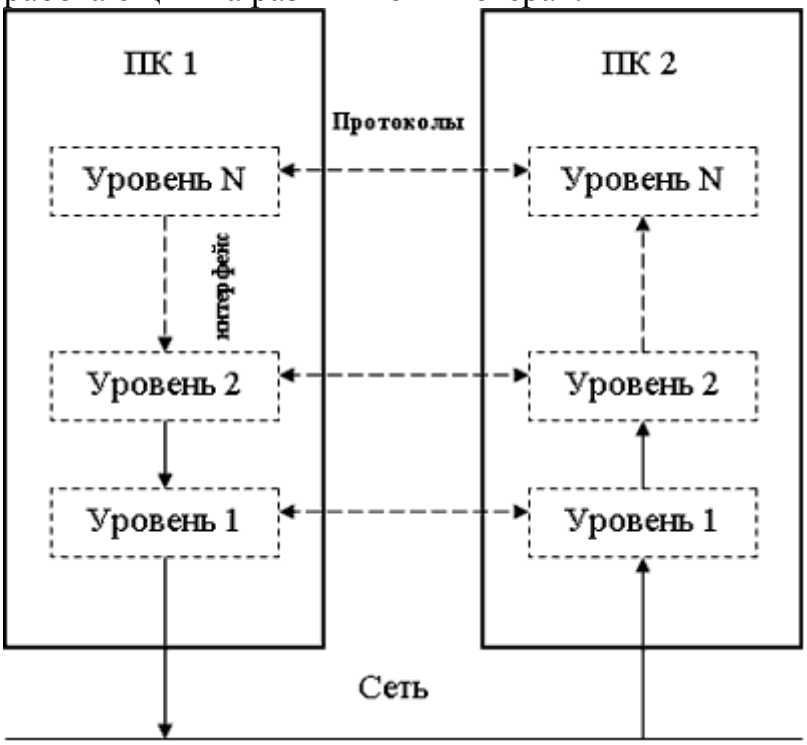



Рис. 1.1. Обобщенная схема обмена согласно OSI\ISO

Оба участника сетевого обмена должны принять множество соглашений. Соглашения должны быть приняты для всех уровней, начиная от самого низкого –

уровня передачи битов – до самого высокого, реализующего сервис для пользователя. Декомпозиция предполагает четкое определение функции каждого уровня и интерфейсов между уровнями.
Взаимодействие одноименных функциональных уровней по горизонтали осуществляется посредством протоколом. Протоколом называется набор правил и методов взаимодействия одноименных функциональных уровней объектов сетевого обмена.
Взаимодействия функциональных уровней по вертикали осуществляется через интерфейсы. Интерфейс определяет набор функций, которые нижележащий уровень предоставляет вышележащему уровню.
Таким образом, механизм передачи какого-либо пакета информации через сеть от клиентской программы, работающей на одном компьютере ПК 1, к клиентской программе, работающей на другом компьютере ПК 2, можно условно представить в виде последовательной пересылки этого пакета сверху вниз от верхнего уровня, обеспечивающего взаимодействие с пользовательским приложением, к нижнему уровню, организующему интерфейс с сетью, его трансляции на компьютер ПК 2 и обратной передачи верхнему уровню уже на ПК 2.
Коммуникационные протоколы могут быть реализованы как программно, так и аппаратно. Протоколы нижних уровней часто реализуются комбинацией программных и аппаратных средств, а протоколы верхних уровней – как правило, чисто программными средствами. Протоколы реализуются не только компьютерами, но и другими сетевыми устройствами – концентраторами, мостами, коммутаторами, маршрутизаторами и т.д. В зависимости от типа устройств в нем должны быть встроенные средства, реализующие тот или иной набор протоколов.
Иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети, называется стеком коммуникационных протоколов. В сети Интернет базовым набором протоколов является стек протоколов TCP/IP.

## 1.2 Обмен данных и взаимодействие компьютеров в сети

### 1.2.1 Общие понятия.

Главная цель, которая преследуется при соединении компьютеров в сеть - это возможность использования ресурсов каждого компьютера всеми пользователями сети. Для того, чтобы реализовать эту возможность, компьютеры, подсоединенные к сети, должны иметь необходимые для этого средства взаимодействия с другими компьютерами сети. Задача разделения сетевых ресурсов является сложной, она включает в себя решение множества проблем - выбор способа адресации компьютеров и согласование электрических сигналов при установление электрической связи, обеспечение надежной передачи данных и обработка сообщений об ошибках, формирование отправляемых и интерпретация полученных сообщений, а также много других не менее важных задач.

Обычным подходом при решении сложной проблемы является ее декомпозиция на несколько частных проблем - подзадач. Для решения каждой подзадачи назначается некоторый модуль. При этом четко определяются функции каждого модуля и правила их взаимодействия.

Частным случаем декомпозиции задачи является многоуровневое представление, при котором все множество модулей, решающих подзадачи, разбивается на иерархически упорядоченные группы - уровни. Для каждого уровня определяется набор функций-запросов, с которыми к модулям данного уровня могут обращаться модули выше лежащего уровня для решения своих задач. Такой формально определенный набор функций, выполняемых данным уровнем для выше лежащего уровня, а также форматы сообщений, которыми обмениваются два соседних уровня в ходе своего взаимодействия, называется *интерфейсом*.

### 1.2.2 Виды протоколов.

Интерфейс определяет совокупный сервис, предоставляемый данным уровнем выше лежащему уровню.

При организации взаимодействия компьютеров в сети каждый уровень ведет "переговоры" с соответствующим уровнем другого компьютера. При передаче сообщений оба участника сетевого обмена должны принять множество соглашений. Например, они должны согласовать уровни и форму электрических сигналов, способ определения длины сообщений, договориться о методах контроля достоверности и т.п. Другими словами, соглашения должны быть приняты для всех уровней, начиная от самого низкого уровня передачи битов, до самого высокого уровня, детализирующего, как информация должна быть интерпретирована.

Правила взаимодействия двух машин могут быть описаны в виде набора процедур для каждого из уровней. Такие формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне, но в разных узлах, называются *протоколами*.

Из приведенных определений можно заметить, что понятия "интерфейс" и "протокол", в сущности, обозначают одно и то же, а именно - формализовано заданные процедуры взаимодействия компонент, решающих задачу связи компьютеров в сети. Однако довольно часто в использовании этих терминов имеется некоторый нюанс: понятие "протокол" чаще применяют при описании правил взаимодействия компонент одного уровня, расположенных на разных узлах сети, а "интерфейс" - при описании правил взаимодействия компонентов соседних уровней, расположенных в пределах одного узла.

Согласованный набор протоколов разных уровней, достаточный для организации межсетевого взаимодействия, называется *стеком протоколов*.

Программные средства, реализующие некоторый протокол, также называют протоколом. При этом соотношение между протоколом - формально определенной процедурой взаимодействия, и протоколом - средством, реализующим эту процедуру, аналогично соотношению между алгоритмом решения некоторой задачи и программой, решающей эту задачу. Понятно, что один и тот же алгоритм может быть запрограммирован с разной степенью эффективности. Точно также и протокол может иметь несколько программных реализаций, например, протокол

IPX, реализованный компанией Microsoft для Windows NT в виде программного продукта NWLink, имеет характеристики, отличающиеся от реализации этого же протокола компанией Novell. Именно поэтому, при сравнении протоколов следует учитывать не только логику их работы, но и качество программных решений. Более того, на эффективность взаимодействия устройств в сети влияет качество всей совокупности протоколов, составляющих стек, то есть, насколько рационально распределены функции между протоколами разных уровней и насколько хорошо определены интерфейсы между ними.

Есть еще группа протоколов, которая относится к обеспечивающим реализацию распределенных приложений – это DCOM и CORBA, которые используют распределенные протоколы.

Протоколы реализуются не только программно-аппаратными средствами компьютеров, но и коммуникационными устройствами. Действительно, в общем случае связь компьютеров в сети осуществляется не напрямую - "компьютер-компьютер", а через различные коммуникационные устройства такие, например, как концентраторы, коммутаторы или маршрутизаторы. В зависимости от типа устройства, в нем должны быть встроены средства, реализующие некоторый набор сетевых протоколов.

При организации взаимодействия могут быть использованы два основных типа протоколов. В протоколах *с установлением соединения* (connection-oriented network service, CONS) перед обменом данными отправитель и получатель должны сначала установить логическое соединение, то есть договориться о параметрах процедуры обмена, которые будут действовать только в рамках данного соединения. После завершения диалога они должны разорвать это соединение. Когда устанавливается новое соединение, переговорная процедура выполняется заново. Телефон - это пример взаимодействия, основанного на установлении соединения.

Другая группа протоколов - протоколы *без предварительного установления* соединения (connectionless network service, CLNS). Такие протоколы называются также *дейтаграммными* протоколами. Отправитель просто передает сообщение, когда оно готово. Опускание письма в почтовый ящик - это пример связи без установления соединения.

### 1.3 Модель ISO/OSI

Из того, что протокол является соглашением, принятым двумя взаимодействующими объектами, в данном случае двумя работающими в сети компьютерами, совсем не следует, что он обязательно представляет собой стандарт. Но на практике при реализации сетей стремятся использовать стандартные протоколы. Это могут быть фирменные, национальные или международные стандарты.

Международная Организация по Стандартам (International Standards Organization, ISO) разработала модель, которая четко определяет различные уровни взаимодействия систем, дает им стандартные имена и указывает, какую работу

должен делать каждый уровень. Эта модель называется моделью взаимодействия открытых систем (Open System Interconnection, OSI) или моделью ISO/OSI.

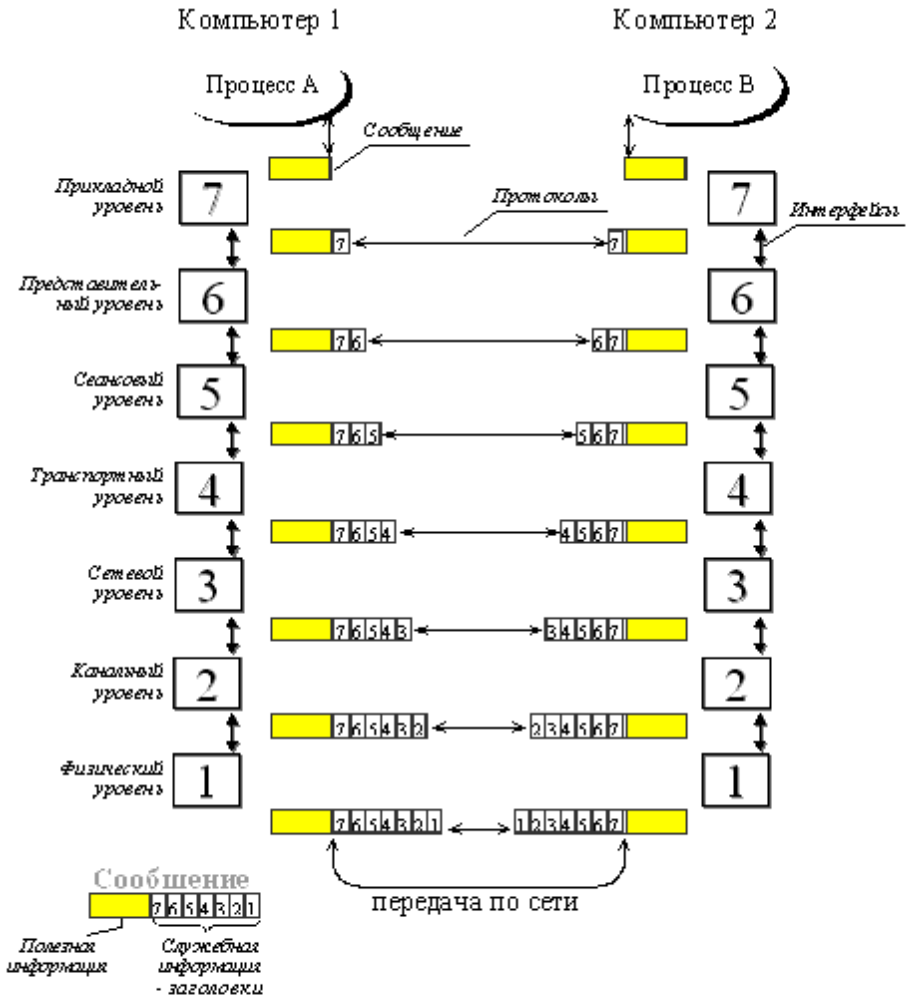



Рис. 1. Модель взаимодействия открытых систем ISO/OSI

В модели OSI взаимодействие делится на семь уровней или слоев (рис.1). Каждый уровень имеет дело с одним определенным аспектом взаимодействия. Таким образом, проблема взаимодействия декомпозирована на 7 частных проблем, каждая из которых может быть решена независимо от других. Каждый уровень поддерживает интерфейсы с выше- и нижележащими уровнями.

Модель OSI описывает только системные средства взаимодействия, не касаясь приложений конечных пользователей. Приложения реализуют свои собственные протоколы взаимодействия, обращаясь к системным средствам. Следует иметь в виду, что приложение может взять на себя функции некоторых верхних уровней модели OSI, в таком случае, при необходимости межсетевого обмена оно обращается напрямую к системным средствам, выполняющим функции оставшихся нижних уровней модели OSI.

Приложение конечного пользователя может использовать системные средства взаимодействия не только для организации диалога с другим приложением, выполняющимся на другой машине, но и просто для получения услуг того или

иного сетевого сервиса, например, доступа к удаленным файлам, получение почты или печати на разделяемом принтере.

Итак, пусть приложение обращается с запросом к прикладному уровню, например к файловому сервису. На основании этого запроса программное обеспечение прикладного уровня формирует сообщение стандартного формата, в которое помещает служебную информацию (заголовок) и, возможно, передаваемые данные. Затем это сообщение направляется представительному уровню. Представительный уровень добавляет к сообщению свой заголовок и передает результат вниз сеансовому уровню, который в свою очередь добавляет свой заголовок и т.д. Некоторые реализации протоколов предусматривают наличие в сообщении не только заголовка, но и концевика. Наконец, сообщение достигает самого низкого, физического уровня, который действительно передает его по линиям связи.

Когда сообщение по сети поступает на другую машину, оно последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует, обрабатывает и удаляет заголовок своего уровня, выполняет соответствующие данному уровню функции и передает сообщение вышележащему уровню.

Кроме термина "сообщение" (message) существуют и другие названия, используемые сетевыми специалистами для обозначения единицы обмена данными. В стандартах ISO для протоколов любого уровня используется такой термин как "протокольный блок данных" - Protocol Data Unit (PDU). Кроме этого, часто используются названия кадр (frame), пакет (packet), дейтаграмма (datagram).

### 1.4 Функции уровней модели ISO/OSI

***Физический уровень.*** Этот уровень имеет дело с передачей битов по физическим каналам, таким, например, как коаксиальный кабель, витая пара или оптоволоконный кабель. К этому уровню имеют отношение характеристики физических сред передачи данных, такие как полоса пропускания, помехозащищенность, волновое сопротивление и другие. На этом же уровне определяются характеристики электрических сигналов, такие как требования к фронтам импульсов, уровням напряжения или тока передаваемого сигнала, тип кодирования, скорость передачи сигналов. Кроме этого, здесь стандартизуются типы разъемов и назначение каждого контакта.

Функции физического уровня реализуются во всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом.

Примером протокола физического уровня может служить спецификация 10Base-T технологии Ethernet, которая определяет в качестве используемого кабеля неэкранированную витую пару категории 3 с волновым сопротивлением 100 Ом, разъем RJ-45, максимальную длину физического сегмента 100 метров, манчестерский код для представления данных на кабеле, и другие характеристики среды и электрических сигналов.

***Канальный уровень.*** На физическом уровне просто пересылаются биты. При этом не учитывается, что в некоторых сетях, в которых линии связи используются (разделяются) попеременно несколькими парами взаимодействующих компьютеров, физическая среда передачи может быть занята. Поэтому одной из задач канального уровня является проверка доступности среды передачи. Другой задачей канального уровня является реализация механизмов обнаружения и коррекции ошибок. Для этого на канальном уровне биты группируются в наборы, называемые кадрами (frames). Канальный уровень обеспечивает корректность передачи каждого кадра, помещая специальную последовательность бит в начало и конец каждого кадра, чтобы отметить его, а также вычисляет контрольную сумму, суммируя все байты кадра определенным способом и добавляя контрольную сумму к кадру. Когда кадр приходит, получатель снова вычисляет контрольную сумму полученных данных и сравнивает результат с контрольной суммой из кадра. Если они совпадают, кадр считается правильным и принимается. Если же контрольные суммы не совпадают, то фиксируется ошибка.

В протоколах канального уровня, используемых в локальных сетях, заложена определенная структура связей между компьютерами и способы их адресации. Хотя канальный уровень и обеспечивает доставку кадра между любыми двумя узлами локальной сети, он это делает только в сети с совершенно определенной топологией связей, именно той топологией, для которой он был разработан. К таким типовым топологиям, поддерживаемым протоколами канального уровня локальных сетей, относятся общая шина, кольцо и звезда. Примерами протоколов канального уровня являются протоколы Ethernet, Token Ring, FDDI, 100VG-AnyLAN.

В локальных сетях протоколы канального уровня используются компьютерами, мостами, коммутаторами и маршрутизаторами. В компьютерах функции канального уровня реализуются совместными усилиями сетевых адаптеров и их драйверов.

В глобальных сетях, которые редко обладают регулярной топологией, канальный уровень обеспечивает обмен сообщениями между двумя соседними компьютерами, соединенными индивидуальной линией связи. Примерами протоколов "точка - точка" (как часто называют такие протоколы) могут служить широко распространенные протоколы PPP и LAP-B.

***Сетевой уровень.*** Этот уровень служит для образования единой транспортной системы, объединяющей несколько сетей с различными принципами передачи информации между конечными узлами. Рассмотрим функции сетевого уровня на примере локальных сетей. Протокол канального уровня локальных сетей обеспечивает доставку данных между любыми узлами только в сети с соответствующей *типовой топологией*. Это очень жесткое ограничение, которое не позволяет строить сети с развитой структурой, например, сети, объединяющие несколько сетей предприятия в единую сеть, или высоконадежные сети, в которых существуют избыточные связи между узлами. Для того, чтобы, с одной стороны, сохранить простоту процедур передачи данных для типовых топологий, а с другой стороны, допустить использование произвольных топологий, используется дополнительный сетевой уровень. На этом уровне вводится понятие "сеть". В

данном случае под сетью понимается совокупность компьютеров, соединенных между собой в соответствии с одной из стандартных типовых топологий и использующих для передачи данных один из протоколов канального уровня, определенный для этой топологии.

Таким образом, внутри сети доставка данных регулируется канальным уровнем, а вот доставкой данных между сетями занимается сетевой уровень.

Сообщения сетевого уровня принято называть *пакетами (packets)*. При организации доставки пакетов на сетевом уровне используется понятие *"номер сети"*. В этом случае адрес получателя состоит из номера сети и номера компьютера в этой сети.

Для того, чтобы передать сообщение от отправителя, находящегося в одной сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество транзитных передач (hops) между сетями, каждый раз выбирая подходящий маршрут. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет.

Проблема выбора наилучшего пути называется *маршрутизацией* и ее решение является главной задачей сетевого уровня. Эта проблема осложняется тем, что самый короткий путь не всегда самый лучший. Часто критерием при выборе маршрута является время передачи данных по этому маршруту, оно зависит от пропускной способности каналов связи и интенсивности трафика, которая может изменяться с течением времени. Некоторые алгоритмы маршрутизации пытаются приспособиться к изменению нагрузки, в то время, как другие принимают решения на основе средних показателей за длительное время. Выбор маршрута может осуществляться и по другим критериям, например, надежности передачи.

На сетевом уровне определяется два вида протоколов. Первый вид относится к определению правил передачи пакетов с данными конечных узлов от узла к маршрутизатору и между маршрутизаторами. Именно эти протоколы обычно имеют в виду, когда говорят о протоколах сетевого уровня. К сетевому уровню относят и другой вид протоколов, называемых *протоколами обмена маршрутной информацией*. С помощью этих протоколов маршрутизаторы собирают информацию о топологии межсетевых соединений. Протоколы сетевого уровня реализуются программными модулями операционной системы, а также программными и аппаратными средствами маршрутизаторов.

Примерами протоколов сетевого уровня являются протокол межсетевого взаимодействия IP стека TCP/IP и протокол межсетевого обмена пакетами IPX стека Novell.

***Транспортный уровень.*** На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Хотя некоторые приложения имеют собственные средства обработки ошибок, существуют и такие, которые предпочитают сразу иметь дело с надежным соединением. Работа транспортного уровня заключается в том, чтобы обеспечить приложениям или верхним уровням стека - прикладному и сеансовому - передачу данных с той степенью надежности, которая им требуется. Модель OSI

определяет пять классов сервиса, предоставляемых транспортным уровнем. Эти виды сервиса отличаются качеством предоставляемых услуг: срочностью, возможностью восстановления прерванной связи, наличием средств мультиплексирования нескольких соединений между различными прикладными протоколами через общий транспортный протокол, а главное - способностью к обнаружению и исправлению ошибок передачи, таких как искажение, потеря и дублирование пакетов.

Выбор класса сервиса транспортного уровня определяется, с одной стороны, тем, в какой степени задача обеспечения надежности решается самими приложениями и протоколами более высоких, чем транспортный, уровней, а с другой стороны, этот выбор зависит от того, насколько надежной является вся система транспортировки данных в сети. Так, например, если качество каналов передачи связи очень высокое, и вероятность возникновения ошибок, не обнаруженных протоколами более низких уровней, невелика, то разумно воспользоваться одним из облегченных сервисов транспортного уровня, не обремененных многочисленными проверками, квитированием и другими приемами повышения надежности. Если же транспортные средства изначально очень ненадежны, то целесообразно обратиться к наиболее развитому сервису транспортного уровня, который работает, используя максимум средств для обнаружения и устранения ошибок - с помощью предварительного установления логического соединения, контроля доставки сообщений с помощью контрольных сумм и циклической нумерации пакетов, установления тайм-аутов доставки и т.п.

Как правило, все протоколы, начиная с транспортного уровня и выше, реализуются программными средствами конечных узлов сети - компонентами их сетевых операционных систем. В качестве примера транспортных протоколов можно привести протоколы TCP и UDP стека TCP/IP и протокол SPX стека Novell.

***Сеансовый уровень.*** Сеансовый уровень обеспечивает управление диалогом для того, чтобы фиксировать, какая из сторон является активной в настоящий момент, а также предоставляет средства синхронизации. Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, вместо того, чтобы начинать все с начала. На практике немногие приложения используют сеансовый уровень, и он редко реализуется.

***Уровень представления.*** Этот уровень обеспечивает гарантию того, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе. При необходимости уровень представления выполняет преобразование форматов данных в некоторый общий формат представления, а на приеме, соответственно, выполняет обратное преобразование. Таким образом, прикладные уровни могут преодолеть, например, синтаксические различия в представлении данных. На этом уровне может выполняться шифрование и дешифрование данных, благодаря которому секретность обмена данными обеспечивается сразу для всех прикладных сервисов. Примером протокола, работающего на уровне представления, является протокол Secure Socket Layer (SSL), который обеспечивает секретный обмен сообщениями для протоколов прикладного уровня стека TCP/IP.

***Прикладной уровень***. Прикладной уровень - это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например, с помощью протокола электронной почты. Единица данных, которой оперирует прикладной уровень, обычно называется *сообщением (message)*.

Существует очень большое разнообразие протоколов прикладного уровня. Приведем в качестве примеров хотя бы несколько наиболее распространенных реализаций файловых сервисов: NCP в операционной системе Novell NetWare, SMB в Microsoft Windows NT, NFS, FTP и TFTP, входящие в стек TCP/IP.

Модель OSI представляет хотя и очень важную, но только одну из многих моделей коммуникаций. Эти модели и связанные с ними стеки протоколов могут отличаться количеством уровней, их функциями, форматами сообщений, сервисами, предоставляемыми на верхних уровнях и прочими параметрами

# 2 Протоколы взаимодействия приложений

## 2.1 Сетенезависимые протоколы.

Функции всех уровней модели OSI могут быть отнесены к одной из двух групп: либо к функциям, зависящим от конкретной технической реализации сети, либо к функциям, ориентированным на работу с приложениями.

Три нижних уровня - физический, канальный и сетевой - являются сетезависимыми, то есть протоколы этих уровней тесно связаны с технической реализацией сети, с используемым коммуникационным оборудованием. Например, переход на оборудование FDDI означает полную смену протоколов физического и канального уровня во всех узлах сети.

Три верхних уровня - сеансовый, уровень представления и прикладной - ориентированы на приложения и мало зависят от технических особенностей построения сети. На протоколы этих уровней не влияют никакие изменения в топологии сети, замена оборудования или переход на другую сетевую технологию. Так, переход от Ethernet на высокоскоростную технологию ATM не потребует никаких изменений в программных средствах, реализующих функции прикладного, представительного и сеансового уровней.

Транспортный уровень является промежуточным, он скрывает все детали функционирования нижних уровней от верхних уровней. Это позволяет разрабатывать приложения, независящие от технических средств, непосредственно занимающихся транспортировкой сообщений.

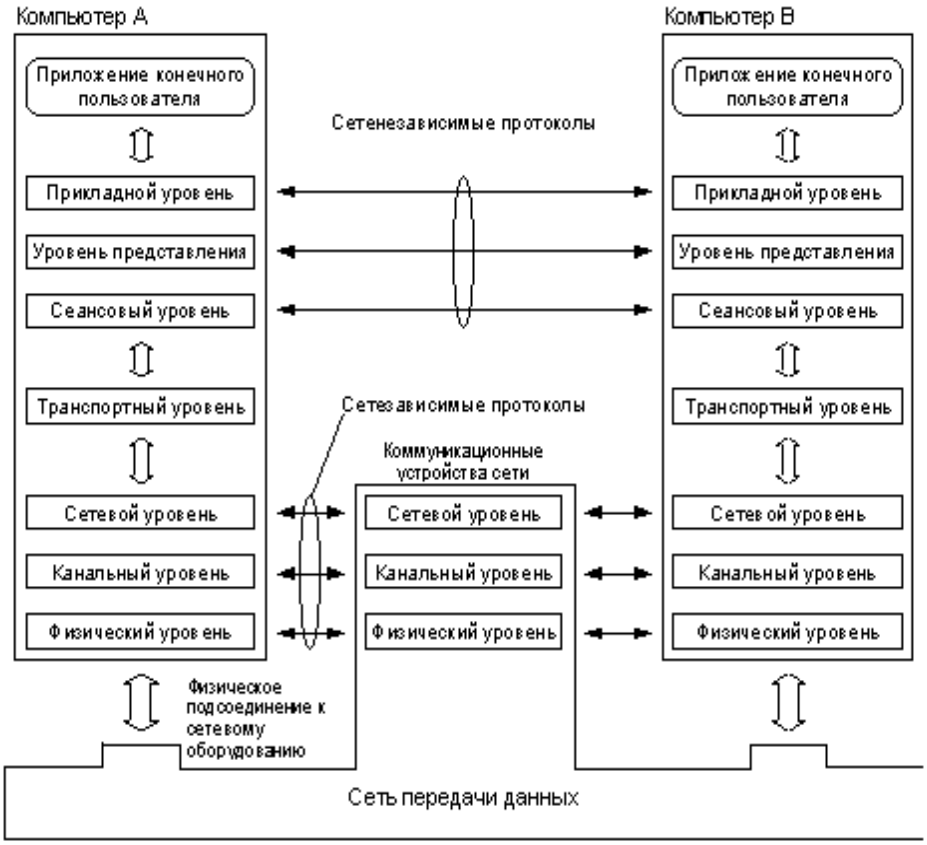



Рис. 2. Сетезависимые и сетенезависимые уровни модели OSI

Рисунок 2 показывает уровни модели OSI, на которых работают различные элементы сети. Компьютер, с установленной на нем сетевой ОС, взаимодействует с другим компьютером с помощью протоколов всех семи уровней. Это взаимодействие компьютеры осуществляют через различные коммуникационные устройства: концентраторы, модемы, мосты, коммутаторы, маршрутизаторы, мультиплексоры. В зависимости от типа, коммуникационное устройство может работать либо только на физическом уровне (повторитель), либо на физическом и канальном (мост и коммутатор), либо на физическом, канальном и сетевом, иногда захватывая и транспортный уровень (маршрутизатор).

## 2.2 Функциональное соответствие видов коммуникационного оборудования уровням модели OSI

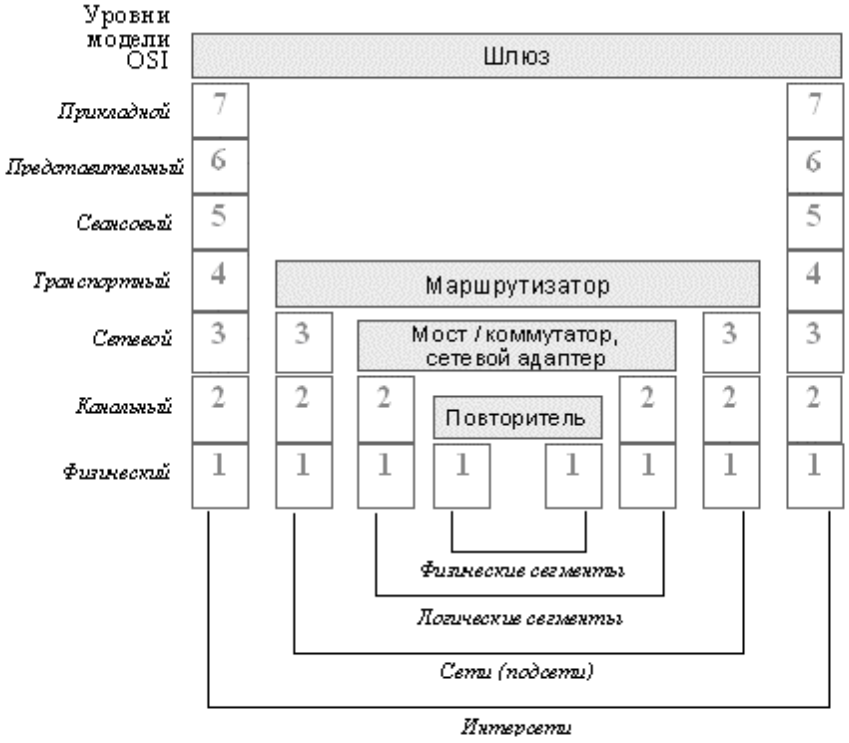



Рис. 3. Соответствие функций коммуникационного оборудования модели OSI

Лучшим способом для понимания отличий между сетевыми адаптерами, повторителями, мостами/коммутаторами и маршрутизаторами является рассмотрение их работы в терминах модели OSI. Соотношение между функциями этих устройств и уровнями модели OSI показано на рисунке 3.

Повторитель, который регенерирует сигналы, за счет чего позволяет увеличивать длину сети, работает на физическом уровне.

Сетевой адаптер работает на физическом и канальном уровнях. К физическому уровню относится та часть функций сетевого адаптера, которая связана с приемом и передачей сигналов по линии связи, а получение доступа к разделяемой среде передачи, распознавание MAC-адреса компьютера - это уже функция канального уровня.

Мосты выполняют большую часть своей работы на канальном уровне. Для них сеть представляется набором MAC-адресов устройств. Они извлекают эти адреса из заголовков, добавленных к пакетам на канальном уровне, и используют их во время обработки пакетов для принятия решения о том, на какой порт отправить тот или иной пакет. Мосты не имеют доступа к информации об адресах сетей, относящейся к более высокому уровню. Поэтому они ограничены в принятии решений о возможных путях или маршрутах перемещения пакетов по сети.

Маршрутизаторы работают на сетевом уровне модели OSI. Для маршрутизаторов сеть - это набор сетевых адресов устройств и множество сетевых путей. Маршрутизаторы анализируют все возможные пути между любыми двумя узлами

сети и выбирают самый короткий из них. При выборе могут приниматься во внимание и другие факторы, например, состояние промежуточных узлов и линий связи, пропускная способность линий или стоимость передачи данных.

Для того, чтобы маршрутизатор мог выполнять возложенные на него функции ему должна быть доступна более развернутая информация о сети, нежели та, которая доступна мосту. В заголовке пакета сетевого уровня кроме сетевого адреса имеются данные, например, о критерии, который должен быть использован при выборе маршрута, о времени жизни пакета в сети, о том, какому протоколу верхнего уровня принадлежит пакет.

Благодаря использованию дополнительной информации, маршрутизатор может осуществлять больше операций с пакетами, чем мост/коммутатор. Поэтому программное обеспечение, необходимое для работы маршрутизатора, является более сложным.

На рисунке 3 показан еще один тип коммуникационных устройств - шлюз, который может работать на любом уровне модели OSI. Шлюз (gateway) - это устройство, выполняющее трансляцию протоколов. Шлюз размещается между взаимодействующими сетями и служит посредником, переводящим сообщения, поступающие из одной сети, в формат другой сети. Шлюз может быть реализован как чисто программными средствами, установленными на обычном компьютере, так и на базе специализированного компьютера. Трансляция одного стека протоколов в другой представляет собой сложную интеллектуальную задачу, требующую максимально полной информации о сети, поэтому шлюз использует заголовки всех транслируемых протоколов.

## 2.3 Особенности протоколов, используемых в локальных и глобальных сетях

В настоящее время наблюдается тенденция к сближению протоколов локальных и глобальных сетей. Ярким примером являются протоколы технологии ATM, работающие без изменений как в тех, так и в других сетях. Тем не менее, большинство протоколов, используемых сегодня, относятся либо к локальным, либо к глобальным сетям и не могут применяться не по прямому назначению.

Различия между протоколами локальных и глобальных сетей происходят в основном из-за различий между свойствами каналов, использующихся в этих сетях.

Каналы локальных сетей имеют небольшую длину и высокое качество, а каналы глобальных сетей - наоборот, большую длину и низкое качество.

Небольшая длина каналов локальных сетей создала возможность совместного использования их узлами сети в режиме разделения времени. Практически все протоколы локальных сетей имеют версию работы на разделяемых средах передачи данных, хотя более поздние протоколы (Fast Ethernet, Gigabit Ethernet) имеют также и версию работы на индивидуальных каналах в полнодуплексном режиме. Большая протяженность каналов глобальных сетей делает

нерациональными любые процедуры разделения канала во времени, так как длительность этих процедур становится слишком большой. Поэтому каналы глобальных сетей используются всегда на индивидуальной основе как связи типа "точка - точка".

Высокое качество кабелей локальных сетей послужило причиной отказа от использования в протоколах локальных сетей процедур восстановления искаженных и потерянных кадров. Этих процедур нет ни в протоколах семейства Ethernet, ни у протокола Token Ring, ни у протокола FDDI. В то же время в протоколах глобальных сетей, ориентирующихся на каналы плохого качества, процедурам восстановления кадров всегда уделялось большое внимание. Например, в сетях X.25 восстановлением кадров занимаются сразу два смежных протокола - LAP-B на канальном уровне и протокол X.25/3 - на сетевом.

Начало массового использования цифровых оптоволоконных каналов в глобальных сетях, обеспечивающих высокое качество передачи данных, послужило причиной разработки протоколов глобальных сетей нового поколения, в которых отсутствуют процедуры восстановления кадров. Такой особенностью обладают, например, сети frame relay и ATM.

Таким образом, одно из отличий протоколов локальных и глобальных сетей преодолено за счет продвижения глобальных сетей навстречу локальным. Второе отличие сегодня снимается за счет быстрого внедрения в локальные сети техники микросегментации, отказывающейся от использования разделяемых сред и предоставляющей каждому узлу сети индивидуальный коммутируемый канал. В результате, протоколы локальных и глобальных сетей все больше сближаются, а существование технологии ATM доказывает, что принципиальных причин для существования между этими классами протоколов четкой границы сегодня не существует и ее окончательное исчезновение - не за горами.

# 3 Протоколы предачи данных в современных вычислительных сетях

## 3.1 Характеристика популярных стеков коммуникационных протоколов

Существует достаточно много стеков протоколов, широко применяемых в сетях. Это и стеки, являющиеся международными и национальными стандартами, и фирменные стеки, получившие распространение благодаря распространенности оборудования той или иной фирмы. Примерами популярных стеков протоколов могут служить стек IPX/SPX фирмы Novell, стек TCP/IP, используемый в сети Internet и во многих сетях на основе операционной системы UNIX, стек OSI международной организации по стандартизации, стек DECnet корпорации Digital Equipment и некоторые другие.

Использование в сети того или иного стека коммуникационных протоколов во многом определяет лицо сети и ее характеристики. В небольших сетях может использоваться исключительно один стек. В крупных корпоративных сетях, объединяющих различные сети, параллельно используются, как правило, несколько стеков.

В коммуникационном оборудовании реализуются протоколы нижних уровней, которые в большей степени стандартизованы, чем протоколы верхних уровней, и это является предпосылкой для успешной совместной работы оборудования различных производителей. Перечень протоколов, поддерживаемых тем или иным коммуникационным устройством, является одной из наиболее важных характеристик этого устройства.

Компьютеры реализуют коммуникационные протоколы в виде соответствующих программных элементов сетевой операционной системы, например, протоколы канального уровня, как правило, выполнены в виде драйверов сетевых адаптеров, а протоколы верхних уровней в виде серверных и клиентских компонент сетевых сервисов.

Умение хорошо работать в среде той или иной операционной системы является важной характеристикой коммуникационного оборудования. Часто можно прочитать в рекламе сетевого адаптера или концентратора, что он разрабатывался специально для работы в сети NetWare или UNIX. Это означает, что разработчики аппаратуры оптимизировали ее характеристики применительно к тем протоколам, которые используются в этой сетевой операционной системе, или к данной версии их реализации, если эти протоколы используются в различных ОС. Из-за особенностей реализации протоколов в различных ОС, в качестве одной из характеристик коммуникационного оборудования используется его сертифицированность на возможность работы в среде данной ОС.

На нижних уровнях - физическом и канальном - практически во всех стеках используются одни и те же протоколы. Это хорошо стандартизованные протоколы Ethernet, Token Ring, FDDI и некоторые другие, которые позволяют использовать во всех сетях одну и ту же аппаратуру.

Протоколы сетевого и более высоких уровней существующих стандартных стеков отличаются большим разнообразием и, как правило, не соответствуют рекомендуемому моделью ISO разбиению на уровни. В частности, в этих стеках функции сеансового и представительного уровня чаще всего объединены с прикладным уровнем. Такое несоответствие связано с тем, что модель ISO появилась как результат обобщения уже существующих и реально используемых стеков, а не наоборот.

## 3.2 Стек TCP/IP

Стек TCP/IP, называемый также стеком DoD и стеком Internet, является одним из наиболее популярных и перспективных стеков коммуникационных протоколов. Если в настоящее время он распространен в основном в сетях с ОС UNIX, то реализация его в последних версиях сетевых операционных систем для персональных компьютеров (Windows NT, NetWare ) является хорошей предпосылкой для быстрого роста числа установок стека TCP/IP.

Стек был разработан по инициативе Министерства обороны США (Department of Defence, DoD) более 20 лет назад для связи экспериментальной сети ARPAnet с другими сателлитными сетями как набор общих протоколов для разнородной

вычислительной среды. Сеть ARPA поддерживала разработчиков и исследователей в военных областях. В сети ARPA связь между двумя компьютерами осуществлялась с использованием протокола Internet Protocol (IP), который и по сей день является одним из основных в стеке TCP/IP и фигурирует в названии стека.

Большой вклад в развитие стека TCP/IP внес университет Беркли, реализовав протоколы стека в своей версии ОС UNIX. Широкое распространение ОС UNIX привело и к широкому распространению протокола IP и других протоколов стека. На этом же стеке работает всемирная информационная сеть Internet, чье подразделение Internet Engineering Task Force (IETF) вносит основной вклад в совершенствование стандартов стека, публикуемых в форме спецификаций RFC.

Так как стек TCP/IP был разработан до появления модели взаимодействия открытых систем ISO/OSI, то, хотя он также имеет многоуровневую структуру, соответствие уровней стека TCP/IP уровням модели OSI достаточно условно.

Структура протоколов TCP/IP приведена на рисунке 4. Протоколы TCP/IP делятся на 4 уровня.

Самый нижний (уровень IV) - уровень межсетевых интерфейсов - соответствует физическому и канальному уровням модели OSI. Этот уровень в протоколах TCP/IP не регламентируется, но поддерживает все популярные стандарты физического и канального уровня: для локальных каналов это Ethernet, Token Ring, FDDI, для глобальных каналов - собственные протоколы работы на аналоговых коммутируемых и выделенных линиях SLIP/PPP, которые устанавливают соединения типа "точка - точка" через последовательные каналы глобальных сетей, и протоколы территориальных сетей X.25 и ISDN. Разработана также специальная спецификация, определяющая использование технологии ATM в качестве транспорта канального уровня.

Следующий уровень (уровень III) - это уровень межсетевого взаимодействия, который занимается передачей дейтаграмм с использованием различных локальных сетей, территориальных сетей X.25, линий специальной связи и т. п. В качестве основного протокола сетевого уровня (в терминах модели OSI) в стеке используется протокол **IP**, который изначально проектировался как протокол передачи пакетов в составных сетях, состоящих из большого количества локальных сетей, объединенных как локальными, так и глобальными связями. Поэтому протокол IP хорошо работает в сетях со сложной топологией, рационально используя наличие в них подсистем и экономно расходуя пропускную способность низкоскоростных линий связи. Протокол IP является дейтаграммным протоколом.

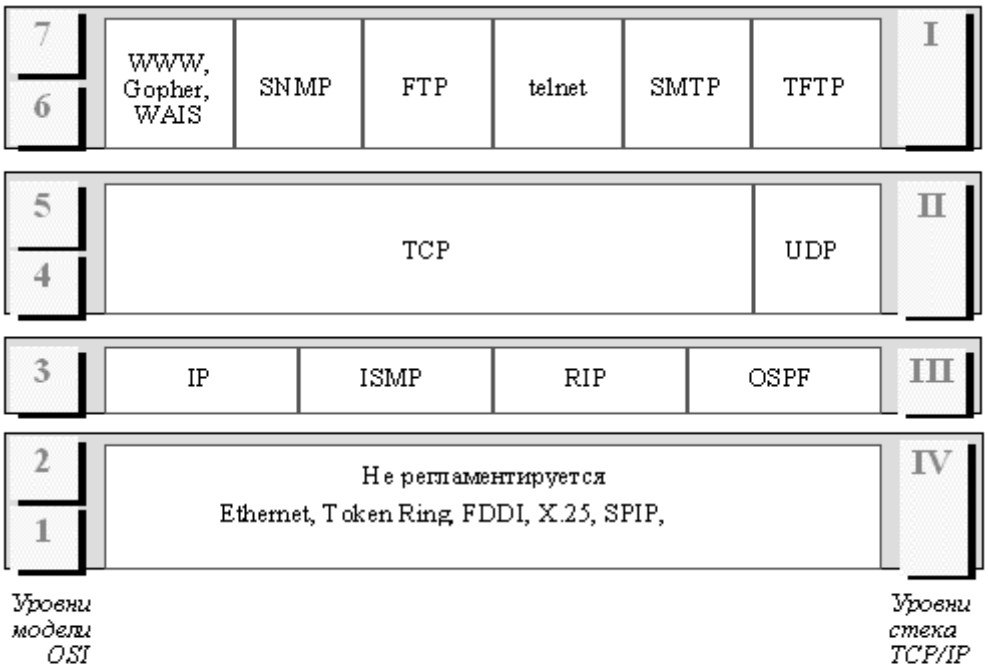



Рис.4. Стек TCP / IP

К уровню межсетевого взаимодействия относятся и все протоколы, связанные с составлением и модификацией таблиц маршрутизации, такие как протоколы сбора маршрутной информации **RIP** (Routing Internet Protocol) и **OSPF** (Open Shortest Path First), а также протокол межсетевых управляющих сообщений **ICMP** (Internet Control Message Protocol). Последний протокол предназначен для обмена информацией об ошибках между маршрутизатором и шлюзом, системой-источником и системой-приемником, то есть для организации обратной связи. С помощью специальных пакетов ICMP сообщается о невозможности доставки пакета, о превышении времени жизни или продолжительности сборки пакета из фрагментов, об аномальных величинах параметров, об изменении маршрута пересылки и типа обслуживания, о состоянии системы и т.п.

Следующий уровень (уровень II) называется основным. На этом уровне функционируют протокол управления передачей **TCP**(Transmission Control Protocol) и протокол дейтаграмм пользователя **UDP** (User Datagram Protocol). Протокол TCP обеспечивает устойчивое виртуальное соединение между удаленными прикладными процессами. Протокол UDP обеспечивает передачу прикладных пакетов дейтаграммным методом, то есть без установления виртуального соединения, и поэтому требует меньших накладных расходов, чем TCP.

Верхний уровень (уровень I) называется прикладным. За долгие годы использования в сетях различных стран и организаций стек TCP/IP накопил большое количество протоколов и сервисов прикладного уровня. К ним относятся такие широко используемые протоколы, как протокол копирования файлов FTP, протокол эмуляции терминала telnet, почтовый протокол SMTP, используемый в электронной почте сети Internet и ее российской ветви РЕЛКОМ, гипертекстовые сервисы доступа к удаленной информации, такие как WWW и многие другие.

Остановимся несколько подробнее на некоторых из них, наиболее тесно связанных с тематикой данного курса.

Протокол **SNMP** (Simple Network Management Protocol) используется для организации сетевого управления. Проблема управления разделяется здесь на две задачи. Первая задача связана с передачей информации. Протоколы передачи управляющей информации определяют процедуру взаимодействия сервера с программой-клиентом, работающей на хосте администратора. Они определяют форматы сообщений, которыми обмениваются клиенты и серверы, а также форматы имен и адресов. Вторая задача связана с контролируемыми данными. Стандарты регламентируют, какие данные должны сохраняться и накапливаться в шлюзах, имена этих данных и синтаксис этих имен. В стандарте SNMP определена спецификация информационной базы данных управления сетью. Эта спецификация, известная как база данных MIB (Management Information Base), определяет те элементы данных, которые хост или шлюз должен сохранять, и допустимые операции над ними.

Протокол пересылки файлов **FTP** (File Transfer Protocol) реализует удаленный доступ к файлу. Для того, чтобы обеспечить надежную передачу, FTP использует в качестве транспорта протокол с установлением соединений - TCP. Кроме пересылки файлов протокол, FTP предлагает и другие услуги. Так пользователю предоставляется возможность интерактивной работы с удаленной машиной, например, он может распечатать содержимое ее каталогов, FTP позволяет пользователю указывать тип и формат запоминаемых данных. Наконец, FTP выполняет аутентификацию пользователей. Прежде, чем получить доступ к файлу, в соответствии с протоколом пользователи должны сообщить свое имя и пароль.

В стеке TCP/IP протокол FTP предлагает наиболее широкий набор услуг для работы с файлами, однако он является и самым сложным для программирования. Приложения, которым не требуются все возможности FTP, могут использовать другой, более экономичный протокол - простейший протокол пересылки файлов **TFTP** (Trivial File Transfer Protocol). Этот протокол реализует только передачу файлов, причем в качестве транспорта используется более простой, чем TCP, протокол без установления соединения - UDP.

Протокол **telnet** обеспечивает передачу потока байтов между процессами, а также между процессом и терминалом. Наиболее часто этот протокол используется для эмуляции терминала удаленной ЭВМ.

### 3.3 Стек IPX/SPX

Этот стек является оригинальным стеком протоколов фирмы Novell, который она разработала для своей сетевой операционной системы NetWare еще в начале 80-х годов. Протоколы Internetwork Packet Exchange (IPX) и Sequenced Packet Exchange (SPX), которые дали имя стеку, являются прямой адаптацией протоколов XNS фирмы Xerox, распространенных в гораздо меньше степени, чем IPX/SPX. По количеству установок протоколы IPX/SPX лидируют, и это обусловлено тем, что сама ОС NetWare занимает лидирующее положение с долей установок в мировом масштабе примерно в 65%.

Семейство протоколов фирмы Novell и их соответствие модели ISO/OSI представлено на рисунке 5.

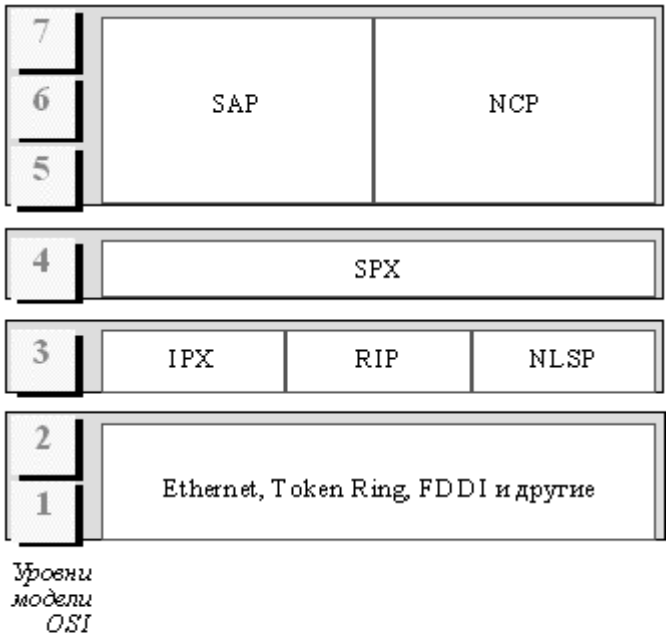



Рис.5. Стек IPX / SPX

На физическом и канальном уровнях в сетях Novell используются все популярные протоколы этих уровней (Ethernet, Token Ring, FDDI и другие).

На сетевом уровне в стеке Novell работает протокол **IPX**, а также протоколы обмена маршрутной информацией **RIP** и **NLSP** (аналог протокола OSPF стека TCP/IP). IPX является протоколом, который занимается вопросами адресации и маршрутизации пакетов в сетях Novell. Маршрутные решения IPX основаны на адресных полях в заголовке его пакета, а также на информации, поступающей от протоколов обмена маршрутной информацией. Например, IPX использует информацию, поставляемую либо протоколом RIP, либо протоколом NLSP (NetWare Link State Protocol) для передачи пакетов компьютеру назначения или следующему маршрутизатору. Протокол IPX поддерживает только дейтаграммный способ обмена сообщениями, за счет чего экономно потребляет вычислительные ресурсы. Итак, протокол IPX обеспечивает выполнение трех функций: задание адреса, установление маршрута и рассылку дейтаграмм.

Транспортному уровню модели OSI в стеке Novell соответствует протокол SPX, который осуществляет передачу сообщений с установлением соединений.

На верхних прикладном, представительном и сеансовом уровнях работают протоколы NCP и SAP. Протокол **NCP** (NetWare Core Protocol) является протоколом взаимодействия сервера NetWare и оболочки рабочей станции. Этот протокол прикладного уровня реализует архитектуру клиент-сервер на верхних уровнях модели OSI. С помощью функций этого протокола рабочая станция производит подключение к серверу, отображает каталоги сервера на локальные буквы дисководов, просматривает файловую систему сервера, копирует удаленные

файлы, изменяет их атрибуты и т.п., а также осуществляет разделение сетевого принтера между рабочими станциями.

**SAP** (Service Advertising Protocol) - протокол объявления о сервисе - концептуально подобен протоколу RIP. Подобно тому, как протокол RIP позволяет маршрутизаторам обмениваться маршрутной информацией, протокол SAP дает возможность сетевым устройствам обмениваться информацией об имеющихся сетевых сервисах.

Серверы и маршрутизаторы используют SAP для объявления о своих сервисных услугах и сетевых адресах. Протокол SAP позволяет сетевым устройствам постоянно корректировать данные о том, какие сервисные услуги имеются сейчас в сети. При старте серверы используют SAP для оповещения оставшейся части сети о своих услугах. Когда сервер завершает работу, то он использует SAP для того, чтобы известить сеть о прекращении действия своих услуг.

В сетях Novell серверы NetWare 3.x каждую минуту рассылают широковещательные пакеты SAP. Пакеты SAP в значительной степени засоряют сеть, поэтому одной из основных задач маршрутизаторов, выходящих на глобальные связи, является фильтрация трафика SAP-пакетов и RIP-пакетов.

Особенности стека IPX/SPX обусловлены особенностями ОС NetWare, а именно ориентацией ее ранних версий (до 4.0) на работу в локальных сетях небольших размеров, состоящих из персональных компьютеров со скромными ресурсами. Поэтому Novell нужны были протоколы, на реализацию которых требовалось минимальное количество оперативной памяти (ограниченной в IBM-совместимых компьютерах под управлением MS-DOS 640 Кбайтами) и которые бы быстро работали на процессорах небольшой вычислительной мощности. В результате, протоколы стека IPX/SPX до недавнего времени хорошо работали в локальных сетях и не очень - в больших корпоративных сетях, так как слишком перегружали медленные глобальные связи широковещательными пакетами, которые интенсивно используются несколькими протоколами этого стека (например, для установления связи между клиентами и серверами).

Это обстоятельство, а также тот факт, что стек IPX/SPX является собственностью фирмы Novell и на его реализацию нужно получать у нее лицензию, долгое время ограничивали распространенность его только сетями NetWare. Однако к моменту выпуска версии NetWare 4.0, Novell внесла и продолжает вносить в свои протоколы серьезные изменения, направленные на приспособление их для работы в корпоративных сетях. Сейчас стек IPX/SPX реализован не только в NetWare, но и в нескольких других популярных сетевых ОС - SCO UNIX, Sun Solaris, Microsoft Windows NT

## 4 Протоколы компьютерных сетей

Сетевым протоколом называется набор правил, позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть компьютерами.Фактически разные протоколы зачастую описывают лишь разные стороны одного типа связи; взятые вместе, они образуют так называемый стек

протоколов. Названия <протокол> и <стек протоколов> также указывают на программное обеспечение, которым реализуется протокол

### 4.1 Уровни протоколов

Наиболее распространённой системой классификации сетевых протоколов является так называемая модель OSI. В соответствии с ней протоколы делятся на 7 уровней по своему назначению - от физического (формирование и распознавание электрических или других сигналов) до прикладного (API для передачи информации приложениями):

- **Прикладной уровень (Application layer).** Верхний (7-й) уровень модели, обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя доступ к сетевым службам, таким как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты. Также отвечает за передачу служебной информации, предоставляет приложениям информацию об ошибках и формирует запросы к уровню представления. Пример: HTTP, POP3, SMTP.
- **Уровень представления (Presentation layer).** 6-й уровень отвечает за преобразование протоколов и кодирование/декодирование данных. Запросы приложений, полученные с уровня приложений, он преобразует в формат для передачи по сети, а полученные из сети данные преобразует в формат, понятный приложениям. На уровне представления может осуществляться сжатие/распаковка или кодирование/декодирование данных, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально.
- **Сеансовый уровень (Session layer).** 5-й уровень модели отвечает за поддержание сеанса связи, что позволяет приложениям взаимодействовать между собой длительное время. Сеансовый уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений. Синхронизация передачи обеспечивается помещением в поток данных контрольных точек, начиная с которых возобновляется процесс при нарушении взаимодействия.
- **Транспортный уровень (Transport layer).** 4-й уровень модели, предназначен для доставки данных без ошибок, потерь и дублирования в той последовательности, как они были переданы. При этом неважно, какие данные передаются, откуда и куда, то есть он предоставляет сам механизм передачи. Блоки данных он разделяет на фрагменты, размер которых зависит от протокола, короткие объединяет в один, а длинные разбивает. Протоколы этого уровня предназначены для взаимодействия типа точка-точка. Пример: TCP, UDP
- **Сетевой уровень (Network layer).** 3-й уровень сетевой модели OSI, предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и заторов в сети. На этом уровне работает такое сетевое устройство, как маршрутизатор.

- **Уровень звена данных (Data Link layer).** Часто это уровень называется канальным. Этот уровень предназначен для обеспечения взаимодействия сетей на физическом уровне и контроля за ошибками, которые могут возникнуть. Данные, полученные с физического уровня, он упаковывает во фреймы, проверяет на целостность, если нужно исправляет ошибки и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием. Спецификация IEEE 802 разделяет этот уровень на 2 подуровня - MAC (Media Access Control) регулирует доступ к разделяемой физической среде, LLC (Logical Link Control) обеспечивает обслуживание сетевого уровня. На этом уровне работают коммутаторы, мосты. В программировании этот уровень представляет драйвер сетевой платы, в операционных системах имеется программный интерфейс взаимодействия канального и сетевого уровней между собой, это не новый уровень, а просто реализация модели для конкретной ОС. Примеры таких интерфейсов: ODI, NDIS
- **Физический уровень (Physical layer).** Самый нижний уровень модели, предназначен непосредственно для передачи потока данных. Осуществляет передачу электрических или оптических сигналов в кабель или в радиоэфир и соответственно их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов. Другими словами, осуществляет интерфейс между сетевым носителем и сетевым устройством. На этом уровне работают концентраторы (хабы), повторители (ретрансляторы) сигнала и медиаконверторы. Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом.

В основном используются протокол TCP/IP

Определение:

*Transmission Control Protocol/Internet Protocol, TCP/IP* (Протокол управления передачей/Протокол Интернета)

Большинство операционных систем сетевых серверов и рабочих станций поддерживает TCP/IP, в том числе серверы NetWare, все системы Windows, UNIX, последние версии Mac OS, системы OpenMVS и z/OS компании IBM, а также OpenVMS компании DEC. Кроме того, производители сетевого оборудования создают собственное системное программное обеспечение для TCP/IP, включая средства повышения производительности устройств. Стек TCP/IP изначально применялся на UNIX-системах, а затем быстро распространился на многие другие типы сетей.

## 4.2 Протоколы локальных сетей

**Протоколы локальных сетей, испоьзуемые в настоящее время следующие:**

- IPX/SPX;
- NetBEUI;
- AppleTalk;
- TCP/IP;
- SNA;
- DLC;
- DNA;

## 4.3 Свойства протоколов локальной сети

В основном протоколы локальных сетей имеют такие же свойства, как и Другие коммуникационные протоколы, однако некоторые из них были разработаны давно, при создании первых сетей, которые работали медленно, были ненадежными и более подверженными электромагнитным и радиопомехам. Поэтому для современных коммуникаций некоторые протоколы не вполне пригодны. К недостаткам таких протоколов относится слабая защита от ошибок или избыточный сетевой трафик. Кроме того, определенные протоколы были созданы для небольших локальных сетей и задолго до появления современных корпоративных сетей с развитыми средствами маршрутизации.

Протоколы локальных сетей должны иметь следующие основные характеристики:

- обеспечивать надежность сетевых каналов;
- обладать высоким быстродействием;
- обрабатывать исходные и целевые адреса узлов;
- соответствовать сетевым стандартам, в особенности - стандарту IEEE 802.

В основном все протоколы, рассматриваемые в этой главе, соответствуют перечисленным требованиям, однако, как вы узнаете позднее, у одних протоколов возможностей больше, чем у других.

В таблице перечислены протоколы локальных сетей и операционные системы, с которыми эти протоколы могут работать. Далее в главе указаны протоколы и системы (в частности, операционные системы серверов и хост компьютеров) будут описаны подробнее.

***Таблица 4.1*** *Протоколы локальных сетей и сетевые операционные системы*

| Протокол | Соответствующая операционная система |
|---|---|
| IPX/SPX | Novell NetWare |
| NetBEUI | Первые версии операционных систем Microsoft Windows |
| AppleTalk | Apple Macintosh |
| TCP/IP | UNIX, Novel NetWare, современные версии операционных систем Microsoft Windows, операционные системы мэйнфреймов IBM |
| SNA | Операционные системы мэйнфреймов и миникомпьютеров IBM |
| DLC | Клиентские системы, взаимодействующие с мэйнфреймами IBM, настроенными на работу с протоколом SNA |

## 4.4 Понятие протокола Интернет

Очевидно, что рано или поздно компьютеры, расположенные в разных точках земного шара, по мере увеличения своего количества должны были обрести некие средства общения. Такими средствами стали компьютерные сети. Сети бывают локальными и глобальными. Локальная сеть - это сеть, объединяющая компьютеры, географически расположенные на небольшом расстоянии друг от друга - например, в одном здании. Глобальные сети служат для соединения сетей и компьютеров, которых разделяют большие расстояния - в сотни и тысячи километров. Интернет относится к классу глобальных сетей.

Простое подключение одного компьютера к другому - шаг, необходимый для создания сети, но не достаточный. Чтобы начать передавать информацию, нужно убедиться, что компьютеры "понимают" друг друга. Как же компьютеры "общаются" по сети? Чтобы обеспечить эту возможность, были разработаны специальные средства, получившие название "протоколы". Протокол - это совокупность правил, в соответствии с которыми происходит передача информации через сеть. Понятие протокола применимо не только к компьютерной индустрии. Даже те, кто никогда не имел дела с Интернетом, скорее всего работали в повседневной жизни с какими-либо устройствами, функционирование которых основано на использовании протоколов. Так, обычная телефонная сеть общего пользования тоже имеет свой протокол, который позволяет аппаратам, например, устанавливать факт снятия трубки на другом конце линии или распознавать сигнал о разъединении и даже номер звонящего.

Исходя из этой естественной необходимости, миру компьютеров потребовался единый язык (то есть протокол), который был бы понятен каждому из них.

Основные протоколы используемые в работе Интернет:

**TCP/IP, POP3, SMTP, FTP, HTTP**

**IMAP4, WAIS, Gorpher, WAP**

## 4.5 Краткое описание протоколов Интернет

### TCP/IP

Над созданием протоколов, необходимых для существования глобальной сети, трудились лучшие умы человечества. Одним из них был Винтон Серф (Vinton G. Cerf). Сейчас этого человека называют "отцом Интернета". В 1997 году Президент США Билл Клинтон наградил Винтона Серфа и его коллегу Роберта Кана (Robert E. Kahn) Национальной медалью за заслуги в области технологии, отметив их вклад в становление и развитие Интернета. Ныне Винтон Серф занимает пост старшего вице-президента по Интернет-архитектуре в корпорации MCI WorldCom Inc.

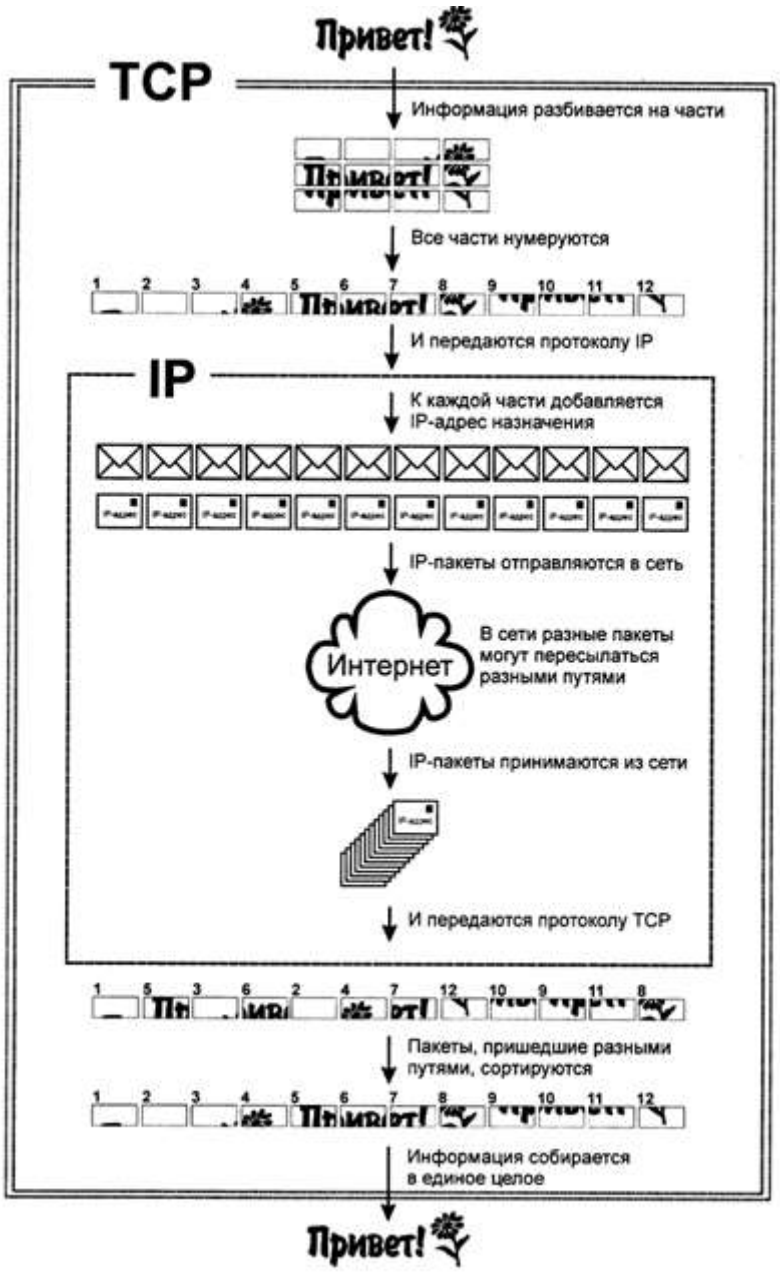
В 1972 году группа разработчиков под руководством Винтона Серфа разработала протокол TCP/IP - Transmission Control Protocol/Internet Protocol (Протокол управления передачей/Протокол Интернета).

Эксперимент по разработке этого протокола проводился по заказу Министерства обороны США. Данный проект получил название ARPANet (Advanced Research Projects Agency Network - Сеть агентства важных исследовательских проектов). Очевидно, что в обстановке войны, когда необходимость в обмене информацией встает как никогда остро, возникает проблема непредсказуемости состояния пути, по которому будет передана та или иная информация - любой из узлов передачи в любой момент может быть выведен из строя противником. Поэтому главной задачей при разработке сетевого протокола являлась его "неприхотливость" - он должен был работать с любым сетевым окружением и, кроме того, обладать гибкостью в выборе маршрута при доставке информации.

Позже TCP/IP перерос свое изначальное предназначение и стал основой стремительно развивавшейся глобальной сети, ныне известной как Интернет, а также небольших сетей, использующих технологии Интернета - интранет. Стандарты TCP/IP являются открытыми и непрерывно совершенствуются.

На самом деле TCP/IP является не одним протоколом, а целым набором протоколов, работающих совместно. Он состоит из двух уровней. Протокол верхнего уровня, TCP, отвечает за правильность преобразования сообщений в пакеты информации, из которых на приемной стороне собирается исходное послание. Протокол нижнего уровня, IP, отвечает за правильность доставки сообщений по указанному адресу. Иногда пакеты одного сообщения могут доставляться разными путями.

Схема функционирования протокола TCP/IP:




**HTTP**

Протокол HTTP (Hypertext Transfer Protocol - Протокол передачи гипертекста) является протоколом более высокого уровня по отношению к протоколу TCP/IP - протоколом уровня приложения. HTTP был разработан для эффективной передачи по Интернету Web-страниц. Именно благодаря HTTP мы имеем возможность созерцать страницы Сети во всем великолепии. Протокол HTTP является основой системы World Wide Web.

Вы отдаете команды HTTP, используя интерфейс броузера, который является HTTP-клиентом. При щелчке мышью на ссылке броузер запрашивает у Web-сервера данные того ресурса, на который указывает ссылка - например, очередной Web-страницы.

Чтобы текст, составляющий содержимое Web-страниц, отображался на них определенным образом - в соответствии с замыслом создателя страницы - он размечается с помощью особых текстовых меток - тегов языка разметки гипертекста (HyperText Markup Language, HTML).

Адреса ресурсов Интернета, к которым вы обращаетесь по протоколу HTTP, выглядит примерно следующим образом: http://www.tut.by

**FTP**

Протокол FTP (File Transfer Protocol - Протокол передачи файлов) специально разработан для передачи файлов по Интернету. Позже мы поговорим о нем подробно. Сейчас скажем лишь о том, что адрес FTP-ресурса в Интернете выглядит следующим образом: ftp://ftp.netscape.com

**TELNET**

С помощью этого протокола вы можете подключиться к удаленному компьютеру как пользователь (если наделены соответствующими правами, то есть знаете имя пользователя и пароль) и производить действия над его файлами и приложениями точно так же, как если бы работали на своем компьютере.

Telnet является протоколом эмуляции терминала. Работа с ним ведется из командной строки. Если вам нужно воспользоваться услугами этого протокола, не стоит рыскать по дебрям Интернета в поисках подходящей программы. Telnet-клиент поставляется, например, в комплекте Windows 98.

Чтобы дать команду клиенту Telnet соединиться с удаленным компьютером, подключитесь к Интернету, выберите в меню Пуск (Start) команду Выполнить (Run) и наберите в строке ввода, например, следующее: telnet lib.ru

(Вместо lib.ru вы, разумеется, можете ввести другой адрес.) После этого запустится программа Telnet, и начнется сеанс связи.

## 4.6 Список сетевых протоколов

Здесь представлен неполный список сетевых протоколов, отсортированных по ближайшим им уровням модели Open Systems Interconnection (OSI). Однако, многие из данных протоколов изначально основаны на стеке протоколов TCP/IP и прочих моделях, поэтому они не могут быть однозначно соотнесены с уровнями модели OSI.

### 4.6.1 Протоколы уровня 1 (Физический уровень)

- ADSL Asymmetric Digital Subscriber Line
- ISDN Integrated Services Digital Network
- PDH Плезиохронная цифровая иерархия
  - Т-канал (T1, T3, etc.)
  - Е-канал (E1, E3, etc.)
- RS-232, a serial line interface originally developed to connect modems and computer terminals
- SDH Синхронная Цифровая Иерархия
- SONET Synchronous Optical NETworking

- Стандартные модемные протоколы/Протоколы серии ITU V, используемые в соединениях между аналоговыми модемами по телефонной линии.
- Физический уровень CCITT G.hn

### 4.6.2 Протоколы уровня 1+2

- Ethernet
- GFP ITU-T G.7041 Обобщенная процедура разбиения на фреймы
- OTN ITU-T G.709 Оптическая передача данных по сети, также называемая «Оптической оболочкой канала» (Optical Channel Wrapper) или «Цифровой технологией обертывания» (Digital Wrapper Technology)

### 4.6.3 Протоколы уровня 2 (Канальный уровень)

- ARCNET Attached Resource Computer NETwork
- CDP Протокол обнаружения Cisco
- DCAP Data Link Switching Client Access Protocol
- Dynamic Trunking Protocol Динамический протокол группобразования
- Econet
- FDDI Волоконно-оптический интерфейс по распределенным данным
- Frame Relay
- Канальный уровень CCITT G.hn
- HDLC High-Level Data Link Control
- IEEE 802.11 WiFi
- IEEE 802.16 WiMAX
- LocalTalk
- L2F Layer 2 Forwarding Protocol
- L2TP Layer 2 Tunneling Protocol
- LAPD Процедуры доступа к соединению на D-канале
- LLDP Link Layer Discovery Protocol
- LLDP-MED Link Layer Discovery Protocol — Media Endpoint Discovery
- PPP Point-to-Point Protocol
- PPTP Point-to-Point Tunneling Protocol
- Q.710 Упрощенный Message Transfer Part
- NDP Протокол обнаружения соседей
- RPR IEEE 802.17 Resilient Packet Ring
- Shortest Path Bridging
- SLIP Serial Line Internet Protocol (устарел)
- StarLAN
- STP Spanning Tree Protocol
- Token ring — по сути является не протоколом, а топологией
- VTP VLAN Trunking Protocol

### 4.6.4 Протоколы уровня 2+3

- ATM Asynchronous Transfer Mode
- Frame relay, упрощенная версия X.25
- MPLS Multi-protocol label switching
- X.25
- ARP Протокол разрешения адреса
- RARP Протокол разрешения обратного адреса

Протоколы уровня 1+2+3

---

- MTP Message Transfer Part
- NSP Network Service Part

### 4.6.5 Протоколы уровня 3 (Сетевой уровень)

- CLNP Сетевой протокол без установки соединения
- EGP протокол внешнего шлюза (устарел)
- EIGRP Enhanced Interior Gateway Routing Protocol
- ICMP Internet Control Message Protocol
- IGMP Internet Group Management Protocol
- IGRP Interior Gateway Routing Protocol
- IPv4 Internet Protocol version 4
- IPv6 Internet Protocol version 6
- IPsec Internet Protocol Security
- IPX Internetwork Packet Exchange
- SCCP Signalling Connection Control Part
- AppleTalk DDP

### 4.6.6 Протоколы уровня 3 (управление на сетевом уровне)

- IS-IS Intermediate System-to-Intermediate System
- OSPF Open Shortest Path First
- BGP Border Gateway Protocol
- RIP Routing Information Protocol
- IRDP: Реализация RFC 1256
- Gateway Discovery Protocol (GDP) — протокол, разработанный Cisco, схожий с IRDP

Протоколы уровня 3.5

---

- HIP Протокол идентификации хоста

Протоколы уровня 3+4

---

- Xerox Network Systems

### 4.6.7 Протоколы уровня 4 (Транспортный уровень)

- AH Аутентификационный заголовок по IP или IPSec
- ESP Encapsulating Security Payload over IP or IPSec
- GRE Generic Routing Encapsulation для туннелирования
- IL Первоначально разработан как транспортный уровень для 9P
- SCTP Stream Control Transmission Protocol
- Sinec H1 для удаленного контроля
- SPX Sequenced Packet Exchange
- TCP Transmission Control Protocol
- UDP User Datagram Protocol

### 4.6.8 Протоколы уровня 5 (Сеансовый уровень)

- 9P — протокол распределенной файловой системы, разработанный как часть Plan 9
- NCP NetWare Core Protocol
- NFS — Сетевая файловая система
- SMB Server Message Block
- SOCKS «SOCKetS»

### 4.6.9 Прочие протоколы

- Controller Area Network (CAN)
- Общепромышленный протокол (CIP)
- Цифровое управление командами (DCC)
- Financial Information eXchange (FIX)
- I²C
- Modbus
- DECnet — семейство протоколов от Digital Equipment Corporation (ныне HP)
- Service Location Protocol SLP
- Service Advertising Protocol SAP

### 4.6.10 Протоколы уровня 7 (Прикладной уровень)

- ADC — peer-to-peer-протокол обмена файлами
- AFP, Apple Filing Protocol
- BACnet, Building Automation and Control Network protocol
- BitTorrent — peer-to-peer-протокол обмена файлами
- BOOTP, Bootstrap Protocol
- DIAMETER — протокол аутентификации, авторизации и работы с аккаунтами
- DICOM содержит определение сетевого протокола
- DICT — словарный протокол
- DNS — система доменных имен
- DHCP, Dynamic Host Configuration Protocol
- ED2K — peer-to-peer-протокол обмена файлами
- FTP — протокол передачи файлов
- Finger — протокол, возвращающий информацию о пользователях на удаленном компьютере
- Gnutella — peer-to-peer-протокол скачивания файлов
- Gopher — иерархический протокол на основе гиперссылок
- HTTP, Hypertext Transfer Protocol
- IMAP, Internet Message Access Protocol
- IRC — протокол для чата
- ISUP, ISDN User Part
- XMPP — протокол мгновенного обмена сообщениями
- LDAP Lightweight Directory Access Protocol

- MIME, Multipurpose Internet Mail Extensions
- MSNP, Microsoft Notification Protocol (используется в Windows Live Messenger)
- MAP, Mobile Application Part
- NetBIOS — протокол общего пользования файлами и разрешения имен — основа обмена файлами в Windows.
- NNTP — сетевой протокол передачи новостей
- NTP — сетевой протокол времени
- NTCIP, National Transportation Communications for Intelligent Transportation System Protocol
- POP3 — почтовый протокол версии 3
- RADIUS — протокол аутентификации, авторизации и работы с аккаунтами
- Rlogin — протокол удаленного входа в UNIX
- rsync — протокол передачи файлов для резервного копирования, копирования и зеркалирования
- RTP, Real-time Transport Protocol
- RTSP, Real-time Transport Streaming Protocol
- SSH, Secure Shell
- SISNAPI, Siebel Internet Session Network API
- SIP, Session Initiation Protocol, сигнальный протокол
- SMTP, Simple Mail Transfer Protocol
- SNMP, Simple Network Management Protocol
- SOAP, Simple Object Access Protocol
- STUN, Session Traversal Utilities for NAT
- TUP, Telephone User Part
- Telnet — протокол удаленного доступа к терминалу
- TCAP, Transaction Capabilities Application Part
- TFTP, Trivial File Transfer Protocol, простой протокол передачи файлов
- WebDAV, Web Distributed Authoring and Versioning
- DSM CC Digital Storage Media Command and Control
  - **UDP** — Название: User Datagram Protocol Уровень (по модели OSI): Транспортный Семейство: TCP/IP (иногда называют UDP/IP) Порт/ID: 17 (в IP) Спецификация: RFC 768 / STD 6 Основ
  - **TCP/IP** — Стек протоколов TCP/IP (англ. Transmission Control Protocol/Internet Protocol) набор сетевых протоколов разных уровней модели сетевого взаимодействия DOD, используемых в сетях. Протоколы работают друг с другом в стеке (англ. stack, стопка)
  - **TCP** — Название: Transport Control Protocol Уровень (по модели OSI): Транспортный Семейство: TCP/IP Порт/ID: 6/IP Спецификация: RFC 793 / STD 7 Основные реализации
  - **UDP-пакет** — UDP Название: User Datagram Protocol Уровень (по модели OSI): Транспортный Семейство: TCP/IP (иногда называют UDP/IP) Порт/ID: 17 (в IP) Спецификация: RFC 768 / STD 6 Основные реализации (клиенты): Ядро Windows, Linux, UNIX
  - **Udp** — Название: User Datagram Protocol Уровень (по модели OSI): Транспортный Семейство: TCP/IP (иногда называют UDP/IP) Порт/ID: 17 (в

IP) Спецификация: RFC 768 / STD 6 Основные реализации (клиенты): Ядро Windows, Linux, UNIX .

- **UDP Lite** — (облегченный UDP) протокол без установки соединения, весьма похожий на UDP. В отличие от UDP, в котором защищены контрольной суммой (checksum) или все пакеты или ни один из них, UDP Lite допускает возможность частичных контрольных сумм.
- **Список UNIX-демонов** — Список UNIX демонов, которые могут быть в различных модификациях UNIX. UNIX демоны обычно включают окончание d как аббревиатуру от англ. daemon; например, клиентская программа называется telnet , а отвечающий ей с серверной стороны демон.
- **TCP-порт** — Сетевой порт параметр протоколов UDP, определяющий назначение пакетов данных в формате Это условное число от 0 до 65535, позволяющие различным программам, выполняемым на одном хосте, получать данные независимо друг от друга (предоставляют так.
- **UDP-порт** — Сетевой порт параметр протоколов UDP, определяющий назначение пакетов данных в формате Это условное число от 0 до 65535, позволяющие различным программам, выполняемым на одном хосте, получать данные независимо друг от друга (предоставляют так… … *Википедия*
- **UDP/IP** — Стек протоколов TCP/IP (англ. Transmission Control Protocol/Internet Protocol) собирательное название для сетевых протоколов разных уровней, используемых в сетях. Слово «стек» (англ. stack, стопка) подразумевает, что протокол IP.

Основные протоколы TCP/IP по уровням модели OSI(**Список портов TCP и UDP**)
Физический
Ethernet • RS-232 • EIA-422 • RS-449 • RS-485
Канальный
Ethernet • PPPoE • PPP • L2F • 802.11 Wi-Fi • 802.16 WiMax • Token ring • ARCNET •FDDI • HDLC • SLIP • ATM • CAN • DTM • X.25 • Frame relay • SMDS • STP • ERPS
Сетевой
IPv4 • IPv6 • IPsec • ICMP • IGMP • ARP • RARP • RIP2 • OSPF
Транспортный
TCP • UDP • SCTP • DCCP • RDP/RUDP • RTP • GRE
Сеансовый
ADSP • H.245 • iSNS • NetBIOS • PAP • RPC • L2TP • PPTP • RTCP • SMPP • SCP • ZIP •SDP
Представления
XDR • SSL • TLS
Прикладной
BGP • HTTP • HTTPS • DHCP • IRC • SNMP • DNS • DNSSEC • NNTP • XMPP • SIP • IPP •NTP • SNTP • Электронная почта (SMTP • POP3 • IMAP4)
Передача файлов (FTP •TFTP • SFTP)
Удалённый доступ (rlogin • Telnet • SSH • RDP)
Другие прикладные OSCAR • CDDB • Multicast FTP • Multisource FTP • BitTorrent • Gnutella • Skype

***В компьютерных сетях из стека сетевых протоколов транспортного уровня чаще всего используютсяпротоколы TCP и UDP, а также другие протоколы, которые используют идентификацию структуры данных спредставлением конечных точек (хостов) в виде числовой последовательности. Такие конечные точкиназываются портами и идентифицируются согласно номерам портов. Номера портов, используемые дляконкретных специфических целей, выделяет и регистрирует IANA (Internet Assigned Numbers Authority).***

## 5 Порты для сетевых протоколов

Различают общеизвестные порты, Зарегистрированные порты и динамически выделяемые порты. Список первых двух типов можно найти в [5].

*Динамически выделяемые / приватные порты*

По умолчанию порты в эфемерном диапазоне портов не могут быть зарегистриров аны. Этипорты используются временными (короткоживущими) соединениями кли ент- сервер в определенных частныхслучаях.

Например, ядра и дистрибутивы Linux многих версий используют следующую оп цию для задания диапазона используемых портов от 32768 до 61000. В следующем системном файле:/proc/sys/net/ipv4/ip_local_port_range указывается диапазон порт ов для использования.

## 6 Языки описания в протоколах

(на примеASN.1)

### 6.1 Назначение языков

Язык описания протоколов PDL (Protocol Definition Language) предназначен для описания сообщений протоколов средствам формализованных конструкций языка с целью дальнейшей их унифицированной обработки. Язык обеспечивает работу с семейством протоколов с заголовком фиксированной длины. Данное семейство протоколов обладает следующими свойствами:

- протокол образует множество сообщений, идентифицируемых уникальным номером сообщения, именуемым типом сообщения;
- каждое сообщение состоит из полей определённого типа определённой длины;
- сообщения одного типа содержат одни и те же поля, расположенные в одном и том же порядке;
- каждое сообщение содержит одинаковый заголовок, расположенный в начале сообщения;
- заголовок содержит поле, определяющее номер сообщения;
- длина сообщений может быть как фиксированной, так и переменной;
- для протоколов с переменной длиной сообщения заголовок должен содержать поле, определяющее длину сообщения (включая заголовок).

### 6.2 Выполняемые функции

Основной областью применения языков PDL и XDL являются программные средства обработки сообщений, предполагающие универсальный подход к обработке сообщений, например:

- средства, осуществляющие перераспределение потоков данных на логическом уровне – для выделения отдельных сообщений из потока («нарезка»);
- средства, осуществляющие преобразование сообщений одного протокола в сообщения другого протокола на логическом уровне («шлюзы»);
- средства для обмена информацией с БД на основе протоколов из семейства, описанного выше;
- средства автоматической регистрации сообщений;
- графические средства для просмотра зарегистрированной информации в удобочитаемой форме (в виде сообщений);
- графические средства для построения отчётов на основе зарегистрированной информации (документирование);
- имитационные средства, обеспечивающие имитацию на уровне сообщений;
- графические средства представления информации, отображающие информацию в виде таблиц;
- другие средства и инструменты, используемые при разработке.

### 6.3 Язык описания маршрутной политики RPSL

Язык описания маршрутной политики RPSL (Routing Policy Specification Language) позволяет оператору сети специфицировать маршрутную политику на различных уровнях иерархии Интернет, например, на уровне автономных систем (AS). Так что низкоуровневые конфигурации маршрутизаторов могут генерироваться на основании спецификаций RPSL. Язык RPSL является расширяемым и не препятствует внедрению новых протоколов маршрутизации.

Язык RPSL призван заменить язык спецификации маршрутной политики, используемый в настоящее время и описанный в документах RIPE-181 [6] или RFC-1786 [7]. RIPE-81 [8] был первым языком, использованным в Интернет для спецификации маршрутных политик. В процессе использования RIPE-181 стало понятно, что некоторые политики не могут быть специфицированы и необходим более совершенный язык.

Язык RPSL был сконструирован так, чтобы описание всей политики маршрутизации записывалось в одну распределенную базу данных, улучшая целостность маршрутизации в Интернет. RPSL не является языком конфигурации маршрутизатора. Язык RPSL сформирован так, чтобы конфигурация маршрутизатора осуществлялась на основе описания маршрутной политики автономной системы (класс aut-num) в сочетании с описанием маршрутизатора (класс inet-rtr), которое предоставляет идентификатор маршрутизатора, номер автономной системы, описания интерфейсов и партнеров маршрутизатора. Эти данные используются совместно с глобальной базой данных карты автономных систем (класс as-set), и информацией об автономных системах отправителях и о маршрутных префиксах (классы route и route-set).

Язык RPSL является объектно-ориентированным, то есть, объекты содержат блоки описания политики и административную информацию. Эти объекты регистрируются в IRR (Internet Routing Registry) авторизованными организациями. О IRR смотри [1, 17, 4].

Далее рассматриваются классы, которые используются для определения различных политик и административных объектов. Класс "mntner" определяет средства, предназначенные для добавления, уничтожения и модификации набора объектов. Классы "person" и "role" описывают технический и административный персонал, с которым можно установить контакт. Автономные системы (AS) специфицируются с помощью класса "aut-num". Маршруты специфицируются посредством класса "route". Наборы объектов могут быть определены с помощью классов "as-set", "route-set", "filter-set", "peering-set" и "rtr-set". Класс "dictionary" предоставляет возможность расширения возможностей языка. Класс "inet-rtr" используется для спецификации маршрутизаторов. Многие из этих классов были определены ранее, смотри [6, 13, 16, 12, 5].

### 6.3.1 Имена, зарезервированные слова и представления RPSL

Каждый класс имеет набор атрибутов, где записывается некоторая информация об объектах класса. Атрибуты могут быть обязательными и опционными. Обязательный атрибут должен быть определен для всех объектов класса. Опционный атрибут может отсутствовать. Атрибуты могут быть одно- и многозначными. Каждый объект однозначно идентифицируется набором атрибутов класса "key".

Значение любого атрибута имеет определенный тип. Язык RPSL не чувствителен к тому, в каком регистре записаны те или иные выражения. Далее перечислены наиболее часто используемые типы атрибутов.

Многие объекты в RPSL имеют имя. <object-name> составляется из букв, чисел, а также символов подчеркивания ("_") и дефисов ("-"), первым символов всегда должна быть буква, а последним символом - буква или цифра. Следующие слова зарезервированы в RPSL, и не могут использоваться в качестве имен:

any as-any rs-any peeras

and or not

atomic from to at action accept announce except refine

networks into inbound outbound

Имена, начинающиеся с определенных префиксов зарезервированы для определенных типов объектов. Имена, начинающиеся с "as-" зарезервированы для имен автономных систем. Имена, начинающиеся с "rs-" зарезервированы для набора имен маршрутов. Имена, начинающиеся с "rtrs-" зарезервированы для набора имен маршрутизаторов. Имена, начинающиеся с "fltr-" зарезервированы для набора имен фильтров. Имена, начинающиеся с "prng-" зарезервированы для набора имен партнеров.

**<as-number>** Номер AS x представляется в виде строки "ASx". То есть автономная система 226 характеризуется с помощью AS226.

**<ipv4-address>** Адрес IPv4 характеризуется последовательностью из

четырех целых чисел, лежащих в диапазоне от 0 до 255, разделенные символом точка ".". Например, 192.148.166.48.

**<address-prefix>** Адресный префикс представляет собой IPv4-адрес, за которым следует символ косой черты "/" и без пробела целое число, лежащее в диапазоне 0-32. Примерами адресных префиксов могут служить:

192.224.128.5/32, 192.148.0.0/16, 0.0.0.0/0. Следующие адресные префиксы являются неверными: 0/0, 192.10/16, так как 0 или 192.10 не являются строками, содержащими четыре целые числа.

**<address-prefix-range>** Диапазоном адресных префиксов является адресный префикс, за которым следует опционный оператор диапазона. Операторами диапазона являются:

^- оператор значений “больше, исключая”. Он служит для выделения адресных префиксов больше указанного, но исключая пограничное значение префикса. Например, 128.9.0.0/16^- содержит все префиксы больше 128.9.0.0/16, исключая значение 128.9.0.0/16.

^+ Оператор значений “больше, включая”. Он служит для выделения адресных префиксов больше указанного, включая пограничное значение префикса. Например, 5.0.0.0/8^+ содержит все префиксы больше 5.0.0.0/8, включая 5.0.0.0/8.

**^n** где n целое, выделяет из адресного префикса все значения с длиной n. Например, 30.0.0.0/8^16 содержит все префиксы более 30.0.0.0/8, которые имеют длину 16, такие как 30.9.0.0/16.

**^n-m** где n и m целые числа, выделяет из адресного префикса все значения с длинами в интервале от n до m. Например, 30.0.0.0/8^24-32 содержит все значения из префикса 30.0.0.0/8, которые имеют длины в интервале 24-32, такие как 30.9.9.96/28.

Операторы диапазона могут применяться для наборов адресных префиксов. Например, для набора маршрутов rs-foo, rs-foo^+ (определенный ниже) содержит все специфические данные о префиксах в rs-foo.

Применение двух операторов диапазона последовательно является ошибкой (напр. 30.0.0.0/8^24-28^+ - ошибка). Однако оператор диапазона может быть применен к набору адресных префиксов, содержащий диапазоны адресных префиксов (напр. {30.0.0.0/8^24-28}^27-30 не является ошибкой). В этом случае, внешний оператор ^n-m располагается над внутренним оператором ^k-l и становится оператором ^max(n,k)-m, если m больше или равно max(n,k), в противном случае префикс удаляется из набора. Заметим, что оператор ^n эквивалентен ^n-n. Префикс/l^+ эквивалентен префиксу prefix/l^l-32. Префикс/l^- эквивалентен префиксу prefix/l^(l+1)-32. Префикс {prefix/l^n-m}^+ эквивалентен префиксу {prefix/l^n-32}, а префикс {prefix/l^n-m}^- эквивалентен префиксу {prefix/l^(n+1)-32}. Например,

{128.9.0.0/16^+}^- == {128.9.0.0/16^-}

| | |
|---|---|
| {128.9.0.0/16^-}^+ | == {128.9.0.0/16^-} |
| {128.9.0.0/16^17}^24 | == {128.9.0.0/16^24} |
| {128.9.0.0/16^20-24}^26-28 | == {128.9.0.0/16^26-28} |
| {128.9.0.0/16^20-24}^22-28 | == {128.9.0.0/16^22-28} |
| {128.9.0.0/16^20-24}^18-28 | == {128.9.0.0/16^20-28} |
| {128.9.0.0/16^20-24}^18-22 | == {128.9.0.0/16^20-22} |
| {128.9.0.0/16^20-24}^18-19 | == {} |

| | |
|---|---|
| **<date>** | Дата представляется в виде восьми десятичных цифр вида YYYYMMDD, где YYYY отображает год, MM представляет месяц (01 - 12) и DD характеризует день месяца (01 - 31). Все даты, если не определено что-то иное, задаются в стандарте UTC. Например, 07 июля 1938 представляется в виде 19380707. |
| **<email-address>** | описано в RFC-822 [10]. |
| **<dns-name>** | описано в RFC-1034 [17]. |
| **<nic-handle>** | представляет собой уникальный идентификатор, используемый при маршрутизации, присвоении адресов и т.д. для того, чтобы обеспечить однозначную ссылку на контактную информацию. Классы person и role связывают указатели NIC с действительными именами людей и контактной информацией. |
| **<free-form>** | представляет собой последовательность ASCII-символов. |
| **<X-name>** | является именем объекта типа X. То есть <mntner-name> является именем объекта mntner. |
| **<registry-name>** | является именем регистратора IRR. |

Значением атрибута может быть также список одного из этих типов. Элементы списка отделяются друг от друга запятыми. Например, "AS1, AS2, AS3, AS4". Заметим, что значение-список ортогонально многозначности. Многозначный атрибут имеет более одного значения, каждое из которых может быть, а может и не быть списком. С другой стороны однозначный атрибут может иметь значение-список. Объект RPSL текстуально представляется в виде списка пар атрибут-значение. Каждая пара атрибут-значение записывается на отдельной строке. Имя атрибута начинается с колонки 0 и завершается двоеточием. Далее следует значение атрибута. Атрибут, который имеет то же имя, что и класс объекта должен быть специфицирован раньше. Представление объекта

завершается пустой строкой. Значение атрибута может занимать несколько строк, для этого очередная строка должна начинаться с символа пробел, табулятор или знак плюс ('+'). Символ "+" для строки продолжения позволяет значениям атрибута содержать пустые строки. Порядок размещения пар атрибут-значение является существенным.

Описание атрибута может содержать комментарий. Комментарий может размещаться где угодно в определении объекта, он начинается с символа "#" и завершается первым же символом перехода на новую строку.

Целые числа могут задаваться:

1. в нотации языка программирования C (напр. 1, 12345),
2. в виде последовательности четырех одно-октетных целых чисел (в диапазоне 0-255), разделенных символом точка "." (например, 1.1.1.0, 255.255.0.0), в этом случае 4-октетное целое образуется объединением этих однооктетных целых чисел, начиная с наиболее значимых,
3. в виде двух 2-октетных целых чисел (в диапазоне 0 - 65535), разделенных символом двоеточие ":" (например, 3561:70, 3582:10), в этом случае 4-октетное целое число образуется путем объединения всех октетов в порядке их значимости.

### 6.3.2 Контактная информация

Классы mntner, person и role, а также атрибуты admin-c, tech-c, mnt-by, changed и всех классов характеризуют контактную информацию. Класс mntner специфицирует также аутентификационную информацию, необходимую для того, чтобы создать, ликвидировать или модифицировать другие объекты. Эти классы не специфицируют маршрутную политику и каждый реестр может иметь различные или дополнительные требования. В документе "Routing Policy System Security" [20] описана модель аутентификации и авторизации.

## 7 Веб-сервисы как средство интеграции приложений в WWW, использующие распределенный протокол

**Всемирная паутина является готовой платформой для создания и использования распределенных машинно-ориентированных систем на основе веб-сервисов. Веб-сервер выступает в качестве сервера приложений, к которым обращаются не конечные пользователи, а сторонние приложения. Это позволяет многократно использовать функциональные элементы, устранить дублирование кода, упростить решение задач интеграции приложений.**

Веб-служба, **веб-сервис** (англ. web-service) — это сетевая технология, обеспечивающая межпрограммное взаимодействие на основе веб-стандартов. Консорциум W3C определяет веб-сервис, как «программную систему, разработанную для поддержки интероперабельного межкомпьютерного (machine-to-machine) взаимодействия через сеть»

## 7.1 Веб-службы: концепции и протоколы

Веб-сервис идентифицируется строкой URI. Веб-сервис имеет программный интерфейс, представленный в машинно-обрабатываемом формате WSDL. Другие системы взаимодействуют с этим веб-сервисом путем обмена сообщениями протокола SOAP. В качестве транспорта для сообщений используется протокол HTTP. Описание веб-сервисов и их API могут быть найдены средствами UDDI. Концептуальная схема технологии приведена на рис. 7.1., а связь между протоколами — на рис. 7.2.


Рис.7. 1. Концепция веб-сервиса

- **SOAP** (Simple Object Access Protocol) — протокол обмена сообщениями между потребителем и поставщиком веб-сервиса;
- **WSDL** (Web Services Description Language) — язык описания внешних интерфейсов веб-службы;
- **UDDI** (Universal Discovery, Description and Integration) — универсальный интерфейс распознавания, описания и интеграции, используемый для формирования каталога веб-сервисов и доступа к нему.


Рис.7.2. Протоколы веб-сервисов

Все спецификации, используемые в технологии, основаны на XML и, соответственно, наследуют его преимущества (структурированность, гибкость и т.д.) и недостатки (громоздкость, медлительность).

## 7.2 SOAP

SOAP (изначально *Simple Object Access Protocol*, а в версии 1.2 официальная расшифровка аббревиатуры отсутствует) — простой протокол доступа к объектам (компонентам распределенной вычислительной системы), основанный на обмене структурированными сообщениями. Как любой текстовый протокол, SOAP может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTPS и др., но чаще всего SOAP используется поверх HTTP.

Все сообщения SOAP оформляются в виде структуры, называемой *конвертом* (envelop), включающей следующие элементы:

- Идентификатор сообщения (локальное имя).
- Опциональный элемент Header (заголовок):
  - Ноль или более ссылок на используемые пространства имен;
  - Ноль или более свойств, доступных в этом пространстве имен.
- Обязательный элемент Body (тело сообщения)
  - Ноль или более ссылок на используемые пространства имен;
  - Дочерние элементы тела сообщения

Развернутый список элементов сообщения SOAP приведен в схеме данных (для SOAP версии 1.2).

Пример сообщения SOAP:

<env:**Envelope** xmlns:env="*http://www.w3.org/2003/05/soap-envelope*">

<env:**Header**>

<n:alertcontrol xmlns:n="*http://example.org/alertcontrol*">

<n:priority>1</n:priority>

<n:expires>2001-06-22T14:00:00-05:00</n:expires>

</n:alertcontrol>

</env:**Header**>

<env:**Body**>

<m:alert xmlns:m="http://example.org/alert">

<m:msg>Get up at 6:30 AM</m:msg>

</m:alert>

</env:**Body**>

</env:**Envelope**>

XML-RPC: Не конкурент, а альтернатива SOAP

XML-RPC — очень простой и эффективный протокол взаимодействия веб-сервисов. Он не предназначен для решения глобальных задач, как SOAP, но широко используется во многих веб-разработках см. рис.7.3.


Рис.7.3 Схема связи для протокола XML-RPC

*Концепция XML-RPC*

XML-RPC — это "... спецификация и набор реализаций, которые позволяют программному обеспечению, работающему на разных операционных системах и в различных условиях, вызывать процедуры через Интернет. Это удаленный вызов процедуры с использованием HTTP как транспорта и XML как способа кодирования. XML-RPC разработан настолько простым, насколько это возможно для сложных структур данных, подлежащих передаче, обработке и приему". — [xmlrpc.com]

"Мы хотели, чистый, расширяемый и очень простой формат. Он должен представлять HTML-кодеру возможность заглянуть в файл, содержащий описание XML-RPC вызова, понять, что тот делает и быть в состоянии изменить его, чтоб он заработал с первой или второй попытки... Мы также хотели, чтобы это был легко реализуемый протокол, который может быть быстро адаптирован для работы в другой среде или на других операционных системах."- [xmlrpc.com].

### 7.3 WSDL

Язык описания веб-сервисов (*Web services Description Language*, WSDL) предназначен для унифицированного представления внешних интерфейсов веб-службы. Текущая версия протокола (на момент написания этой лекции) WSDL 2.0 и она имеет некоторые отличия от предыдущих версий (см.табл. 7.1 и рис. 7.4).

Таблица 7.1. Основные элементы протокола WSDL.

| **Элемент WSDL 1.1** | **Элемент WSDL 2.0** | **Краткое описание** |
|---|---|---|
| PortType | Interface | Представляет описание интерфейса веб-сервиса (список операций и их параметров). |
| Service | Service | Список системных функций |
| Binding | Binding | Специфицирует интерфейсы и задает параметры связывания с протоколом SOAP: стиль связывания (RPC/Document) и транспорт (SOAP). Эта секция доступна и для каждой из операций |
| Operation | Operation | Определяет операцию, представляемую веб-сервером. WSDL-операция — это аналог традиционным функциям и процедурам. |
| Message | не использ. | Сообщение, связанное с определенной операцией. Содержит информацию, необходимую для выполнения данной операции. Каждое сообщение может состоять из нескольких логических частей, описывающих типы данных и имена атрибутов. В версии 2.0 было исключено, т.к. была внедрена поддержка XML Schema для всех элементов. |
| Types | Types | Описание данных в соответствии с XML Schema. |

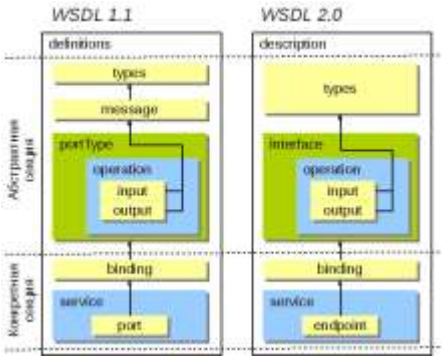



Рис. 7.4. Структура протокола WSDL

В спецификации WSDL 1.1 было определено 4 шаблона обмена сообщениями (типы операций):

- Однонаправленные операции (One-way): операция может принимать сообщение, но не будет возвращать ответ.
- Запрос-ответ (Request-response): операция может принять запрос и должна вернуть ответ.
- Вопрос-ответ (Solicit-response): операция может послать запрос и будет ждать ответ на него.
- Извещение (Notification): операция может послать сообщение, но не будет ожидать ответ.

В версии WSDL 2.0 эти шаблоны изменены и расширены в сторону поддержки сообщений об ошибках (например, шаблон Robust-in-only), но для обратной совместимости поддерживаются типы WSDL 1.1

Пример описания веб-сервиса на языке WSDL (версия 2.0).

### 7.4 UDDI

*Universal Description, Discovery and Integration* (UDDI, универсальный интерфейс распознавания, описания и интеграции) — открытый стандарт, утвержденный OASIS, определяющий методы публикации и обнаружения сетевых программных компонентов сервис-ориентированной архитектуры (SOA). В практической реализации UDDI представляет собой сетевой реестр (службу каталогов), представляющий данные и метаданные о веб-сервисах и доступный по адресу http://uddi.xml.org/services.

UDDI опирается на отраслевые стандарты HTTP, XML, XML Schema (XSD), SOAP и WSDL. Концептуальная связь между UDDI и другими протоколами стека веб-сервисов показана на рисунке 7.5.


Рис. 7.5. Место UDDI в стеке протоколов веб-служб

Функциональное назначение реестра UDDI — представление данных и метаданных о веб-службах. Он может использоваться как в сети общего пользования, так и в пределах внутренней инфраструктуры организации. Реестр UDDI предлагает основанный на стандартах механизм классификации, каталогизации и управления веб-службами, позволяющий применять их (веб-сервисы) другими приложениями. Этот механизм включает средства для поиска сервиса, вызова этой службы, а также для управления метаданными об этой службе.

Ключевыми функциями UDDI являются публикация информации о службе в реестре и поиск этой информации сторонними приложениями. Наряду с этими, реализованы и типовые для службы каталогов функции: представление модели хранимых данных и структуры информационной базы, отношения между объектами реестра, репликация, обеспечение безопасности и т.д. —Все основные функции реестра представлены в виде веб-сервисов и доступны через API UDDI.

Веб-сервисы: Pro et Contra (За и Против)

Достоинства

- Обеспечивают взаимодействие программных систем независимо от платформы.
- Основаны на базе открытых стандартов и протоколов.
- Использование HTTP позволяет приложениям взаимодействовать через межсетевой экран.

Недостатки

- Меньшая производительность и больший объем сетевого трафика по сравнению с такими технологиями как CORBA или DCOM.

# 8 Средства анализа и управления сетями

## 8.1 Стандарты систем управления программно-аппаратными средствами в сети

### 8.1.1 Стандартизуемые элементы системы управления

При формализации схемы «менеджер - агент» могут быть стандартизованы следу-ющие аспекты ее функционирования:

- протокол взаимодействия агента и менеджера;
- интерфейс «агент - управляемый ресурс»;
- интерфейс «агент - модель управляемого ресурса»;
- интерфейс «менеджер - модель управляемого ресурса»;
- справочная система о наличии и местоположении агентов и менеджеров, упрощающая построение распределенной системы управления;
- язык описания моделей управляемых ресурсов, то есть язык описания MIB;
- схема наследования классов моделей объектов (дерево наследования), которая позволяет строить модели новых объектов на основе моделей более общих объектов, например, модели маршрутизаторов на основе модели обобщенного коммуникационного устройства;
- схема иерархических отношений моделей управляемых объектов (дерево включения), которая позволяет отразить взаимоотношения между отдельными элементами реальной системы, например, принадлежность модулей коммутации определенному коммутатору или отдельных коммутаторов и концентраторов определенной подсети.

Существующие стандарты на системы управления отличаются тем, что в них может быть стандартизованы не все перечисленные выше аспекты схемы «менеджер - агент».

В стандартах систем управления как минимум стандартизуется некоторый способ формального описания моделей управляемых объектов, а также определяется протокол взаимодействия между менеджером и агентом.

Сегодня на практике применяются два семейства стандартов управления сетями - стандарты Internet, построенные на основе протокола SNMP (Simple Network

Management Protocol), и международные стандарты ISO/ITU-T, использующие в качестве протокола взаимодействия агентов и менеджеров протокол CMIP (Common Management Information Protocol).

Стандарты систем управления, основанных на протоколе SNMP, формализуют минимум аспектов системы управления, а стандарты ISO/ITU-T - максимум аспектов, как и большинство стандартов, разработанных ITU-T. Традиционно, в локальных и корпоративных сетях применяются в основном системы управления на основе SNMP, а стандарты ISO/ITU-T и протокол CMIP находят применение в телекоммуникационных сетях.

### 8.1.2 Стандарты систем управления на основе протокола SNMP

Концепции SNMP-управления

В системах управления, построенных на основе протокола SNMP, стандартизуются следующие элементы:

протокол взаимодействия агента и менеджера;

язык описания моделей MIB и сообщений SNMP - язык абстрактной синтаксической нотации ASN.1 (стандарт ISO 8824:1987, рекомендации ITU-T X.208);

несколько конкретных моделей MIB (MIB-I, MIB-II, RMON, RMON 2), имена объектов которых регистрируются в дереве стандартов ISO. Все остальное отдается на откуп разработчику системы управления. Протокол SNMP и тесно связанная с ним концепция SNMP MIB были разработаны для управления маршрутизаторами Internet как временное решение. Но, как это часто бывает со всем временным, простота и эффективность решения обеспечили успех этого протокола, и сегодня он используется при управлении практически любыми видами оборудования и программного обеспечения вычислительных сетей. И хотя в области управления телекоммуникационными сетями наблюдается устойчивая тенденция применения стандартов ITU-T, в которые входит протокол CMIP, и здесь имеется достаточно много примеров успешного использования SNMP-управления. Агенты SNMP встраиваются в аналоговые модемы, модемы ADSL, коммутаторы ATM и т. д.

SNMP - это протокол прикладного уровня, разработанный для стека TCP/IP, хотя имеются его реализации и для других стеков, например IPX/SPX. Протокол SNMP используется для получения от сетевых устройств информации об их статусе, производительности и других характеристиках, которые хранятся в базе данных управляющей информации MIB (Management Information Base). Простота SNMP во многом определяется простотой MIB SNMP, особенно их первых версий MIB I и MIB II. Кроме того, сам протокол SNMP также весьма несложен.

Существуют стандарты, определяющие структуру MIB, в том числе набор типов ее объектов, их имена и допустимые операции над этими объектами (например, считать»).

Древовидная структура MIB содержит обязательные (стандартные) поддеревья, а также в ней могут находиться частные (private) поддеревья, позволяющие изготовителю интеллектуальных устройств управлять какими-либо специфическими функциями устройства на основе специфических объектов MIB.

Агент в протоколе SNMP - это обрабатывающий элемент, который обеспечивает менеджерам, размещенным на управляющих станциях сети, доступ к значениям переменных MIB и тем самым дает им возможность реализовывать функции по управлению и наблюдению за устройством.

Основные операции по управлению вынесены в менеджер, а агент SNMP выполняет чаще всего пассивную роль, передавая в менеджер по его запросу значения накопленных статистических переменных. При этом устройство работает с минимальными издержками на поддержание управляющего протокола. Оно использует почти всю свою вычислительную мощность для выполнения своих основных функций маршрутизатора, моста или концентратора, а агент занимается сбором статис-тики и значений переменных состояния устройства и передачей их менеджеру системы управления.

## 8.2 Примитивы протокола SNMP

SNMP - это протокол типа «запрос-ответ», то есть на каждый запрос, поступивший от менеджера, агент должен передать ответ. Особенностью протокола являет-ся его чрезвычайная простота - он включает в себя всего несколько команд.

Команда Get-request используется менеджером для получения от агента значения какого-либо объекта по его имени.

Команда GetNext-request используется менеджером для извлечения значения следующего объекта (без указания его имени) при последовательном просмотре таблицы объектов.

С помощью команды Get-response агент SNMP передает менеджеру ответ на команды Get-request или GetNext-request.

Команда Set используется менеджером для изменения значения какого-либо объекта. С помощью команды Set происходит собственно управление устройством. Агент должен понимать смысл значений объекта, который используется для управления устройством, и на основании этих значений выполнять реальное управляющее воздействие - отключить порт, приписать порт определенной VLAN и т. п. Команда Set пригодна также для установки условия, при выполнении которого агент SNMP должен послать менеджеру соответствующее сообщение. Может быть определена реакция на такие события,

как инициализация агента, рестарт агента, обрыв связи, восстановление связи, неверная аутентификация и потеря ближайшего маршрутизатора. Если происходит любое из этих событий, то агент инициализирует прерывание.

Команда Trap используется агентом для сообщения менеджеру о возникновении особой ситуации.

Версия SNMP v.2 добавляет к этому набору команду GetBulk, которая позволяет менеджеру получить несколько значений переменных за один запрос.
На сегодня существует несколько стандартов на базы данных управляющей информации для протокола SNMP. Основными являются стандарты MIB-I и MIB-II, а также версия базы данных для удаленного управления RMON MIB. Кроме этого существуют стандарты для специальных устройств MIB конкретного типа (например, MIB для концентраторов или MIB для модемов), а также частные MIB конкретных фирм-производителей оборудования.

Первоначальная спецификация MIB-I определяла только операции чтения значений переменных. Операции изменения или установки значений объекта являются частью спецификаций MIB-II.

Версия MIB-I (RFC 1156) определяет 114 объектов, которые подразделяются на 8 групп.

System - общие данные об устройстве (например, идентификатор поставщика, время последней инициализации системы).

Interfaces - параметры сетевых интерфейсов устройства (например, их количество, типы, скорости обмена, максимальный размер пакета).

Address Translation Table - описание соответствия между сетевыми и физическими адресами (например, по протоколу ARP).

Internet Protocol - данные, относящиеся к протоколу IP (адреса IP-шлюзов, хостов, статистика о IP-пакетах).

ICMP - данные, относящиеся к протоколу обмена управляющими сообщениями ICMP.

TCP - данные, относящиеся к протоколу TCP (например, о TCP-соединениях)

UDP - данные, относящиеся к протоколу UDP (число переданных, принятых и ошибочных UPD-дейтаграмм).

EGP - данные, относящиеся к протоколу обмена маршрутной информацией Exterior Gateway Protocol, используемому в Internet (число принятых с ошиб-ками и без ошибок сообщений).

Из этого перечня групп переменных видно, что стандарт MIB-I разрабатывался с жесткой ориентацией на управление маршрутизаторами, поддерживающими протоколы стека TCP/IP.

В версии MIB-II (RFC 1213), принятой в 1992 году, был существенно (до 185) расширен набор стандартных объектов, а число групп увеличилось до 10.На рис.8.1 приведен пример древовидной структуры базы объектов MIB-II. На нем показаны две из 10 возможных групп объектов - System (имена объектов начинаются с префикса Sys) и Interfaces (префикс if). Объект SysUpTimeсодержит значение продолжительности времени работы системы с момента последней перезагрузки, объект SysObjectID - идентификатор устройства (например, маршрутизатор).

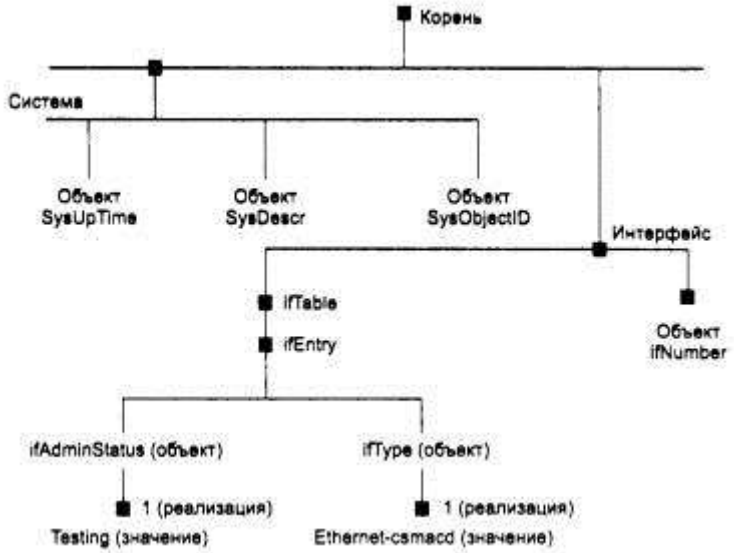



**Рис. 8.1** Стандартное дерево MIB-II (фрагмент)

Объект ifNumber определяет количество сетевых интерфейсов устройства, а объект ifEntry является вершиной поддерева, описывающего один из конкретных интерфейсов устройства. Входящие в это поддерево объекты ifType и ifAdminStatusопределяют соответственно тип и состояние одного из интерфейсов, в данном случае интерфейса Ethernet.
В число объектов, описывающих каждый конкретный интерфейс устройства, включены следующие.

- ifType - тип протокола, который поддерживает интерфейс. Этот объект принимает значения всех стандартных протоколов канального уровня, например rfc877-x25, ethemet-csmacd, iso88023-csmacd, iso88024-tokenBus, iso88025-tokenRlng и т. д.
- ifMtu - максимальный размер пакета сетевого уровня, который можно послать через этот интерфейс.
- ifSpeed - пропускная способность интерфейса в битах в секунду (100 для Fast Ethernet).
- ifPhysAddress - физический адрес порта, для Fast Ethernet им будет MAC - адрес.
- ifAdminStatus - желаемый статус порта.
- up - готов передавать пакеты.
- down - не готов передавать пакеты.

- testing - находится в тестовом режиме.
- ifOperStatus - фактический текущий статус порта, имеет те же значения, что и ifAdminStatus.
- ifInOctets - общее количество байт, принятое данным портом, включая служебные, с момента последней инициализации SNMP-агента.
- iflnUcastPkts - количество пакетов с индивидуальным адресом интерфейса, доставленных протоколу верхнего уровня.
- IflnNUcastPkts - количество пакетов с широковещательным или мультивещательным адресом интерфейса, доставленных протоколу верхнего уровня.
- ifInDiscards - количество пакетов, которые были приняты интерфейсом, оказались корректными, но не были доставлены протоколу верхнего уровня, скорее всего из-за переполнения буфера пакетов или же по иной причине.
- ifin Errors - количество пришедших пакетов, которые не были переданы протоколу верхнего уровня из-за обнаружения в них ошибок.

Кроме объектов, описывающих статистику по входным пакетам, имеются аналогичные объекты, но относящиеся к выходным пакетам.

Как видно из описания объектов MIB-II, эта база данных не дает детальной статистики по характерным ошибкам кадров Ethernet, кроме этого, она не отражает изменение характеристик во времени, что часто интересует сетевого администратора.

Эти ограничения были впоследствии сняты новым стандартом на MIB - RMON MIB, который специально ориентирован на сбор детальной статистики по протоколу Ethernet, к тому же с поддержкой такой важной функции, как построение агентом зависимостей статистических характеристик от времени.

### 8.3 Форматы и имена объектов SNMP MIB

Для именования переменных базы MIB и однозначного определения их форматов используется дополнительная спецификация, называемая SMI - Structure of Management Information. Например, спецификация SMI включает в качестве стандартного имя IpAddress и определяет его формат как строку из 4 байт. Другой пример - имя Counter, для которого определен формат в виде целого числа в диапазоне от 0 до $2^{32}$-1.

При описании переменных MIB и форматов протокола SNMP спецификация SMI опирается на формальный язык ASN.1, принятый ISO в качестве нотации для описания терминов коммуникационных протоколов (правда, многие коммуникационные протоколы, например IP, PPP или Ethernet, обходятся без этой нотации). Нотация ASN. 1 служит для установления однозначного соответствия между терминами, взятыми из стандартов, предназначенных для человеческого использования, и теми данными, которые передаются в коммуникационных протоколах аппаратурой. Достигаемая однозначность очень важна для гетерогенной среды, характерной для корпоративных сетей. Так, вместо того чтобы указать, что некоторая переменная протокола представляет собой целое число, разработчик протокола, использующий нотацию ASN.1, должен точно определить формат и допустимый диапазон переменной. В результате документация на MIB, написанная с помощью нотации ASN.1, может точно и

механически транслироваться в форму кодов, характерных для сообщений протоколов.

Нотация ASN.1 похожа на другие метаязыки, например нормальную Бэкусову форму, используемую при описании языков программирования, в частности Алгола. Нотация ASN.1 поддерживает базовый набор различных типов данных, таких как целое число, строка и т. п., а также позволяет конструировать из этих базовых типов составные данные - массивы, перечисления, структуры.

Существуют правила трансляции структур данных, описанных на ASN.1, в структуры данных языков программирования, например C++. Соответственно, имеются трансляторы, выполняющие эту работу. Примера описаний данных с помощью ASN.1 приведены ниже при описании протокольных блоков данных SNMP.

Нотация ASN.1 широко используется при описании многих стандартов OSI, в частности моделей управляемых объектов и структуры сообщений протокола CMIP.

Имена переменных MIB могут быть записаны как в символьном, так и в числовом форматах. Символьный формат используется для представления переменных в текстовых документах и на экране дисплея, а числовые имена - в сообщениях протокола SNMP. Например, символьному имени SysDescr соответствует числовое имя 1, а более точно 1.3.6.1.2.1.1.1.

Составное числовое имя объекта SNMP MIB соответствует полному имени этого объекта в дереве регистрации объектов стандартизации ISO. Разработчики протокола SNMP не стали использовать традиционный для стандартов Internet способ фиксации численных параметров протокола в специальном RFC, называемом «Assigned Numbers» (там описываются, например, численные значения, которые может принимать поле Protocol пакета IP, и т. п.). Вместо этого они зарегистрировали объекты баз MIB SNMP во всемирном дереве регистрации стандартов ISO, показанном на рис. 6.7.

Как и в любых сложных системах, пространство имен объектов ISO имеет древовидную иерархическую структуру, причем на рис. 8.2 показана только его верхняя часть. От корня этого дерева отходят три ветви, соответствующие стандартам, контролируемым ISO, ITU и совместно ISO-ITU. В свою очередь, организация ISO создала ветвь для стандартов, создаваемых национальными и международными организациями (ветвь огд). Стандарты Internet создавались под эгидой Министерства обороны США (Departament of Defence, DoD), поэтому стандарты MIB попали в поддерево dod-internet, а далее, естественно, в группу стандартов управления сетью - ветвь mgmt. Объекты любых стандартов, создаваемых под эгидой ISO, однозначно идентифицируются составными символьными именами, начинающимися от корня этого дерева. В сообщениях протоколов символьные имена не используются, а применяются однозначно соответствующие им составные числовые имена. Каждая ветвь дерева имен объектов нумеруется в дереве целыми числами слева направо, начиная с единицы, и эти числа и заменяют символьные имена. Поэтому полное символьное имя объекта MIB имеет вид: iso.org.dod.internet.mgmt.mib, а полное числовое имя: 1.3.6.1.2.1.

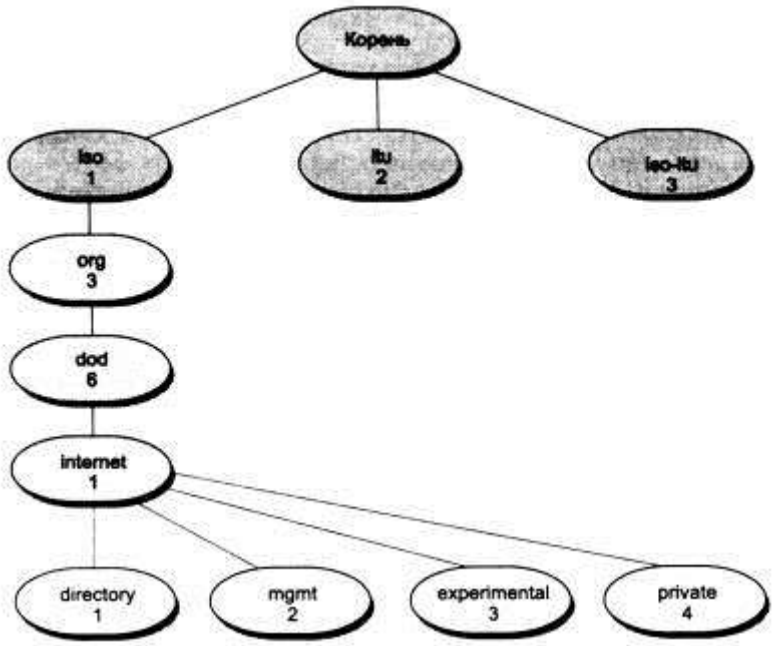



**Рис. 8.2.** Пространство имен объектов ISO

Группа объектов private (4) зарезервирована за стандартами, создаваемыми частными компаниями, например Cisco, Hewlett-Packard и т. п. Это же дерево регистрации используется для именования классов объектов CMIP и TMN.
Соответственно, каждая группа объектов MIB-I и MIB-II также имеет кроме кратких символьных имен, приведенных выше, полные символьные имена и соответствующие им числовые имена. Например, краткое символьное имя группы System имеет полную форму iso.org.dod.internet.mgmt.mib.system, а ее соответствующее числовое имя - 1.3.6.1.2.1. Часть дерева имен ISO, включающая группы объектов MIB, показана на рис. 8.3

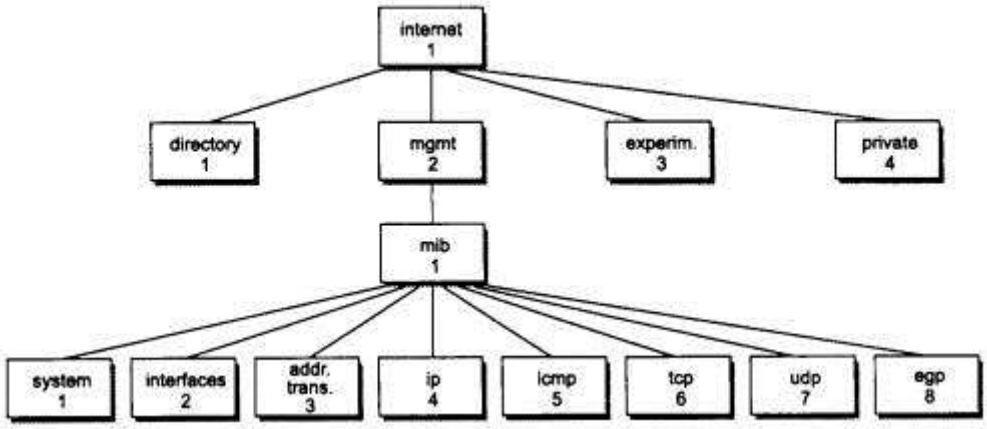



**Рис. 8.3** Часть дерева имен ISO, включающая группы объектов MIB-I

### 8.4 Формат сообщений SNMP

Протокол SNMP обслуживает передачу данных между агентами и станцией, управляющей сетью. SNMP использует дейтаграммный транспортный протокол UDP, не обеспечивающий надежной доставки сообщений. Протокол, организующий надежную передачу дейтаграмм на основе соединений TCP, весьма загружает управляемые устройства, которые на момент разработки протокола SNMP были не очень мощные, поэтому от услуг протокола TCP решили отказаться.

SNMP часто рассматривают только как решение для управления сетями TCP/IP. Хотя SNMP чаще всего и работает над UDP (он может также работать и над TCP), он может работать и над транспортными сетевыми протоколами стека OSI - TPO, TP4, CNLS, а также над протоколами MAC - уровня. Растет поддержка протокола SNMP и в других транспортных средах. Например, фирма Novell начала поддерживать протокол SNMP с версии NetWare 3.11, а некоторые производители оборудования (например, Bay Networks) реализуют в своих устройствах передачу сообщений SNMP с помощью как IP, так и IPX.

Сообщения SNMP, в отличие от сообщений многих других коммуникационных протоколов, не имеют заголовков с фиксированными полями. В соответствии с нотацией ASN.1 сообщение SNMP состоит из произвольного количества полей, и каждое поле предваряется описателем его типа и размера.

Любое сообщение SNMP состоит из трех основных частей: версии протокола (version), идентификатора общности (community), используемого для группирования устройств, управляемых определенным менеджером, и области данных, в которой собственно и содержатся описанные выше команды протокола, имена объектов и их значения. Область данных делится на блоки данных протокола (Protocol Data Unit, PDU).

Общий формат сообщения SNMP в нотации ASN.1 выглядит следующим образом:>

```
SNMP-Message ::=
SEQUENCE {
version INTEGER {
version-1 (0)
},
community
OCTET STRING,
SNMP-PDUs
ANY
}
```

Область данных может содержать пять различных типов PDU, соответствующих пяти командам протокола SNMP:

```
SNMP-PDUs :: =
CHOICE {
get-request
GetRequest-PDU,
get-next-request
GetNextRequest-PDU,
get-response
GetResponse-PDU,
set-request
SetRequest-PDU,
trap
Trap-PDU,
}
```

И наконец, для каждого типа PDU имеется определение его формата. Например, формат блока GetRequest-PDU описан следующим образом:

```
GetRequest-PDU ::=
```

IMPLICIT SEQUENCE {
request-id
RequestID,
error-status
ErrorStatus,
error-index
Errorlndex,
variable-bindings
VarBindList
}
Далее стандарт SNMP определяет соответственно формат переменных блока GetRequest-PDU. Переменная Request ID - это 4-байтовое целое число (используется для установления соответствия ответов запросам), ErrorStatus и Errorlndex - это однобайтовые целые, которые в запросе должны быть установлены в 0. VarBindList - это список числовых имен объектов, значениями которых интересуется менеджер. В нотации ASN.1 этот список состоит из пар «имя - значение». При запросе значение переменной должно быть установлено в null.
Вот пример сообщения протокола SNMP, которое представляет собой запрос о значении объекта SysDescr (числовое имя 1.3.6.1.2.1.1.1).

Как видно из описания, сообщение начинается с кода 30 (все коды шестнадцатеричные), который соответствует ключевому слову SEQUENCE (последовательность). Длина последовательности указывается в следующем байте (41 байт). Далее следует целое число длиной 1 байт - это версия протокола SNMP (в данном случае О, то есть SNMP v.l, а 1 означала бы SNMP v.2). Поле community имеет типstring (строка символов) длиной в 6 байт со значением public. Остальную часть сообщения составляет блок данных GetRequest-PDU . То, что это операция Get-request, говорит код АО (это значение определено в протоколе SNMP, а не в нотации ASN.1), а общая длина блока данных - 28 байт. В соответствии со структурой блокаGetrequest-PDU, далее идет идентификатор запроса (он определен как целое 4-байтовое число). Затем в блоке следуют два однобайтовых целых числа статуса и индекса ошибки, которые в запросе установлены в 0. И наконец, завершает сообщение список объектов, состоящий из одной пары - имени 1.3.6.1.2.1.1.1.0 и значения null.

| 30 | 29 | 02 | 01 | 00 | | | |
|---|---|---|---|---|---|---|---|
| SEQUENCE | len = 41 | INTEGER | len=1 | vers = 0 | | | |
| 04 | 06 | 70 | 75 | 62 | 6C | 69 | 63 |
| string | len = 6 | p | u | b | l | i | c |
| A0 | 1C | 02 | 04 | 05 | AE | 56 | 02 |
| getreq | len = 28 | INTEGER | len = 4 | -------- | requested ID | -------- | -------- |
| 02 | 01 | 00 | 02 | 01 | 00 | | |
| INTEGER | len = 1 | status | INTEGER | len = 1 | error | index | |
| 30 | 0E | 30 | 0C | 06 | 08 | | |
| SEQUENCE | len = 14 | SEQUENCE | len = 12 | objectid | len = 8 | | |
| 2B | 06 | 01 | 02 | 01 | 01 | 01 | 00 |
| 1,3 | 6 | 1 | 2 | 1 | 1 | 1 | 0 |
| 05 | 00 | | | | | | |
| null | len = 0 | | | | | | |

### 8.5 Спецификация RMON MIB

Новейшим добавлением к функциональным возможностям SNMP является спецификация RMON, которая обеспечивает удаленное взаимодействие с базой MIB. До появления RMON протокол SNMP не мог использоваться удаленным образом, он допускал только локальное управление устройствами. База RMON MIB обладает улучшенным набором свойств для удаленного управления, так как содержит агрегированную информацию об устройстве, не требующую передачи по сети больших объемов информации. Объекты RMON MIB включают дополнительные счетчики ошибок в пакетах, более гибкие средства анализа трендов и статистики, более мощные средства фильтрации для захвата и анализа отдельных пакетов, а также более сложные условия установления сигналов предупреждения. Агенты RMON MIB более интеллектуальны по сравнению с агентами MIB-I или MIB-II и выполняют значительную часть работы по обработке информации об устройстве, которую раньше выполняли менеджеры. Эти агенты могут располагаться внутри различных коммуникационных устройств, а также быть выполнены в виде отдельных программных модулей, работающих на универсальных персональных компьютерах и ноутбуках.

Объекту RMON присвоен номер 16 в наборе объектов MIB, а сам объект RMON объединяет 10 групп следующих объектов.

- Statistics - текущие накопленные статистические данные о характеристиках пакетов, количестве коллизий и т. п.
- History - статистические данные, сохраненные через определенные промежутки времени для последующего анализа тенденций их изменений.
- Alarms - пороговые значения статистических показателей, при превышении которых агент RMON посылает сообщение менеджеру.
- Hosts - данные о хостах сети, в том числе и о их MAC - адресах.
- HostTopN - таблица наиболее загруженных хостов сети.
- Traffic Matrix - статистика об интенсивности трафика между каждой парой хостов сети, упорядоченная в виде матрицы.
- Filter - условия фильтрации пакетов.
- Packet Capture - условия захвата пакетов.

- Event - условия регистрации и генерации событий.

Данные группы пронумерованы в указанном порядке, поэтому, например, группа Hosts имеет числовое имя 1.3.6.1.2.1.16.4.

Десятую группу составляют специальные объекты протокола Token Ring.

Всего стандарт RMON MIB определяет около 200 объектов в 10 группах, зафиксированных в двух документах - RFC 1271 для сетей Ethernet и RFC 1513 для сетей Token Ring.

Отличительной чертой стандарта RMON MIB является его независимость от протокола сетевого уровня (в отличие от стандартов MIB-I и MIB-II, ориентированных на протоколы TCP/IP). Поэтому он удобен для гетерогенных сред, использующих различные протоколы сетевого уровня.

Рассмотрим более подробно группу Statistics, которая определяет, какую информацию о кадрах (называемых в стандарте пакетами) Ethernet может предоставить агент RMON. Группа History основана на объектах группы Statistics, так как ее объекты просто позволяют строить временные ряды для объектов группы Statistics.

В группу Statistics входят наряду с некоторыми другими следующие объекты.

- etherStatsDropEvents - общее число событий, при которых пакеты были проигнорированы агентом из-за недостатка его ресурсов. Сами пакеты при этом не обязательно были потеряны интерфейсом.
- etherStatsOrtets - общее число байт (включая ошибочные пакеты), принятых из сети (исключая преамбулу н включая байты контрольной суммы).
- etherStatsPkts - общее число полученных пакетов (включая ошибочные).
- etherStatsBroadcastPkts - общее число хороших пакетов, которые были посланы по широковещательному адресу.
- etherStatsMulticastPkts - общее число хороших пакетов, полученных по мультивещательному адресу.
- etherStatsCRCAlign Errors - общее число полученных пакетов, которые имели длину (исключая преамбулу) между 64 и 1518 байт, не содержали целое число байт (alignment error) или имели неверную контрольную сумму (FCS error).
- etherStatsUndersizePkts - общее число пакетов, которые имели длину меньше, чем 64 байт, но были правильно сформированы.
- etherStatsOversizePkts - общее число полученных пакетов, которые имели длину больше, чем 1518 байт, но были тем не менее правильно сформированы.
- etherStatsFragments - общее число полученных пакетов, которые не состояли из целого числа байт или имели неверную контрольную сумму и имели к тому же длину, меньшую 64 байт.
- etherStatsJabbers - общее число полученных пакетов, которые не состояли из целого числа байт или имели неверную контрольную сумму и имели к тому же длину, большую 1518 байт.
- etherStatsCollisions - наилучшая оценка числа коллизий на данном сегменте Ethernet.
- etherStatsPkts640ctets - общее количество полученных пакетов (включая плохие) размером 64 байт.
- etherStatsPkts65to1270ctets - общее количество полученных пакетов (включая плохие) размером от 65 до 127 байт.

- etherStatsPktsl28to2550ctets - общее количество полученных пакетов (включая плохие) размером от 128 до 255 байт.
- etherStatsPkts256to5110ctets - общее количество полученных пакетов (включая плохие) размером от 256 до 511 байт.
- etherStatsPkts512tol0230ctets - общее количество полученных пакетов (включая плохие) размером от 512 до 1023 байт.
- etherStatsPktsl024tol5180ctets - общее количество полученных пакетов (включая плохие) размером от 1024 до 1518 байт.

Как видно из описания объектов, с помощью агента RMON, встроенного в повторитель или другое коммуникационное устройство, можно провести очень детальный анализ работы сегмента Ethernet или Fast Ethernet. Сначала можно получить данные о встречающихся в сегменте типах ошибок в кадрах, а затем целесообразно собрать с помощью группыHistoryзависимости интенсивности этих ошибок от времени (в том числе и привязав их ко времени). После анализа временных зависимостей часто уже можно сделать некоторые предварительные выводы об источнике ошибочных кадров и на этом основании сформулировать более тонкие условия захвата кадров со специфическими признаками (задав условия в группеFilter ), соответствующими выдвинутой версии. После этого можно провести еще более детальный анализ за счет изучения захваченных кадров, извлекая их из объектов группыPacket Capture.

Позже был принят стандарт RMON 2, который распространяет идеи интеллектуальной RMON MIB на протоколы верхних уровней, выполняя часть работы анализаторов протоколов.

### 8.6 Недостатки протокола SNMP

Протокол SNMP служит основой многих систем управления, хотя имеет несколько принципиальных недостатков, которые перечислены ниже.

- Отсутствие средств взаимной аутентификации агентов и менеджеров. Единственным средством, которое можно было бы отнести к средствам аутентификации, является использование в сообщениях так называемой «строки сообщества» - «community string». Эта строка передается по сети в открытой форме в сообщении SNMP и служит основой для деления агентов и менеджеров на «сообщества», так что агент взаимодействует только с теми менеджерами, которые указывают в поле community string ту же символьную строку, что и строка, хранящаяся в памяти агента. Это, безусловно, не способ аутентификации, а способ структурирования агентов и менеджеров. Версия SNMP v.2 должна была ликвидировать этот недостаток, но в результате разногласий между разработчиками стандарта новые средства аутентификации хотя и появились в этой версии, но как необязательные.
- Работа через ненадежный протокол UDP (а именно так работает подавляющее большинство реализации агентов SNMP) приводит к потерям аварийных сообщений (сообщений trap) от агентов к менеджерам, что может привести к некачественному управлению. Исправление ситуации путем перехода на надежный транспортный протокол с установлением соединений чревато потерей связи с огромным количеством встроенных агентов SNMP, имеющихся в установленном в сетях оборудовании. (Протокол CMIP

изначально работает поверх надежного транспорта стека OSI и этим недостатком не страдает.) Разработчики платформ управления стараются преодолеть эти недостатки. Например, в платформе HP 0V Telecom DM TMN, являющейся платформой для разработки многоуровневых систем управления в соответствии со стандартами TMN и ISO, работает новая реализация SNMP, организующая надежный обмен сообщениями между агентами и менеджерами за счет самостоятельной организации повторных передач сообщений SNMP при их потерях.

### Simple Object Access Protocol (SOAP) 1.1[7]

**W3C Note 08 May 2000** (Copyright© 2000 DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software)

**This version:**

http://www.w3.org/TR/2000/NOTE-SOAP-20000508

**Latest version:**

http://www.w3.org/TR/SOAP

**Authors (alphabetically):**

Don Box, DevelopMentor
David Ehnebuske, IBM
Gopal Kakivaya, Microsoft
Andrew Layman, Microsoft
Noah Mendelsohn, Lotus Development Corp.
Henrik Frystyk Nielsen, Microsoft
Satish Thatte, Microsoft
Dave Winer, UserLand Software, Inc.

### Web Services Description Language (WSDL) 1.1[8]

### W3C Note 15 March 2001

**This version:**

http://www.w3.org/TR/2001/NOTE-wsdl-20010315

**Latest version:**

http://www.w3.org/TR/wsdl

**Authors (alphabetically):**

Erik Christensen, Microsoft
Francisco Curbera, IBM Research
Greg Meredith, Microsoft
Sanjiva Weerawarana, IBM Research



## 9 Распределенные протоколы на основе технологий DCOM и CORBA

Распределенные протоколы предназначены для связывания частей приложения, часть которых расположена на компьютере клиента (пользователя), а

другая часть выполняется на сервере приложений согласно архитектуре «клиент-сервер».

Для технологии Майкрософт DCOM существует обязательное требование согласования на сервере приложений и на клиенте версий *.dll- библиотек и операционных систем, что приводит к построению и реализации только одноплатформенных (гомогенных) систем обработки и соответствующим ограничениям и отсутствию возможности использовать компоненты программ, которые написаны на разных языках программирования. Это позволяет реализовывать приложения распределенные только на уровне данных (т.е. с СУБД под одинаковыми или разными ОС).

Для технологии CORBA нет таких ограничений – она создавалась именно для связывания разнородных программных компонентов, которые написаны на разных языках программирования, для использования в гетерогенной среде (т.е. для случая, когда сервер и рабочая станция клиента используют разные ОС от разных вендоров), что позволяет реализовывать распределенные на уровне программ приложения.

## 9.1 Распределенный протокол на базе технологии DCOM

Распределенный COM Microsoft (DCOM) развивает многокомпонентную модель (COM) до уровня поддержки связи между объектами различных компьютеров - по локальным сетям, сетям межкорпоративного уровня или даже Интернет. Исполнение вашего приложения при помощи DCOM может быть распределено по местам, наиболее удобным для пользователя.

DCOM берет на себя низкоуровневые детали сетевых протоколов, а вы сможете сфокусироваться на вашем настоящем деле: создании хороших решений для пользователей.

Далее дается общий обзор возможности использования DCOM для решения сложнейших проблем, связанных с распределенными приложениями, а также использованием этой технологии и ее компонентов на основе реализации распределенных сетевых протоколов.

Если Ваши приложения когда-нибудь нуждались (или будут нуждаться) в:

- устойчивой работе в случае аварии аппаратного обеспечения
- высокоэффективной работе как по обслуживанию небольшой рабочей группы, так и в качестве большого корпоративного сайта
- устойчивости приложений в их работе при авариях сети
- в улучшеной работе на клиентских машинах с различными параметрами или в различных географических зонах с различными удаленными функциональными серверами
- достижения большей эффективности приложений в условиях загруженности сети,

то ниже будет описано, каким образом DCOM поможет вам решить эти и другие проблемы для новых или уже существующих приложений.

Ниже дано объяснение предложенных решений, которые предлагаются DCOM и Майкрософт:

- Независимость от местоположения.
- Управление соединением.
- Масштабируемость.
- Быстродействие.
- Полоса пропускания и латентность.
- Безопасность.
- Баланс загрузки.
- Устойчивость к авариям.
- Гибкость в перераспределении ресурсов.
- Нейтральность протокола взаимодействия.
- Нейтральность платформы.
- Безболезненная интеграция с другими протоколами Интернет.

Начав с этого материала, вам следует обратиться к другим страницам серии для более подробного ознакомления по определенным вопросам.

Если вы хотите узнать, как использовать DCOM в определенных приложениях и решениях, обратитесь к материалу Microsoft “DCOM - Решения в работе”. Это - хороший способ применения многих идей, использованных в этих решениях, в вашей собственной работе.

Если вы действительно хотите понять сам механизм совместной работы компонентов, обратитесь к тем статьям, ссылки на которые есть на этих страницах и которые вы можете увидеть на “Technology Preview CD”.

“DCOM-Архитектура”. Это пособие по внутренней и внешней работе DCOM, воспользовавшись которым, вы увидите, как легко DCOM реализует работу в распределенной среде без ущерба гибкости, масштабируемости или отказоустойчивости.

Эти страницы вводят вас в мир распределенного программирования. Здесь показаны некоторые проблемы, с которыми вы столкнетесь или уже столкнулись, и даны некоторые соображения, как можно их устранить, используя уникальные возможности DCOM.

### 9.1.1 Как получить реализацию DCOM?

В данный момент DCOM поставляется с Windows 2003. COM и DCOM больше не являются с 2000 года монополией Microsoft, а управляются независимым консорциумом ActiveX (ActiveX Consortium).

Распределение приложений - это не самоцель. Распределенные приложения предоставляют вам абсолютно новые решения в проектировании и развитии. Для того, чтобы получить эти дополнительные возможности, необходимо сделать значительные вложения. Некоторые приложения являются распределенными изначально: многопользовательские игры, приложения для обмена мнениями, для

телеконференций - все это примеры подобных приложений. Для них преимущества устойчивой инфраструктуры очевидны. Многие другие приложения также являются распределенными в том смысле, что они имеют как минимум два компонента, работающие на различных машинах. Но, поскольку эти приложения не были созданы для использования в распределенной среде, они достаточно ограничены в масштабируемости и в гибкости перераспределения. Любой тип поточных или групповых приложений, большинство приложений клиент/сервер и даже некоторые desktop-приложения обязательно управляют способом коммуникаций и кооперации своих пользователей. Рассмотрение таких приложений в качестве распределенных и работа с нужным компонентом в нужном месте предоставляют пользователю преимущества и оптимизируют использование сетевых и компьютерных ресурсов. Приложения, которые разрабатывались как распределенные, могут совмещать различных клиентов с различными мощностями посредством работы компонента со стороны клиента, если это возможно, и - со стороны сервера, когда это необходимо.

Разработка распределенных приложений дает системному менеджеру большое преимущество в виде гибкости в перераспределении нагрузки между серверами.

К тому же, распределенные приложения являются гораздо более масштабируемыми, нежели их монолитные собратья. Если вся логика комплексного приложения сосредоточена в едином модуле, есть единственный способ ускорить работу без настройки приложения: более скоростное аппаратное обеспечение. Сегодняшние серверы и операционные системы легко модифицируются, однако чаще бывает дешевле приобрести еще одну такую же машину, нежели сделать upgrade, чтобы ускорить сервер вдвое. При правильно сконструированном распределенном приложении в начале работы все компоненты могут запуститься с одного сервера. При увеличении загрузки некоторые компоненты могут перераспределяться на дополнительные, менее дорогостоящие машины.

### 9.1.2 Архитектура DCOM

DCOM является развитием многокомпонентной модели (COM). COM определяет, каким образом компоненты взаимодействуют со своими клиентами. Это взаимодействие осуществляется таким образом, чтобы клиент и компонент могли соединяться без необходимости использовать некоторый промежуточный компонент системы и клиент мог вызывать методы компонента. Рисунок 9.1 иллюстрирует это с точки зрения многокомпонентной модели.

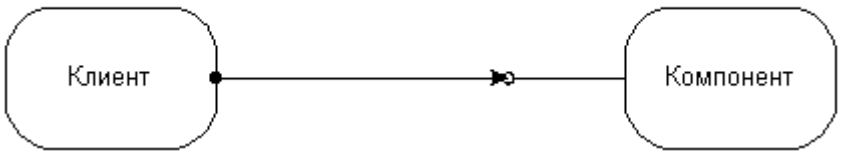



Рис. 9.1 – COM-компоненты в одном процессе

В сегодняшних операционных системах процессы изолированы друг от друга. Клиент, которому нужно связаться с компонентом другого объекта, не может вызвать компонент напрямую, а должен использовать некоторую форму связи между процессами, предусмотренную операционной системой. COM организует подобное соединение в полностью прозрачной манере: он перехватывает вызовы со стороны клиента и адресует их компоненту другого процесса. Рис.9.2 показывает, как run-time библиотеки COM/DCOM организуют соединение между клиентом и компонентом.

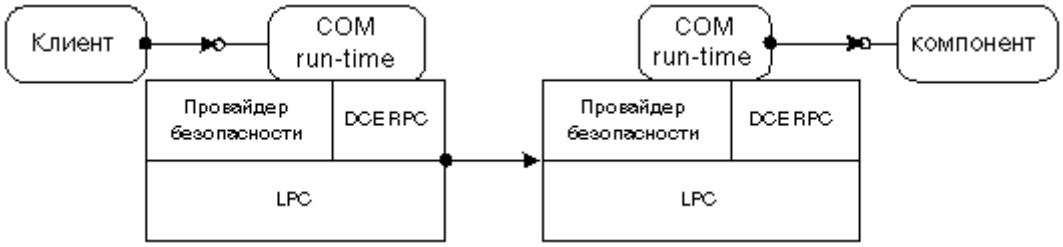



Рис. 9.2 – COM-компоненты в разных процессах

Когда клиент и компонент находятся на различных машинах, DCOM просто заменяет локальное соединение между процессами сетевым протоколом. При этом ни клиент, ни компонент и не подозревают, что провода, соединяющие их, стали чуть длиннее.

Рис.9.3 показывает архитектуру DCOM в общем: COM run-time предлагает клиентам и компонентам объектно-ориентированные сервисы и использует RPC и провайдер безопасности для генерации стандартных сетевых пакетов, соответствующих стандарту протокола DCOM.

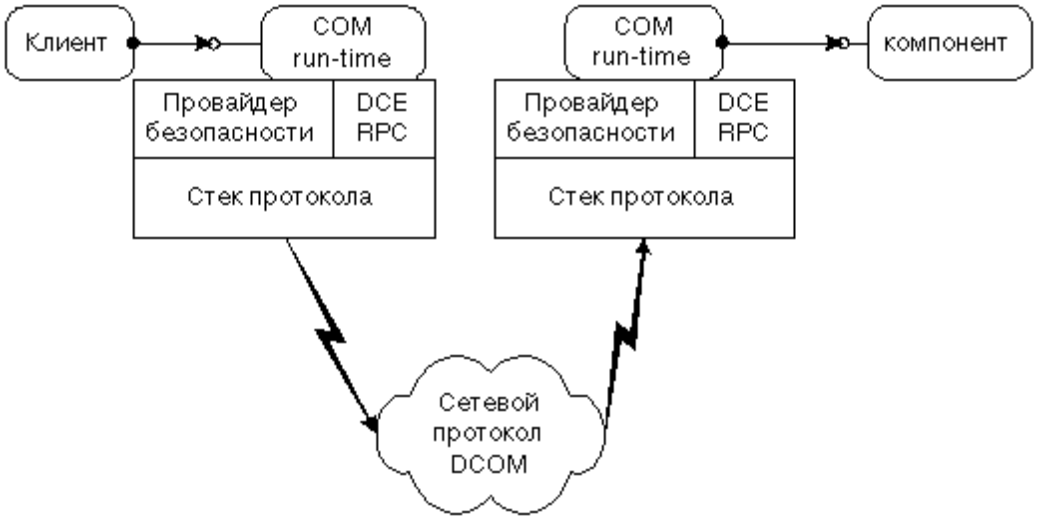



Рис. 9.3 – DCOM: COM-компоненты на различных машинах

### 9.1.3 Компоненты и их повторное использование

Большая часть распределенных приложений разрабатывалась не наспех и не в вакууме. Существующая инфраструктура аппаратного обеспечения, существующее программное обеспечение, существующие компоненты и инструменты должны быть интегрированы и призваны уменьшить время и

стоимость разработки и перераспределения. В смысле инвестиций DCOM имеет несомненные преимущества перед любыми COM компонентами и инструментами. Огромный рынок компонентов позволяет уменьшить время разработки с помощью интегрирования стандартизованных решений в пользовательское приложение. Многие разработчики знакомы с COM и могут легко применить свои знания в создании распределенных DCOM-приложений.

Любой компонент, разрабатывавшийся как часть распределенного приложения - кандидат для повторного использования в будущем. Организация процесса разработки по компонентному принципу позволяет непрерывно повышать уровень функциональности новых приложений и уменьшать время разработки, используя уже выполненную работу в качестве основы для развития. Проектирование в COM и DCOM обеспечивает использование ваших компонентов и сегодня, и в будущем.

### 9.1.4 Независимость от местоположения

Когда вы начинаете использовать распределенное приложение в реальной сети, выявляется несколько особенностей:

- Компоненты, взаимодействующие чаще, должны быть “ближе” друг к другу.
- Некоторые компоненты могут работать на определенных машинах или в определенных местах.
- Использование меньших по размеру компонентов увеличивает гибкость в перераспределении, но при этом увеличивается сетевой трафик.
- Компоненты большего размера уменьшают сетевой траффик, но зато уменьшается и гибкость в перераспределении.

С помощью DCOM эти критические моменты обходятся достаточно легко, поскольку в исходном коде не определены детали перераспределения. DCOM полностью скрывает местоположение компонента, будь он в том же процессе, что и клиент, или на другом конце света. В любом случае способы, которыми клиент соединяется с компонентом и вызывает методы компонента, идентичны. DCOM не нуждается не только в изменениях исходного кода, но даже и в рекомпиляции программы. Простая реконфигурация изменяет способ, которым компоненты соединяются друг с другом.

Независимость DCOM от местоположения значительно упрощает задачу распределения компонентов приложения для получения оптимального быстродействия в целом. Представим, например, что определенные компоненты должны находиться на определенных машинах или в определенных местах. Если приложение имеет большое число маленьких компонентов, вы можете уменьшить загрузку сети перемещением их в нужный сегмент ЛВС, на нужную машину или даже в нужный процесс. Если приложение состоит из меньшего числа больших компонентов, загрузка сети - меньшая из проблем, так как вы можете разместить компоненты на более быстрые из доступных машин.

Рис. 9.4 демонстрирует, как один и тот же компонент может быть перенесен на машину клиента при удовлетворительной полосе пропускания между машиной

“клиент” и машиной “среднего слоя”, и на машине сервера, если клиент работает с приложением по медленному сетевому соединению.

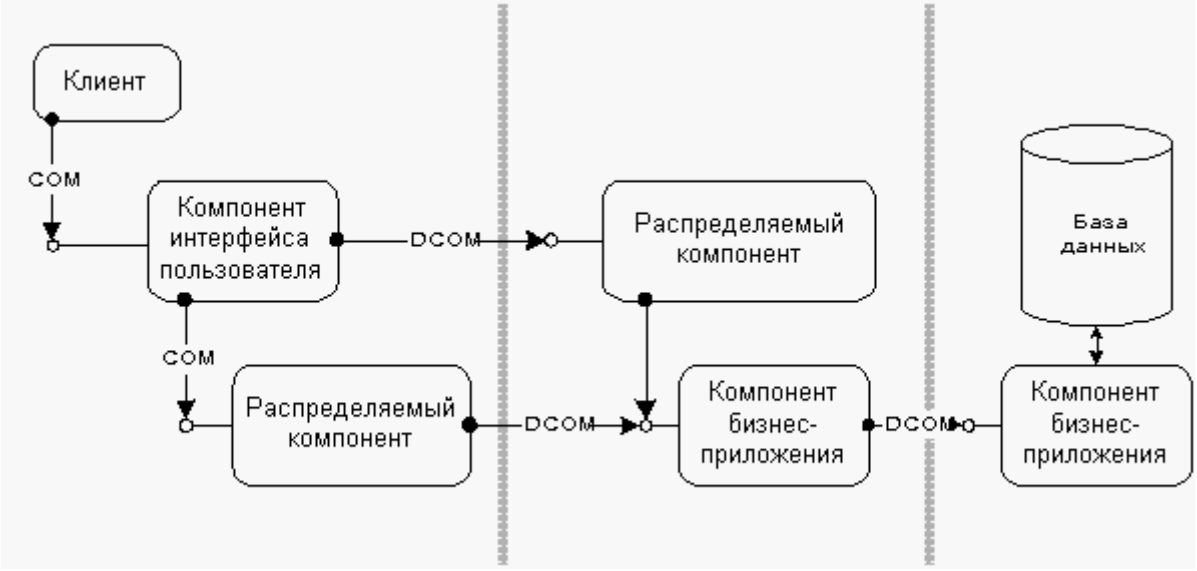



Рис.9.4 – Независимость от местоположения

Благодаря независимости DCOM от местоположения, приложение может перемещать взаимодействующие компоненты с “близлежащих” машин на одну и ту же машину или даже в один процесс. Даже если функциональность большого логического модуля организуется большим числом маленьких компонентов, они могут по-прежнему эффективно взаимодействовать друг с другом. Компоненты могут работать на машине, на которой это наиболее целесообразно: пользовательский интерфейс находится на машине клиента или “рядом” с ней. Компонент, работающий с базой данных, – на сервере “близко” к базе данных.

### 9.1.5 Независимость от языка

Общим вопросом при проектировании и разработке распределенного приложения является выбор языка или инструмента для данного компонента. Язык обычно выбирают с учетом затрат на разработку, с учетом имеющейся квалификации и необходимого быстродействия. Как развитие COM, DCOM абсолютно не зависит от языка. Теоретически для создания COM-компонентов может использоваться любой язык и эти компоненты могут использоваться большим числом языков и инструментов. Java, Microsoft Visual C++, C#, Microsoft Visual Basic, Delphi, PowerBuilder и Micro Focus COBOL - все они хорошо взаимодействуют с DCOM.

Нейтральность DCOM по отношению к языку позволяет разработчику приложения выбирать язык и инструменты, с которыми он чувствует себя наиболее свободно. Независимость от языка, кроме того, позволяет выполнять быстрое прототипирование: вначале компоненты могут быть разработаны на языке высокого уровня, таком как Microsoft Visual Basic, а позже - на другом языке, таком как C++ или Java, лучше использующем преимущества таких продвинутых функций DCOM, как многопоточность и объединение потоков.

### 9.1.6 Управление соединением

Сетевые соединения не так устойчивы, как соединения внутри машины. Компоненты распределенного приложения должны знать, что клиент более не активен, даже – или в особенности – в случае сетевой или аппаратной аварии.

DCOM управляет соединением компонентов, предназначенных для одного клиента, так же, как и компонентов, обслуживающих несколько клиентов, используя счетчик ссылок для каждого компонента. Когда клиент соединяется с компонентом, DCOM увеличивает значение счетчика ссылок компонента. Когда клиент разрывает соединение, DCOM уменьшает значение счетчика ссылок компонента. Если значение счетчика достигает нуля – компонент свободен.

Для определения активности клиента DCOM использует эффективный протокол отслеживания (см. раздел 0 “Управление распределенным соединением между приложениями”). Машина клиента посылает периодическое сообщение. При нарушении соединения DCOM уменьшает счетчик и освобождает компонент, если значение счетчика станет равным нулю. С точки зрения компонента, случай отключения клиента, случай аварии сети и случай поломки машины клиента регистрируются одним и тем же механизмом подсчета. Приложения могут использовать такой механизм подсчета соединений для своего высвобождения.

Во многих случаях поток информации между компонентом и его клиентами не однонаправлен: компоненту нужно инициировать некоторые действия со стороны клиента, например, извещение о том, что длительный процесс завершился, обновляются данные, просматриваемые пользователем (обзор новостей), или поступило следующее сообщение в телеконференции или многопользовательской игре. Многие протоколы усложняют процесс осуществления такого типа симметричной связи. В DCOM каждый компонент может быть одновременно провайдером и потребителем функциональности. Один и тот же механизм, одни и те же возможности управляют связью в обоих направлениях, облегчая организацию связи и взаимодействия клиент/сервер.

DCOM предлагает устойчивый распределенный механизм регистрации соединений, который полностью прозрачен для приложения. DCOM - это одновременно симметричный сетевой протокол и модель программирования, предлагающие не только традиционное взаимодействие клиент/сервер, но и полноценную связь между клиентами и серверами.

***Масштабируемость***

Критическим фактором распределенных приложений является их способность модифицироваться в соответствии с числом пользователей, объемом данных и требуемой функциональностью. Приложение должно быть маленьким и быстрым при минимальной потребности в нем, но оно должно обеспечить и дополнительные потребности без ущерба быстродействию или надежности. DCOM предлагает ряд возможностей, улучшающих масштабируемость вашего приложения.

### 9.1.7 Симметричная многопроцессорная обработка (SMP)

DCOM использует возможность Windows NT поддерживать симметричную многопроцессорную обработку. Для приложений, использующих модель свободных потоков, DCOM организует объединение потоков входящих запросов. На многопроцессорных машинах это объединение потоков оптимизируется в соответствии с числом доступных процессоров: чрезмерно большие потоки приводят к слишком частому переключению между содержимым, в то время как слишком маленькие потоки могут привести к простою некоторых процессоров. DCOM ограждает разработчика от деталей управления потоками и обеспечивает оптимальное быстродействие, что может обеспечить лишь кодирование управления объединением потоков вручную, требующее больших затрат.

Используя поддержку симметричной многопроцессорной обработки Windows NT, DCOM-приложения могут безболезненно масштабироваться от уровня однопроцессорных машин до уровня огромных многопроцессорных систем.

***Гибкость в перераспределении***

Самые скоростные многопроцессорные машины могут не справиться с удовлетворением потребностей при увеличении загруженности приложения (даже если ваш бюджет выдержит приобретение такой машины). Независимость DCOM от местоположения облегчает осуществление перераспределения компонентов на другие компьютеры, предлагая, тем самым, более легкую и недорогую организацию масштабируемости.

Перераспределение проще всего выполняется для компонентов, которые не должны находиться в определенном месте или делить свое местонахождение с другими компонентами. В этом случае есть возможность работы многочисленных копий компонента на различных машинах. Чаще всего нагрузка может быть разделена между машинами с учетом таких критериев, как емкость машины или даже текущая загруженность. DCOM облегчает изменение способа соединения клиентов с компонентами и компонентов между собой. Одни и те же компоненты могут быть динамично перераспределены без выполнения повторной работы или даже рекомпиляции. Все, что необходимо - это обновить регистрацию, файловую систему или базу данных, в которых занесены местоположения каждого компонента.

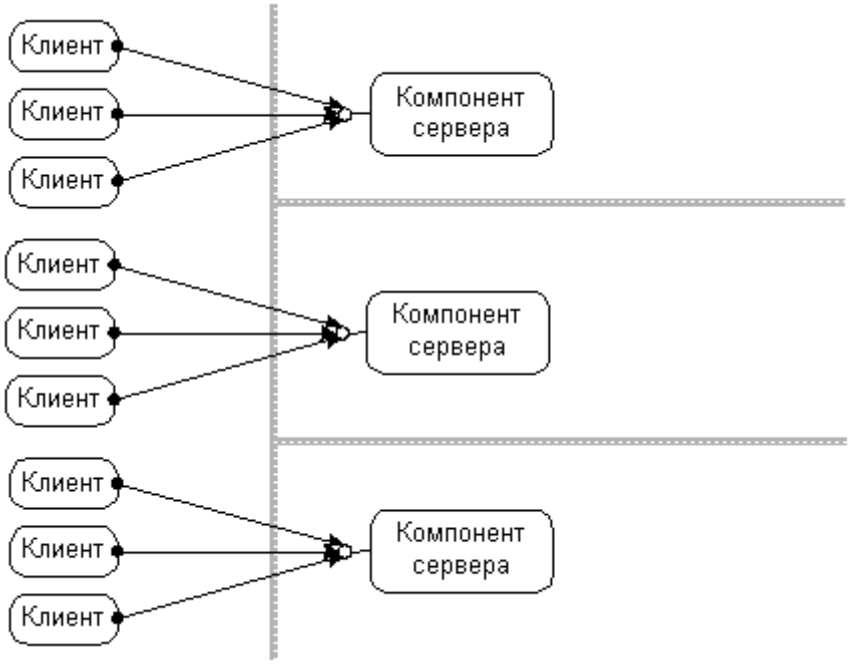



Рис.9.5 – Параллельное перераспределение

**Пример.** Организация, офисы которой расположены в Нью-Йорке, Лондоне, Сан-Франциско и Сиднее (Австралия) может установить компоненты на свои серверы. Двести пользователей одновременно получают доступ к 50 компонентам сервера с определенным быстродействием. По мере поставки пользователям новых бизнес-приложений и приложений, частично использующих уже имеющиеся бизнес-компоненты, а частично - новые, нагрузка на сервер увеличивается до 600 пользователей и скорость работы в пиковые часы становится неприемлемой. Администратор добавляет второй сервер и перераспределяет 30 компонентов, размещая их только на новой машине. Двадцать компонентов остаются только на старом сервере, а оставшиеся 20 работают на обеих машинах.

Большая часть реальных распределенных приложений имеют один или более критический компонент, который участвует в большинстве операций. Это могут быть компоненты базы данных или компоненты, связанные с бизнесом, доступ к которым должен осуществляться постоянно для реализации политики “первым пришел - первым обслужен”. Компоненты такого типа не могут дублироваться, поскольку их предназначение заключается в организации единственной точки синхронизации среди всех пользователей приложения. Для увеличения быстродействия распределенного приложения в целом такие компоненты, создающие “узкие места”, должны перераспределяться на специально выделенный мощный сервер. DCOM помогает вам изолировать такие критические компоненты на ранних этапах проектирования, размещать первоначально компоненты на одной машине, а позже - переносить критические компоненты на отдельные машины, что показано на Рис. 9.6

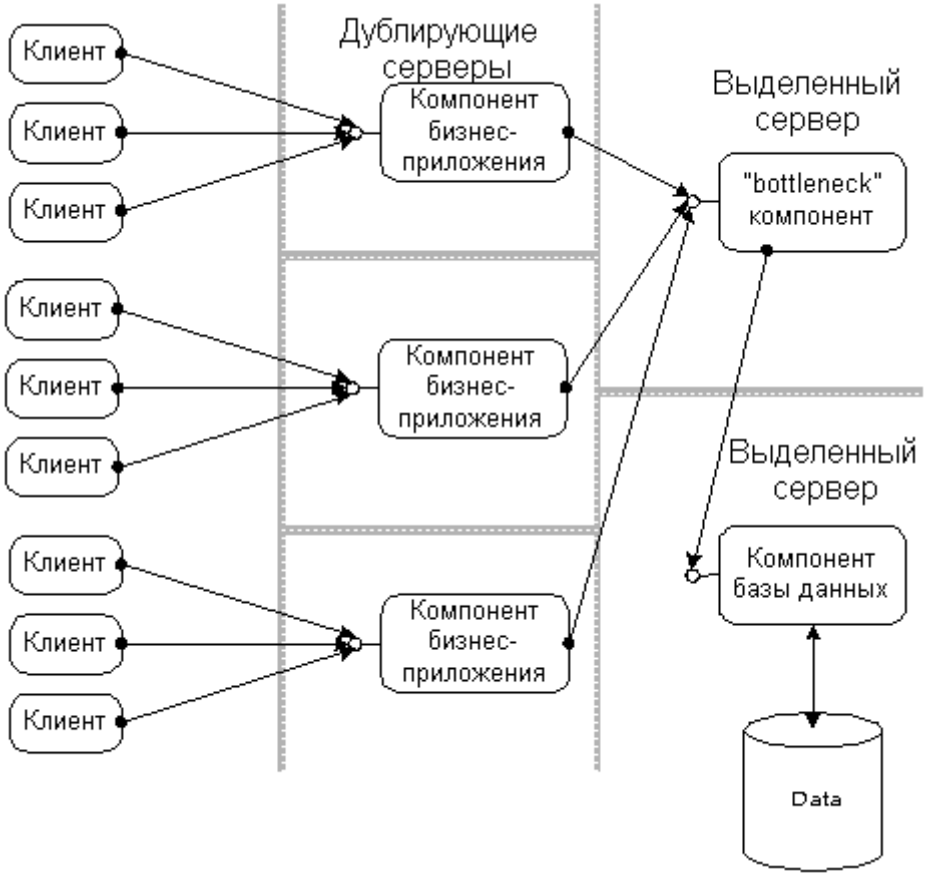



Рис. 9.6 – Изоляция критических компонентов

Для таких критических компонентов DCOM в целом может организовать более быстрое выполнение задачи. Подобные компоненты - это обычно часть последовательности процесса организации в электронной торговой системе заказов на покупку или продажу: запросы должны обрабатываться по мере поступления (первым пришел – первым обслужен). Одно из решений заключается в разделении задачи на меньшие компоненты с перераспределением каждого компонента на отдельную машину. Результат этого подобен поточному методу (pipelining), используемому в современных микропроцессорах: первый запрос обрабатывается первым компонентом (например, выполняющим проверку содержимого) и передается на следующий компонент (который, например, выполняет обновление базы данных). Как только первый компонент передал запрос на следующий, он готов обработать следующий запрос. Фактически, две машины параллельно обрабатывают множество запросов в определенном порядке их поступления. То же самое возможно выполнять и на одной машине (при использовании DCOM): многочисленные компоненты могут выполняться в различных потоках или процессах. В дальнейшем, когда потоки смогут быть распределены на одной многопроцессорной машине или на различные машины, этот подход облегчит масштабируемость, что показано на Рис.9.7.

Модель программирования DCOM с ее независимостью от местоположения облегчает схемы перераспределения по мере разрастания приложения: первоначально машина сервера может содержать все компоненты, соединяясь с ними как с очень эффективными серверами в процессе. Фактически приложение выглядит как хорошо отлаженное монолитное приложение. По мере увеличения

потребностей можно добавить другие машины с перераспределением компонентов на эти машины без всякого изменения кода.

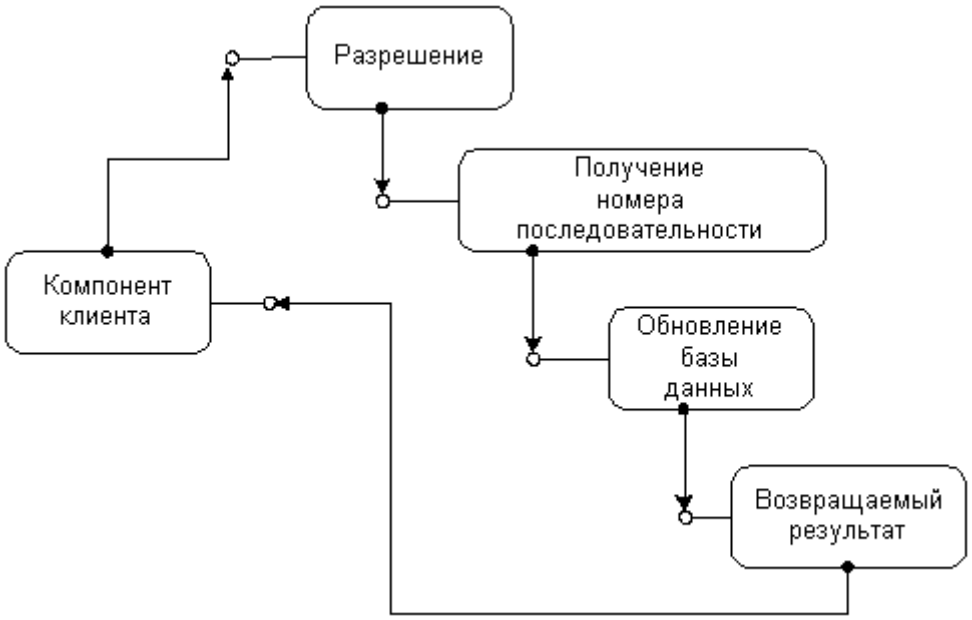



Рис. 9.7 – Схема реализации Поточного метода

### 9.1.8 Эволюция функциональности – контроль версий

Кроме масштабируемости при изменении числа пользователей или количества транзакций, приложению необходимо масштабироваться при возникновении необходимости в новых функциях. С течением времени должны внедряться новые задачи, а существующие – модифицироваться. В идеале все клиенты и компоненты должны модифицироваться одновременно, либо старый компонент должен модифицироваться после обновления всех клиентов – серьезная, с трудом решаемая в условиях большой географической дисперсии сайтов или пользователей проблема.

DCOM предоставляет гибкие механизмы эволюции для клиентов и компонентов. COM и DCOM позволяют клиенту динамично запрашивать функциональность компонента. Вместо объявления своей функциональности в виде монолитной совокупности методов и возможностей, COM-компонент может отвечать различным клиентам по-разному. Клиент, использующий определенную возможность, нуждается только в определенных методах. Клиент может использовать более одной возможности компонента одновременно. Добавление к компоненту новых возможностей никак не влияет на старого, не осведомленного о них клиента, что показано на Рис.9.8.

При такой структуризации компонента появляется новый тип развития: изначально компонент объявляет основной набор возможностей COM-интерфейса, к которым может получить доступ любой клиент. После приобретения компонентом новых возможностей, большая часть этих интерфейсов (а чаще - все) будет по-прежнему необходима; новые же функции и возможности появятся в дополнительных интерфейсах без изменения первоначальных интерфейсов.

Старые клиенты так же имеют доступ к основному набору интерфейсов, как если бы ничего и не менялось. Новые клиенты могут проверить наличие новых интерфейсов и использовать их, если они есть; если же их нет - клиент может корректно деградировать до набора старых интерфейсов.

Поскольку функциональность в DCOM-модели программирования сгруппирована в интерфейсы, вы можете проектировать новые клиентские приложения для работы со старыми серверами, новые серверы – для работы со старыми клиентами или комбинировать эти возможности для удовлетворения ваших запросов. В обычных моделях объектов даже малейшее изменение метода серьезно сказывается на взаимодействии между клиентом и компонентом. Некоторые модели позволяют добавлять новые методы в конец списка методов, но при этом нет возможности безопасной проверки взаимодействия новых методов со старыми компонентами. Рассмотрение этого вопроса в сетевом аспекте вносит еще большие осложнения, ведь обычно кодирование и наличие реальных параметров проводной связи влияет на набор используемых методов и параметров. Добавление или изменение методов и параметров значительно изменяет и сетевой протокол. DCOM устраняет все эти проблемы единственным, изящным, унифицированным решением и для объектной модели, и для сетевого протокола.

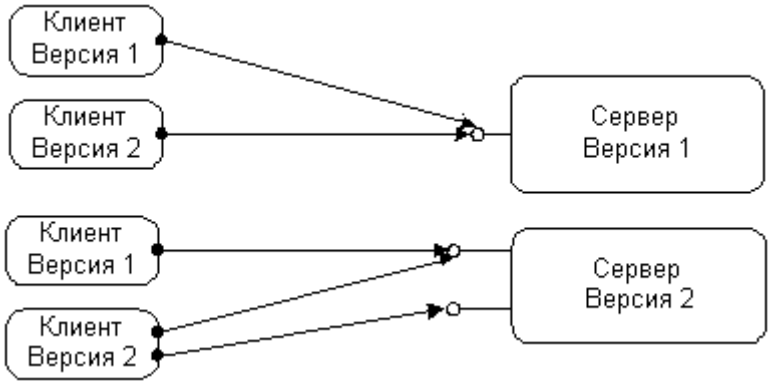



Рис. 9.8 – Структура для устойчивого контроля версий приложений

### 9.1.9 Быстродействие

Масштабируемость нельзя назвать серьезным достоинством при изначально неудовлетворительном быстродействии. Всегда приятно осознавать, что более скоростное аппаратное обеспечение может обеспечить вашему приложению следующий шаг в развитии, но ведь нужно подумать и о потребностях на сегодняшнем уровне. Вы считаете, что не все возможности масштабируемости реализуемы? Что делать, если нет удовлетворительного быстродействия поддержки всех современных языков? Что делать, если компонент, из-за его удаленного местоположения, исключается из работы в процессе в качестве клиента?

В COM и DCOM клиент никогда не видит сам объект сервера, но клиент никогда и не отделяется от сервера системным компонентом до тех пор, пока это абсолютно необходимо. Такая прозрачность достигается поразительно просто: единственный способ, которым клиент может общаться с компонентом - это посредством методов последнего. Клиент получает адреса этих методов из таблицы адресов

методов (vtable). Чтобы вызвать метод компонента, клиент получает адрес метода и вызывает его. Только преимущества программирования в COM по сравнению с традиционной функцией вызова C или Ассемблера позволяют легко находить адрес метода (косвенный вызов функции вместо прямого вызова функции). Если компонент является компонентом в процессе, работающим в том же потоке, что и клиент, вызов метода адресуется компоненту напрямую. При этом не добавляется никакого COM- или системного кода; COM лишь определяет стандарт таблицы адресов методов.

Что случается, когда клиент и компонент в действительности расположены не так близко - в различных потоках, процессах или на машинах, далеко отстоящих друг от друга? COM размещает код своего вызова удаленной процедуры (RPC) в таблице адресов методов и затем, упаковывая каждый вызов метода в стандартное представление, посылает его клиенту; затем происходит его распаковка и восстановление оригинального вызова метода: COM предлагает объектно-ориентированный механизм вызова удаленной процедуры.

Насколько быстр этот механизм RPC? Для определения этого есть различные параметры быстродействия:

- Насколько быстр вызов пустого метода?
- Насколько быстры реальные вызовы методов, посылающие и возвращающие данные?
- Насколько быстра работа по сети?

Вызовам пересекающихся процессов нужно, чтобы параметры заносились в буфер и пересылались в другой процесс. Быстродействие порядка 2000 вызовов в секунду на стандартном desktop-компьютере удовлетворяет большинству требований. Все локальные вызовы ограничиваются скоростью процессора (и в определенной мере доступной памятью) и хорошо масштабируются на многопроцессорных машинах. Удаленные вызовы ограничиваются параметрами первичной сети и в этих условиях быстродействие DCOM на 35% выше быстродействия TCP/IP (для TCP/IP время вызова по сети составляет 2 мс).

Вскоре Microsoft обнародует значения быстродействия для большинства платформ, что покажет способность DCOM к масштабируемости с учетом числа клиентов и числа процессоров сервера.

Эти неофициальные, но повторяемые значения быстродействия показывают 35-процентное превышение быстродействия DCOM над TCP/IP для “пустых” вызовов. Это соотношение уменьшается при реальной работе сервера. Если серверу нужна 1 мс, например, для обновления базы данных, то соотношение уменьшается до 23 % и до 17 % - если серверу необходимо 2 мс.

Преимущества DCOM в масштабируемости и быстродействии в целом могут быть получены только при использовании хорошо организованного управления объединением потоков и тестовых протоколов (pinging protocols). Большая часть распределенных приложений даже при значительных вложениях не получат

приближенно такого же увеличения быстродействия, как при использовании стандартизованных DCOM-протокола и модели программирования.

### 9.1.10 Полоса пропускания и латентность

Распределенные приложения пользуются преимуществами сети, связывая компоненты. DCOM полностью скрывает тот факт, что компоненты работают на разных компьютерах. Однако на практике приложения вынуждены учитывать два параметра сетевого соединения:

- Полоса пропускания: размер параметров вызова метода постоянно влияет на время, которое требуется для выполнения вызова.
- Латентность: физическое расстояние и число задействованных элементов сети (таких как маршрутизаторы и линии коммуникаций) значительно влияют на задержку даже маленьких пакетов данных. В случае глобальной сети, такой как Интернет, эта задержка может измеряться секундами.

Как DCOM помогает приложению работать при таких ограничениях? DCOM минимизирует передачу данных по сети там, где это возможно для того, чтобы уменьшить латентность сети. Наиболее предпочтительный транспортный протокол для DCOM - это UDP-набор протокола TCP/IP без функции соединения: природа этого протокола позволяет выполнить определенную оптимизацию посредством слияния многих низкоуровневых пакетов с данными и тестовыми сообщениями. Даже при работе с протоколами, ориентированными на соединение, DCOM продолжает предоставлять значительные преимущества по сравнению с пользовательскими протоколами приложений.

### 9.1.11 Распределенное управление соединением между протоколами

Многие протоколы нуждаются в постоянном управлении. Компонентам нужно знать, что на машине клиента возникла серьезная авария аппаратного обеспечения или сетевое соединение между клиентом и компонентом прервалось на длительное время.

Общим решением этой проблемы является постоянная пересылка сообщений с определенной периодичностью (pinging). Если сервер не получает сообщения в течение определенного времени, это значит, что связи с клиентом нет.

DCOM использует такие сообщения для каждой из машин. Даже если машина клиента использует 100 компонентов с машины сервера, единственное тестовое сообщение поддерживает связь всех клиентов. В дополнение к использованию тестовых сообщений, DCOM минимизирует размер этих тестовых сообщений путем посылки разницы между ними (delta-pinging). Вместо посылки 100 идентификаторов клиентов создается мета-идентификатор, представляющий все 100 ссылок. При изменении набора ссылок пересылается только разница между двумя последовательными наборами ссылок. И, наконец, DCOM накладывает тестовое сообщение на обычные пересылаемые сообщения. Только в случае, если машина клиента простаивает по отношению к машине сервера, с 2-минутным интервалом пересылаются сами тестовые сообщения (рис.9.9).

DCOM позволяет различным приложениям (даже из различных источников) использовать единое, оптимизированное, постоянное управление и протокол определения аварий сети, незначительно уменьшая полосу пропускания. Если на сервере работает 100 разных приложений с сотней различных пользовательских протоколов, этот сервер должен принять по одному тестовому сообщению для каждого из приложений, от каждого из подключенных клиентов. Загрузка сети в целом может быть уменьшена только в случае, если эти протоколы некоторым образом координируют свою тестовую стратегию. DCOM автоматически предоставляет такое координирование среди пользовательских COM-протоколов.

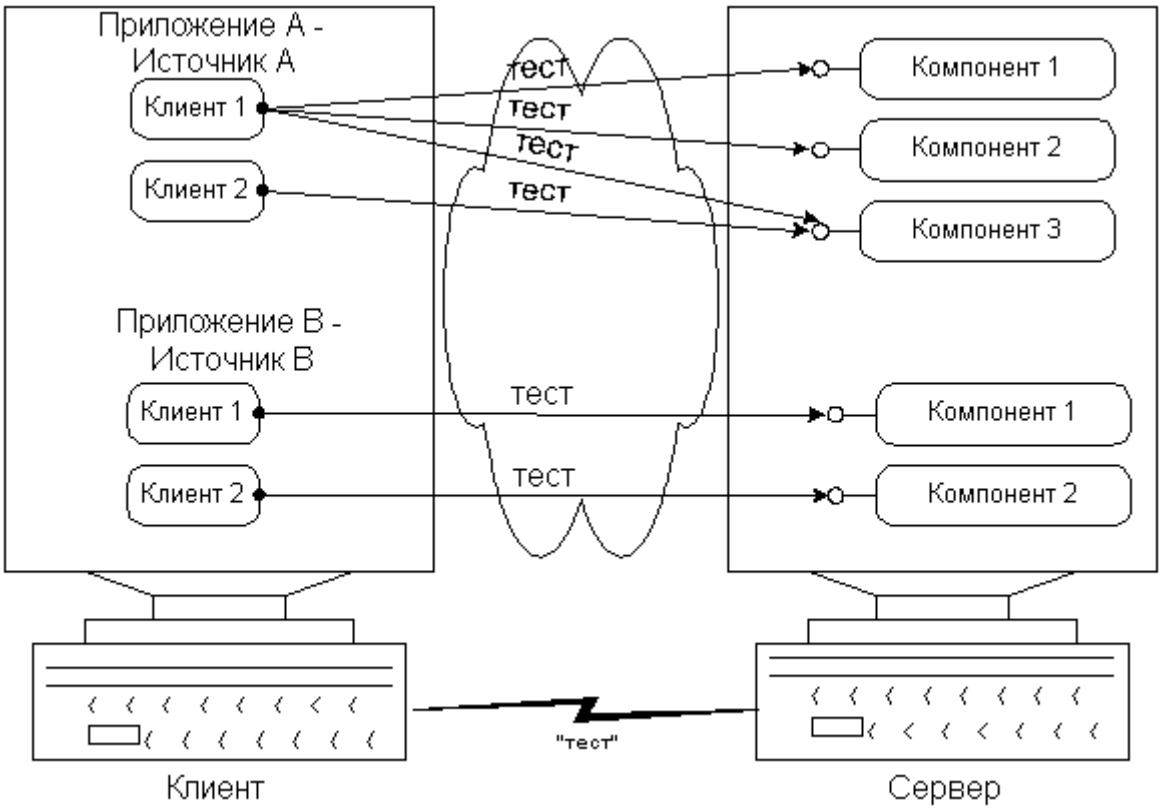



Рис. 9.9 - Совмещенное постоянное управление

### 9.1.12 Оптимизация сетевого обмена

Общей проблемой при разработке распределенных приложений является интенсивный сетевой обмен между компонентами различных машин. В Интернет каждый из таких сетевых обменов вызывает задержку в 1 секунду, а чаще – значительно большую. Даже в случае быстрой локальной сети время обмена измеряется в миллисекундах – достаточно много для таких масштабов.

Общим способом уменьшения интенсивности сетевого обмена является “увязывание” многочисленных вызовов методов в единственный запрос метода (пакетирование или упаковка). DCOM использует этот способ для таких задач, как подключение к объекту или создание нового объекта и запрос его функциональности. Недостатком такого способа для большинства компонентов является то, что модель программирования сильно отличается в локальном и удаленном случаях.

**Пример.** Компонент базы данных предусматривает метод перечисления результатов запроса по одной или более строк за один раз. В локальном случае разработчик может просто использовать этот метод для последовательного добавления строк к окну списка. В удаленном случае это решение приводит к сетевому обмену при перечислении каждой строки. При использовании метода со способом пакетирования разработчику необходимо локализовать достаточно большой буфер для того, чтобы все строки хранились в очередности и пересылались одним вызовом с последующим поочередным добавлением к окну. Поскольку при этом модель программирования значительно изменяется, то, чтобы приложение работало эффективно и в распределенном окружении, разработчик должен пойти на некоторые компромиссы в конструировании..

DCOM облегчает дизайнерам компонентов выполнение подобного пакетирования без того, чтобы клиентам нужно было использовать API пакетирования. Механизм маршалинга DCOM позволяет компоненту передавать код, называемый “заместителем объекта” (proxy object) и позволяющий перехватывать многочисленные вызовы методов и упаковывать их в единственный вызов удаленной процедуры клиенту:

**Пример.** Разработчик из предыдущего примера продолжает перечислять методы один за другим, поскольку это способ, который необходим логике приложения. (Это нужно списку API). Однако, первый вызов для начала перечисления пересылается в заместитель объекта, определенный приложением, который вызывает все строки (или набор строк) и кэширует их в объект-заместитель. Затем последующие вызовы исходят из этого кэша без дополнительных сетевых обменов. Разработчик продолжает работать с простой моделью программирования, хотя в целом приложение уже оптимизировано (рис.9.10).

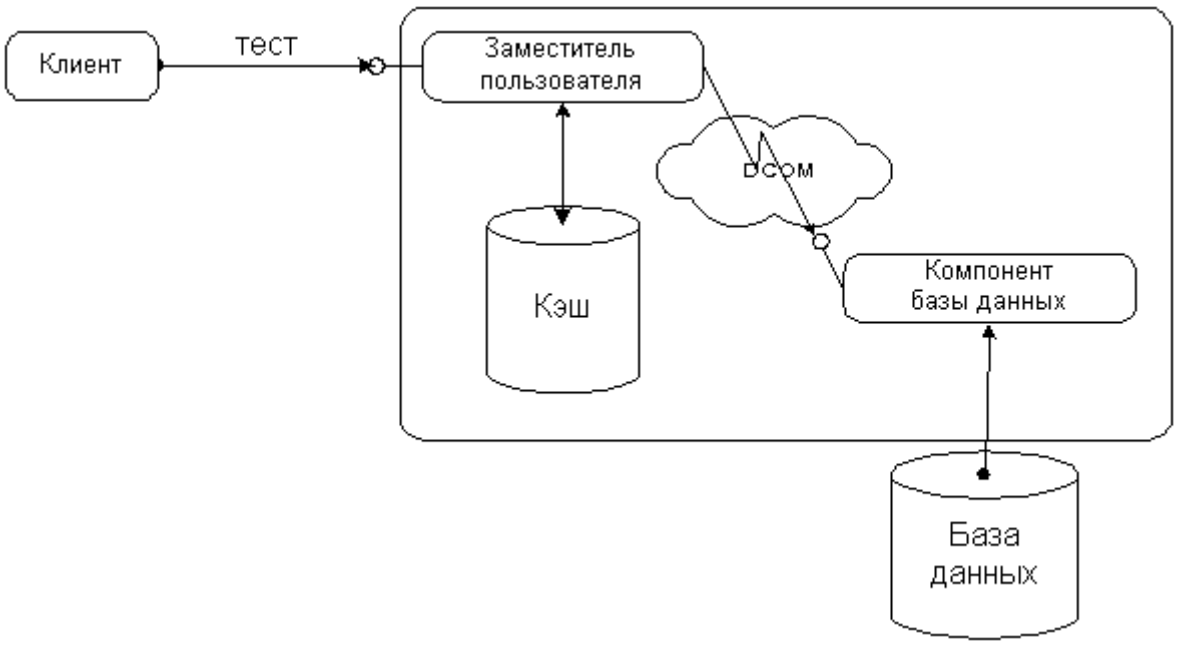



Рис. 9.10 – Компонентная модель: кэширование со стороны клиента

Кроме того, DCOM позволяет выполнять эффективную переадресацию с одного компонента на другой. Если компонент содержит ссылку на другой компонент

второй машины, он может передать эту ссылку клиенту, работающему на третьей машине (ссылка клиента на другой компонент, работающий на другой машине). Когда клиент использует эту ссылку, он напрямую соединяется со вторым компонентом. DCOM стыкует эти ссылки и позволяет оригиналу компонента и машине выйти из этой схемы взаимодействия. Это предоставляет пользователю сервисы директорий, позволяющие возвращать ссылки на широкий диапазон удаленных компонентов.

**Пример 1.** Приложение “шахматы” позволяет игрокам, ищущим партнера, регистрироваться в сервисе каталога шахмат. Другие игроки могут запросить этот список или встать в очередь. При выборе игроком партнера сервис каталога шахмат возвращает ссылку на компонент клиента партнера. DCOM автоматически соединяет двух игроков; в дальнейших транзакциях сервис каталога более не участвует.

**Пример 2.** Компонент “Брокер” имеет соединение с 20 машинами с серверами, на которых работают идентичные компоненты. Происходит постоянная проверка загрузки серверов и определение того, изменилось ли количество серверов. Когда клиент нуждается в компоненте, он подключается к компоненту “брокер”, который возвращает ссылку на компонент сервера с минимальной загрузкой. DCOM автоматически соединяет клиента с сервером; после этого компонент “брокер” выпадает из схемы взаимодействия, что показано на Рис.9.11.

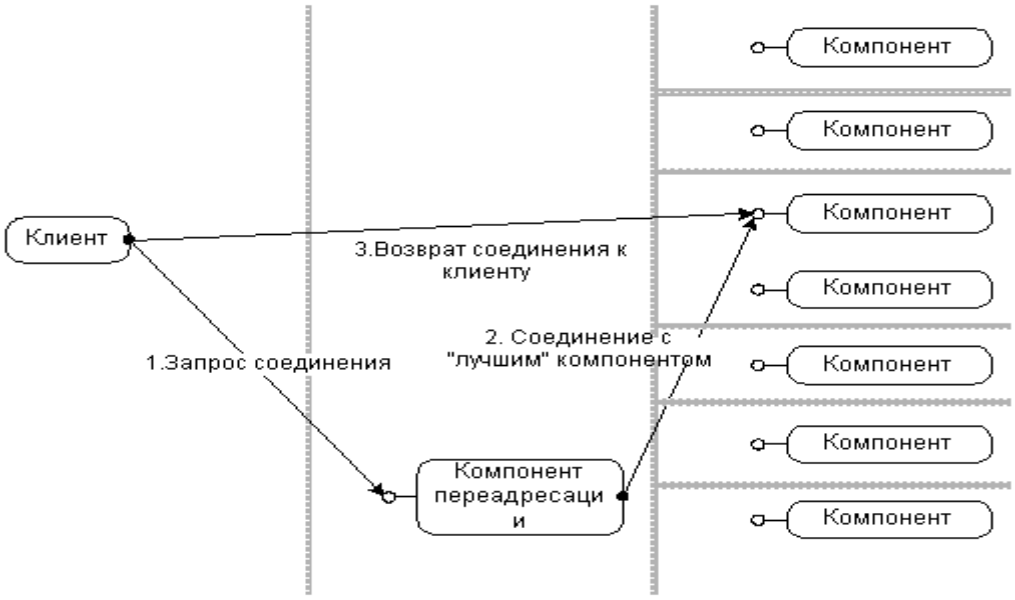



Рис. 9.11 – Схема переадресации компонентов

Если необходимо, DCOM даже позволяет компонентам включаться в произвольные пользовательские протоколы, использование которых подразумевает работу вне механизма DCOM. Компонент может использовать пользовательский маршалинг для пересылки заместителя объекта в процесс клиента, который в дальнейшем может использовать любой произвольный протокол для общения с компонентом.

**Пример 1.** Компонент со стороны сервера может использовать ODBC-соединение для работы с базой данных SQL-сервера. При обращении клиента к этому объекту, прямая связь между машиной клиента и SQL-сервером базы данных (с использованием ODBC) может оказаться более эффективной, нежели использование DCOM для связи с машиной сервера, которая работает с SQL-сервером базы данных. Используя пользовательский маршалинг DCOM, компонент базы данных может первоначально скопироваться на машину клиента и подключиться к SQL-серверу даже без уведомления клиента, что тот подключен не к компоненту базы данных сервера, а к локальной копии компонента той же базы данных.

**Пример 2.** Торговой системе необходимо наличие двух типов коммуникационных механизмов: безопасного канала с аутентификацией от клиента до центральной системы, который используется для размещения и удаления объектов, и канала распространения, который пересылает информацию о заказах одновременно всем подключенным клиентам. В то время, как канал клиент/сервер эффективно и легко организуется современным безопасным и синхронизированным DCOM-соединением, канал оповещения для обеспечения распределения информации между многими слушателями требует более сложного механизма. DCOM позволяет безболезненно встроить такой пользовательский протокол (“надежное оповещение”) в архитектуру приложения: компонент приемника данных может инкапсулировать этот протокол и сделать его полностью прозрачным как для клиента, так и для сервера, что показано на Рис.9.12. Для небольших систем с малым количеством пользователей могут использоваться стандартные протоколы DCOM “точка-точка “(point-to-point), в то время как сайты с большим числом пользователей должны все-таки использовать усложненный протокол оповещения пользователей. Если DCOM предложит подобный стандартизованный транспортный механизм в будущем, приложение сможет безболезненно перейти к новому протоколу.

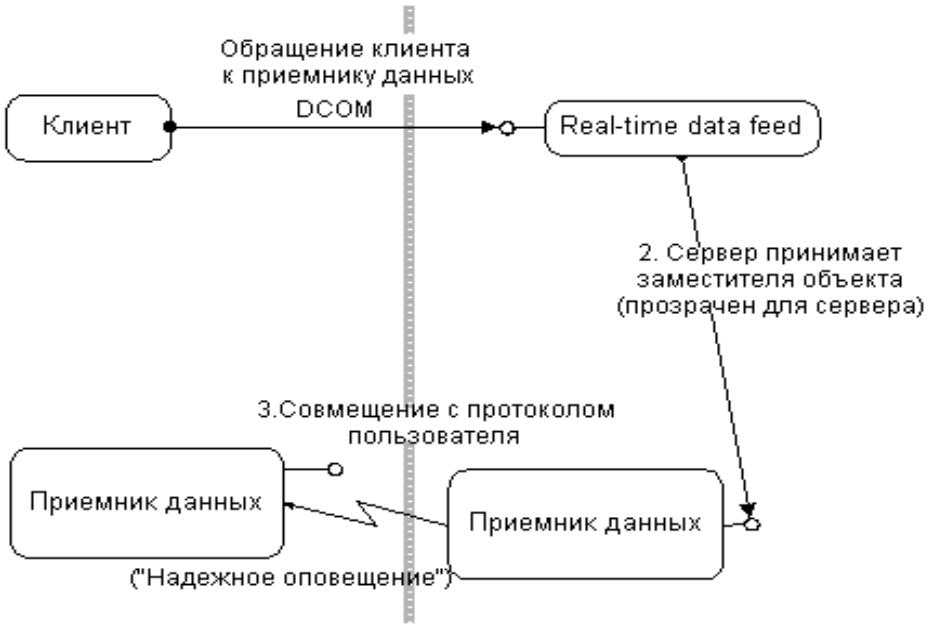



Рис. 9.12 – Замена DCOM пользовательскими протоколами

DCOM предоставляет множество способов совместить реальный сетевой протокол с сетевым траффиком без изменения взаимодействия клиента с компонентом: кэширование со стороны клиента, переадресация ссылок и замена сетевого транспорта при необходимости – лишь часть возможных способов.

### 9.1.13 Безопасность

Использование сети для распределения приложения - это сложная задача не только из-за таких физических ограничений, как полоса пропускания и латентность. Это также вынуждает использовать меры безопасности для клиентов и компонентов. Поскольку сейчас любому физически доступно большое число операций при доступе к сети, доступ к таким действиям должен ограничиваться на более высоком уровне.

При отсутствии поддержки безопасности со стороны разработанной распределенной платформы, каждое приложение должно было бы применять свои механизмы обеспечения безопасности. Обычный механизм использует запрос имени пользователя и пароля – обычно зашифрованного. Приложение сопоставляет эти идентификаторы с каталогом или базой данных пользователя и возвращает некоторый динамичный идентификатор, используемый для будущих вызовов метода. При всех последующих вызовах метода безопасности клиенты должны будут пройти этот идентификатор безопасности. Каждое приложение должно уметь хранить список имен пользователей и паролей, управлять им, защищать каталог пользователя от несанкционированного доступа и управлять изменением паролей, а также обеспечивать надежность при таком действии, как пересылка паролей по сети.

Распределенная платформа должна предусматривать наличие framework, отвечающего за безопасность для того, чтобы была обеспечена возможность различать отдельных клиентов и отдельные группы клиентов. Тогда система или приложение будут иметь возможность определять, кто пытается начать работать с компонентом. DCOM использует framework повышенной безопасности, предлагаемый Windows NT. Windows NT предусматривает серьезный набор встроенных провайдеров безопасности, поддерживающий многочисленные механизмы идентификации и аутентификации, от традиционных моделей доменов доверия (trusted-domain) до нецентрализованно управляемых, хорошо масштабируемых механизмов безопасности. Центральной частью framework, отвечающего за безопасность, является каталог пользователя, который содержит информацию, необходимую для подтверждения прав пользователя (имя пользователя, пароль). Большинство DCOM-реализаций на отличных от Windows NT платформах предусматривают такой же или подобный механизм, какой бы провайдер безопасности ни был доступен для данной платформы. Большая часть UNIX-реализаций DCOM будут включать Windows NT-совместимый провайдер безопасности.

Перед тем, как ближе ознакомиться с Windows NT-провайдерами каталогов и безопасности, давайте обсудим, как DCOM использует этот общий framework безопасности для облегчения создания защищенных приложений.

### 9.1.14 Конфигурирование безопасности

DCOM может обеспечить безопасность распределенному приложению без встраивания специализированного кодирования безопасности в клиент или компонент. Модель программирования DCOM скрывает потребность в безопасности со стороны компонента точно так же, как скрывает и его местонахождение. Тот же двоичный код, который работает на одной машине, где безопасность – не главное требование, может использоваться и в распределенном приложении с обеспечением защищенности.

DCOM достигает подобной прозрачности в безопасности, позволяя разработчикам и администраторам конфигурировать установки защищенности для каждого компонента. Точно так же, как файловая система Windows NT позволяет администратору установить списки управления доступом (ACL) для файлов и каталогов, DCOM содержит списки управления доступом для компонентов. Эти списки показывают, какие пользователи или группы пользователей имеют право доступа к компонентам определенного класса. Эти списки могут легко конфигурироваться инструментом конфигурирования DCOM (DCOMCNFG) или программироваться с помощью реестра Windows NT и функций обеспечения безопасности Win32.

Когда бы клиент ни вызвал метод или создал пример компонента, DCOM получает текущее пользовательское имя клиента, связанного с текущим процессом (на самом деле – текущий исполняемый поток). Windows NT гарантирует, что данный пользовательский идентификатор аутентичен. После этого DCOM передает имя пользователя машине или процессу, где работает этот компонент. На машине компонента DCOM снова проверяет имя пользователя, используя аутентификационный механизм, и проверяет список управления доступом для данного компонента (на самом деле — для первого компонента, работающего в процессе, содержащего интересующий компонент. Более детально это описано в Белых Страницах “Архитектура DCOM”.) Если имя клиента не включено в этот список (прямо или косвенно – в качестве члена группы пользователей), DCOM просто отклоняет этот вызов еще до того, как компонент вовлекся в работу. Этот обычный механизм безопасности полностью прозрачен как для клиента, так и для компонента и оптимизирован. Он базируется на framework безопасности Windows NT, который является наиболее интенсивно используемой (и оптимизированной!) частью операционной системы Windows NT: при каждом доступе к файлу или даже к такому примитиву синхронизации потоков, как сигнализация или событие, Windows NT выполняет идентичную проверку доступа. Тот факт, что Windows NT по-прежнему может неплохо конкурировать в быстродействии с другими операционными системами и сетевыми операционными системами, показывает, насколько эффективен этот механизм безопасности (рис.9.13).

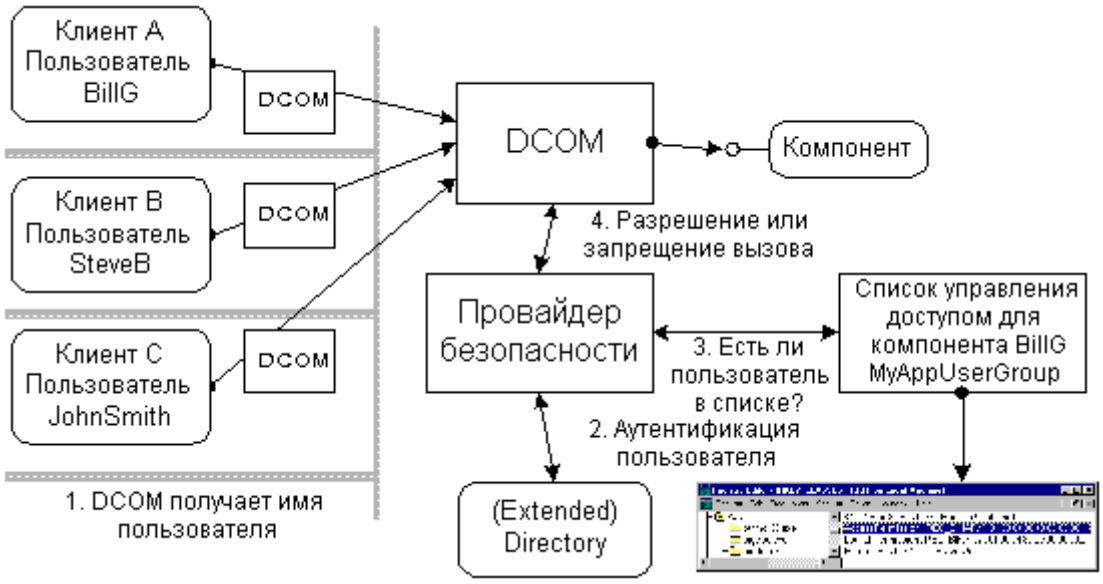



Рисунок 9.13 – Структура конфигурирования безопасности

DCOM предлагает чрезвычайно эффективный механизм обеспечения безопасности, который позволяет разработчику создавать распределенные приложения, не опасаясь за его защищенность. Любой провайдер безопасности, поддерживаемый Windows ОС, может использоваться механизмом безопасности DCOM.

### 9.1.15 DOT.NET и будущее COM

В 2002 году была официально выпущена платформа Microsoft .NET, которая на сегодняшний день объявлена Microsoft рекомендуемой основой для создания приложений и компонентов под Windows. По этой причине в.NET включены и средства, позволяющие обращаться к компонентам COM из приложений .NET, и наоборот. По словам представителей Майкрософт, COM (точнее, COM+) и .NET являются отлично взаимодополняющими технологиями.

**DCOM через интернет и решение проблемы XP SP2**

В 2009 году DComLab опубликовал коммерческий продукт ComBridge. При использовании ComBridge для работы по DCOM через Интернет не требуется CIS, не используется 135 порт, в локальной сети не требуются настройки dcomcnfg. ComBridge встраивается в транспортный уровень DCOM, полностью выделяя весь трафик созданного объекта и всех полученных из него объектов в отдельный поток.

### 9.1.16 Другие распределенные технологии Майкрософт – OPC и OLE

**Программные технологии OPC**

OPC (OLE for Process Control) — семейство программных технологий, предоставляющих единый интерфейс для управления объектами автоматизации и технологическими процессами. Многие из OPC-протоколов базируются на Windows-технологиях: OLE, ActiveX, COM/DCOM. Такие OPC-протоколы, как OPC XML DA и OPC UA, являются платформо-независимыми.

**Технология OLE**

OLE (англ. Object Linking and Embedding, произносится как oh-lay [олэй] — Связывание и встраивание объекта) — технология связывания и внедрения объектов в другие документы и объекты, разработанные корпорацией Майкрософт.

OLE позволяет передавать часть работы от одной программы редактирования к другой и возвращать результаты назад. Например, установленная на персональном компьютере издательская система может послать некий текст на обработку в текстовый редактор, либо некоторое изображение в редактор изображений с помощью OLE-технологии.

Технология часто критикуется за неоправданную сложность, конкретно:

- необходимость использования двух языков программирования (.idl для описания интерфейсов и, обычно, C++ для написания реализаций). Необходимость возникает только при создании собственных интерфейсов, и не возникает в случае, если разработчик ограничил себя использованием готовых интерфейсов.
- необходимость «прокладочного» кода (в его роли обычно выступает ATL) для того, чтобы создать COM-объект на базе C++ класса. Хотя этот код и тривиален в использовании для опытного человека, он не очень прост для начинающих. Как и в предыдущем пункте, эта проблема возникает только при написании собственных классов и не возникает при одном лишь использовании стандартных чужих классов (для которых MS разработал библиотеку смарт-пойнтеров — comdef.h, _com_ptr_t<Interface>, эта библиотека делает использование COM-объектов тривиальным).
- необходимость регистрации компонент в реестре операционной системы, причем при этом в качестве идентификатора класса используется нечитаемый человеком GUID (хотя его и возможно дополнить читаемым именем).
- инфраструктура remoting (удаленного вызова методов) использует бинарный формат запросов и ответов, являясь расширением DCE RPC. Это приводит к возникновению огромной «поверхности уязвимости» с точки зрения безопасности, и не раз приводило к крупным эпидемиям вредоносного ПО (MSBlaster).
- инфраструктура remoting использует по умолчанию (вслед за DCE RPC) динамически назначаемые номера TCP- и UDP-портов, что делает её крайне сложной в настройке при наличии межсетевых экранов.
- обработка ошибок. В COM принято использовать 32-битные коды ошибки HRESULT, которые имеют значения вроде 0x80070123, и совершенно не читаемы человеком (хотя в последнее время все они легко ищутся поисковыми машинами Интернета).

Кроме того, runtime type information в COM, известная под названием type libraries, поддерживается только для т. н. Automation-compatible интерфейсов, имеющих огромные ограничения на типы параметров (массивы — только SAFEARRAY, строки — только BSTR, никаких произвольных структур, только числа, дата/время, массивы, строки и ссылки на другие Automation-compatible объекты).

Заметно, однако, что многие из этих недостатков являются платой за достоинство COM — независимость от языка программирования и исполняющей среды, и не существуют в «истинно объектных» языках, таких, как C#, или же (прекращенная) реализация Java компании Microsoft. Эти языки предоставляют и полную runtime type information, и отсутствие необходимости регистрации, и возможность написания как интерфейсов, так и классов стандартным для языка образом, без «прокладок» вроде ATL. Так, в MS J++ любой класс Java тривиально публиковался внешнему миру как класс COM, достаточно было лишь регистрации. То же существует и в C#.

С противоположной стороны, «истинно объектные» языки либо вообще не способны стыковаться с компонентами из других объектных языков и требуют написания всей системы (и нижележащих подсистем и фреймворков) «сверху донизу» на одном языке в одной исполняющей среде (Java, Objective C), либо же налагают такое же требование хотя и не на язык, но на исполняющую среду (.NET, языки C#, C++ managed и VB.NET).

Более новые аналогичные технологии (например, в мире .NET) пытаются решить эти проблемы. Там обычно стек remoting полиморфен и кастомизируем, что дает возможность самостоятельно выбирать формат вопросов/ответов и транспортный протокол (по умолчанию используется уже не DCE RPC, а SOAP, в качестве формата данных — XML, а в качестве транспорта — HTTP, который не полагается на динамические номера портов).

Использование механизма позднего связывания может существенно снизить производительность по сравнению, например, с вызовом экспортируемой функции из динамической библиотеки. Однако этот механизм применяется только в скриптовых языках, и только в том случае, если язык не поддерживает объявление ссылок на объекты как ссылок на COM-интерфейсы из type libraries (в виде Dim obj As Excel.Workbook), а поддерживает только абстрактные COM-объекты (в виде Dim obj As Object).

## 9.2 Распределенный протокол на основе технологии Corba

### 9.2.1 Объектная модель CORBA

Объектная модель CORBA определяет взаимодействие между клиентами и серверами. Клиенты – это приложения, которые запрашивают сервисы. Серверы – это приложения, представляющие сервисы.

Объекты-серверы содержат набор сервисов, разделяемых между многими клиентами. Операция указывает запрашиваемый сервис объекта-сервера. Интерфейсы объектов есть описание множества операций, которые могут быть вызваны клиентами данного объекта. Реализации объектов – это приложения, реально исполняющие сервисы, запрашиваемые клиентами.

### 9.2.2 Брокеры запросов Их функции и особенности

Необходимая спецификация доступа к распределенным приложениям в гетерогенной среде была разработана OMG и получила название Object Management Architecture (OMA).

OMA состоит из четырех основных компонент, представляющих собой спецификации различных уровней поддержки приложений (рис. 9.14):

- архитектуры брокера запросов объектов (CORBA – Common Object Request Broker Architecture) – устанавливает базовые механизмы взаимодействия объектов в гетерогенной зоне;
- сервисов объектов (Object services) – являются основными системными службами, используемыми разработчиками для создания приложений;
- универсальных средств (Common Facilities) – ориентированы на поддержку пользовательских приложений, таких, как электронная почта, средства печати и т.д.;
- объектов приложений (Application Objects) – предназначены для решения конкретных прикладных задач.

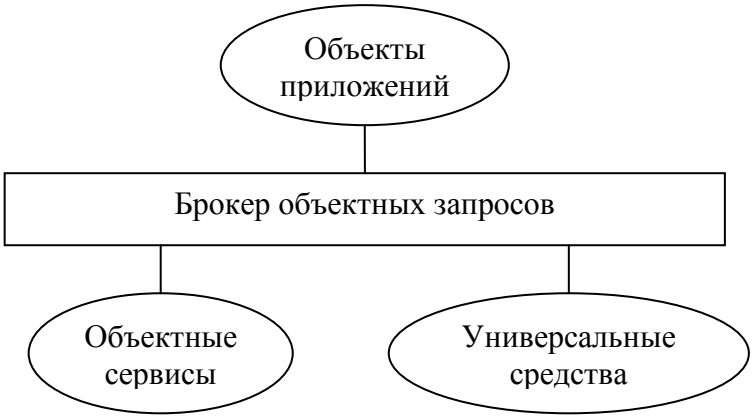



Рис. 9.14. Архитектура управления объектами (OMA)

COBRA определяет механизм, обеспечивающий взаимодействие приложений в распределенной среде.

Главными компонентами стандарта CORBA являются:

- объектный брокер запросов (Object Request Broker);
- язык определения интерфейсов (Interface Definition Language);
- объектный адаптер (Object Adapter);
- депозитарий интерфейсов (Interface Repository).

### 9.2.3 Место CORBA в семиуровневой модели OSI

Наличие большого количества сетевых протоколов в разных операционных системах потребовало разработки спецификации сетевых соединений. Семиуровневая модель Open System Interconnection (OSI) обеспечивает описание сервисов, которые каждый уровень должен представлять для реализации соединения. Низший уровень – физический – определяет доступ к физической линии. Уровень данных обеспечивает достоверную передачу данных по физической линии. Сетевой уровень имеет дело с установкой соединения и маршрутизацией. Транспортный уровень отвечает за достоверную передачу до точки назначения. Уровень сессий обеспечивает управление соединением. Уровень представлений описывает синтаксис данных и обеспечивает прозрачность

для приложений. Последний уровень – уровень приложений – обеспечивает соответствующие сетевые функции для конечного пользователя.

Концептуально CORBA относится к уровням приложений и представлений. Она обеспечивает возможность построения распределенных систем и приложений на самом высоком уровне абстракции в рамках стандарта OSI. С ее помощью возможно изолировать клиентские программы от низкоуровневых, гетерогенных характеристик информационных систем. Характерные особенности разработки по технологии CORBA заключаются в следующем.

Язык описания интерфейсов OMG IDL позволяет определить интерфейс, независимый от языка программирования, используемого для реализации.

Высокий уровень абстракции CORBA в семиуровневой модели OSI позволяет программисту не работать с низкоуровневыми протоколами.

Программисту не требуется информация о реальном месте расположения сервера и способе его активизации.

Разработка клиентской программы не зависит от серверной операционной системы и аппаратной платформы.

### 9.2.4 Брокерная архитектура CORBA

***Объектный брокер запросов (ORB)***

Спецификация CORBA разработана для обеспечения возможности интеграции совершенно различных объектных систем.

Его задачей является представление механизма выполнения запроса, сделанного клиентом: поиск объекта, к которому относится данных запрос, передача необходимых данных, подготовка объекта к обработке. Интерфейс, с помощью которого клиент может запрашивать выполнение необходимых операций, не зависит от местонахождения объекта и языка программирования, с помощью которого он реализован. Клиент может запрашивать выполнение операций с помощью ORB несколькими способами. На рис. 5.3 показаны способы возможного взаимодействия объектов.

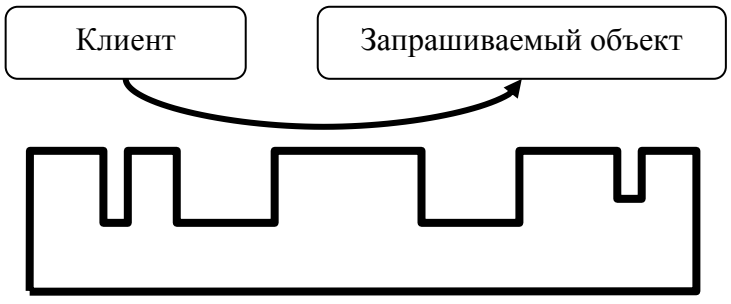



Рис. 9.15. Схема работы объектного брокера (ORB)

***Структура ORB***

Рис. 9.15 показывает запрос, посылаемый пользователем к системе объектной реализации. Клиент – это пользователь, который желает выполнить операцию над объектом в системе объектной реализации.

Система объектной реализации – это фактически код и данные, которые физически содержатся в объекте. ORB ответствен за все механизмы, требуемые,

чтобы найти объектную реализацию для запроса, подготовить ее, реализовать объект и содержащиеся в нем метод и отправить данные, составляющие запрос, пользователю. Интерфейс, с которым имеет дело пользователь, полностью независим от того, где объект размещен, какой язык программирования в нем используется, где это выполнено, а также в отношении любого другого аспекта, который совершенно не влияет на интерфейс объекта.

Рис. 9.16 показывает структуру индивидуального ORB. Интерфейсы к ORB показываются закрашенными прямоугольниками, и стрелки указывают, вызывается ли ORB или выполняет вызов-запрос при помощи интерфейса.

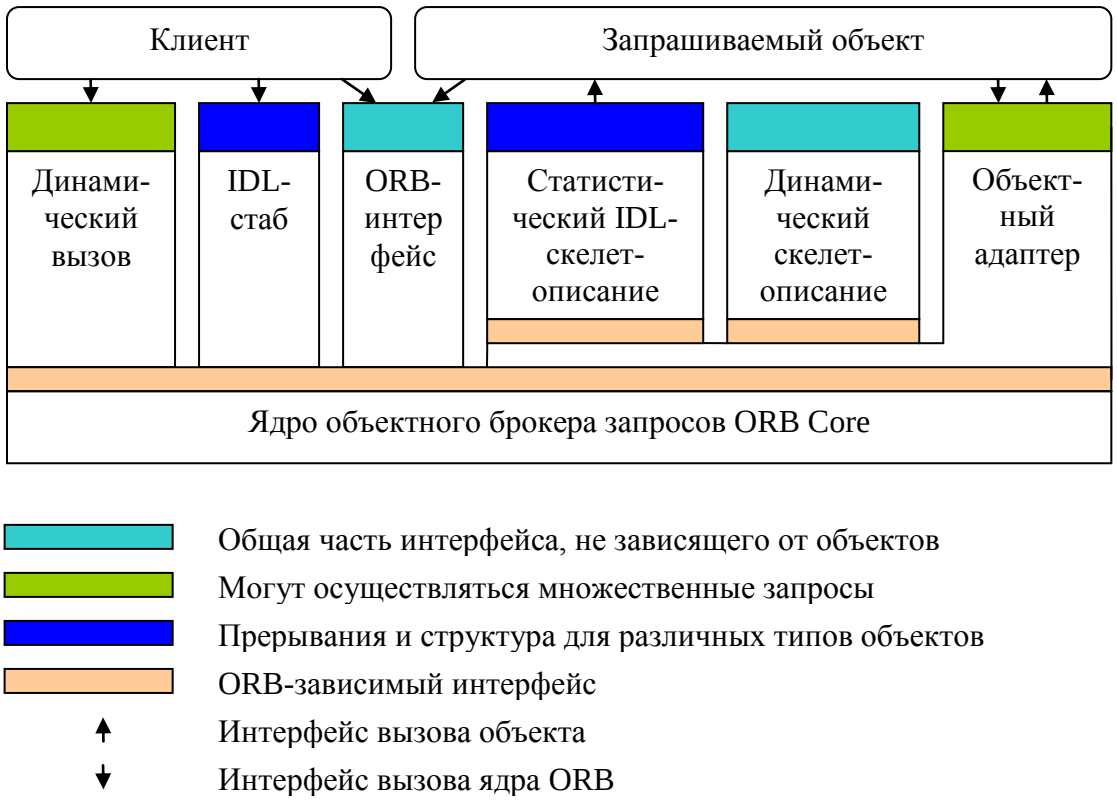



Рис. 9.16. Структура интерфейса объектного брокера запросов

Вызов операций разделяемого объекта-сервера может быть сделан статическим (IDL-стаб) и динамическим (Dynamic Invocation Interface) способом. Интерфейсы описываются, используя язык определения интерфейсов, получивший название OMG Interface Definition Language (IDL). В случае статического вызова эти описания интерфейсов отображаются в программный код на языках C, C++, Smalltalk. Информация об интерфейсах объектов может быть получена клиентом двумя способами: статически (compile time) и динамически (runtime). Интерфейсы могут быть также указаны с помощью службы репозитория интерфейсов (Interface Repository). Этот сервис представляет интерфейсы как объекты, обеспечивая доступ к ним во время работы приложения (в runtime-режиме). (Описание динамического вызова интерфейсов и репозитория интерфейсов см. ниже).

Пользователь выполняет запрос при наличии доступа к объекту, когда знает и тип объекта и желаемую функцию выполнения. Пользователь инициализирует запрос, вызывая подпрограммы статической настройки, которые являются специфическими для объекта, или строит динамически оформляемый запрос.

### 9.2.5 Объектный адаптер (Object Adapter)

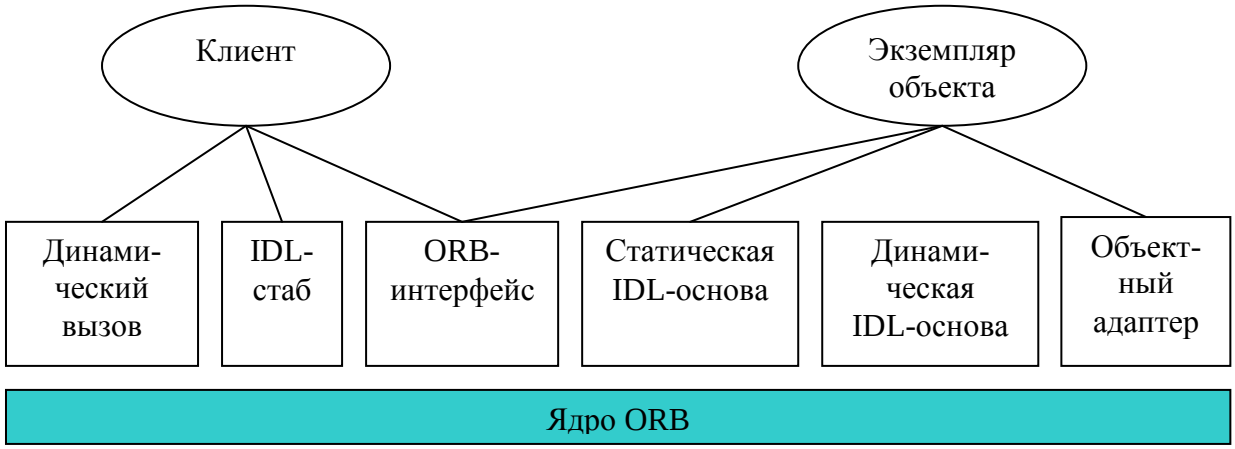



Рис.9.17. Объектный адаптер как компонент в структуре интерфейса ORB

Главная функция объектного адаптера, используемого для реализации объекта, - предоставление клиентам доступа к сервисам объектного брокера запросов ORB, что показано на Рис.9.17. Объектный адаптер обеспечивает все низкоуровневые средства для связи объекта с его клиентами. Основными задачами объектного адаптера являются:

- генерация ссылок на удаленные объекты;
- вызов метода объекта, определенного в IDL;
- обеспечение безопасности взаимодействия;
- активизация и деактивизация объектов;
- установление соответствия между ссылками на удаленные объекты (proxy) и реальными экземплярами объектов;
- регистрация объектов.

Спецификация OMG CORBA определяет базовый объектный адаптер (Basic Object Adapter – BOA), который должен быть реализован во всех брокерах запросов.

## См. также

- Список протоколов автоматизации
- **Systems Network Architecture** (SNA), разработанная корпорацией IBM
- **Distributed Systems Architecture** (DSAbg), разработанная компанией Honeywell Bull

# Литература


- *Network Protocols Handbook*. — *Javvin Technologies, 2005*. — *ISBN 9780974094526*.


# Общие ссылки

1. Protocol Encapsulation Chart (англ.) — PDF-файл, описывающий отношения между общими протоколами и сетевой моделью OSI.
2. Network Protocols Acronyms and Abbreviations (англ.) — список сетевых протоколов и их аббревиатуры, упорядоченный по алфавиту.
3. http://book.itep.ru/4/4/rpsl.htm
4. http://bourabai.kz/dbt/protocols.htm
5. http://dic.academic.ru/dic.nsf/ruwiki/685997
6. http://compnets.narod.ru/6-2.html
7. https://www.w3.org/TR/soap11
8. https://www.w3.org/TR/wsdl

**Ссылки к разд. 6.3**


[1] Internet routing registry. procedures. http://www.ra.net/RADB.tools.docs/, http://www.ripe.net/db/doc.html.

[4] C. Alaettinouglu, D. Meyer, and J. Schmitz. Application of routing policy specification language (rpsl) on the internet. Work in Progress.

[5] T. Bates. Specifying an `internet router' in the routing registry. Technical Report RIPE-122, RIPE, RIPE NCC, Amsterdam, Netherlands, October 1994.

[6] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of ip routing policies in a routing registry. Technical Report ripe-181, RIPE, RIPE NCC, Amsterdam, Netherlands, October 1994.

[7] Bates, T., Gerich, E., Joncheray, L., Jouanigot,

J-M., Karrenberg, D., Terpstra, M. and J. Yu, " Representation of IP Routing Policies in a Routing Registry", RFC-1786, March 1995.

[8] T. Bates, J-M. Jouanigot, D. Karrenberg, P. Lothberg, and M. Terpstra. Representation of ip routing policies in the ripe database. Technical Report ripe-81, RIPE, RIPE NCC, Amsterdam, Netherlands, February 1993.

[12] D. Karrenberg and T. Bates. Description of inter-as networks in the ripe routing registry. Technical Report RIPE-104, RIPE, RIPE NCC, Amsterdam, Netherlands, December 1993.

[13] D. Karrenberg and M. Terpstra. Authorisation and notification of changes in the ripe database. Technical Report ripe-120, RIPE, RIPE NCC, Amsterdam, Netherlands, October 1994.

[16] A. M. R. Magee. Ripe ncc database documentation. Technical Report RIPE-157, RIPE, RIPE NCC, Amsterdam, Netherlands, May 1997.

[17] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC-1034, November 1987.

[20] C. Villamizar, C. Alaettinouglu, D. Meyer, S. Murphy, and C. Orange. Routing policy system security", Work in Progress.