Технология .NET Remoting

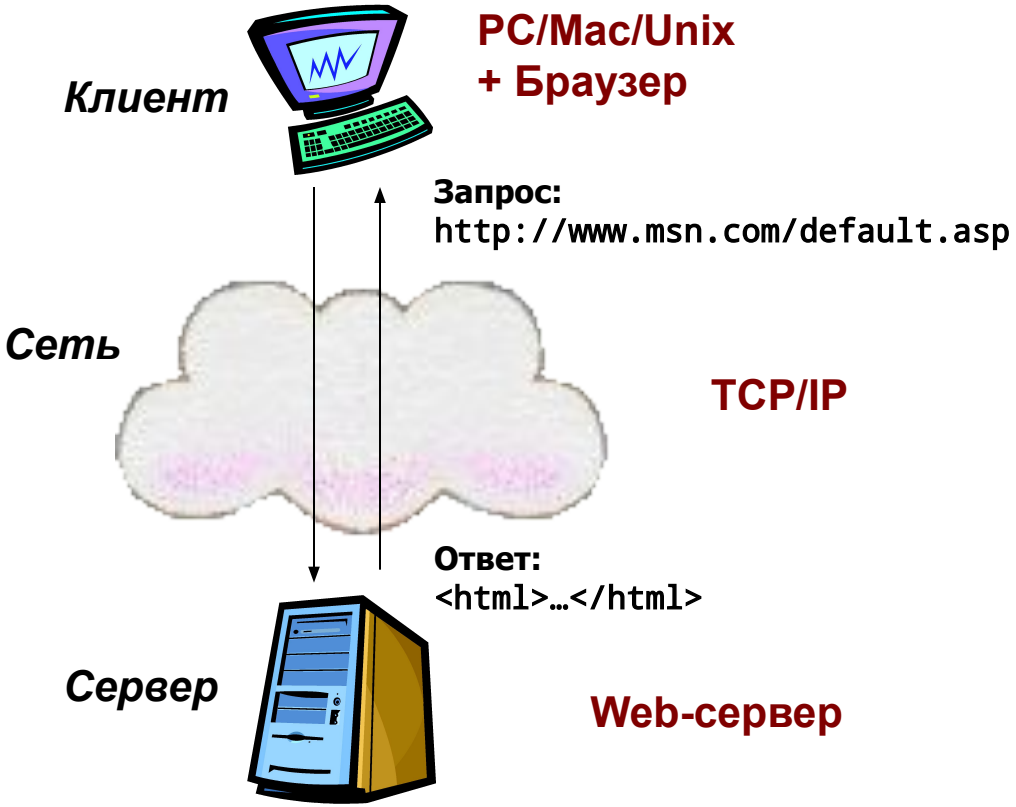
Распределенные приложения

Технология .NET Remoting

Для описания взаимодействия двух сущностей применяется модель «*клиент-сервер*», в которой одна из сторон (*клиент*) инициирует обмен данными, посылая запрос другой стороне (*серверу*). Сервер так или иначе обрабатывает запрос и при необходимости посылает ответ клиенту.

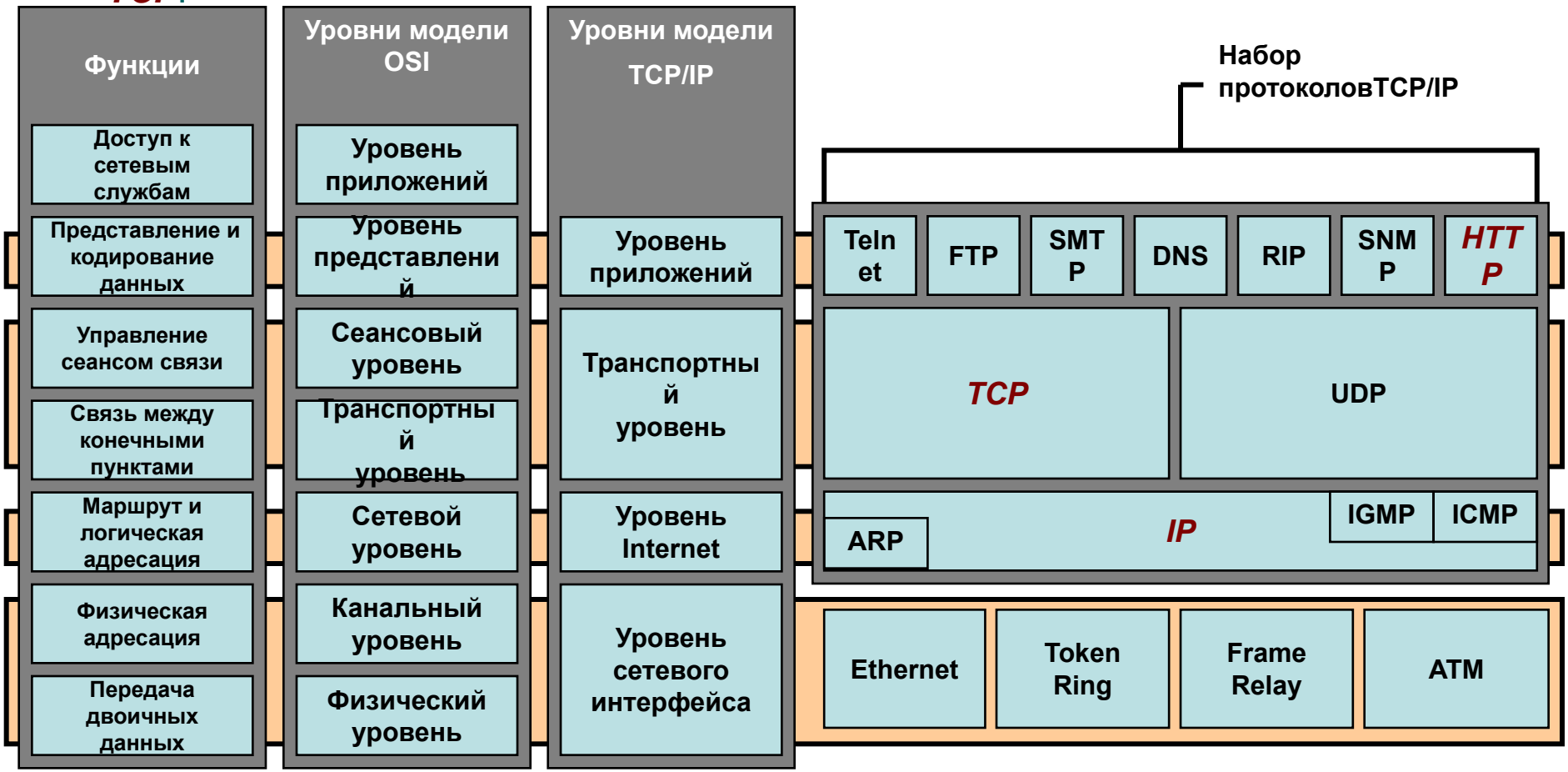
Взаимодействие является *синхронным*, если клиент ожидает завершения обработки своего запроса сервером, и *асинхронным*, если клиент посылает серверу запрос и продолжает свою работу без ожидания ответа сервера. В зависимости от того, как распределены логические компоненты приложения между *клиентами* и *серверами*, различают четыре основные модели архитектуры *«клиент-сервер»*:

- модель «*файл-сервер*»;
- модель «*сервер базы данных*»;
- модель «*сервер транзакций*»;
- модель «*сервер приложений*».



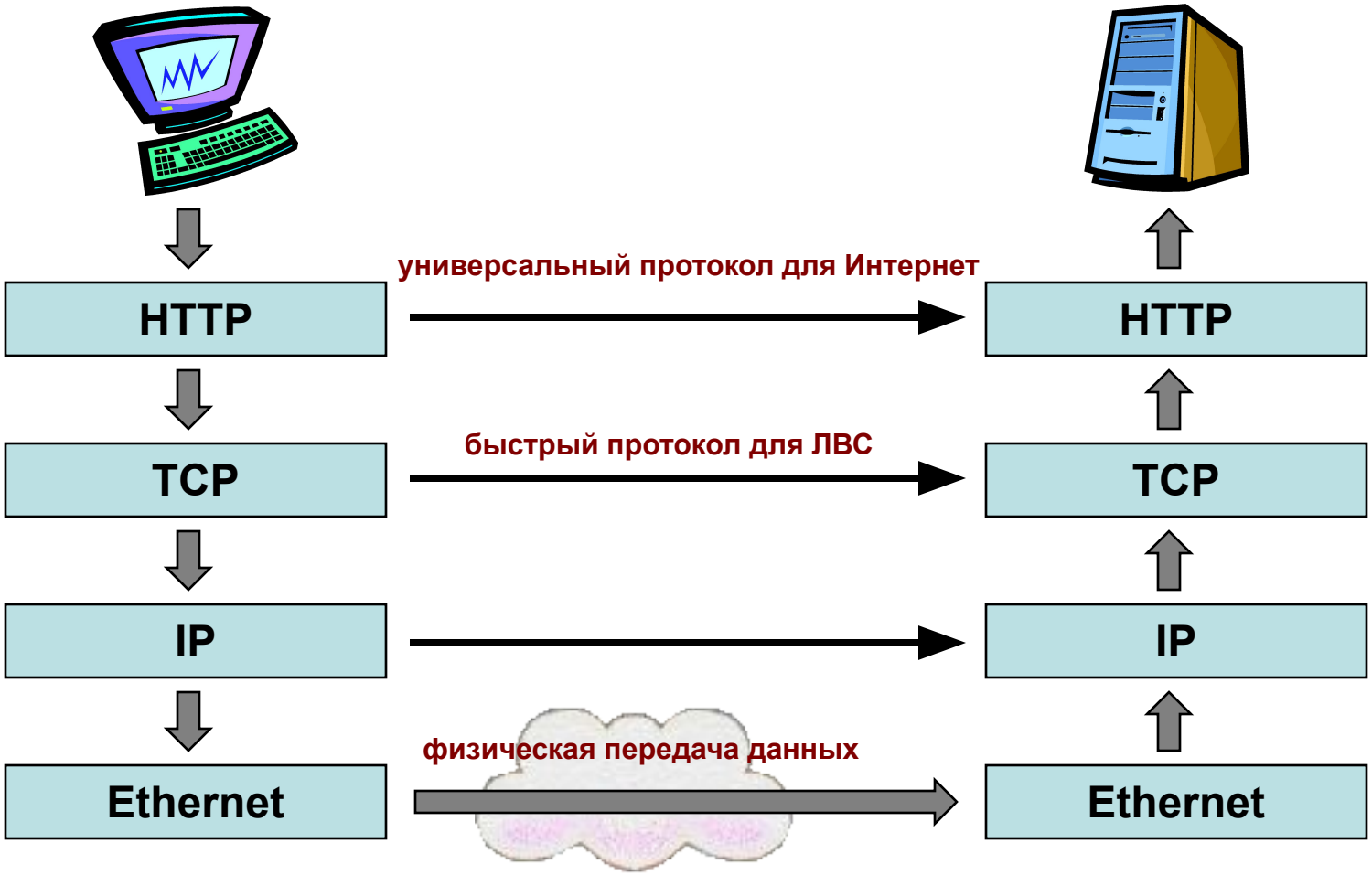
Технология .NET Remoting

Существуют разные сетевые модели коммуникаций и описания сетевых протоколов, см. рисунок. Из ниже перечисленных в *.NET Remoting* применяются протоколы ***HTTP*** и ***TCP***.



Технология .NET Remoting

Стек протоколов ***HTTP, TCP, IP***.



Технология .NET Remoting

Технология *.NET Remoting* разработана для создания распределенных приложений. С её помощью можно обращаться к экземплярам классов *.NET Framework*, находящимся за пределами собственного *домена приложения*. Это может быть другое приложение внутри одного процесса *Windows*, другой процесс *Windows* на той же машине, или процесс на другой машине (в том числе подключенной через Интернет).

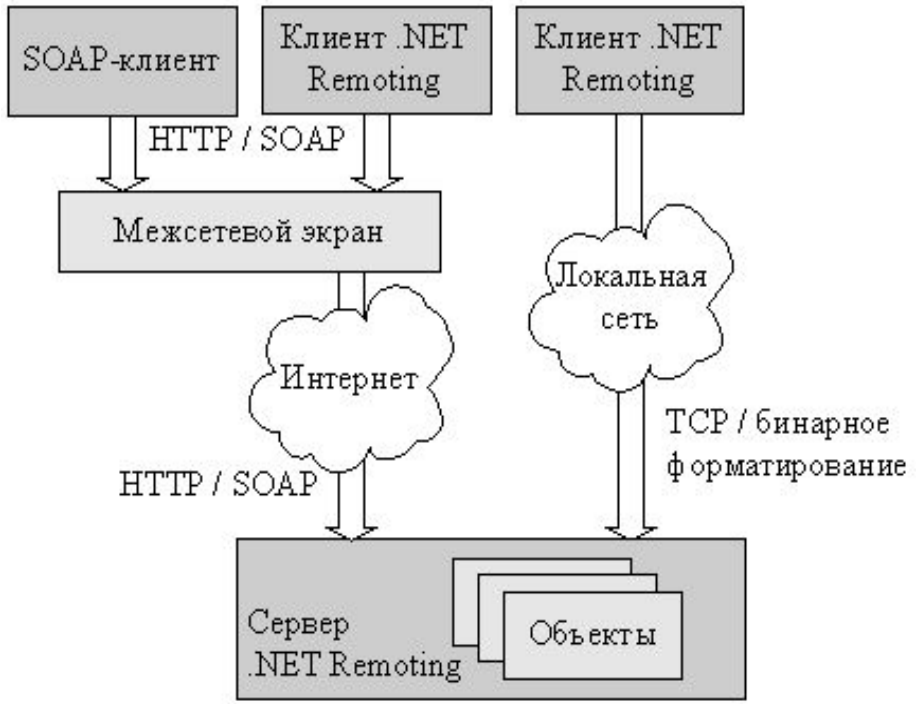
При запуске *.NET Remoting* настраивается для использования конкретного вида транспортного протокола (*channel*) и протокола доступа (*formatter*). Затем регистрируются все классы, к которым будет обеспечен удаленный доступ.

Первый вариант настройки: транспортный протокол (*channel*) есть *TCP*. В качестве протокола доступа (*formatter*) применяется собственный бинарный формат, который при желании можно изменить на другой.

Второй вариант настройки: транспортный протокол (*channel*) есть *HTTP*. В качестве протокола доступа применяется *SOAP*.

Вариант настройки *HTTP / SOAP* следует применять в случае, если:

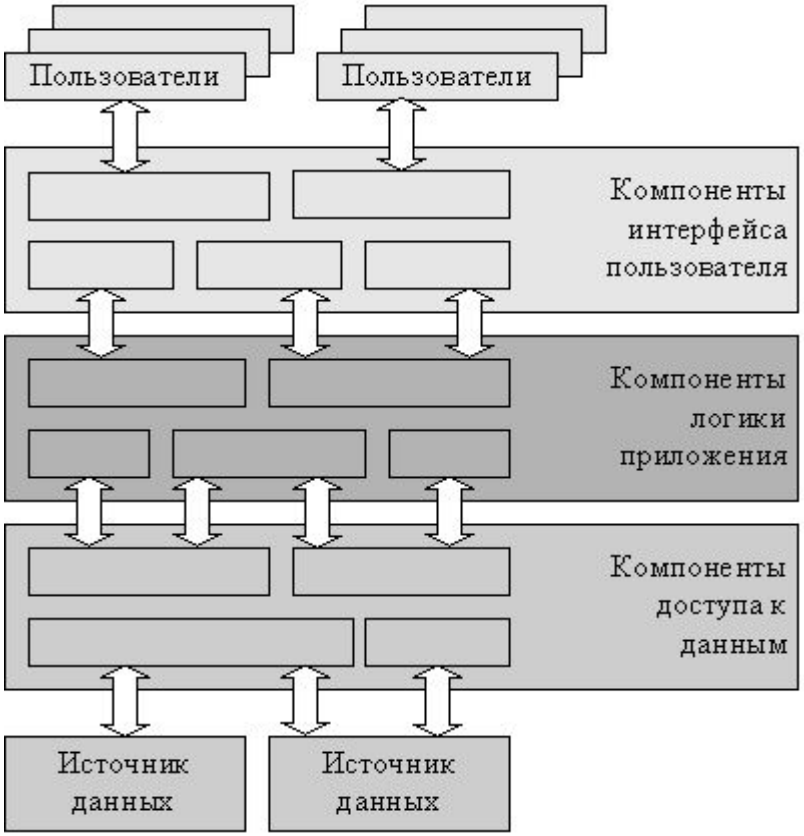
- необходим прокси-сервер, который нельзя настроить на определенный порт;
- возможны клиенты, созданные без использования *.Net Remoting*;
- необходимо контролировать содержимое сетевого обмена.

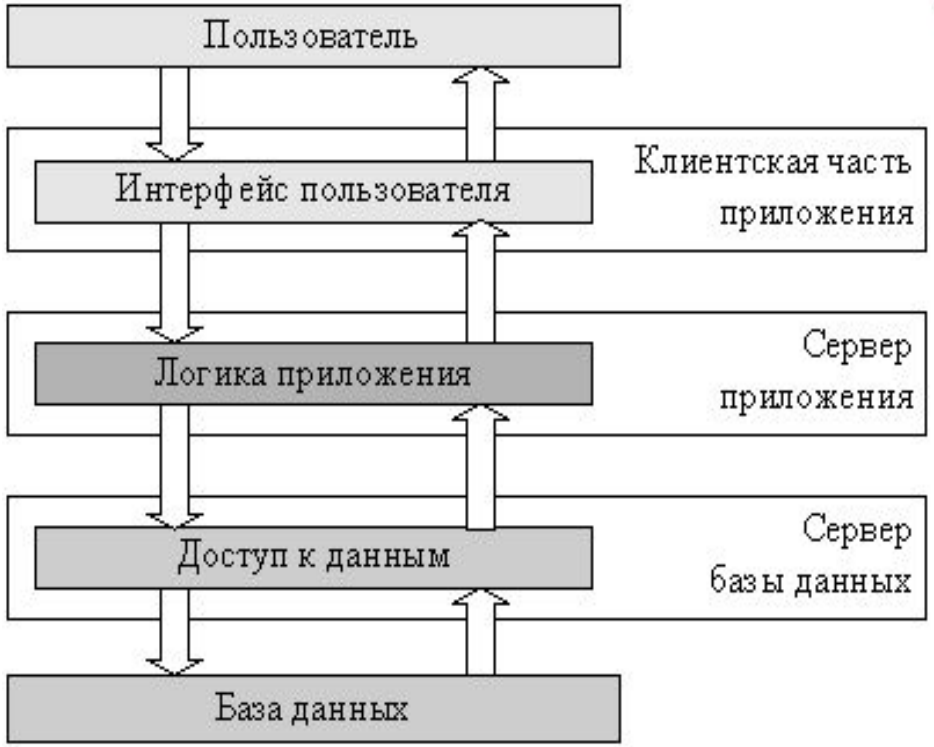


Технология .NET Remoting

Типичная архитектура приложений *.NET Remoting* такова, что компоненты интерфейса пользователя, логика приложения (бизнес-логика) и компоненты доступа к данным отделены друг от друга и развернуты на разных узлах. Пример – система платежных терминалов и банкоматов *CyberPlat* (до 40% рынка в РФ)

Трехзвенная архитектура модели взаимодействия «клиент-сервер»





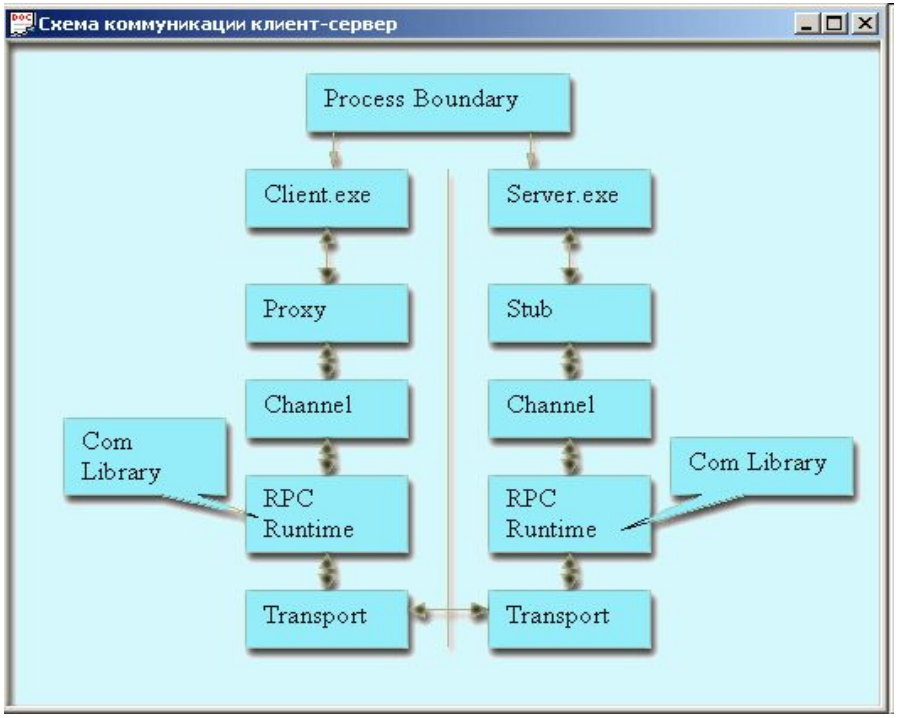
Технология .NET Remoting

Удаленный вызов процедур

Идея удаленного вызова процедур (*Remote Procedure Call, RPC*) появилась в середине 80-х годов и заключалась в том, что при помощи промежуточного ПО функцию на удаленном компьютере можно вызывать так же, как и функцию на локальном компьютере. Чтобы удаленный вызов происходил прозрачно с точки зрения вызывающего приложения, промежуточная среда должна предоставить процедуру-заглушку (*stub*) или представителя (*proxy*), которые будут вызываться клиентом.

После вызова клиентом процедуры-заглушки промежуточная среда преобразует переданные ей аргументы в вид, пригодный для передачи по транспортному протоколу, и передает их на сервер вместе с вызываемой функцией.

На сервере аргументы функции извлекаются промежуточной средой из сообщения транспортного уровня и передаются непосредственно вызываемой функции. Аналогичным образом на клиентскую машину передается результат выполнения вызванной функции.



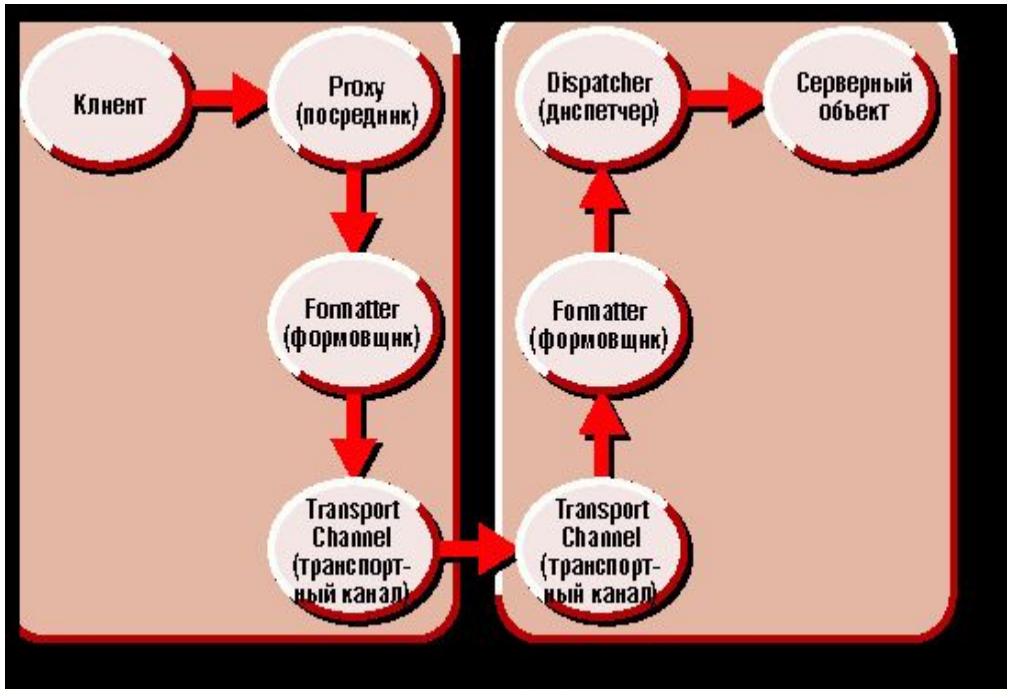
Технология .NET Remoting

Использование удаленных объектов

При необходимости доступа к удаленным объектам клиент указывает протоколы доступа и посылает запрос на сервер. Сервер в ответ на этот запрос создает у себя затребованный объект и передает его идентификатор клиенту. Клиент создает у себя *proxy*-класс, который затем и использует, как если бы это был объект в его собственном домене.

В *.NET Remoting* объекты могут передаваться по значению (*by value*) и по ссылке (*by ref*). Создавая классы, экземпляры которых предполагается передавать (возвращать) по значению, необходимо их пометить атрибутом [*Serializable*].

При передаче объектов по ссылке передается только идентификатор (спец. объект *ObjRef*). Клиент создает у себя *proxy*-класс, который перенаправляет вызовы методов удаленного объекта на сервер и получает возвращаемые значения. Объект, передаваемый по ссылке, должен наследоваться от класса *MarshalByRefObject*.



У клиента создается впечатление, что удаленный объект находится в его адресном пространстве, но на самом деле вызовы методов *сериализуются* и передаются по каналам связи на сервер.

Технология .NET Remoting

Маршалинг и сериализация

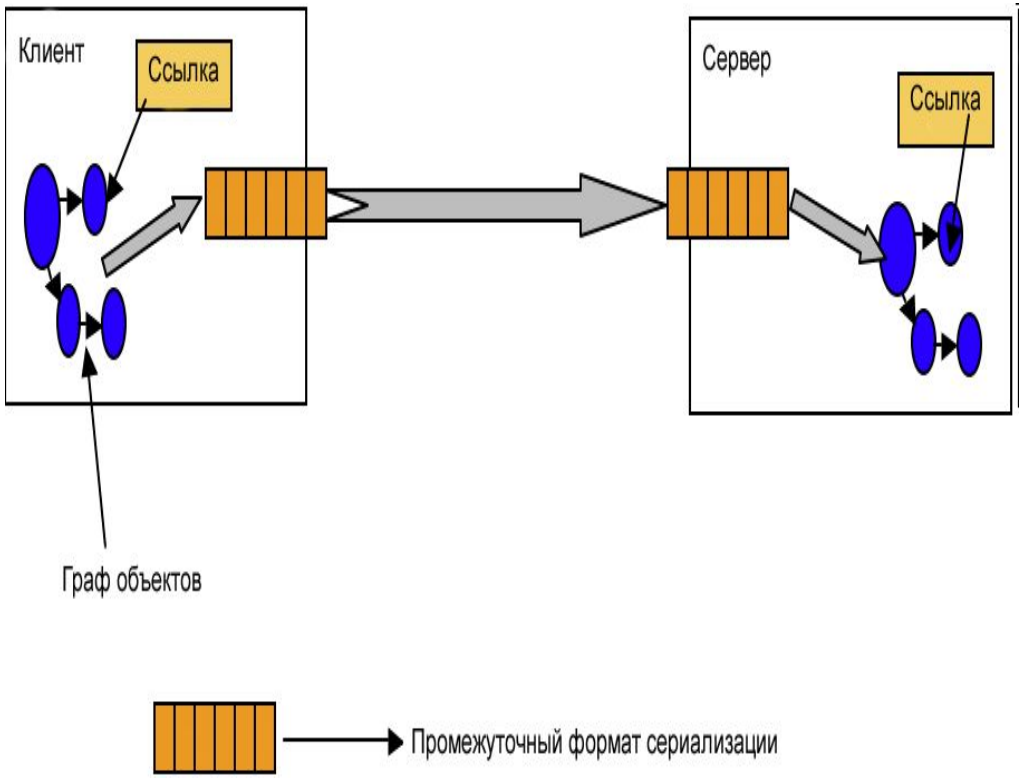
Маршалинг (*marshaling*) есть процесс упаковки данных и формирования пакета сообщений для передачи данных по телекоммуникационному каналу.

Сериализация (*serialization*) есть процесс преобразования данных в последовательную форму (в поток байтов) для передачи данных по телекоммуникационному каналу.

Маршалинг и *сериализация* примитивных типов не представляет проблем, поскольку такие объекты имеют простую линейную структуру.

В общем же случае объекты имеют сложную внутреннюю структуру и, кроме того, могут ссылаться друг на друга, образуя направленный граф объектов (*object graph*).

Внутренняя структура и *граф объектов* должны быть сохранены в потоке байтов и восстановлены после передачи данных по каналу связи. В этом состоит главная задача, которая решается в ходе *сериализации*.



Технология .NET Remoting

Представители (proxy)

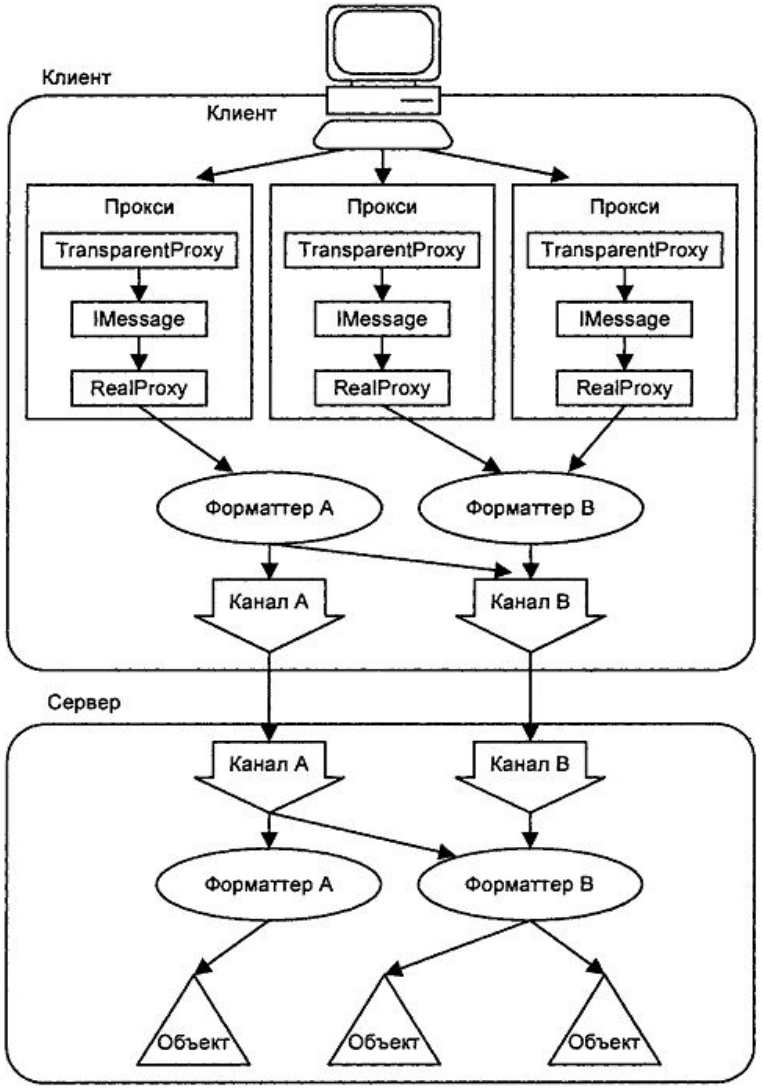
В *.NET Remoting proxy* представляет собой объект-заменитель, который создаётся внутри клиентского процесса и предоставляет клиенту те же интерфейсы, что и сам удаленный объект, с которым клиент желает иметь дело.

Имеются два вида прокси – «прозрачные» (*transparent proxy)* и «реальные» (*real proxy*).

Прозрачные *proxy* создаются «на лету» во время выполнения кода и предназначены для перехвата клиентских вызовов удаленного объекта. Получив управление, *transparent proxy* упаковывает клиентский вызов в пакет сообщений *IMessage* в соответствии с транспортным протоколом (*channel*) и передаёт её далее в *real proxy*.

Real proxy сериализует пакет сообщений *IMessage* и передает его в канал связи. На серверной стороне полученный вызов *десериализуется*, распаковывается и достигает собственно удаленного объекта.

Передача данных от удаленных объектов обратно в клиентскую часть осуществляется по этой же цепочке.



Координация потоков

Синхронизация в C#. Инструменты .NET Framework

Координация потоков

Средства синхронизации потоков в C#

C# поддерживает параллельное выполнение кода через многопоточность. Поток – это независимая нить исполнения, способная выполняться одновременно с другими потоками. Программа на *C#* запускается как единственный поток, автоматически создаваемый *CLR* и операционной системой («главный» поток), и становится многопоточной после создания дополнительных потоков.

Управление многопоточностью осуществляет планировщик потоков, эту функцию *CLR* обычно делегирует операционной системе. На однопроцессорных компьютерах планировщик потоков осуществляет квантование времени. На многопроцессорных компьютерах многопоточность реализована как смесь квантования времени и подлинного параллелизма, когда разные потоки выполняют код на разных *CPU*.

Потоки и процессы ОС отчасти схожи. Например, время разделяется между процессами, исполняющимися на одном компьютере, так же, как между потоками одного приложения. Но процессы *ОС* полностью изолированы друг от друга. Потоки же разделяют динамическую память (кучу) с другими потоками этого же приложения.

Основные средства синхронизации потоков в *C#* см. в таблицах и в рекомендованной литературе.

Конструкция	**Назначение**
Sleep	Блокировка потока на указанное время
Join	Ожидание окончания другого потока

Координация потоков

Конструкция	Назначение	Доступна из других потоков?	Быстро-действие
lock	Гарантирует, что только один поток может получить доступ к общему ресурсу или секции кода	Нет	Быстро
Mutex	Гарантирует, что только один поток может получить доступ к общему ресурсу или секции кода. Используется для предотвращения запуска нескольких экземпляров приложения	Да	Средне
Semaphore	Гарантирует, что не более заданного числа потоков может получить доступ к общему ресурсу или секции кода	Да	Средне
EventWaitHandle	Позволяет потоку ожидать сигнала от другого потока	Да	Средне
Wait and Pulse	Позволяет потоку ожидать, пока не выполнится заданное условие блокировки	Нет	Средне
Interlocked	Обеспечивает выполнение простых не блокирующих атомарных операций	Да – через разделяемую память	Очень быстро
volatile	Применяется для безопасного не блокирующего доступа к полям	Да – через разделяемую память	Очень быстро