Распределенные информационно-аналитические системы

Лекция № 15.

Технология DCOM

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по основным понятиям DCOM; технологии и архитектуре DCOM; механизмам работы удаленного вызова процедур в DCOM.

Учебные вопросы:

1. Основные понятия DCOM.

2. Технология и архитектура DCOM.

3. Механизм работы удаленного вызова процедур в DCOM.

1. Основные понятия DCOM

Для того чтобы различные фрагменты сложного приложения могли работать вместе через Интернет, необходимо обеспечить между ними надежные и защищенные соединения, а также создать специальную систему, которая направляет программный трафик. С начала 1990-х годов над решением этой задачи трудятся две конкурирующие группы разработчиков.

Одну из них – консорциум Object Management Group (OMG) – поддерживают компании IBM, Sun Microsystems и ряд других производителей. В 1991 году OMG предложил архитектуру распределенных вычислений, получившую название Common Object Request Broker Architecture (CORBA). Сегодня она применяется многими крупными организациями для развертывания распределенных вычислений в масштабах предприятия на базе Unix-серверов и мэйнфреймов.

Другое технологическое направление представляет Microsoft, создавшая модель компонентных объектов (Component Object Model – COM) и распределенная модель компонентных объектов (Distributed Component Object Model – DCOM), которые встраиваются в операционные системы Windows.

COM (Component Object Model) – модель компонентных объектов от Microsoft. Стандартный механизм, включающий интерфейсы, с помощью которых одни объекты предоставляют свои сервисы другим.

COM представляет собой основанную на объектах клиент-серверную модель, разработанную Microsoft для обеспечения взаимодействия между компонентами программного обеспечения и приложениями. Microsoft расширила эту технологию добавлением элементов ActiveX, которые представляют результат развития технологий OLE и OCX. OCX (OLE Custom eXtension) – это программируемые компоненты-приложения с интерфейсом на базе OLE, позволяющим легко включать их в другие приложения. С 1996 года они стали называться ActiveX.

Сейчас Microsoft предлагает использовать термин Active technologies вместо ActiveX, включая в новые технологии такие составляющие:

- Active Documents (активные документы);
- ActiveX (управляющие элементы);
- Active Scripting controls;
- Automation (автоматизация, прежде известная как OLE Automation).

Ключевым аспектом COM является то, что эта технология обеспечивает связь между клиентами и серверами посредством ***интерфейсов***. Именно интерфейс предоставляет клиенту способ «узнать» у сервера, какие именно возможности он поддерживает на этапе выполнения. Для расширения возможностей сервера необходимо просто добавить новый интерфейс к существующим.

DCOM (Distributed Component Object Model) – распределенная модель компонентных объектов. Расширение модели COM фирмы Microsoft, ориентированное на поддержку и интеграцию распределенных объектных приложений, функционирующих в сети.

DCOM – программная архитектура для распределения приложений между несколькими компьютерами в сети. Программный компонент на одной из машин может использовать DCOM для передачи сообщения (его называют удаленным вызовом процедуры) к компоненту на другой машине. DCOM автоматически устанавливает соединение, передает сообщение и возвращает ответ удаленного компонента.

COM и DCOM – технологии, обеспечивающие взаимодействие между компонентами приложения и позволяющие развертывать распределенное приложение на платформе Windows.

COM является моделью программирования на основе объектов, которая упрощает взаимодействие различных приложений и компонентов, а DCOM – это своего рода «клей», связывающий воедино разнообразные технологии, применяемые в распределенных приложениях.

DCOM дает возможность двум или нескольким компонентам легко взаимодействовать друг с другом независимо от того, когда и на каком языке программирования они были написаны, а также где именно они находятся и в какой операционной системе работают (рисунок 1).

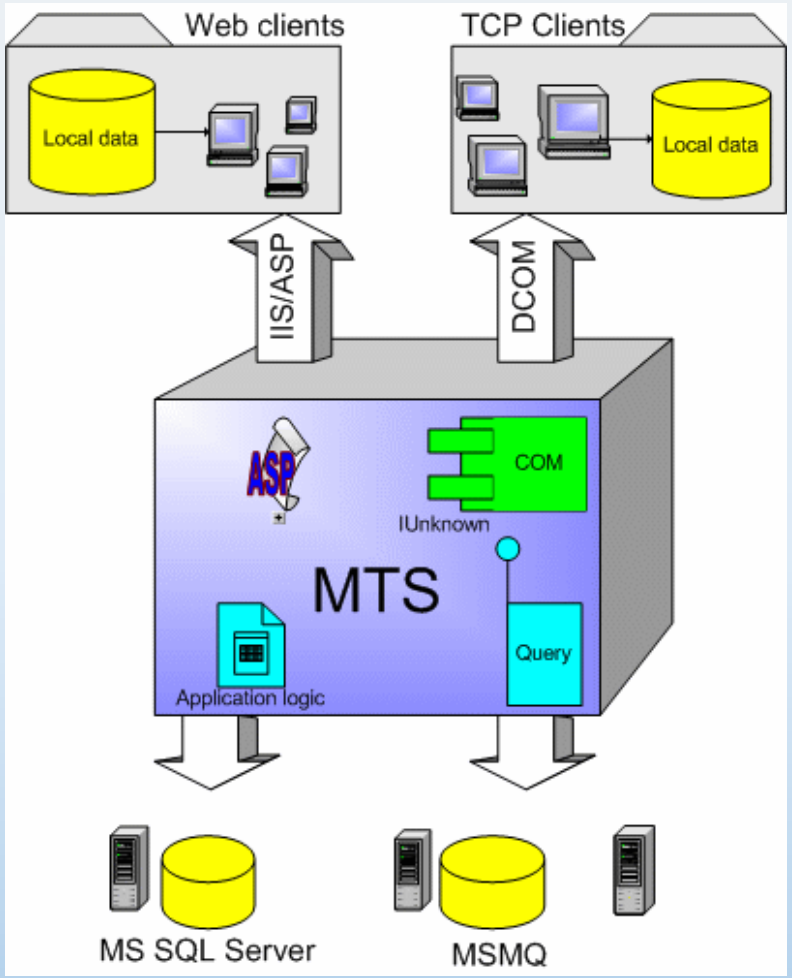

Рисунок 1 – Взаимодействие компонентов на основе DCOM

Распределенные многопользовательские приложения могут модифицировать таблицы базы данных и взаимодействовать даже с другими сущностями, не имеющими отношения к базам данных, например с очередями сообщений.

Транзакции реализуют простой механизм выполнения операции по принципу «все или ничего». Операция считается выполненной только при успешной модификации всех участвующих в ней объектов, а при неудачном изменении одного из них вся операция отменяется.

Преимуществом DCOM является значительная простота использования. Если программисты пишут свои Windows-приложения с помощью ActiveX (предлагаемого Microsoft способа организации программных компонентов), то операционная система будет автоматически устанавливать необходимые соединения и перенаправлять трафик между компонентами, независимо от того, размещаются ли компоненты на той же машине или нет.

Способность DCOM связывать компоненты позволила Microsoft наделить Windows рядом важных дополнительных возможностей, в частности, реализовать сервер Microsoft Transaction Server, отвечающий за выполнения транзакций баз данных через Интернет. Новая же версия COM+ еще больше упростит программирование распределенных приложений, в частности, благодаря таким компонентам, как базы данных, размещаемые в оперативной памяти.

Однако у DCOM есть ряд недостатков. Это решение до сих пор ориентировано исключительно на системы Microsoft. DCOM изначально создавалась под Windows. Хорошо известно, что Microsoft заключила соглашение с компанией Software AG, предмет которого – перенос DCOM на другие платформы.

В числе недостатков и то, что архитектура предусматривает использование для поиска компонентов в сети разработанной Microsoft сетевой службы каталогов Active Directory, появившейся в Windows начиная с версии 2000. Это приводит к зависимости пользователя от соответствующих служб Microsoft и резко снижает безопасность использования такой технологии.

DCOM против CORBA. Обе системы DCOM компании Microsoft и CORBA консорциума OMG поддерживают распределенные вычисления. Однако, две эти технологии развиваются в разных направлениях.

Microsoft расширила DCOM, добавив службы обработки транзакций, упростив программирование распределенных приложений и усовершенствовав поддержку Unix и других платформ.

OMG расширяет свою компонентную модель за счет служб, ориентированных на конкретные отрасли, то есть телекоммуникации, производство, электронную коммерцию, финансы, медицину, транспорт и коммунальные услуги.

2. Технология и архитектура DCOM

Обычные COM объекты имеют один важный недостаток – их можно использовать только в пределах отдельного компьютера, то есть и COM-сервер, и COM-клиент должны работать в рамках одной операционной системы. Их нельзя разместить на разных компьютерах и организовать общение через сетевую среду, то есть в рамках технологии обычной COM нельзя создать распределенную программную систему.

Для того, чтобы устранить этот недостаток, специалисты Microsoft разработали распределенную модель COM – DCOM. Модель DCOM расширяет возможности COM, позволяя обращаться к COM объектам в режиме удаленного доступа.

Распределенная модель DCOM разработана для того, чтобы обеспечить взаимодействие между COM-объектами в режиме удаленного доступа. Это позволяет COM-серверу и COM-клиентам располагаться на разных компьютерах и связываться по сети.

По сути, механизм поддержки DCOM реализует те же задачи, что и механизм поддержки COM, но дополнительно он обеспечивает безопасность приложений в сетевой среде. Создание DCOM стало возможным в результате расширения возможностей механизма ***удаленного вызова процедур (Remote Procedure Calls – RPC)*** и превращение его в объектный (object) RPC. В настоящее время эту модель можно рассматривать как средство реализации распределенных многоплатформенных приложений.

В модели DCOM используется концепция «прозрачности размещения», которая подразумевает, что можно, не изменяя программного кода, выполнить приложение COM сервера на разных компьютерах сети. При этом ни приложение-сервер, ни приложение-клиент не «знают», на каком компьютере фактически работает «собеседник». Они могут выполняться на одном компьютере, на разных компьютерах локальной сети, или даже на разных компьютерах, подключенных к глобальной сети, например Интернет. Приложение-клиент «узнает» о существовании сервера из данных в системном реестре, которые могут быть легко изменены.

Но за такую «прозрачность» приходится платить. В данном случае приходится предпринимать дополнительные меры для обеспечения безопасности. В средствах поддержки DCOM эта задача решается в рамках сетевой системы доменов, уже давно реализованной в Microsoft.

Клиентские и серверные DCOM-приложения должны следовать правилам аутентификации и авторизации. Это означает, что приложение должно пройти через определенную процедуру регистрации при попытке получить доступ к компьютеру, на котором выполняется серверное приложение. Сервер должен аутентифицировать (идентифицировать) клиента – проверить, действительно ли он является тем, за кого себя выдает, и авторизировать доступ – выяснить, имеет ли пользователь необходимые привилегии доступа.

Приложения DCOM могут полагаться на те параметры безопасности, которые хранятся в системном реестре Registry, или изменить эти параметры программно. Эти два подхода к реализации подсистемы безопасности получили наименования ***декларативной*** и ***программируемой безопасности***.

Главной проблемой при программировании DCOM-приложений является как раз подсистема обеспечения безопасности.

Модель DCOM базируется на модели COM. Целью создания COM была поддержка разработки компонентов, которые могли бы динамически активизироваться и взаимодействовать друг с другом. Компонент в COM – это исполняемый код, содержащийся в динамически компонуемой библиотеке (DLL) или исполняемой программе.

Сама по себе модель COM существует в виде библиотек, компонуемых с процессом. Изначально она разрабатывалась для поддержки так называемых составных документов (compound documents).

Составные документы – это документы, построенные из разнородных частей, таких как текст (форматированный), изображения, электронные таблицы и т.п. Каждая из этих частей может быть отредактирована при помощи ассоциированного с ней приложения.

Для поддержки бесчисленного множества составных документов Microsoft нужен был обобщенный метод для разделения отдельных частей и объединения их в единую сущность.

Вначале это привело к ***технологии связывания и внедрения объектов (Object Linking and Embedding – OLE)***. Первая версия OLE использовала для передачи информации между частями документа примитивный и негибкий способ обмена сообщениями. Вскоре она была заменена следующей версией, также называвшейся OLE, но построенной на базе более гибкого механизма COM, что привело к появлению структуры, представленной на рисунке 2.

Рисунок 2 – Общая структура ActiveX, OLE и COM

На рисунке 2 также показан элемент ActiveX – этим термином в настоящее время называют все, что относится к OLE, плюс некоторые новшества, к которым относят гибкую (как правило) способность компонентов выполняться в разных процессах, поддержку сценариев и более или менее стандартную группировку объектов в так называемые элементы управления ActiveX.

Среди экспертов по DCOM (даже внутри Microsoft) не существует согласия по вопросу о точном определении ActiveX, поэтому мы даже не будем пытаться дать подобное определение.

Модель DCOM добавила к этой структуре весьма существенную вещь – способность процесса работать с компонентами, размещенными на другой машине. Однако базовый механизм обмена информацией между компонентами, принятый в DCOM, очень часто совпадает с соответствующими механизмами COM. Другими словами, для программиста разница между COM и DCOM часто скрыта за различными интерфейсами.

Как мы увидим, DCOM в первую очередь предоставляет прозрачность доступа. Другие виды прозрачности менее очевидны.

Один объект может реализовывать одновременно несколько интерфейсов. Однако в отличие от CORBA в DCOM имеются только ***бинарные интерфейсы (binary interfaces)***. Такой интерфейс, в сущности, представляет собой таблицу с указателями на реализации методов, которые являются частью интерфейса. Разумеется, для определения интерфейса по-прежнему удобно использовать специальный язык определения интерфейса (IDL). В DCOM также имеется такой язык под названием ***MIDL (Microsoft IDL – язык IDL от Microsoft)***, при помощи которого генерируются бинарные интерфейсы стандартного формата.

Достоинство бинарных интерфейсов состоит в том, что они не зависят от языка программирования. В случае CORBA каждый раз для поддержки нового языка программирования необходимо отображать описания IDL на конструкции этого языка. Подобная стандартизация в случае бинарных интерфейсов не нужна. Разницу между этими двумя подходами иллюстрирует рисунке 3.

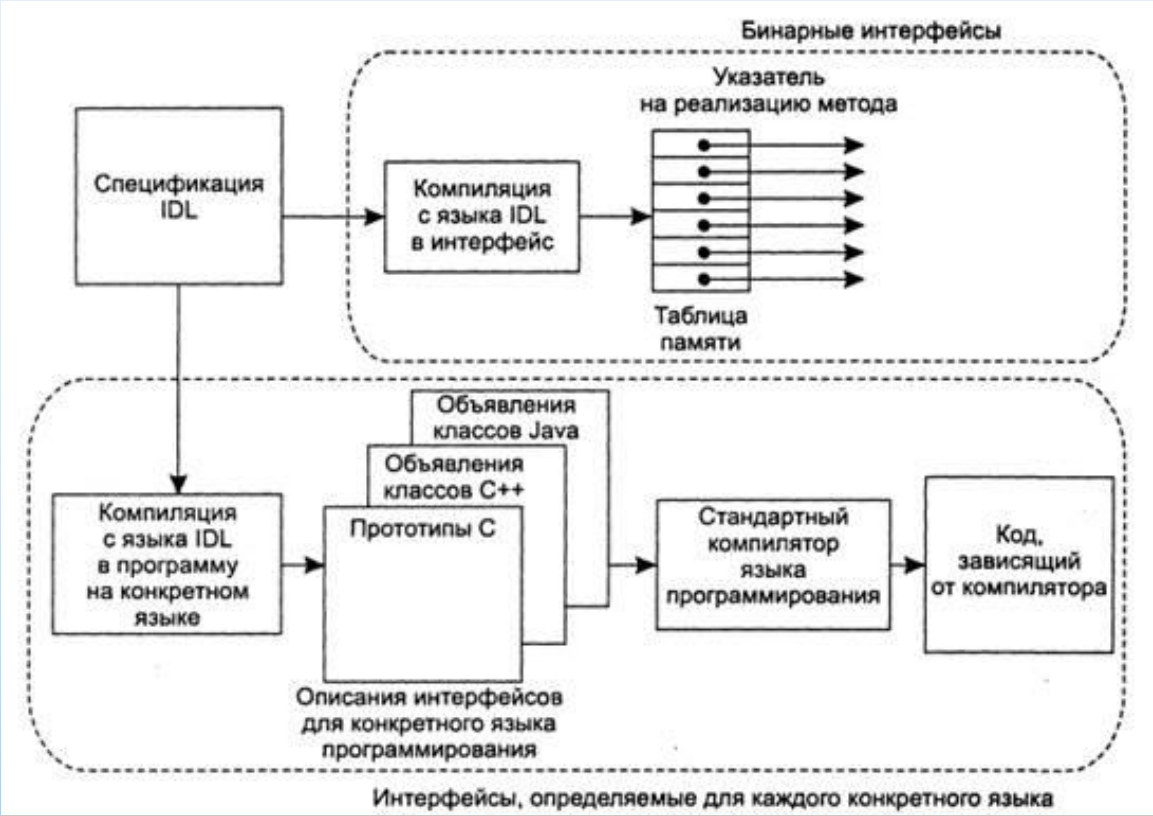

Рисунок 3 – Разница между интерфейсами CORBA и DCOM

Каждый интерфейс в DCOM имеет уникальный 128-битный идентификатор, который называется ***идентификатором интерфейса (Interface Identifier – IID)***. Каждый IID абсолютно уникален, не существует двух интерфейсов с одинаковым идентификатором IID. Этот идентификатор создается путем комбинирования большого случайного числа, локального времени и адреса сетевого интерфейса текущего хоста. Вероятность того, что два сгенерированных индикатора совпадут, практически равна нулю.

Объект DCOM создается как экземпляр класса. Чтобы создать такой объект, необходимо иметь доступ к соответствующему классу. Для этой цели DCOM содержит ***объекты класса (class objects)***.

Формально таким объектом может быть все, что поддерживает интерфейс IClassFactory. Этот интерфейс содержит метод CreateInstance, который похож на оператор new в языках C++ и Java. Вызов метода CreateInstance для объекта класса приводит к созданию объекта DCOM, содержащего реализацию интерфейсов объекта класса.

Таким образом, объект класса представляет собой коллекцию объектов одного типа, то есть реализующих один и тот же набор интерфейсов. Объекты, принадлежащие к одному классу, обычно различаются только в части текущего состояния. Путем создания экземпляров объекта класса становится возможным обращение к методам этих интерфейсов. В DCOM на любой объект класса можно сослаться по его глобальному уникальному идентификатору класса (Class Identifier, CLSID).

Все объекты реализуют стандартный объектный интерфейс IUnknown. Когда путем вызова CreateInstance создается новый объект, объект класса возвращает указатель на этот интерфейс. Самый важный метод, содержащийся в IUnknown, – метод QueryInterface, который на основании IID возвращает указатель на другой интерфейс, реализованный в объекте.

Важное отличие от объектной модели CORBA состоит в том, что ***все объекты в DCOM нерезидентные***. Другими словами, как только у объекта не остается ссылающихся на него клиентов, этот объект удаляется. Подсчет ссылок производится путем вызова методов AddRef и Release, входящих в интерфейс IUnknown. Наличие исключительно нерезидентных объектов делает невозможным хранение каждым из объектов своего глобально уникального идентификатора. По этой причине объекты в DCOM могут вызываться только по указателям на интерфейсы. Соответственно, для передачи ссылки на объект другому процессу необходимо предпринять специальные меры.

DCOM также требует динамического обращения к объектам. Объекты, запросы к которым могут создаваться динамически, во время выполнения, должны иметь реализацию интерфейса IDispatch. Этот интерфейс подобен интерфейсу динамического обращения (DII) в системе CORBA.

Общее описание подсистемы вызова удаленных процедур по технологии DCOM. Серверы автоматизации могут выполняться как в адресном пространстве контроллера, так и в собственном адресном пространстве.

Отметим, что локальный COM-сервер можно превратить в удаленный (то есть запускаемый на компьютере сети, отличном от компьютера, содержащего контроллер). Такой способ использования серверов особенно удобен в случае, когда сервер требует для своей работы наличия особых ресурсов, недоступных на всех компьютерах, содержащих контроллеры, таких, как дополнительный объем оперативной памяти, уникальное оборудование, дополнительное программное обеспечение, требующее сложной конфигурации, настройки и поддержки (а если обратить внимание на другие механизмы использования сервисов, доступных в сети, отличные от COM, то, возможно, и другой операционной системы).

Один из часто встречающихся примеров практического использования удаленного доступа – использование удаленного запуска MS Excel, установленного на одном-единственном компьютере локальной сети, для печати отчетов и диаграмм. В этом случае можно снизить требования к ресурсам рабочих станций, содержащих приложения-контроллеры Excel (и сэкономить некоторые средства на покупке нескольких копий Excel).

Второй пример использования удаленного запуска серверов автоматизации, вызывающий в последнее время немалый интерес разработчиков информационных систем – это пример трехзвенной информационной системы с «тонким» клиентом – контроллером автоматизации, и сервером автоматизации (в данном случае он иногда называется сервером приложений), предоставляющим «тонкому» клиенту сервисы, связанные с доступом к данным, содержащимся, в свою очередь, в какой-либо серверной СУБД. В этом случае, помимо экономии ресурсов компьютеров, содержащих контроллеры, имеется еще одно преимущество, связанное с тем, что для доступа к данным используется программное обеспечение, требующее отдельной установки, сложной настройки и поддержки, такое как клиентская часть серверной СУБД, а также, как правило, библиотека Borland Database Engine, драйверы SQL Links, а иногда и ODBC-драйверы. Если это программное обеспечение используется только сервером, но не используется контроллерами, сопровождение такой системы значительно облегчается.

Использование интерфейсов. Единственным способом взаимодействия клиента с COM-объектом является использование интерфейсов. Интерфейс связан с группой функций, связанных, в свою очередь, с COM-объектом, и не содержит их реализации. На этапе выполнения интерфейс всегда представляется указателем, идентифицируемым с помощью идентификатора IID (Interface Identifier – по существу, это тот же GUID) и указывающим на другой указатель, который, в свою очередь, указывает на таблицу, содержащую адреса реализации каждой функции из этой группы (рисунок 4).

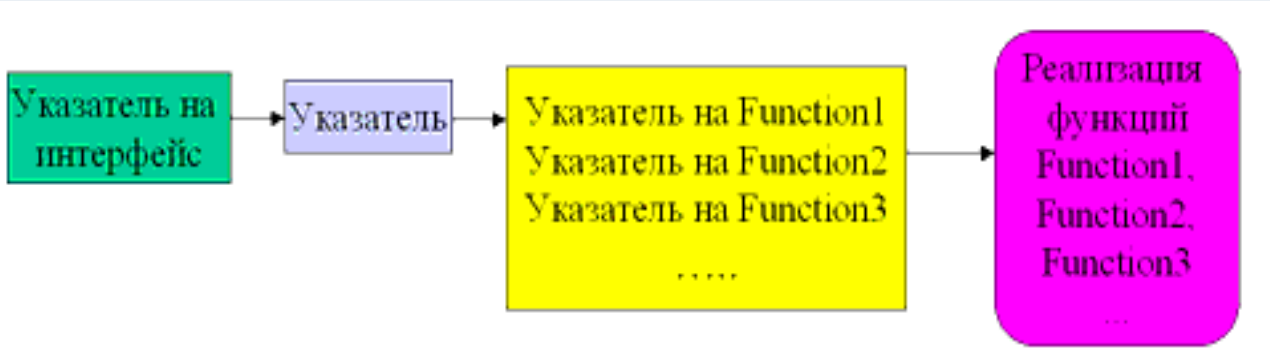

Рисунок 4 – Структура интерфейса

Эта структура позволяет обеспечить маршалинг – пересылку указателя между процессами (и в общем случае между компьютерами).

Реализация интерфейса COM-объекта представляет собой создание в памяти подобной структуры и предоставление указателя на нее. Фактически такая структура похожа на структуру, используемую для построения таблиц виртуальных функций в C++, поэтому таблица указателей на функции в этой структуре иногда называется «vtable» (или таблицей виртуальных методов).

Весьма существенно, что адреса самих функций определяются на этапе выполнения с помощью указателя на интерфейс в момент обращения к ним, а не хранятся в памяти статически. Отметим также, что при наличии указателя на функцию в таблице указателей всегда существует и ее реализация, поэтому клиент не должен ни обрабатывать сообщений об обращении к несуществующим функциям, ни иметь собственной версии их реализации.

Когда клиенту требуется COM-объект, он пытается найти сервер, посылает серверу запрос на создание объекта, а затем получает от него указатель на исходный интерфейс, с помощью которого можно получить дополнительные указатели на другие интерфейсы этого объекта.

COM находит местоположение сервера на основании записи в реестре, загружает его в оперативную память, запрашивает у сервера нужный объект (инициируя его создание) и получает указатель на интерфейс. Если сервер и клиент выполняются в разных процессах, COM автоматически передает указатель на интерфейс в процесс клиента.

Маршалинг и удаленный доступ. Если сервер является внутренним («in-process»), то есть выполненным в виде DLL, она загружается в адресное пространство клиента с помощью функции Win32 API LoadLibrary. В этом случае значение указателя на интерфейс непосредственно доступно клиенту.

Если сервер локальный, COM использует функцию CreateProcess, загружая исполняемый файл и инициализируя COM в адресном пространстве последнего. Затем COM в процессе сервера связывается с COM в процессе клиента каким-либо доступным способом. В этом случае обычно нет возможности передать клиенту точное значение указателя на интерфейс, так как это указатель на объект в другом адресном пространстве. Тогда используется маршалинг, создающий так называемый «marshalling packet», содержащий необходимую информацию для соединения с процессом, в котором создан объект.

Этот пакет создается с помощью функции COM API CoMarshalInterface, затем он передается процессу клиента любым доступным способом, где другая функция CoUnMarshalInterface превращает этот пакет в указатель на интерфейс.

В действительности маршалинг создает в обоих адресных пространствах два объекта: «stub» (заглушка) – представитель клиента в адресном пространстве сервера, имеющий дело с реальным указателем на интерфейс, и «proxy» – представитель сервера в адресном пространстве клиента.

Эти два объекта соединяются между собой каким-либо доступным способом, и представитель сервера передает клиентскому процессу указатель на интерфейс. Сходная технология используется при осуществлении ***удаленного вызова процедур (Remote Procedure Calls – RPC)***.

Естественно, proxy-объект не содержит реализации интерфейса. Все аргументы вызываемых функций помещаются в пакет, передаваемый stub-объекту с использованием RPC.

Stub-объект распаковывает переданные аргументы, помещает их в стек и обращается к реальному объекту, используя существующий указатель на интерфейс. Результат выполнения функции упаковывается в пакет и посылается proxy-объекту, который распаковывает его и передает клиенту.

В случае сервера, расположенного на удаленном компьютере, при обращении к серверу COM соединяется со специальным резидентным процессом удаленного компьютера, контролирующим удаленный запуск сервисов на нем. Этот процесс, называемый иногда ***менеджером управления службами (Service Control Manager – SCM)***, осуществляет запуск сервера на удаленном компьютере и возвращает указатель на интерфейс клиентскому компьютеру и клиентскому процессу.

В качестве SCM могут быть использованы средства различных производителей, как основанные непосредственно на технологии Microsoft DCOM (Distributed COM), так и являющиеся расширением этой технологии (например, ObjectFactory из комплекта поставки OLEnterprise корпорации Inprise).

Обязательное наличие такого процесса (и возможностей его конфигурации) диктуется элементарными соображениями безопасности – было бы неестественно, если бы любой пользователь сети мог запустить любой процесс на любом из компьютеров, к этой сети подключенных.

В остальном маршалинг осуществляется точно так же, как и в случае локального сервера, за исключением того, что proxy и stub, общаясь с помощью того же самого механизма RPC, физически находятся на разных компьютерах (рисунок 5).

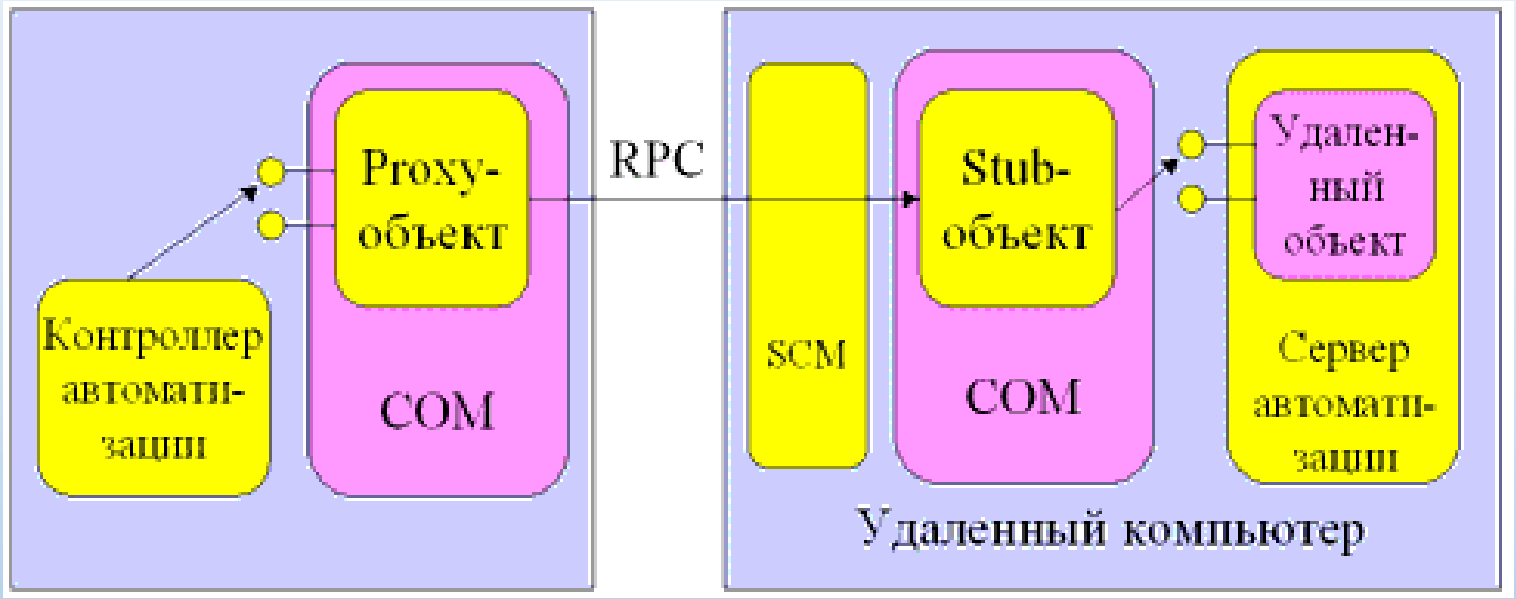

Рисунок 5 – Осуществление удаленного доступа к серверу автоматизации

Библиотека типов обычно ассоциируется с приложением или другим крупным компонентом, содержащим различные объекты классов. Сама по себе библиотека может храниться в отдельном файле или быть частью приложения. В любом случае библиотека типов в первую очередь требуется для точного описания сигнатуры динамически вызываемых методов. Кроме того, библиотеки типов могут использоваться в системах программирования, например, для отображения структуры разрабатываемых интерфейсов на экране.

Для того чтобы действительно активизировать объект, то есть гарантировать, что он создан и помещен в процесс, из которого будет в состоянии принимать обращения к методам, DCOM использует реестр Windows в комбинации с менеджером управления службами SCM. Кроме всего прочего, реестр используется для записи отображения идентификаторов классов (CLSID) на локальные имена файлов, содержащих реализации классов. Чтобы создать объект, процесс сначала должен убедиться, что соответствующий объект класса загружен.

Если объект выполняется на удаленном хосте, дело происходит иначе. В этом случае клиент контактирует с менеджером управления службами этого хоста, который представляет собой процесс, ответственный за активизацию объектов, подобно хранилищу реализаций в CORBA. SCM этого удаленного хоста просматривает свой локальный реестр в поисках файла, ассоциированного с CLSID, после чего запускает процесс, содержащий этот объект. Сервер выполняет маршалинг указателя на интерфейс и возвращает его клиенту, который выполняет демаршалинг указателя и передает его заместителю.

Процесс клиента получает доступ к SCM и реестру, что помогает ему найти удаленный объект и выполнить привязку к нему. Клиенту предлагается заместитель, реализующий интерфейсы этого объекта.

Серверы DCOM. Предполагается, что объекты DCOM являются более или менее самодостаточными в том смысле, что различные возможности, которые в CORBA предоставляет переносимый адаптер объектов, жестко закодированы в каждом объекте DCOM.

DCOM предлагает стандартный способ активизации объектов, в том числе и размещенных на удаленном хосте.

На хосте, поддерживающем объекты DCOM, должен выполняться менеджер управления службами SCM, отвечающий за активизацию объектов, которая производится следующим образом. Предположим, что клиент владеет CLSID объекта класса. Чтобы создать новый объект, он передает этот идентификатор CLSID в свой локальный реестр, где выполняется поиск хоста, на котором должен находиться нужный сервер объектов. Затем CLSID передается менеджеру SCM, ассоциированному с найденным хостом. В поисках файла, позволяющего загрузить объект класса, который может создавать экземпляры нового объекта, SCM ищет этот идентификатор CLSID уже в своем локальном реестре.

SCM обычно начинает выполнение нового процесса с загрузки найденного объекта класса, после чего создает экземпляр объекта. В SCM регистрируется также порт, из которого «свежеиспеченный» сервер будет получать входящие запросы, а также идентификатор нового объекта. Эта информация привязки возвращается клиенту, который после ее получения может работать с сервером объекта напрямую, без вмешательства SCM.

Такой подход оставляет решение всех вопросов, связанных с управлением объектами на сервере, разработчику. Чтобы немного облегчить ему жизнь, DCOM предоставляет средства управления объектами.

Одно из таких средств носит название ***активизации «на лету» («just-in-time» activation)***, или ***JIT-активизации (JIT-activation)***. Под этим названием скрывается механизм, при помощи которого сервер может управлять активизацией и удалением объектов. Он работает следующим образом.

Обычно объект активизируется в результате запроса клиента на создание экземпляра нового объекта данного класса. Сервер объекта хранит объект в памяти до тех пор, пока существуют клиенты, хранящие ссылку на этот объект. Однако при JIT-активизации сервер может удалить объект тогда, когда захочет. Так, например, если сервер обнаруживает, что у него не осталось памяти, он может освободить место для новых объектов, удалив старые.

Когда клиент вызывает метод ранее удаленного объекта, сервер немедленно создает новый объект того же класса и обращается к указанному методу этого объекта. Разумеется, такой подход допустим только при работе с объектами без фиксации состояния, то есть объектами, которые в паузах между обращениями не хранят никакой информации о своем внутреннем состоянии. Все, что остается для работы с таким объектом, – его методы. Подобные объекты использовались для вызова библиотечных функций еще в те дни, когда не придумали объектно-ориентированного программирования. Если же предполагается сохранять информацию о состоянии объекта, после каждого обращения ее придется записывать на диск и при каждом создании объекта вновь загружать.

Защита. Как и в CORBA, система защиты DCOM в основном обеспечивает защиту обращений к объектам. Однако способ реализации защищенных обращений здесь абсолютно иной. Хотя DCOM чаще всего входит в одну из операционных систем семейства Windows, разработчики DCOM совершенно оправданно решили по возможности отделить защиту DCOM от служб защиты Windows. Таким образом они сохранили возможность реализации DCOM на других операционных системах с сохранением защиты обращений.

В DCOM существует два направления реализации защиты. Важная роль в защите DCOM отводится реестру, при помощи которого осуществляется декларативная защита. Кроме того, защита может быть реализована и программным путем, в том смысле, что приложение может само решать вопросы контроля доступа и аутентификации.

3. Механизм работы удаленного вызова процедур в DCOM

Перед непосредственным вызовом на клиентской и серверной стороне должны быть созданы специальные структуры (процедуры, файлы) – это так называемые ***клиентский стаб (stub)*** и ***серверный скелетон (skeleton)***, которые необходимы для корректной работы удаленного вызова процедур RPC. Чаще всего, они генерируются автоматически специальными утилитами по основному коду программы.

Идея, положенная в основу RPC, состоит в том, чтобы сделать удаленный вызов процедуры выглядящим по возможности также, как и локальный вызов 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 – клиентский стаб.

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.

Этапы выполнения удаленного вызова процедур RPC. Взаимодействие программных компонентов при выполнении удаленного вызова процедур иллюстрируется на рисунке 6.

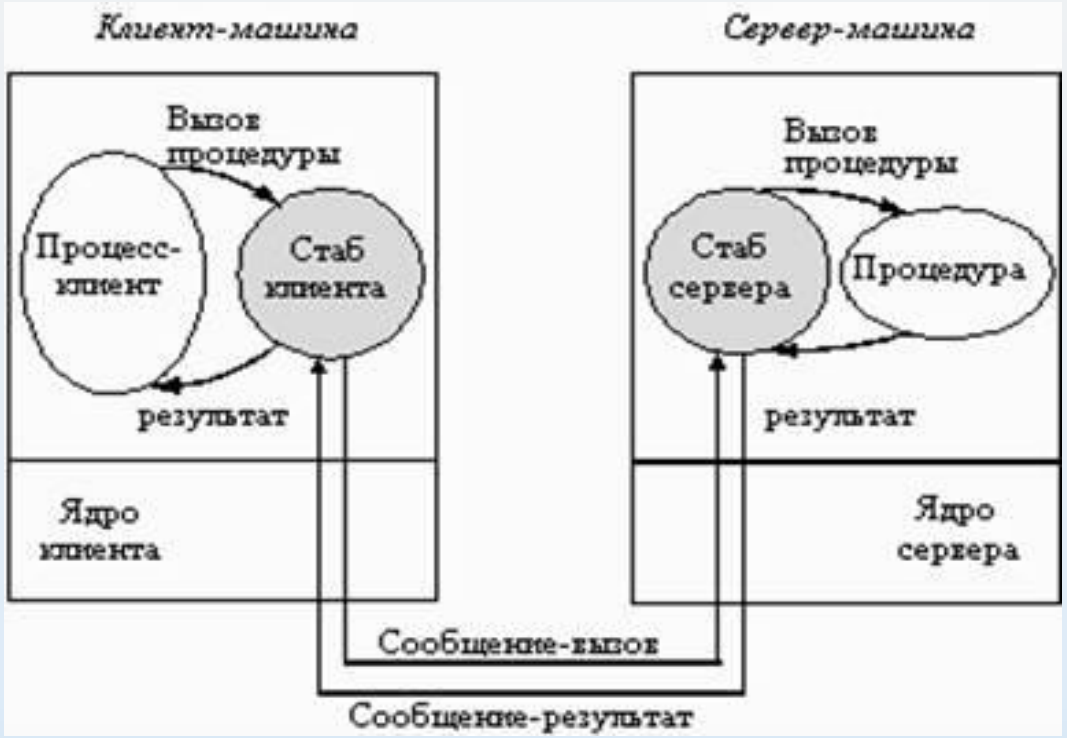

Рисунок 6 – Механизм работы удаленного вызова процедур RPC

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.

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. 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.

Этапы работы механизма удаленного вызова процедур RPC проиллюстрированы на рисунке 7.

Рисунок 7 – Этапы работы механизма удаленного вызова процедур RPC

При удаленном вызове процедуры в распределенной системе происходят следующие действия:

1. Процедура клиента вызывает стаб как обычную процедуру. Стаб упаковывает параметры (маршализация, marshaling).

2. Стаб обращается к ядру ОС.

3. Ядро посылает сообщение на удаленную машину (ядру удаленного ПК).

4. Передача полученного сообщения скелетону серверного процесса.

5. Распаковка параметров (демаршализация, unmarshaling). Вызов требуемой процедуры.

6. Процедура на сервере выполняется. Возвращает результаты скелетону.

7. Скелетон упаковывает результат.

8. Передача результата ядру.

9. Ядро сервера передает сообщение по сети ядру клиента.

10. Ядро клиента обращается к стабу, который распаковывает полученный результат.

11. Передача от стаба клиентскому процессу.

Порядок выполнения удаленного вызова процедуры проиллюстрирован на рисунке 8.

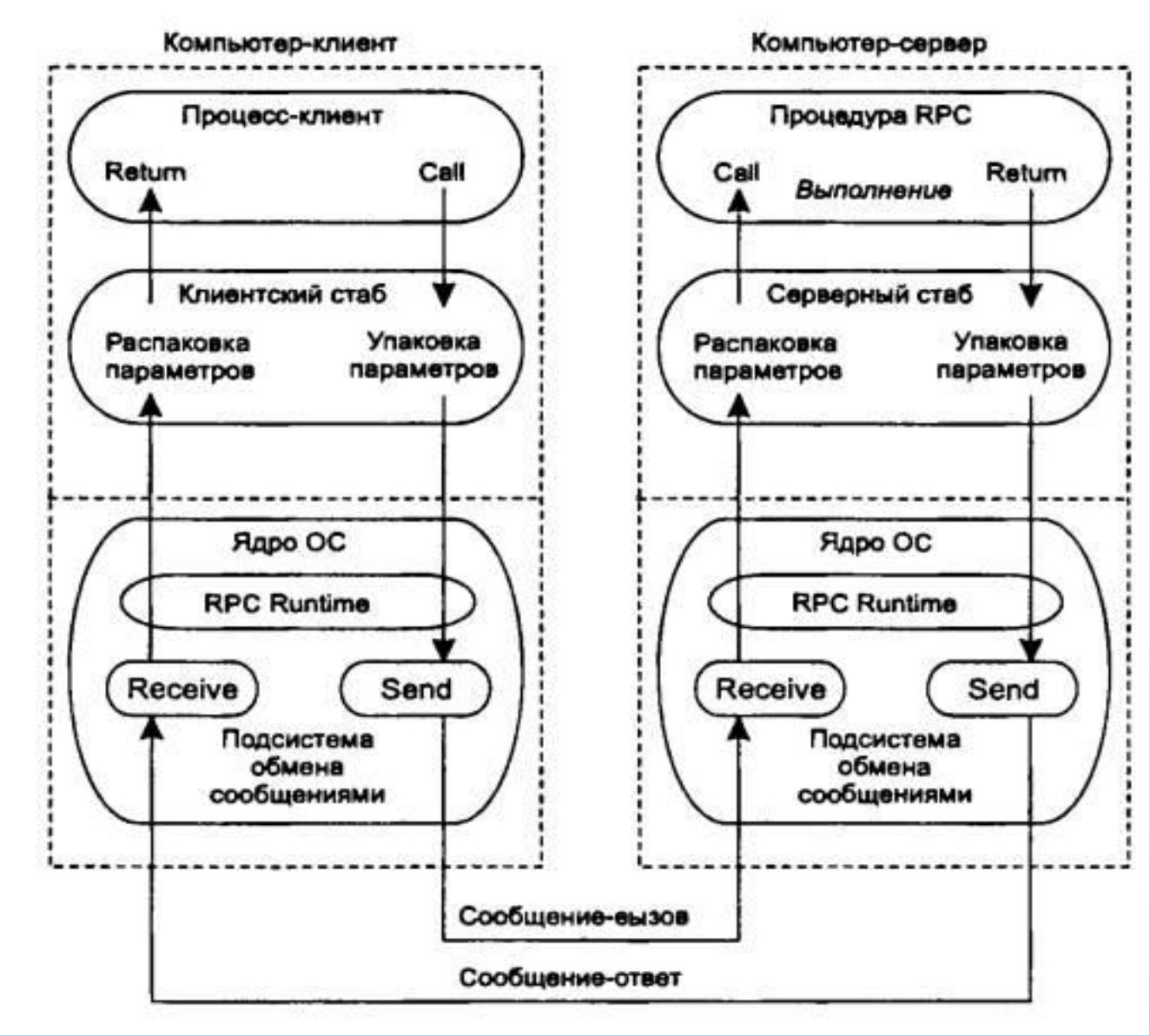

Рисунок 8 – Порядок выполнения удаленного вызова процедуры

Выводы

В ходе лекции рассмотрены следующие вопросы:

- основные понятия DCOM;

- технология и архитектура DCOM;

- механизм работы удаленного вызова процедур в DCOM.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №15. Аутентификация