# Системная интеграция и корпоративные информационные системы

## Лекция 4: Стандарты объектно-ориентированного взаимодействия

# Модель OSI

**Уровни модели OSI/ISO и их характеристики**

| № | Название | Функции | Пример протокола | Реализация |
|---|---|---|---|---|
| 1 | Физический | Передача битов по физическим линиям связи. Определяет механические, электрические и оптические характеристики кабелей и разъемов физического интерфейса сети | Спецификация 10Base-T | Канал связи |
| 2 | Канальный | Обеспечивает взаимодействие между двумя компьютерами. Функции — проверка доступности среды передачи, реализация механизмов обнаружения и коррекции ошибок. В протоколах канального уровня, используемых в локальных сетях, заложены определенная структура связей между компьютерами и способы их адресации | Ethernet, Token Ring, FDDI, 100VG-AnyLAN | Операционная система |
| 3 | Сетевой | Реализует взаимодействие узлов сети. Этот уровень служит для образования единой транспортной системы, объединяющей несколько сетей с различными принципами передачи информации между конечными узлами | Протокол межсетевого взаимодействия IP стека TCP/IP и протокол межсетевого обмена пакетами IPX стека Novell | |
| 4 | Транспортный | Обеспечивает возможность передачи более длинных сообщений между хостами | Протоколы TCP и UDP стека TCP/IP и протокол SPX стека Novell | |
| 5 | Сеансовый | Устанавливает и поддерживает соединение между компонентами распределенной системы. Использует механизм соединений транспортного уровня | | Промежуточная среда |
| 6 | Представительский | Устраняет различия в представлении данных. Этот уровень гарантирует то, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе. При необходимости уровень преобразовывает данные в некоторый общий формат представления или выполняет обратное преобразование | Secure Socket Layer (SSL) | |
| 7 | Прикладной | Обеспечивает функционирование приложения. Набор протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам | Протокол электронной почты | Компонент информационной системы |

# СОКЕТЫ

- Сокет – это программный интерфейс для обеспечения обмена данными между процессами; абстрактный объект, представляющий конечную точку соединения
- Обеспечивают примитивы низкого уровня для непосредственного обмена потоком байт между двумя процессами
- Могут быть использованы для организации простейшего взаимодействия удаленных приложений

Пусть клиенту банка необходимо обеспечить возможность перевода денег на свой счет из другого банка, используя Web-приложение (задача интеграции Web-приложения с финансовой системой). Простейшее решение может быть получено с использованием протокола TCP/IP. Для определенности представим, что необходимо передать только имя клиента и сумму денежного перевода. Приведенный ниже пример кода (C#) демонстрирует вариант реализации конечной точки Web-приложения, в котором создается сокет с адресом my.bank.com и по сети пересылается сумма и имя клиента

```
String hostName = "my.bank.com";
//Задание DNS-имени удаленного компьютера
int port = 80;
IPHostEntry hostInfo = Dns.GetHostByName (hostName);
//Преобразование имени удаленного компьютера в IP-адрес
IPAddress address = hostInfo.AddressList [0];
IPEndPoint endpoint = new IPEndPoint (address, port);
Socket socket = new Socket (address.AddressFamily, SocketType.
Stream, ProtocolType.Tcp);
socket.Connect (endpoint);
byte [] amount = BitConverter.GetBytes (1000);
//Преобразование данных в поток байтов
byte [] name = Encoding.ASCII.GetBytes ("Joe");
int bytesSent = socket.Send (amount);
bytesSent += socket.Send (name);
socket.Close ();
```

# Проблемы и ограничения использования сокетов

- Связываемые системы должны иметь одинаковое внутреннее представление целых чисел (32 или 64 бит) и одинаковый порядок следования байтов (прямой или обратный). В противном случае передаваемые данные будут неверно интерпретированы приложением-получателем;

- Компьютер должен иметь неизменяемый адрес. Перемещение приложения получателя на другой компьютер потребует изменения программного кода;

- Интегрируемые приложения должны быть доступны в одно и то же время, поскольку протокол TCP/IP является протоколом с установкой соединения. Если одно из взаимодействующих приложений в момент установки соединения недоступно, то соединение не будет установлено;

- Использование соглашения о формате данных, предаваемых между интегрируемыми приложениями. Любое изменение в формате потребует изменений в программном коде как на стороне отправителя, так и на стороне получателя.

***использование механизма сокетов приводит к по- лучению плохо маситабируемого, неустойчивого, трудно поддерживаемого интеграционного решения, обеспечивающего «жесткую» связь между приложениями.***

# Промежуточная среда

Промежуточная среда (middleware – связующее программное обеспечение) выполняет функции сеан- сового и представительского уровней модели OSI/ISO.

Наличие промежуточной среды позволяет создать открытые, масштабируемые и отказоустойчивые распределенные системы.

Промежуточная среда предоставляет:

- единый, независимый от операционной системы механизм использования одними программными компонентами сервисов других компонентов;
- безопасность – аутентификация и авторизация всех пользователей сервисов компонента и защита передаваемой между компонентами информации от искажения и чтения третьей стороной;
- целостность данных – управление транзакциями, распределенными между удаленными компонентами системы;
- балансировку нагрузки на серверы с программными компонентами;
- бнаружение удаленных компонентов.

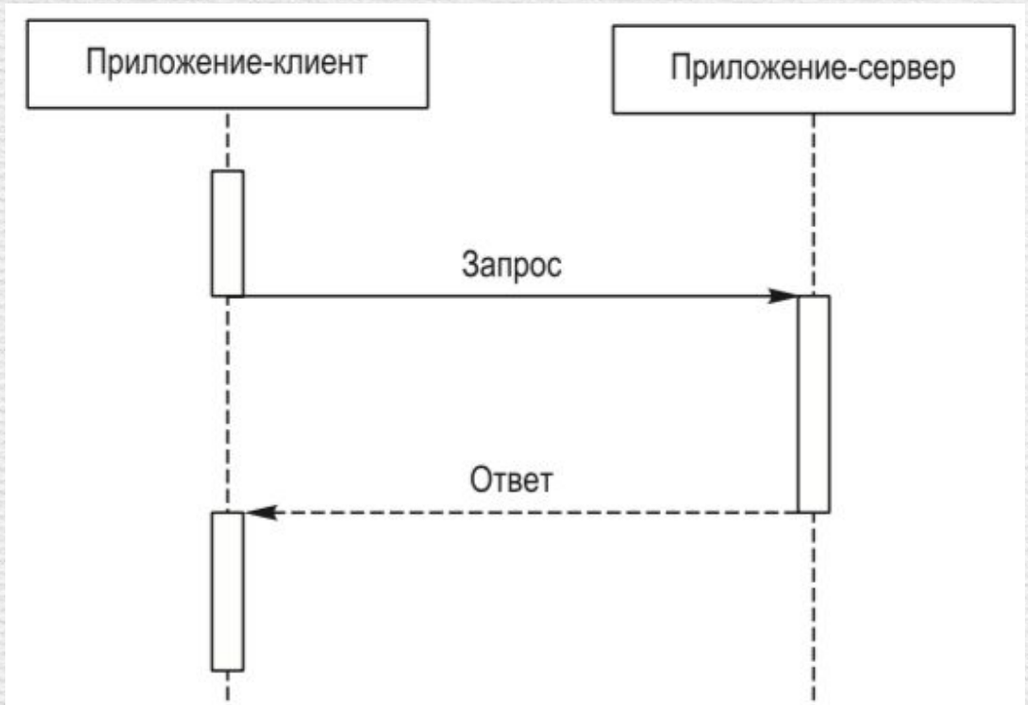
# Типы промежуточных сред

- ориентированные на посылку сообщений – поддерживают связь между компонентами распределенной системы посредством обмена сообщениями (Microsoft Message Queuing, IBM MQSeries, Sun Java System Message Queue);

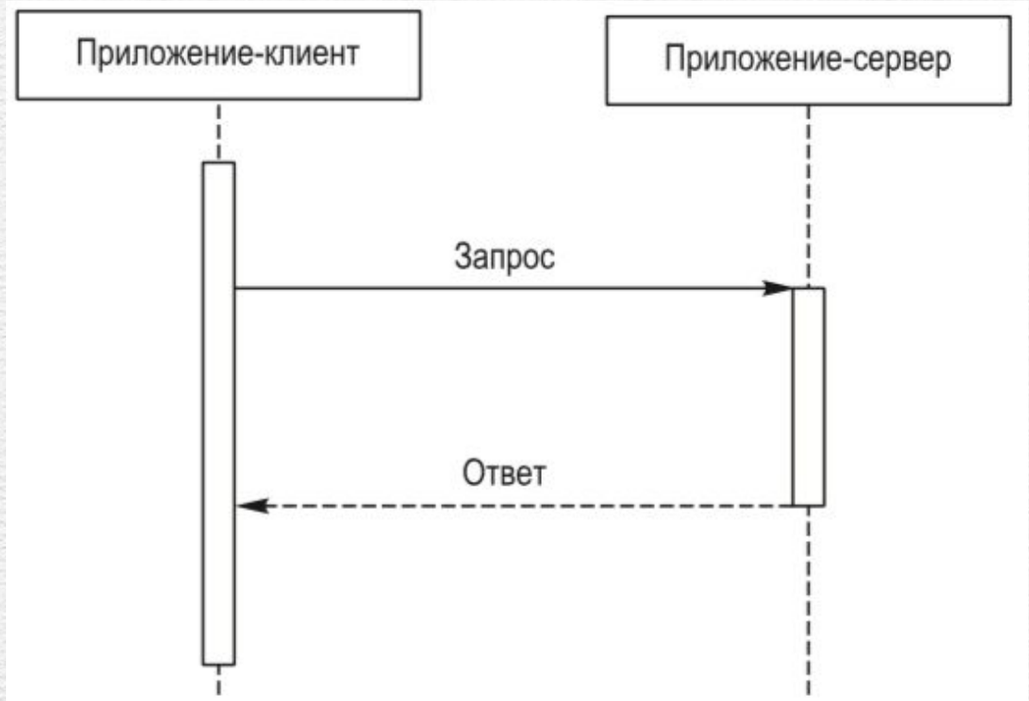
- объектно-ориентированные – построены на модели удаленного обращения к методу (J2EE, .NET, CORBA);

- транзакционно-ориентированные – ориентированы на поддержку транзакций в системах распределенных баз данных (мониторы транзакций, все современные промышленные СУБД, например MS SQL SERVER).

# Модели взаимодействия приложений




Синхронным называется такое взаимодействие, при котором приложение, выступающее в роли клиента, отослав запрос, блокируется и может продолжать работу только после получения ответа от приложения, выступающего в роли сервера. Иногда такой вид взаимодействия называют блокирующим.




В случае асинхронного взаимодействия приложение-клиент после отправки запроса приложению-серверу может продолжать работу, даже если ответ на запрос еще не пришел. Иногда такой вид взаимодействия называют неблокирующим

# Сложности реализации асинхронного взаимодействия

Сложности реализации асинхронного взаимодействия обусловлены следующими обстоятельствами:

- возможность использования нескольких потоков исполнения, увеличивающая производительность решения, одновременно затрудняет его отладку;
- приложение-клиент должно быть готово принять ответ в любой момент, даже в процессе выполнения другой задачи;
- поскольку асинхронные процессы могут выполняться в любом порядке, приложение-клиент должно уметь обрабатывать результат запроса с учетом времени получения.

# Стандарты объектно-ориентированного взаимодействия

- Включают в себя COM (DCOM, COM+), RMI, CORBA
- Идея заключается в создании виртуального компонента, являющегося прокси-объектом (локальным представителем) удаленного метода или процедуры. Приложение-клиент обращается к прокси-объекту, который и передает сообщение к удаленному объекту.

# Удаленный вызов процедур RPC

Впервые был реализован в начале 1980-х гг. компанией Sun Microsystems и явился прообразом объектно-ориентированного промежуточного слоя применительно к структурной парадигме программирования.

Суть технологии RPC заключается в том, что вызов функции на удаленном компьютере осуществляется с помощью промежуточного программного обеспечения так же, как и на локальном

Для того чтобы организовать удаленный вызов процедуры необходимо:

- описать интерфейс процедуры, которая будет использоваться для удаленного вызова при помощи языка определения интерфейсов (Interface Definition Language, IDL);

- скомпилировать определение процедуры (при этом будут созданы описание этой процедуры на том языке программирования, на котором будут разрабатываться клиент и сервер, и два дополнительных компонента – клиентский и серверный переходники). Клиентский переходник представляет собой процедуру-заглушку (stub), которая будет вызываться клиентским приложением. Клиентский переходник размещается на той же машине, где находится компонент-клиент, а серверный переходник – на той же машине, где находится компонент-сервер.

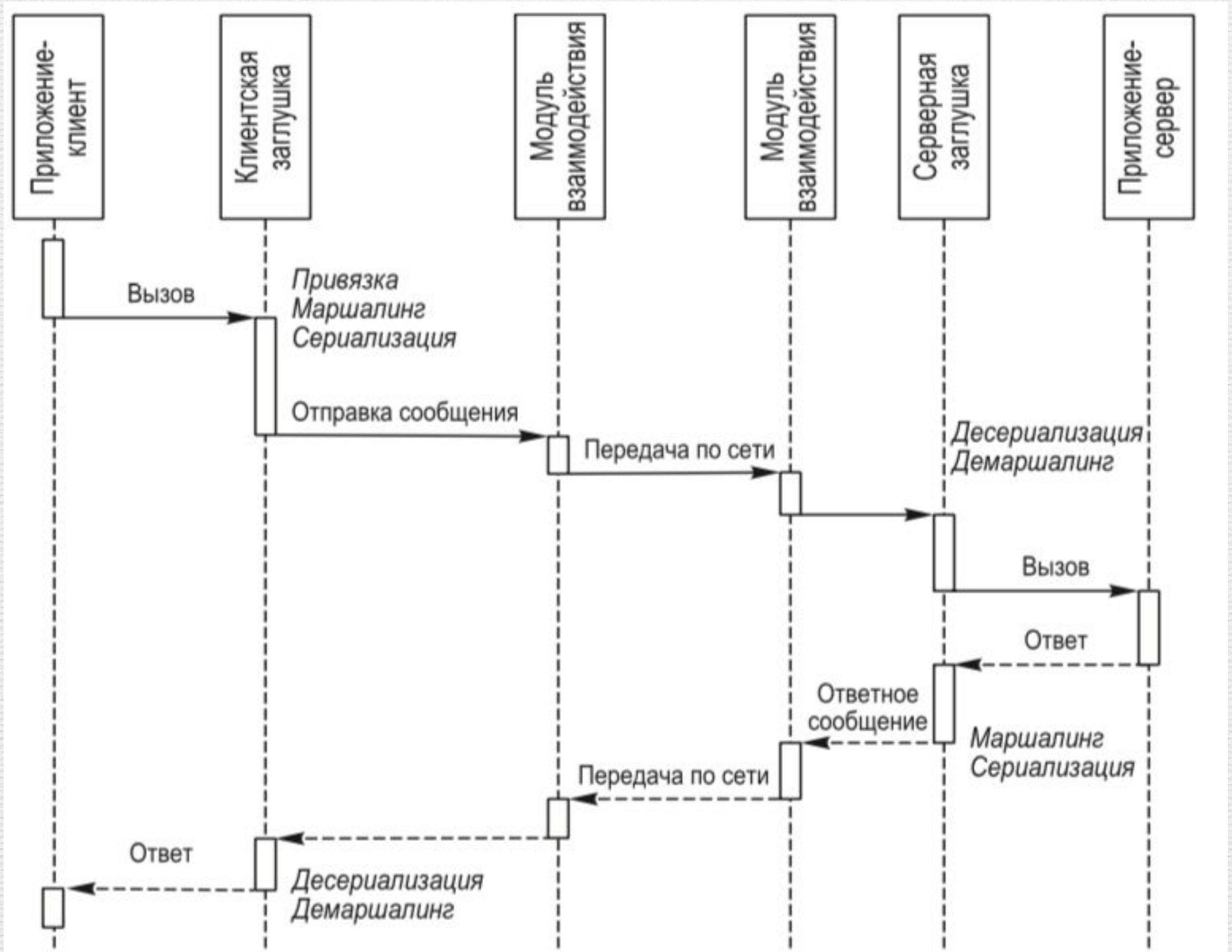
# Организация работы RPC

- клиентский переходник выполняет привязку к серверу (определяет физическое местонахождение (адрес) сервера);
- клиентский переходник выполняет маршалинг (упаковывает вызов процедуры и ее аргументы в сообщение в некотором стан- дартном для данной системы формате);
- клиентский переходник выполняет сериализацию сообщения (преобразует полученное сообщение в поток байтов) и отсылает с помощью какого-либо протокола (транспортного или более высокого уровня) на машину, на которой помещен серверный компонент;
- серверный переходник принимает сообщение, содержащее пара- метры вызова процедуры, распаковывает эти параметры при помощи десериализации и демаршалинга, вызывает локально соответствующую функцию серверного компонента, получает ее результат, упаковывает его и посылает по сети на клиентскую машину;
- после получения ответа от сервера выполняется процедура десериализации.

***Процесс преобразования параметров для передачи их между процессами при удаленном вызове называется маршализацией (marshaling).***

***Процесс преобразования экземпляра какого-либо типа данных в набор байт называется сериализацией.***

# Схема работы RPC



# Удаленный вызов метода

- Является модификацией метода удаленного вызова процедуры для объектной парадигмы программирования.
- В момент вызова удаленного метода на стороне клиента создается клиентская заглушка, называемая посредником (proxy).
- Клиентская заглушка в своем интерфейсе имеет все методы объекта-сервера, выполняет маршалинг и сериализацию параметров вызываемых методов и передает их по сети.
- Поскольку один объект сервера может предоставлять несколько методов, информация о том, какой именно метод вызывается, также упаковывается вместе с аргументами вызова.
- Промежуточная среда на стороне сервера десериализует параметры и передает их заглушке, называемой каркасом, или скелетоном (skeleton), на стороне сервера.

# Стандарты объектно-ориентированного взаимодействия

| Стандарт | Назначение | Достоинства | Ограничения |
|---|---|---|---|
| Com | Средство взаимодействия приложений, функционирующих на одном компьютере | Высокий уровень абстракции<br>Простота использования<br>Очень высокая производительность | COM-приложения способны функционировать только под управлением Windows (двоичная структура объектов компонентной модели Microsoft)<br>Распределенные решения плохо масштабируются (жесткая связь между клиентом и сервером)<br>Невысокая устойчивость к сбоям COM-систем (жесткая привязка сервера к клиенту, двухуровневая архитектура) |
| Com+ | Промежуточная среда для создания распределенных систем, действующих в локальной сети | Поддержка синхронного и асинхронного взаимодействий программных компонентов<br>Динамическая балансировка нагрузки<br>Поддержка логической целостности данных за счет работы с координатором распределенных транзакций<br>Возможность ограничения доступа к компоненту в разрезе ролей, связанных с учетными записями пользователей | Использование ограничено взаимодействием компонентов внутри локальной или виртуальной частной сети, построенной на базе Microsoft Windows, в пределах одного предприятия |

# Стандарты объектно-ориентированного взаимодействия

| Стандарт | Назначение | Достоинства | Ограничения |
|---|---|---|---|
| DCOM | Промежуточная среда для создания распределенных систем, действующих в локальной сети и сети Интернет | Независимость от языка программирования<br>Простота<br>Поддержка распределенных транзакций | Использование ограничено платформами Windows |
| CORBA | Набор спецификаций для промежуточного программного обеспечения объектного типа. Создавалась консорциумом OMG как универсальная методология создания распределенных систем в гетерогенных средах | Кроссплатформенность<br>Высокий уровень устойчивости к внутренним сбоям за счет большей изоляции клиентов и серверов (трехуровневая архитектура)<br>Передача данных между компонентами происходит при помощи протокола UDP (обеспечивает экономию системных ресурсов)<br>Набор сервисов (управление транзакциями, безопасностью, жизненным циклом объектов, событиями и т.д.) | Сложность технологии |
| RMI | Архитектура (Remote Method Invocation, то есть вызов удаленного метода), которая интегрирована с JDK1.1 и реализует распределенную модель вычислений | Самый простой и быстрый способ создания распределенных систем<br>Распределенное автоматическое управление объектами | Поддержка одного языка программирования Java<br>Собственный протокол взаимодействия<br>Трудность интегрирования с существующими приложениями |