Лекция 2.
Организация связи между компонентами

2.1. Роль связи РВС

Понятие «распределенная вычислительная система» подразумевает, что компоненты такой системы распределены, т.е. удалены друг от друга. Очевидно, что функционирование подобных систем невозможно без эффективной связи между ее компонентами. Более того, можно предположить, что большинство существенных для пользователя характеристик и особенностей распределенной системы будут определяться прежде всего именно способом организации связи, его эффективностью и соответствием типу решаемых задач.

Задачи организации обмена между распределенными (территориально, административно и т.д.) компонентами давно и в значительной мере успешно решаются в вычислительных сетях, и естественно, что распределенные ВС используют наработанный опыт. Взаимодействие в вычислительных сетях базируется на протоколах.

Протокол – это набор правил и соглашений, описывающих процедуру взаимодействия между компонентами системы (в том числе и вычислительной).

Если система поддерживает принятый протокол, она, с большой долей вероятности, окажется способна взаимодействовать с другой системой, которая так же поддерживает данный протокол. В области вычислительных коммуникаций уже длительное время существует общепринятая система протоколов - модель OSI (рис. 2.1). Эта модель представляет собой стек протоколов разного уровня, которые позволяют описать практически все аспекты взаимодействия компонентов ВС.

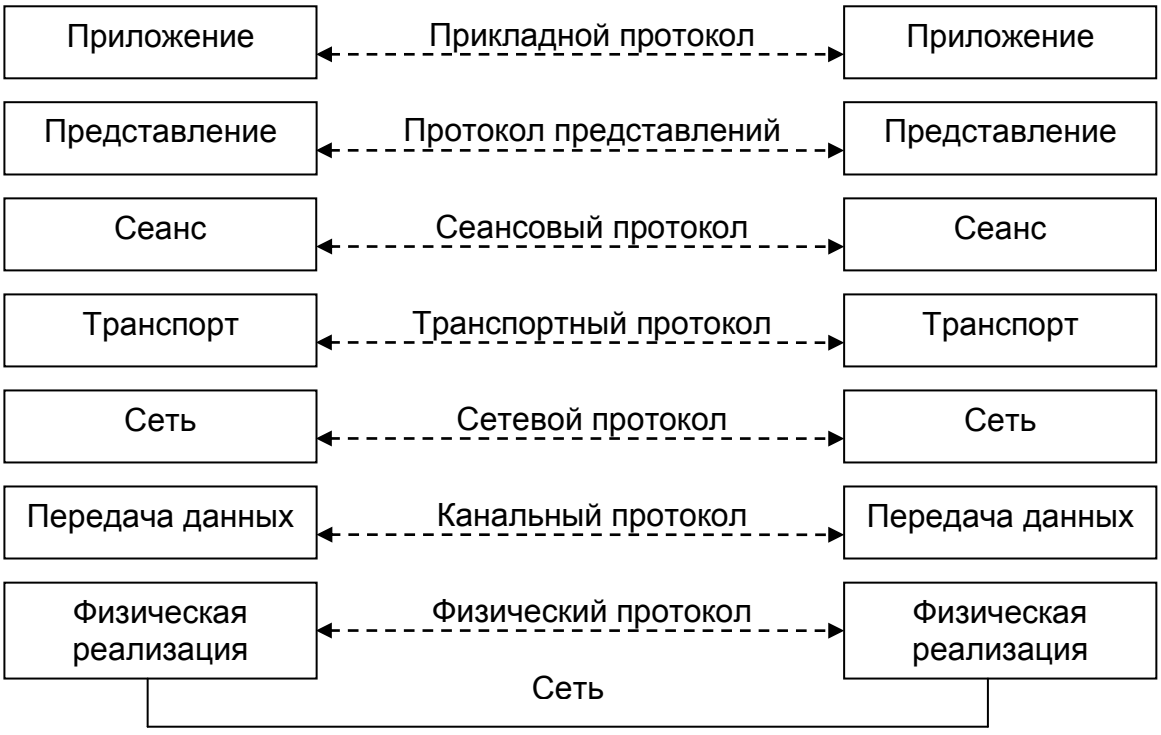


Рис. 2.1. Уровни, интерфейсы и протоколы модели OSI

Детальное рассмотрение стека протоколов и особенностей его различных уровней лежит за рамками данной дисциплины.

2.1.1. Обмен сообщениями

Существует два метода передачи сообщений от одной удаленной системы к другой – *непосредственный обмен сообщениями и использование очередей сообщений*. В первом случае передача происходит напрямую, и она возможна только в том случае, если принимающая сторона готова принять сообщение в этот же момент времени. Во втором случае используется посредник – менеджер очередей сообщений. Компонента посылает сообщение в одну из очередей менеджера, после чего она может продолжить свою работу. В дальнейшем получающая сторона извлечет сообщение из очереди менеджера и приступит к его обработке.

Простейшей реализацией непосредственного обмена сообщениями является использование транспортного уровня сети через интерфейс **сокетов**, минуя какое-либо промежуточное программное обеспечение.

Сокет — абстрактный объект, представляющий конечную точку соединения. При использовании стека протоколов TCP/IP сокет задается комбинацией IP-адреса машины и номера порта, например 10.10.10.10:80. Однако серьезным недостатком способ реализации РВС на основе сокетов обычно является то, что в этом случае реализация всех функций промежуточной среды ложится на разработчиков приложения. При таком подходе сложно получить расширяемую и надежную распределенную систему, поэтому для разработки прикладных распределенных систем обычно используются системы очередей сообщений.

Существует несколько разработок в области промежуточного программного обеспечения, реализующие высокоуровневые сервисы для обмена сообщениями между программными компонентами. К ним относятся, в частности, Microsoft Message Queuing, IBM MQSeries и Sun Java System Message Queue. Такие системы дают возможность приложениям использовать следующие базовые примитивы по использованию очередей:

– добавить сообщение в очередь;

– взять первое сообщение из очереди, процесс блокируется до появления в очереди хотя бы одного сообщения;

– проверить очередь на наличие сообщений;

– установить обработчик, вызываемый при появлении сообщений в очереди.

Менеджер очереди сообщений в таких системах может находиться на компьютере, отличном от компьютеров с участвующими в обмене компонентами. В этом случае сообщение первоначально помещается в исходящую очередь на компьютере с посылающей сообщения компонентой, а затем пересылается менеджеру требуемой. Для создания крупных систем обмена сообщениями может использоваться маршрутизация сообщений, при которой сообщения не передаются напрямую менеджеру, поддерживающему очередь, а проходят через ряд промежуточных менеджеров очередей сообщений (рис. 2.2).

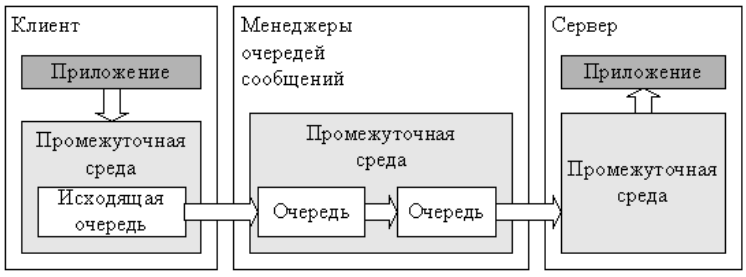


Рис. 2.2. Системы очередей сообщений

Недостатки систем очередей сообщений являются продолжением их достоинств:

– необходимость явного использования очередей распределенным приложением;

– сложность реализации синхронного обмена;

– определенные накладные расходы на использование менеджеров очередей;

– сложность получения ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

2.2. Удаленный вызов процедур.

Идея вызова удаленных процедур (Remote Procedure Call - RPC) состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными [Таненбаум].

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Возникает множество проблем: организация передачи данных с адресного пространства одной машины на другую, включая прозрачное использование нижележащей системы связи; обработка экстренного завершения родительского или дочернего удаленного процесса и др.

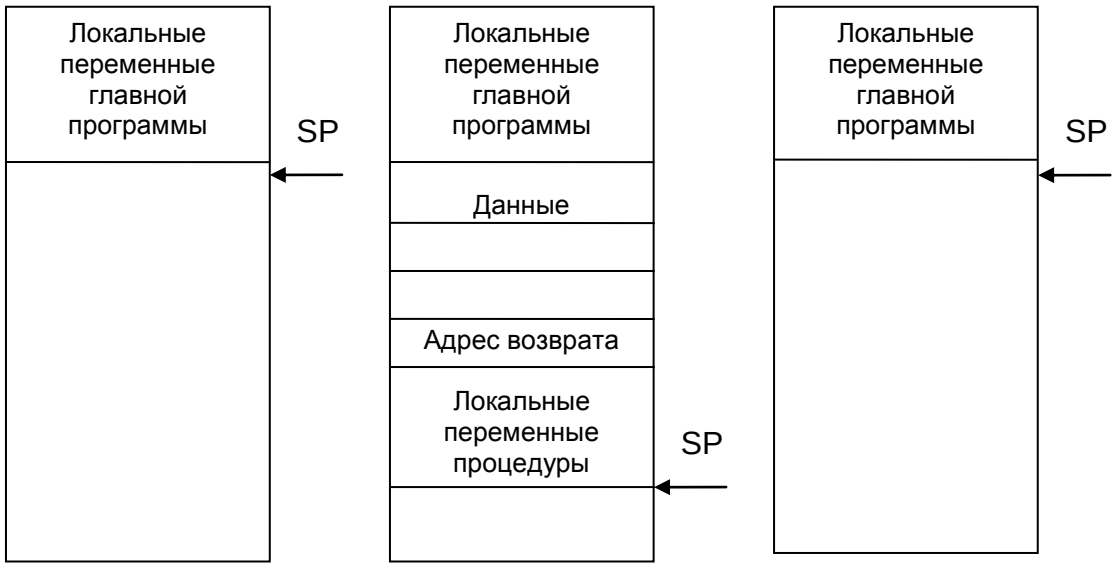
Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Эти и некоторые другие проблемы решает широко распространенная технология RPC, лежащая в основе многих распределенных операционных систем.

2.2.1. Базовые операции RPC

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычной машине, работающей автономно.

Чтобы осуществить вызов, вызывающая процедура заталкивает параметры в стек в обратном порядке (рис. 2.3). После того, как вызов выполнен, он помещает возвращаемое значение в регистр, перемещает адрес возврата и возвращает управление вызывающей процедуре, которая выбирает параметры из стека, возвращая его в исходное состояние.



а) Стек до выполнения вызова;

б) Стек во время выполнения процедуры;

в) Стек после возврата в вызывающую программу

Рис. 2.3. Порядок локального вызова процедуры

Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности так же, как и вызов локальной процедуры. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

RPC достигает прозрачности следующим путем. Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая клиентским стабом (stub - заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности (как на рисунке 2.4), так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

2.2.2. Этапы выполнения RPC

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рисунком 2.5. После того, как клиентский стаб был вызван программой-клиентом, его первой задачей является заполнение буфера отправляемым сообщением. В некоторых системах клиентский стаб имеет единственный буфер фиксированной длины, заполняемый каждый раз с самого начала при поступлении каждого нового запроса. В других системах буфер сообщения представляет собой пул буферов для отдельных полей сообщения, причем некоторые из этих буферов уже заполнены. Этот метод особенно подходит для тех случаев, когда пакет имеет формат, состоящий из большого числа полей, но значения многих из этих полей не меняются от вызова к вызову.

Затем параметры должны быть преобразованы в соответствующий формат и вставлены в буфер сообщения. К этому моменту сообщение готово к передаче, поэтому выполняется прерывание по вызову ядра.

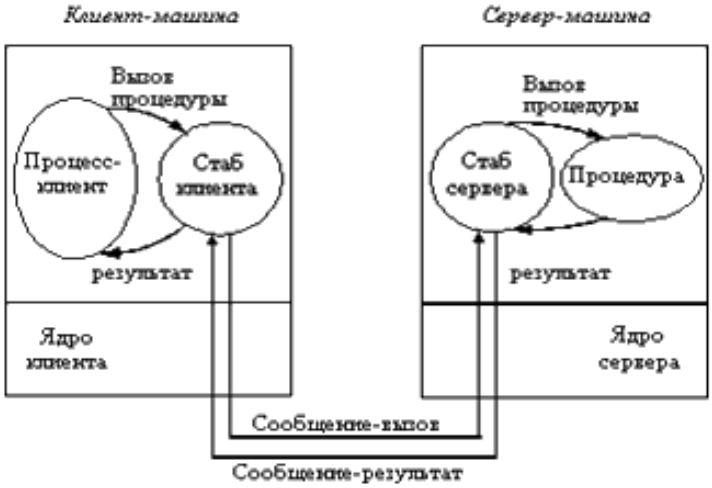


Рис. 2.4. Порядок удаленного вызова процедуры

Когда ядро получает управление, оно переключает контексты, сохраняет регистры процессора и карту памяти (дескрипторы страниц), устанавливает новую карту памяти, которая будет использоваться для работы в режиме ядра. Поскольку контексты ядра и пользователя различаются, ядро должно точно скопировать сообщение в свое собственное адресное пространство, так, чтобы иметь к нему доступ, запомнить адрес назначения (а, возможно, и другие поля заголовка), а также оно должно передать его сетевому интерфейсу. На этом

завершается работа на клиентской стороне. Включается таймер передачи, и ядро может либо выполнять циклический опрос наличия ответа, либо передать управление планировщику, который выберет какой-либо другой процесс на выполнение. В первом случае ускоряется выполнение запроса, но отсутствует мультипрограммирование.

На стороне сервера поступающие биты помещаются принимающей аппаратурой либо во встроенный буфер, либо в оперативную память (рис. 2.5). Когда вся информация будет получена, генерируется прерывание. Обработчик прерывания проверяет правильность данных пакета и определяет, какому стабу следует их передать. Если ни один из стабов не ожидает этот пакет, обработчик должен либо поместить его в буфер, либо вообще отказаться от него. Если имеется ожидающий стаб, то сообщение копируется ему. Наконец, выполняется переключение контекстов, в результате чего восстанавливаются регистры и карта памяти, принимая те значения, которые они имели в момент, когда стаб сделал вызов receive.

Теперь начинает работу серверный стаб. Он распаковывает параметры и помещает их соответствующим образом в стек. Когда все готово, выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

2.3. Организация связи с использованием удаленных объектов

Объектно-ориентированная технология в настоящее время широко применяется при разработке приложений, в том числе и распределенных. Одним из наиболее важных свойств объекта является то, что он скрывает особенности реализации, предоставляя для взаимодействия строго описанный интерфейс. Это позволяет заменять или изменять объекты, оставляя интерфейс неизменным.

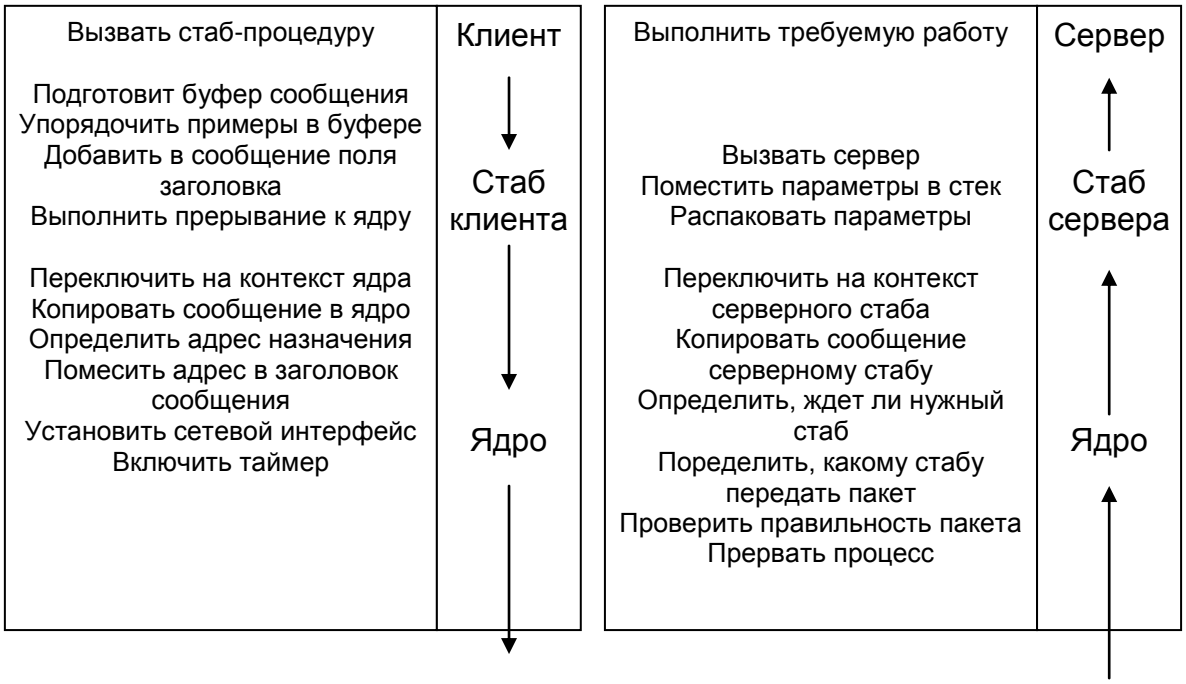


Рис. 2.5. Этапы выполнения процедуры RPC

Развитие клиент-серверной архитектуры в начале 1990-х привело к формированию *объектно-ориентированной концепции* распределенных систем, ориентированной на инкапсуляцию механизма распределенных взаимодействий и уменьшение сложности разработки распределенных приложений посредством методов объектно-ориентированной разработки и удаленных вызовов методов объектов. Основными достоинствами данного подхода стали:

- простота разработки распределенных приложений по сравнению с классическим клиент/серверным подходом;
- возможность разработки приложений для гетерогенных вычислительных сред, обеспечиваемая виртуальной машиной Java и независимым описанием интерфейсов взаимодействующих компонентов;
- возможность отделения интерфейса удаленного объекта от его непосредственной реализации.

Удаленный объект представляет собой некоторые данные, совокупность которых определяет его состояние. Это состояние можно изменять путем вызова его методов. Обычно возможен прямой доступ к данным удаленного объекта, при этом происходит неявный удаленный вызов, необходимый для передачи значения поля данных объекта между процессами. Методы и поля объекта, которые могут использоваться через удаленные вызовы, доступны через некоторый внешний интерфейс класса объекта. Внешний интерфейс компоненты распределенной системы в таких системах обычно совпадает с внешним интерфейсом одного из входящих в компоненту классов.

В момент, когда клиент начинает использовать удаленный объект, на стороне клиента создается клиентская заглушка, называемая посредником (proxy). Посредник реализует тот же интерфейс, что и удаленный объект. Вызывающий процесс использует методы посредника, который маршализирует их параметры для передачи по сети, и передает их по сети серверу. Промежуточная среда на стороне сервера десериализует параметры и передает их заглушке на стороне сервера, которую называют каркасом (skeleton) или, как и в удаленном вызове процедур, заглушкой (рис. 2.6). Каркас связывается с некоторым экземпляром удаленного объекта. Это может быть как вновь созданный, так и существующий экземпляр объекта, в зависимости от применяемой модели использования удаленных объектов, которые будут рассмотрены ниже.

Весь описанный процесс называется маршализацией удаленного объекта по ссылке. В отличие от маршализации по значению, экземпляр объекта находится в процессе сервера и не покидает его, а для доступа к объекту клиенты используют посредников. При маршализации же по значению само значение объекта сериализуется в набор байт для его передачи между процессами, после чего следует создание его копии в другом процессе.

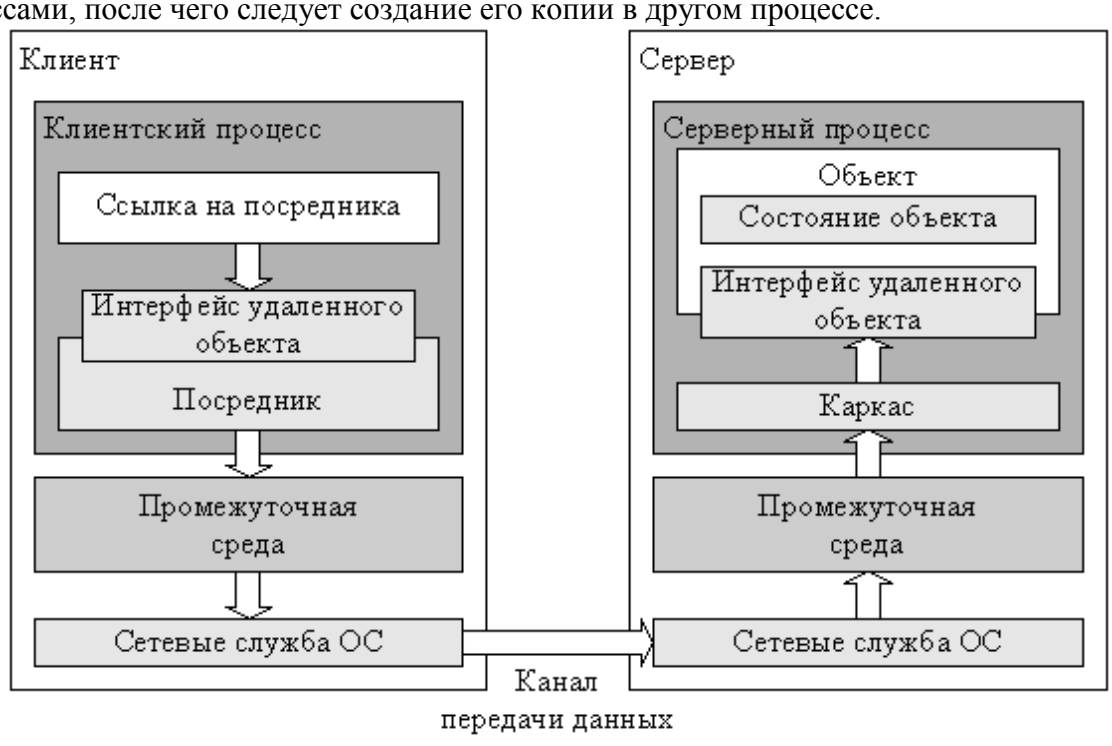


Рис. 2.6. Использование удаленных объектов

При использовании удаленных объектов проблемными являются вопросы о времени их жизни:

– в какой момент времени создается экземпляр удаленного объекта;

– в течение какого промежутка времени он существует.

Для описания жизненного цикла в системах с удаленными объектами используются два дополнительных понятия:

– активация объекта: процесс перевода созданного объекта в состояние обслуживания удаленного вызова, то есть связывания его с каркасом и посредником.

– деактивация объекта: процесс перевода объекта в неиспользуемое состояние.

Для передачи параметров по сети используется **маршализация** (marshalling) - процесс преобразования параметров для передачи их между процессами при удаленном вызове

Маршализация бывает двух типов:

- **по ссылке** – экземпляр удаленного объекта находится на сервере и не покидает его, а для доступа используются посредники;
- **по значению** – удаленный объект сериализируется и его копия передается в другой процесс.

Технология *Java RMI (Remote Method Invocation – удаленный вызов метода)* позволяет обеспечить прозрачный доступ к методам удаленных объектов, обеспечивая доставку параметров вызываемого метода, сообщение объекту о необходимости выполнения метода и передачу возвращаемого значения клиенту обратно.

Распределенное приложение, разработанное на базе технологии Java RMI, состоит из двух отдельных программ: клиента и сервера. Серверное приложение создает удаленный объект, публикует ссылки на него и ожидает, когда клиенты произведут вызов метода данного удаленного объекта. Приложение-клиент получает с сервера ссылку на удаленный объект на сервере, после чего может вызывать его методы. Технология RMI обеспечивает механизм, при помощи которого производится обмен информацией между клиентом и сервером. Процесс публикации ссылки на удаленный объект может быть реализован с помощью специального регистра или же посредством передачи удаленной объектной ссылки как части обычной операции.

Достоинствами использования технологии Java RMI для разработки распределенного приложения можно назвать возможность разрабатывать систему целиком основываясь на объектно-ориентированной концепции, не погружаясь в разработку собственных протоколов взаимодействия между распределенными компонентами систем, а также кроссплатформенность, предоставляемую виртуальной машиной Java. К недостаткам данного подхода можно отнести:

– строгую ограниченность данной технологии платформой Java;

– необходимость обработки соединений между распределенными компонентами приложения ограничивает масштабируемость используемого подхода.