5. Объектные распределенные системы

5.1 Вызов удаленных 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 вызова удаленных 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 [2].

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; обработку 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.

5.1.1 Базовые операции RPC

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычном компьютере, работающем автономно. Чтобы осуществить вызов, вызывающая процедура помещает параметры в стек в обратном порядке. 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.

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 – 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* (англ. stub – заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности, 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.

5.1.2 Этапы выполнения RPC

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рисунком 10.

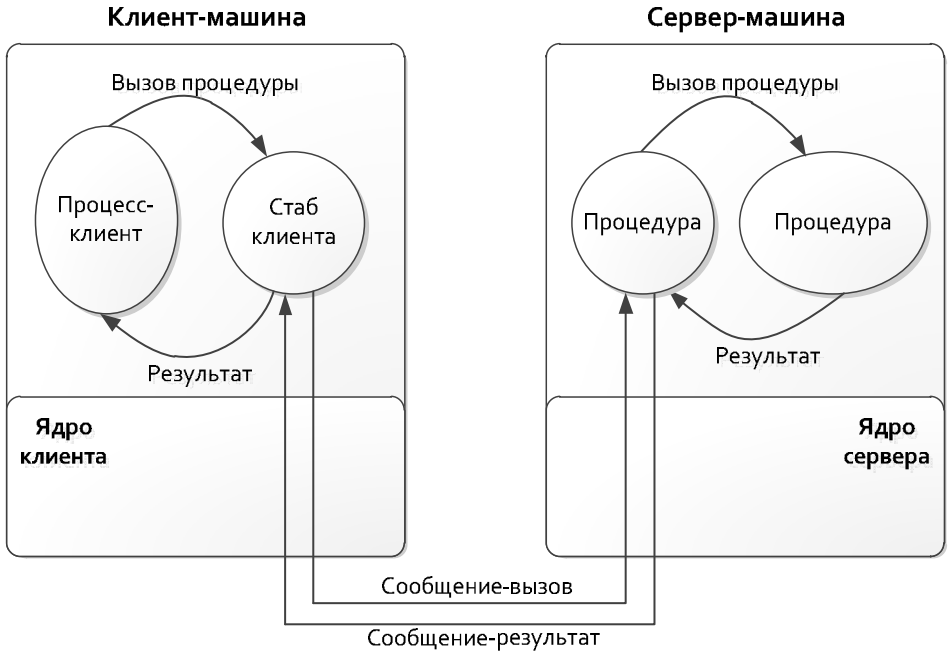

Рис. 10. Порядок удаленного вызова процедуры

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. 11. Этапы выполнения процедуры RPC

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 скопировать сообщение в свое собственное адресное пространство, запомнить адрес назначения, после чего 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. 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. По завершении работы, выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

5.2 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.

Развитие клиент-серверной архитектуры в начале 1990-х годов 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 (обеспечивалась применением виртуальных машин, например, 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*. Если возможен прямой доступ к данным удаленного объекта, это происходит посредством неявного удаленного вызова, необходимого 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.

В момент, когда клиент начинает использовать удаленный объект, на стороне клиента создается клиентская заглушка, называемая *посредником* (англ. proxy). Посредник реализует тот же интерфейс, что и удаленный объект.

Для передачи параметров по сети используется *сериализация* объектов и данных. *Сериализация* – это перевод состояния объекта в последовательность битов (чаще всего, бинарный или XML-файл), после чего его копия может быть передана в другой процесс. Обратный процесс – *десериализация* – это восстановление состояния объекта из принятой последовательности битов.

Вызывающий процесс использует методы посредника, который сериализует 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. 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.

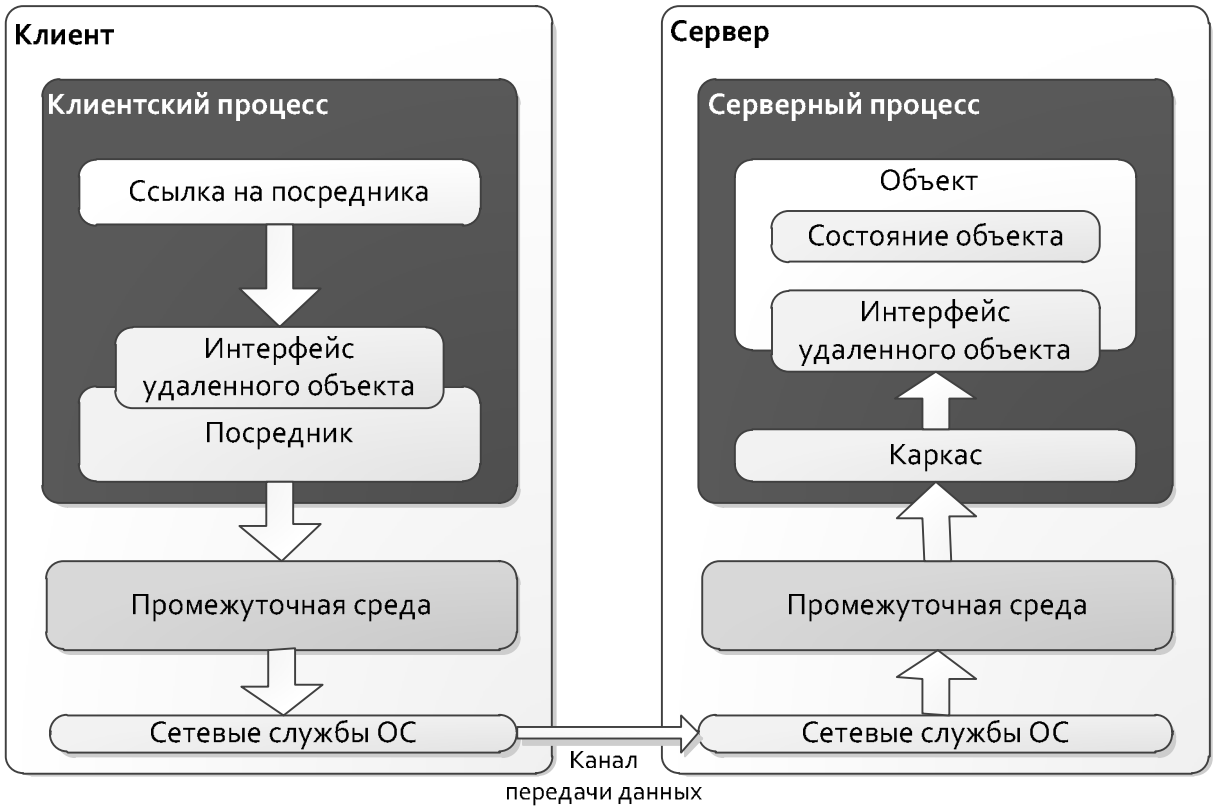

Рис. 12. Использование удаленных объектов

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.

Технология *Java RMI (Remote Method Invocation – вызов удаленных методов)* 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.

5.3 CORBA

5.3.1 Основные понятия CORBA

CORBA (Common Object Request Broker Architecture – общая архитектура брокера объектных запросов) – это технология разработки распределенных приложений, ориентированная на интеграцию распределенных изолированных систем.

В начале 1990-х гг. ночным кошмаром разработчиков было обеспечение общения программ, выполняемых на разных машинах, особенно, если использовались разные аппаратные средства, операционные системы и языки программирования. Программистам приходилось использовать технологию сокетов и самостоятельно реализовывали весь стек протоколов взаимодействия, либо их программы вовсе не взаимодействовали (другие ранние средства промежуточного программного обеспечения были ограничены средой C и Unix и не подходили для использования в неоднородных средах).

Для решения данной проблемы в 1989 г. был создан консорциум OMG (Object Management Group), основной задачей которого стала разработка и продвижение объектно-ориентированных технологий и стандартов. Это некоммерческое объединение, разрабатывающее стандарты для создания корпоративных платформо-независимых приложений.

Концептуальной инфраструктурой, на которой базируются все спецификации OMG, является Object Management Architecture (OMA). В состав OMA входят разнообразные стандартизованные или в настоящий момент стандартизируемые OMG сервисы, программные образцы и шаблоны, язык определения интерфейсов распределенных объектов IDL (Interface Definition Language), стандартизованные или стандартизируемые отображения IDL на языки программирования и, наконец, объектная модель CORBA. Главной особенностью CORBA является использование компонента ORB (Object Resource Broker – брокер ресурсов объектов) для создания экземпляров объектов и вызова их методов. Данный компонент формирует «мост» между приложением и инфраструктурой CORBA.

В 1997 г. консорциум OMG опубликовал спецификацию CORBA 2.0. В ней определялись стандартный протокол и отображение для языка C++, а в 1998 г. было определено отображение для Java. В результате разработчики получили инструментальное средство, позволяющее им относительно легко создавать неоднородные распределенные приложения. CORBA быстро завоевала популяр-

ность, и с использованием этой технологии был создан ряд критически важных приложений.

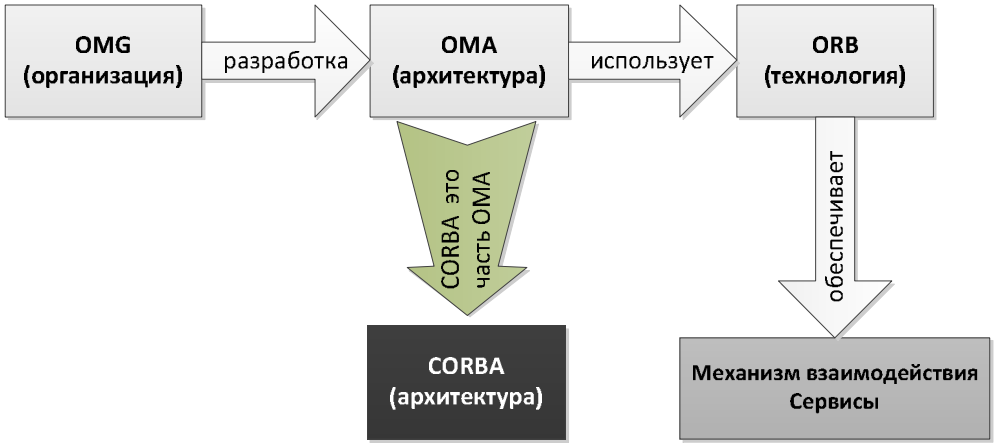

Рис. 13. Основные понятия технологии CORBA

5.3.2 Технология CORBA

Основные компоненты, составляющие архитектуру CORBA, представлены на рисунке 14. Технологический стандарт CORBA определяет язык IDL, применяемый для унифицированного описания интерфейсов распределенных объектов, и его отображения на языки Ada, C, C++, Java, Python, COBOL, Lisp, PL/1 и Smalltalk. Для преобразования описания интерфейса на языке IDL на требуемый язык программирования используется специальный компилятор. В дальнейшем построенный с его помощью программный код может быть преобразован любым стандартным компилятором в исполняемый код.

Главной особенностью CORBA является использование компонента *ORB* (Object Resource Broker – брокер ресурсов объектов) для создания экземпляров объектов и вызова их методов. Данный компонент формирует «мост» между приложением и инфраструктурой CORBA. ORB поддерживает удаленное взаимодействие с другими ORB, а также обеспечивает управление удаленными объектами, включая учет количества ссылок и времени жизни объекта. Для обеспечения взаимодействия между ORB используется протокол *GIOP* (General Inter-ORB Protocol – общий протокол для коммуникации между ORB) [42]. Наиболее распространенной реализацией данного протокола является протокол IIOP (Internet Inter-ORB Protocol – протокол взаимодействия ORB в сети интернет), обеспечивающий отображение сообщений GIOP на стек протоколов TCP/IP.

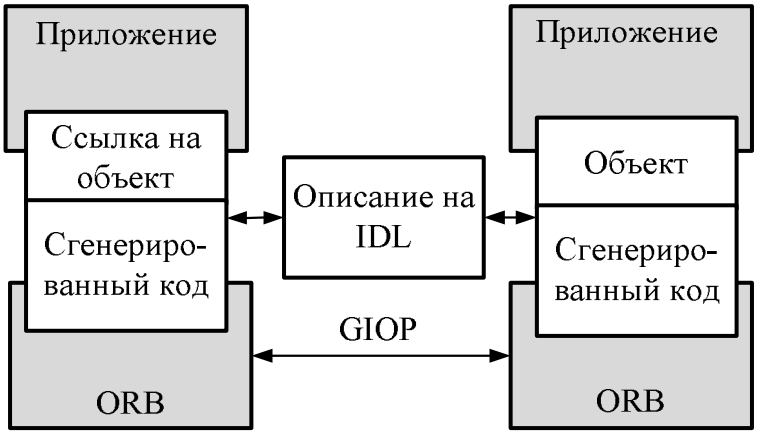

Рис. 14. Ядро архитектуры CORBA

Изначально, технология CORBA ориентирована на предоставление готовой проблемно-ориентированной инфраструктуры для создания РВС в рамках определенной проблемной области. Для этого, в состав CORBA включают набор *стандартных объектных сервисов* и *общих средств*. Спецификация CORBA предусматривает также ряд стандартизованных сервисов (CORBA Services) и горизонтальных и вертикальных Общих Средств (Common Facilities). Сервисы представляют собой обычные CORBA-объекты со стандартизованными (и написанными на IDL) интерфейсами. К таким сервисам относится, например, сервис имен NameService, сервис сообщений, позволяющий CORBA-объектам обмениваться сообщениями, сервис транзакций, позволяющий CORBA-объектам организовывать транзакции. В реальной системе не обязательно должны присутствовать все сервисы, их набор зависит от требуемой функциональности. На сегодня разработано 14 объектных сервисов.

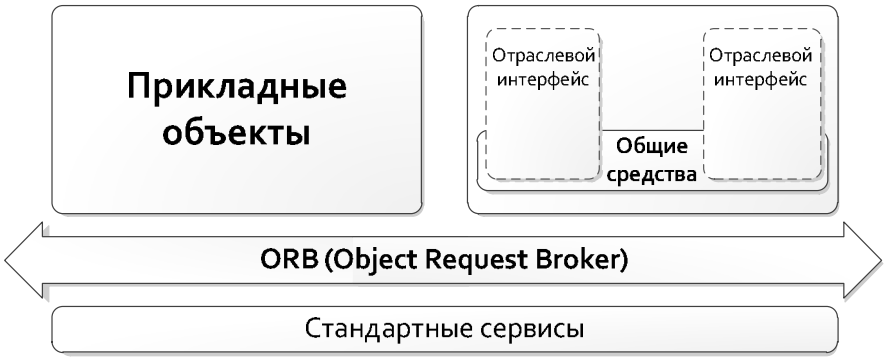

Рис. 15 Схема взаимодействия объектов ОМА

Между объектными сервисами и общими средствами CORBA нет четкой границы. Последние тоже представляют собой CORBA-объекты со стандартизованными интерфейсами. Общие средства делятся на горизонтальные (общие для всех прикладных областей) и вертикальные (для конкретной прикладной

области). Например, разработаны общие средства для медицинских организаций, для ряда производственных сфер и т.п.

5.3.3 Разработка на основе CORBA

Процесс разработки приложения с использованием технологии CORBA состоит из следующих 4-х этапов:

1. Определение интерфейса на IDL.
2. Обработка IDL для создания кода заглушки и скелетона.
3. Создание кода реализации объекта (сервер).
4. Создание кода использования данного объекта (клиент).

Язык определения IDL позволяет независимо от используемого языка программирования создать универсальное описание интерфейса будущей системы.

Созданный на IDL код должен специальным компилятором преобразовываться в код интерфейса объекта на требуемом языке программирования. После чего, на клиенте автоматически генерируется заглушка, преобразующая вызовы методы данного интерфейса в обращения к ORB. На сервере программист на основе сгенерированного интерфейса создает собственную реализацию данного класса. Скелетон автоматизирует получение и обработку удаленного вызова методов, поступающих через ORB.

```
// Модуль системы ценовых предложений
module QuoteSystem
{
  // Структура данных цены
  struct Quote
  {
    string value;
  }

  // Интерфейс сервера ценовых предложений
  interface QuoteServer
  {
    // Атрибут определяющий фондовую биржу
    string exchange;
    // Исключение «Неизвестный идентификатор»
    exception UnknownSymbolException { string message; };
    // Поиск предложения по идентификатору
    Quote getQuote (in string symbol)
                          raises (UnknownSymbolException);
  };
};
```

Рис. 16. Пример описания интерфейса на языке IDL

По сравнению с классическим клиент-серверным подходом, использование технологии CORBA для разработки распределенных приложений имеет следующие преимущества:

- использование IDL для описания интерфейсов позволяет разрабатывать программные компоненты независимо от языка программирования и базовой операционной системы;
- поддержка богатой инфраструктуры распределенных объектов;
- прозрачность вызова удаленных объектов.

Однако программные решения на базе технологии CORBA редко выходят за рамки отдельных предприятий. Разработка крупномасштабных межорганизационных систем на базе технологии CORBA сопряжена со следующими трудностями:

- плохая совместимость различных реализаций технологии CORBA от различных поставщиков;
- проблемы взаимодействия узлов CORBA через Интернет;
- несогласованность многих архитектурных решений CORBA и отсутствие компонентной модели, которая могла бы значительно упростить разработку.

На смену технологии CORBA, пришли стандартизованные протоколы вебсервисов, такие как XML, WSDL, SOAP и др. В настоящее время CORBA используется для реализации узкого круга унаследованных приложений.