

**Communication between
distributed objects
(.NET and COM)**



MÄLARDALEN UNIVERSITY

The Department of Computer
Science and Engineering

Mathias Bartoll,
mbi00001@student.mdh.se
Tobias Helfridsson,
thn00004@student.mdh.se
Daniel Wennström,
dwm00002@student.mdh.se

.NET OVERVIEW	4
THE .NET FRAMEWORK	4
COMMON LANGUAGE RUNTIME	5
BASE CLASS LIBRARY	5
ADO.NET AND XML	6
COMMON LANGUAGE SPECIFICATION	6
VISUAL BASIC .NET	6
C++ .NET	6
C#.NET	7
J#.NET.....	7
ASP.NET.....	7
COM.....	8
COMPONENT-BASED DEVELOPMENT.....	8
INTERFACE-BASED PROGRAMMING.....	9
DISTRIBUTED COM.....	10
MICROSOFT TRANSACTION SERVER (MTS).....	10
JUST IN TIME ACTIVATION	10
DISTRIBUTED TRANSACTION MANAGEMENT	10
ROLE-BASED SECURITY	10
PACKAGING TECHNOLOGY	11
REMOTING SUPPORT	11
CONTEXT MANAGEMENT.....	11
COM+.....	11
LOAD BALANCING	12
IN-MEMORY DATABASE (IMDB)	13
OBJECT POOLING	13
QUEUED COMPONENTS	13
COMMUNICATION BETWEEN DISTRIBUTED OBJECTS.....	14
WHICH PROBLEMS DO WE HAVE TO SOLVE	14
.NET SOLUTIONS FOUND	15
.NET REMOTING	15
<i>Architecture</i>	15
<i>Marshal by value</i>	17
<i>Marshal by reference</i>	17
<i>Activation</i>	17

ASP.NET WEB SERVICES 18

Architecture 18

Building a Web service..... 19

Description..... 19

Discovery..... 19

Serialization 20

Internet Information Service 20

ASP.NET WEB SERVICE VERSUS .NET REMOTING..... 20

Security..... 21

Performance..... 21

Interoperation..... 21

Reliability 21

Summary..... 21

COMMUNICATION IN COM..... 22

 DCOM..... 22

Architecture..... 22

Creating objects 23

Marshalling 23

Security..... 23

COMPARISON .NET VERSUS COM 24

Security..... 24

Interoperation..... 25

Performance 25

Lifetime management 25

CONCLUSION..... 25

REFERENCES 26

.NET Overview

.NET is not only a set of tools for developing software, it is also a vision for how to write software. The main-idea with the vision is to make the connectivity and interoperability between businesses easier. Microsoft explain it like that “the most business cooperate with other business, yet their systems operate in isolation”. With a better communication between supplier and purchaser they can attain much greater effectiveness. For example there is possible to hold a minimum stock if every vendor in a business-chain is connected to each other.

The connectivity will automatically render in security-problems. There is of course important that partners (that also can be competitors) do not get information that they should not have. .NET handles that by the XML-Web services. This is both a methodology and transport layer for passing information between components, on different machines, networks and operating systems.

The XML services are nothing new. But Microsoft had no support for it in their applications before, and now it is supported across their whole product line.

The .NET framework

The .NET framework is much more than XML web-services. It is also a Windows component for building and running .NET applications, what Microsoft likes to call, “The next generation of software applications”. Maybe the most unique aspect of .NET is its multiple language support. Microsoft has already released four commercial programming-languages for the .NET Framework: Visual C# .NET, Visual Basic .NET, the Managed Extensions for C++, and Visual J# .NET. Many other languages are also supported for example: Perl, Python, and COBOL.

The .NET Framework is a set of classes for building applications that run on the common language runtime. These class libraries provide support for tasks as data access, XML Web services, security, file IO, XML manipulation, class reflection, messaging, ASP.NET, and Microsoft Windows services.

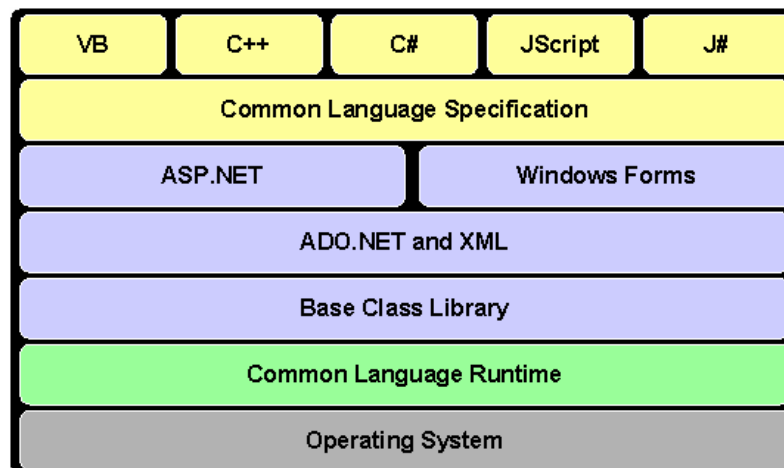


Figure 1: An overview of objects in the .NET framework

Figure 1 describes how the .NET framework is built up, and this is what the first part of this report will explain.

As a help to create languages for the .NET Framework, Microsoft created what they call the Common Language Specification (CLS, More about that later in this report). The CLS describes which features a language must make available in order to use the common language runtime and .NET Framework and also what is needed to interoperate with components in other languages. If a language has the necessary functionality it is “.NET-compliant”. Every .NET-compliant language supports the same data types, uses the same .NET Framework classes, compiles to the same MSIL, and uses the same common language runtime to manage execution. The .NET framework gives the developer freedom to choose the best language for a particular situation, without losing the power and freedom of the platform. That means, components written in one language can interoperate with component written in another language. For example, you can write a class in C# that inherits from a base class written in Visual Basic.

We will now give a short introduction to some of the most important components in the .NET framework.

Common Language Runtime

The Common Language Runtime (CLR) handles run-time services such as language integration, security enforcement, and memory, process, and thread management.

Code that is developed with a language compiler that targets the runtime is called managed code. If the runtime should be able to provide services to managed code, the language compilers must emit metadata that describes the members, types, and references in your code. Metadata is stored with the code; every loadable CLR portable executable file contains metadata.

The CLR makes it unproblematic to design components and applications whose objects interact across languages. Objects written in different programming languages can communicate with each other, and their behaviours can be integrated. For example, you can define a class and then use a different language to call a method on your original class or get a class from your original class. You can also pass an instance of a class to a method of a class written in a different language.

The runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used and those objects are called managed data. Garbage collection eliminates memory leaks and also some other common errors. In a .Net Application it is possible to use managed data, unmanaged data, or both managed and unmanaged data if the code is managed. It can sometimes be difficult to know weather it is managed code or not.

Base Class Library

Base classes provide standard functionality such as input/output, string manipulation, security management, network communications; thread management, text management, and user interface design features. . The .NET Framework Base Class Library (BCL) are a huge collection of managed code classes that integrate with the CLR. One of the main advantages of the .NET BCL is that they have been designed to be easy to use and to be virtually self-documenting. The classes are also organized into namespaces.

ADO.NET and XML

ADO.NET is a set of classes that expose data access services to the programmer. The functionality of ADO.NET is comparable with the functionality that ADO gives to native com developers. ADO.NET gives support for accessing data sources such as OLE DB, XML and Ms SQL-Server. It does also provide components for creating distributed, data-sharing applications.

ADO.NET uses some ADO objects, such as the Connection and Command objects, and also introduces new objects. Key new ADO.NET objects include the DataSet, DataReader, and DataAdapter. In ADO.NET there exists an object, the DataSet, which is separate and distinct from any data stores. Because of that, the DataSet functions as a standalone entity. Inside a DataSet, have a lot of things common with a database, there are tables, columns, relationships, constraints, views, and so on.

Common Language Specification

Applications compiled for .NET are compiled to a language called Microsoft Intermediate Language (MSIL). When you run an application for the first time, the common language runtime compiler compiles the MSIL code into native code before it is executed. The common language runtime is also responsible for providing low-level execution services, such as garbage collection, exception handling, and runtime type-safety checking.

If a object should be able to communicate with other object, independent of programming language, the objects has to expose only the most common features to all the languages they will interoperate with. That's why Common Language Specification (CLS) is needed. CLS is a set of basic language features that is needed by many applications. In that way CLS helps to ensure interoperability between objects in a lot of different programming languages.

Most of the members defined by types in the .NET Framework class library are CLS-compliant. However, some types in the class library have one or more members that are not CLS-compliant. These members enable support for language features that are not in the CLS.

The CLS was designed to be large enough to include the language constructs that are commonly needed by developers, but now almost all languages are supported.

Visual Basic .NET

Visual Basic.NET is designed to be the easiest and most productive tool for creating .NET applications for Windows, Web Services, and Web applications. It is off course a part of the .NET framework and the Common Language Runtime.

Visual Basic.NET has a lot of new features, like Inheritance, method overloading, structured exception handling, and free threading. That makes the VB.NET to a powerful and easy to use object oriented programming language.

C++ .NET

C++ .NET makes it possible for traditional C++ developers to continue creating powerful applications for Windows. There are also extensions for converting regular C++ applications to new C++ .NET applications.

C#.NET

There is no doubt that C# has taken a lot from Java. For example both languages promote one-stop coding, grouping of classes, interfaces and implementation together in one file for easy editing, and so on. C# do also have some features from C++, that Java does not have, like operator overloading and type-safe enumerations. C# does have a lot of similarities with Visual Basic, the user interface with toolbox and click and drag possibilities for example.

J#.NET

Microsoft Visual J# .NET is a development tool for Java-language developers who want to build applications and services on the Microsoft .NET Framework. Visual J#.NET gives the possibility to communicate with the other languages supported by the .NET Framework.

ASP.NET

ASP.NET makes application development much easier than it was with classic ASP, and hence it has dramatically improved developers' productivity. One of the best features is that ASP.NET let you to select any of the available .NET programming languages to compare with classic ASP where you are locked to use VBScript and Jscript.

In addition to the usual Web applications, ASP.NET allows you to create other types, which enables you to extend your applications' reach to new customers and business partners. For example, XML Web services enable sharing of data across the Internet regardless of the operating system and the programming language.

COM

The seed for COM was planted in 1988, when several different teams at Microsoft began to build object-orientated infrastructures. The term COM is a shortening for Component Object Model and is a specification for reusable software that runs in component-based systems. COM makes it possible for clients and objects to communicate across process and host computer boundaries. To achieve the high level of reuse COM is based on object-orientated programming (OOP). COM defines an application-programming interface (API) to allow for the creation of components for use in integrating custom applications or to allow diverse components to interact. These components can interact as long as they follow the binary structure specified by Microsoft. The COM-enabled languages includes: C++, Visual Basic, Java, Delphi and COBOL and allows the user to use the most suitable language. The COM technology is still used although over a decade has gone since the first specification of COM came out.

The goals of COM are the following:

- *Interoperability between different languages*
Different components written in different languages should be able to communicate with each other. This solves many of the problems associated with monolithic applications.
- *A standard for versioning*
With standard versioning means that existing client programs should work with newer versions of the component, and vice versa.
- *Environment for components with different threading requirements*
Communication between components with different types of thread affinity should be possible.
- *Location transparency*
Location transparency is the idea that the code works independent by where the process is running.

Component-Based Development

Before technologies such as COM were available, developers had to compile hundreds or possibly thousands of source files to make one executable file. This approach generated many problems like huge executables and long build times or when a modification was made; the entire application must be rebuilt. This makes it hard for various programming teams to work with large applications.

Component-Based development solves many of the problems associated with earlier approaches. It allows developers to handle binary files instead of source code. Binary components can be updated and replaced independently, making it easier to maintain. Today, Component-based technologies, like COM, are almost a requirement when working with large information systems.

As mentioned earlier the COM technology is built up to be object orientated. It's based on clients, classes and objects. Classes are defined in binary files called *servers*. These servers make it possible for clients to create and connect to objects whose code resides in a separate binary file. Figure 2 shows the relationship between a client and an object served up from a DLL. After a client connects to an object it invokes method calls as in any other OOP environment. COM fully supports object-orientated concepts of encapsulation polymorphism as means to achieve code reuse.

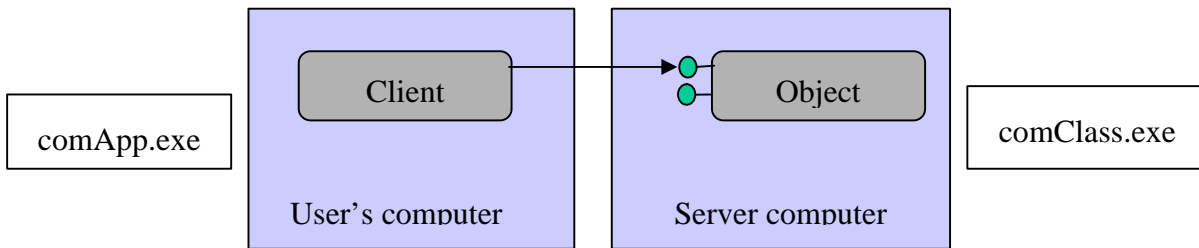


Figure 2 - Client-Object communication

Interface-Based Programming

COM is founded on the idea of interface-based programming which means a separation of interface and implementation. An interface, like a class, is a distinct data type. It defines a set of public methods without including any implementation and defines in that way a protocol for the communication between client and object. The act of decoupling the interface from the class or classes that implements it gives COM-developers the freedom to do many things that would otherwise be impossible. Interfaces are also the key to COM's ability to send remote method calls across host boundaries. A programmer never needs to worry about which platforms or network protocols are involved. A COM object can support any number of interfaces. COM objects and interfaces are specified using Microsoft Interface Definition Language (IDL), an extension of the DCE Interface Definition Language standard. To avoid name collisions, each object and interface must have a unique identifier.

Distributed COM

Distributed COM is an extension to COM that allows network-based component interaction. While COM processes can run on the same machine but in different address spaces, the DCOM extension allows processes to be spread across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the environment.

COM and DCOM should be considered as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across heterogeneous networks. In fact, COM and its DCOM extensions are merged into a single runtime. This single runtime provides both local and remote access.

Microsoft Transaction Server (MTS)

MTS was originally a separate product that worked with Windows NT 4.0 and was part of the NT Option Pack 4.0. It was meant to serve as an object broker—a server product that manages instances of objects. It was also meant to interact with and drive the Distributed Transaction Coordinator (DTC). DTC is a service that manages database transactions. MTS includes many models, some are briefly taken in concern below:

Just In Time Activation

This feature is probably the most important one. COM objects loaded through MTS can be automatically unloaded and reloaded. MTS can cache COM references of an object, so if objects are loaded in quick cycles the servers are never unloaded/reloaded even though the client code releases references. MTS handles all of this through its own abstraction layer, saving resources on the server.

Distributed Transaction Management

MTS allows database-independent management of transactions spanning multiple data sources and multiple COM objects. Essentially MTS can wrap operations into its own transactional monitor, which works through the Distributed Transaction Controller (DTC), causing any ODBC/OleDB database operations to be monitored. The key feature of this technology is that transactions have been abstracted to the COM layer. Rather than coding transactional SQL syntax, code-based logic can be used to deal with transactions.

Role-based security

MTS implements a new security model that allows configure roles for a component. The roles are configured at the component level and mapped to NT users. The code can then query for the roles under which the component is running, rather than using the much more complex NT Security model and functions to figure out user identity. Roles are also useful for deployment because they can be packaged with the component and can be automatically installed on other machines if the necessary accounts exist.

Packaging technology

MTS includes a packaging mechanism that allows taking a package, which might contain multiple COM objects, and packing it up into a distribution file. The distribution file contains all binary files as well as all information about the registry and the security settings of the component. This package can be installed on another machine for instant uptime.

Remoting Support

In Process DLL COM objects cannot be natively remoted to another machine. DCOM requires out of process servers in order to run. With MTS package mechanism the package can be activated across a network connection, which in turn allows In Process objects to be run over the network. If a DCOM connection dies it tends to orphan the running server on the server machine. With MTS the package can catch connection problems and automatically release and possibly even reload the server after a connection has been lost and re-established from the client side.

Context management

MTS includes a static data store object similar to ASP's Application object to allow components to store data for state-keeping and inter-component communication.

The integration between the COM architecture, DCOM and MTS mode is known as COM+.

COM+

COM+ was announced in Sept. 23, 1997 and came with windows 2000. It is an extension of COM and is both an object-oriented programming architecture and a set of operating system services. It adds to COM a new set of services for application components while they are running, such as notifying them of significant events or ensuring they are authorized to run. COM+ is intended to provide a model that makes it relatively easy to create business applications that work well with the MTS in a Windows NT or subsequent system. It is viewed as Microsoft's answer to the Sun Microsystems-IBM-Oracle approach known as Enterprise JavaBeans (EJB). Figure 3 shows a overview over the COM+ technology.

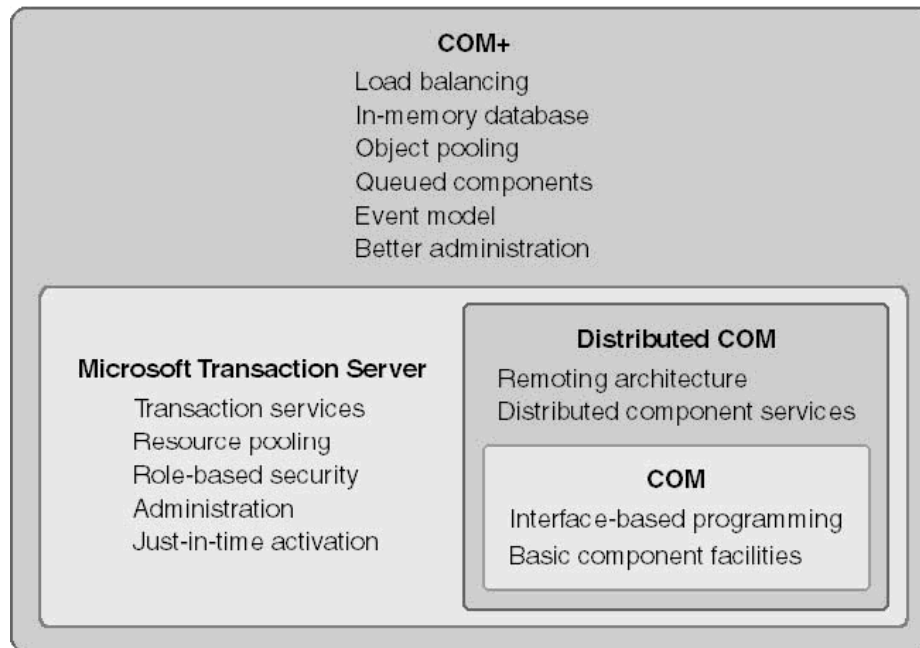


Figure 3: COM+ Overview

But COM+ does more than just make COM easier to use. It also includes a set of new features, including:

- Hiding reference counting from developers.
- Storing meta-data with every COM+ object.
- Adding the notion of interceptors, which are essentially COM+ objects that can be automatically inserted between a client and another COM+ object.
- Largely hiding COM's Interface Definition Language (IDL) from developers.
- An event registry that allows components to publish the possibility of an event and other components to subscribe to be notified when the event takes place. For example, when a sales transaction is completed, it could trigger an event that would allow other programs to be notified for subsequent processing.
- The interception of designated system requests for the purpose of ensuring security
- The queues of asynchronously received requests for a service

Load balancing

COM+ allows load balancing by grouping server machines into application clusters. It divides the work of the COM+ application across several server machines. This can increase the scalability, availability, and flexibility of the application. To a client, an application cluster looks like a single machine, and the client directs all requests to create remote COM objects to one place. This service then silently routes each create request to the most available machine in the cluster, making the decision based on real-time information.

In-memory database (IMDB)

An IMDB provides fast access to data by bypassing the physical disk. It is an OLE Database provider, which means that applications can also access it using Microsoft's current favourite database interface, ActiveX Data Objects (ADO). To an application using it, such as the middle tier of a three-tier application, IMDB provides a convenient place to store data that's not often changed. If this in-memory data is changed, the IMDB can write it to disk as required. The IMDB can also be used to store transient state for objects, subsuming the role of MTS's Shared Property Manager (SPM), and it supports locking and the ability to act as a resource manager for transactional applications, just like a "real" database.

Object pooling

Object pooling is an automatic service provided by COM+ that makes it possible to configure a component so that instances of it are kept active in a pool, ready to be used by any client that requests the component. Once the application is running, COM+ manages the pool, handling the details of object activation and reuse according to specified criteria.

Significant performance and scalability benefits can be achieved by reusing objects in this manner, particularly when they are written to take full advantage of reuse. Object pooling can:

- Speed object use time for each client, factoring out time-consuming initialization and resource acquisition from the actual work that the object performs for clients.
- Share the cost of acquiring expensive resources (such as database connections, sockets, and so on) across all clients.
- Pre-allocate objects at application startup, before any client requests come in.
- Administratively configure pooling to take best advantage of available hardware resources—the pool configuration can change as available hardware resources change.
- Speed reactivation time for objects that use just-in-time (JIT) activation, while deliberately controlling how resources are dedicated to clients.

Queued Components

Queued Components, a key feature of COM+ and based on Microsoft Message Queuing Services (MSMQ), provides an easy way to invoke and execute components asynchronously. Processing can occur without regard to the availability or accessibility of either the sender or receiver. A home shopping network is an example of the type of application that might benefit from asynchronous processing.

Communication between distributed objects

A distributed system is a system where physical separate nodes working together by communicating with each other over an asynchronous network. In fact, when choosing a distributed solution it's not voluntary, cause in most cases it's the only way to solve the problem. There can be many reasons of choosing a distributed solution such as easy as the producing and consuming of data are located on different nodes. Or it can be a hardware related problem where the data storage or the CPU-power isn't enough for the required demands. On these occasions you have to take advantage of the possibility to distribute the work on other machines.

Some of the goals with a distributed solution:

- *Reliability*: The system has the potential to be fault tolerant. The system should be resilient when failures within the system occur. This is managed by redundancy. Having two equal components and if one of them goes down the other will continue to operate for some time.
- *Scalability*: Scalability is a measure of the number of users who can perform a task concurrently. While performance is considered good when a task is performed quickly, the key to building scalable applications is to create a component that performs quickly, using the fewest possible resources.
- *Administration*: Manage the system from one place and changes can be made on the server with little or no interruption of client service.
- *Incremental growth*: Computing power can be added in small increments.

Which problems do we have to solve

Today when a user is interacting with a portal site, it appears to them that they are working with one remote server. In most cases that is normally not the truth, at least for a site of any significant size. There are often various servers and applications behind the scenes that are accessing information on several remote sites. They combine it with information from their user database and merging it all into an integrated product that is delivered to the user via their browser. As useful as these types of applications are, they are all very complex to develop and maintain. When choosing a model to rely on when developing a distributed system you should consider the following criteria.

- *Security*: A distributed application needs to address three main security areas:
 - *Authentication*: Identify such as name and password from a user and validating those credentials against some authority.
 - *Cryptography*: After the server authenticates the client, it must be able to secure the communications.
 - *Access control*: After granted access, it must be able to determine what rights the client posses. For example, what operations can the client perform, and what files can it read or write?
- *Performance*: The biggest bottleneck of a distributed system is when a remote call is made.

- *Interoperation*: We need universal programmatic access to the Internet, some way for a program on one box to call a function on any other box written by anyone. This access has to be independent not only of the operating system but also of the program's internal implementations.

.NET solutions found

The .NET Framework includes support for two distributed programming models: .NET Remoting and ASP.NET Web services. Both techniques support developing distributed applications and application integration. The ASP.NET Web service provides a simple API for Web services using SOAP messages to invoke methods. The clients do not have to know anything about the object model, platform or programming language used to build them and the Web service don't have to know anything about the clients. The only thing the two parts have to agree on is the format on the SOAP messages sent between them. This is defined by the Web service using Web service description (WSDL) and XML Schema (XSD). Compared to ASP.NET Web services, which provides a simple programming model, the .NET Remoting offers a much more complex functionality. Remoting includes support for passing objects by value or by reference, callbacks, multiple-object activation and lifecycle management policies. In order to use .NET Remoting, a client needs to be aware of all these details; in short the client needs to be built using .NET.

.NET Remoting

.NET Remoting is an architecture which enables communication between different application domains or processes using different transportation protocols, serialization formats, object lifetime schemes, and modes of object creation. In .NET Remoting, the remote object is implemented in a class that derives from `System.MarshalByRefObject`. The `MarshalByRefObject` class provides the core foundation for enabling remote access of objects across application domains. A client doesn't call the methods directly; instead a proxy object is used to invoke methods on the remote object. Every public method that we define in the remote object class is available to be called from clients.

Architecture

The remoting architecture provides the programmer with an easy procedure of creating a remote object. With a correct configured client you only need to create a new instance of the remote object using the creation function. The client receives a reference to the server object and can start calling its methods. The remoting system uses proxy objects to create the impression that the server object is in the client's process, see figure 4. The proxy object acts as a representative of the remote object and ensures that all calls made on the proxy are forwarded to the correct remote object instance. When a remote instance is created the remoting infrastructure creates a proxy object that looks exactly like the remote type of your client. The method calls goes from the client to the proxy, and the remoting system receives the call. It then routes it to the server process, invokes the server object and returns the return value to the client's proxy, which returns it to the client.

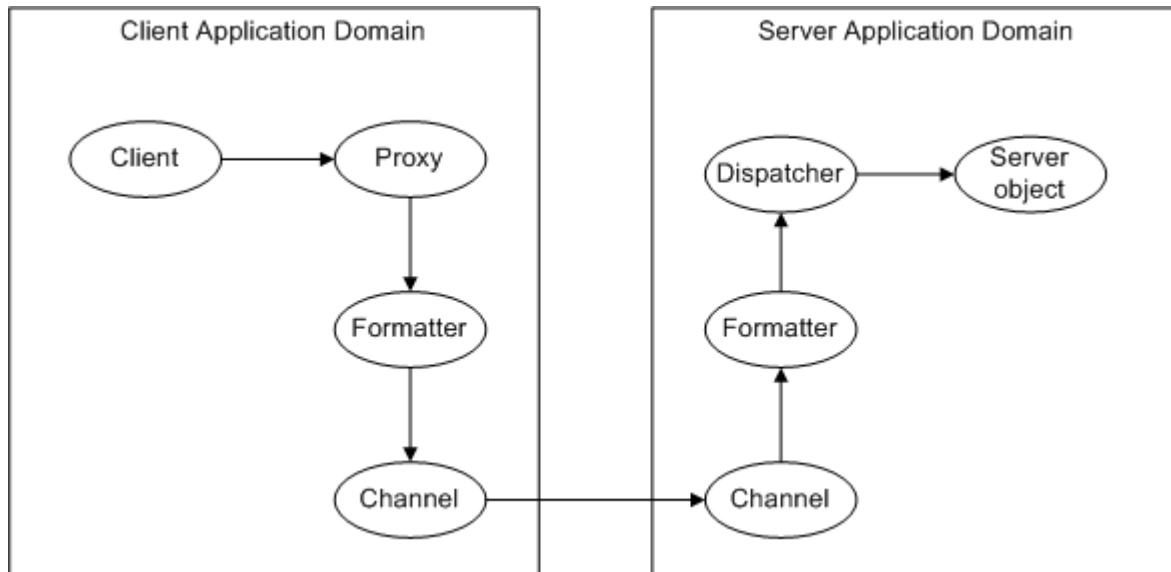


Figure 4 – The .NET Remoting architectural design.

Channels are transport protocols used to transport messages between remote objects. A channel is an object that handles communication between a client and a remote object, across application domain boundaries. There are two types of channels. HTTP channel uses the SOAP protocol. All messages are passed through the SOAP formatter where the messages are changed into XML and serialized; the required headers are added to the stream. The other channel is TCP Channel. It uses a binary formatter to serialize all messages to a binary stream and uses TCP protocol to transport the stream to the target Uniform Resource Identifier (URI).

The common language runtime defines two logical subdivisions for .NET applications: the application domain and the context. Appdomains are separate units of processing that the Common Language Runtime (CLR) recognizes in a running process. They are separate from one another, similar to process boundaries but more lightweight. The CLR further subdivides an application domain into contexts. A context guarantees that a common set of constraints and usage semantics will govern all access to the objects within it.

There are two types of objects, remotable and non-remotable. Nonremotable can't be copied or represented in another application domain. These objects are accessible only from their original application domain. Remotable objects can be accessed outside their application domain or context using a proxy – passed by reference. Or they can be copied and these copies can be passed outside their application domain - passed by value. Passing parameters from one context to another is called marshalling. Marshalling is the process of reading parameters from the stack into a flat memory buffer (that will be transferred across "the wire" later). Unmarshalling is the reverse process, it reads the memory buffer and re-creates the stack that is the same as the caller's stack. The .NET Remoting uses two standard formatters when marshalling: `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` and `System.Runtime.Serialization.Formatters.Soap.SoapFormatter`. As the names suggest they marshal in binary and SOAP format.

Marshal by value

Marshalling objects by value means to serialize their state, including all objects referenced by instance variables to some persistence form from which they can be deserialized in another context. The caller gets a copy of the requested data. Serialization is a way of encoding the present state of an instance object into a sequence of bits. The ability to serialize an object is provided when you set the attribute [Serializable] for a class or by implementing the ISerializable interface. Note that when passing objects by value they are not remote objects. All methods on those objects will be executed locally, in the same context, at the caller and not at the server.

Marshal by reference

A marshal by reference object is a remote object that runs on the server and accepts method calls from the client. When a client activates a remote object, it receives a proxy to the remote object. A proxy object acts as a representative of the remote object and ensures that all calls made on the proxy are forwarded to the correct remote object. All data are stored in the server's memory and the method calls are executed on the server. To define a marshal by reference object the specific class has to derive from System.MarshalByRefObject.

Activation

Before an object can be accessed it must be created and initialized by a process known as activation. Marshal by value types require no activation mechanism because they are copied and activated upon deserialization. Marshal by reference objects on the other hand can be categorized into two groups of activations: *server activation* and *client activation*.

A server-activated object's lifetime is managed by the remote server, not the client that instantiates the object. Server activation is normally used when the remote objects don't need to maintain any state between method calls. It is also used when multiple clients call methods on the same object instance. These objects are comparable to classic stateless Web services. When a client requests a reference to a SAO (server activate object), no message will travel to the server. Only when methods are called on this remote reference, then the server will be notified. A SAO can be marked as either *singleton* or *single call*. The difference in a singleton and singlecall lies in lifetime management.

A singleton object can only exist as one instance at a given time. When the server receives a request from the client, the server checks its internal tables to see if an instance of the class already exists. If not, the object will be created and stored in the table. After the check the singleton instance will handle all requests by either the same client or other clients. The singleton instance can maintain state between method calls. The server guarantees that there will be exactly one or no instance available at a given time.

When using single call mode, a new object will be created for each request and destroyed afterwards. No state is held between calls, and each call (no matter what client it came from) is called on a new object instance. After the method invocation returns, the .NET Remoting makes the remote object instance available for recycling on the next garbage collection.

Client-activated objects are instantiated from the client and the client manages the lifetime of the remote object. The .NET Remoting infrastructure assigns a URI to each instance of a client-activated type when it activates each object instance.

ASP.NET Web services

A Web service is a network accessible interface to application functionality, built using standard Internet technologies. It relies on industry standards, including HTTP, XML, SOAP and WSDL. Web services combine the best aspects of component-based development and the Web. It represents black-box functionality like components that can be reused without worrying about how it is implemented. Web services allow you to provide access to functionality without having to build a complete application. A developer building the consuming client application can use one or more Web services to create a new application or complement an existing one.

Because of the abstraction provided by the standard-based interfaces, it doesn't matter whether the application services are written in Java and the browser in C++, or if the application services are deployed on a Unix system while the browser is deployed on Windows. The interoperability is one of the key benefits gained from implementing Web services.

Architecture

Calls for Web services always come through port 80. For .NET Web services, these calls are always made directly to URLs with an .ASMX extension. Internet Information Service (IIS) intercepts these calls and passes all related packets to the registered ASP.NET ISAPI filter (aspnet_isapi.dll). The filter connects to a worker process named aspnet_wp.exe, which implements the HTTP pipeline that ASP.NET uses to process Web requests. Both executables are made of plain old Win32 code.

In figure 5 there is an example of a 3-tier architecture with a Web server farm supporting different clients. It's in the ASP.NET worker process where all server side code is executed (aspnet_wp.exe).

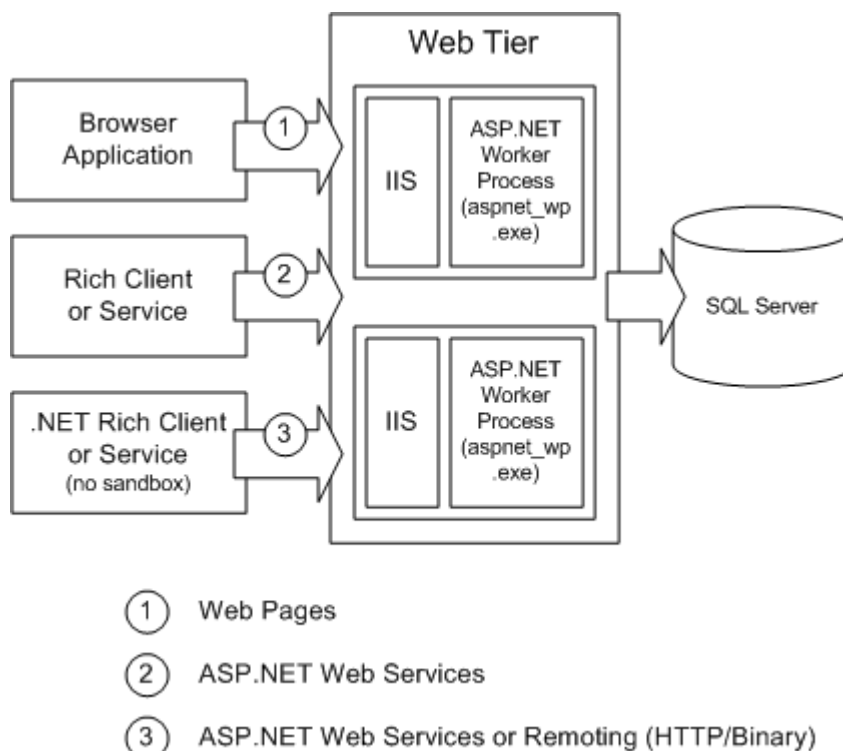


Figure 5 – 3-Tier layer architecture (Dhawan Priya et al., 2002).

The Web service model is supported by Microsoft ASP.NET, which is a unified Web development platform that has grown from Active Server Pages (ASP) technology. The ASP.NET Web services model assumes a stateless service architecture. Stateless architectures are generally more scalable than stateful architectures. Each time a service request is received, a new object is created. There is a request for the method call, the method call is returned, and then the object is destroyed. Services can use the ASP.NET State Management services to maintain a state between requests.

Building a Web service

A .NET Web service is a class that inherits from `System.Web.Services.WebService`. This class is placed in a source file and saved with an `.asmx` extension. The methods that you want to expose, as Web services will be marked with the `WebMethod` attribute. Once this is done, these methods can be invoked by sending HTTP requests using SOAP.

Each Web service should have a unique namespace that allows potential clients to separate it from other namespaces. By default, each .NET Web service is given the same namespace: `http://tempuri.org`. The namespace should be changed as soon as possible before publishing the service on the Web. Using a temporary name does not affect the overall functionality, but will affect consistency and violate Web service naming convention. Although most namespace names out there look like URLs, you don't need to use URLs. A unique name suffices.

Consuming a Web service is very straightforward too. You can very easily create a proxy class for your Web service using either command-line utility (`wsdl.exe`) or the "Add Web Reference" option in VS.NET. The Web service proxy hides the entire network and marshalling plumbing from the application code, so using the Web service looks just like using any other local object. To invoke a Web service requires a wrapper class like the proxy that lives on the client and exposes the same methods as the Web service.

Description

Web Service Description Language (WSDL) is an XML-based language that describes the Web service. It includes details such as where to find the service (its URI), what methods and properties it supports, the data types, and the protocols used to communicate with the service. WSDL describes the Web service method interface thoroughly enough for it to be used to create proxy methods that enable other classes to invoke its members as if they were local methods.

Discovery

Universal Description, Discovery and Integration (UDDI), is a directory that we can use to publish and discover public Web services. It contains information about the technical interfaces of a business services. Through a set of SOAP-based XML API calls we can discover these services, which can be invoked and used. UDDI has two parts: a registry of all a Web service's metadata, and a set of WSDL port type definitions for manipulating and searching the registry.

Web service consumers employ a discovery process to learn that a Web service exists and where to find the current WSDL document. Web service consumers set this discovery process on a target server by providing a URL to a discovery tool. The discovery tool attempts to locate DISCO documents on the target server and informs the consumer of the location of any available WSDL documents. The DISCO document is an XML document that contains pointers to such things as the WSDL file for the Web service.

Serialization

For application data to be moved around the network by the transport layer, it must be packaged in a format that all parties can understand. ASP.NET Web services rely on the System.Xml.Serialization.XmlSerializer class to marshal data to and from SOAP messages at runtime. For metadata, they generate WSDL and XSD definitions that describe what their messages contain. The reliance on pure WSDL and XSD makes ASP.NET Web services metadata portable; it expresses data structures in a way that other Web service on different platforms and with different programming models can understand.

Internet Information Service

A Web server must be an active manager of the application run-time environment and automatically detect and respond to application errors. When an application error occurs, the server needs to be fault-tolerant, meaning it must actively recycle and restart a faulty application while continuing to queue requests for the application and not interrupting the end-user's experience.

The IIS has many features so we will only explain the authentication and authorization part of the provided security options to get some knowledge when comparing with COM.

- **Authentication:** IIS has rich support for various authentication mechanisms such as: Windows Integrated Security (NTLM), Basic authentication, Passport authentication and certificate-based services. NTLM provides the same robust security system as Windows NT. The drawback with NTLM is the loss of authentication support over firewalls and proxy servers. So for Internet based scenarios involving firewalls the choice is on Basic or Passport authentication.
- **Privacy:** After the user is authenticated, your next concern might be to hide sensitive data that's being marshalled between remoting tiers. IIS also provides a strong solution for encryption: Secure Sockets Layer (SSL). SSL is an industry standard for encrypting data, and IIS fully supports SSL by using server side certificates. .NET Remoting clients then only need to specify https (instead of http) as the protocol in the server's URL.

ASP.NET Web service versus .NET Remoting

Both techniques are powerful ones that provide a suitable framework for developing distributed applications. It is important to understand how both technologies work to be able to choose the right model for the application. One guideline when choosing is following arguments: Web services, when different platforms are involved and when HTTP output is a requirement. Remoting, when communication is between .NET components.

Security

.NET Remoting plumbing does not secure cross-process invocations. Plumbing does two things: it marshals instances of programmatic data types into messages that can be sent across the network, and it provides a description of what those messages look like. However a .NET Remoting object hosted in IIS, can leverage all the same security features provided by IIS, including support for secure communication over the wire using SSL. If you are using the TCP channel or the HTTP channel hosted in a container other than IIS, you have to implement authentication, authorization and privacy mechanisms yourself.

ASP.NET Web services rely on HTTP, and integrate with the standard Internet security infrastructure. Because they are hosted in IIS, they benefit from all the security features of IIS. IIS performs authentication before the ASP.NET Web service are called. SSL can be used to secure communication over the wire.

Performance

According to tests that Microsoft has published on the MSDN website (Priya Dhawan, 2002), the .NET Remoting plumbing provides the fastest communication when you use the TCP channel together with the binary formatter. The reason for this is that this approach transmits binary data over raw TCP sockets, which are more efficient than HTTP. The data does not have to be encoded/decoded which results in less processing overhead. The binary formatter is faster than the XML serialization.

In almost all the other tests the ASP.NET Web services outperformed the .NET Remoting that uses SOAP formatter with either the HTTP or the TCP channel. That is because the ASP.NET XML serialization is more efficient than the .NET Remoting SOAP serialization.

Interoperation

Interoperability is an application's ability to communicate and work side-by-side with other applications that are not necessarily targeted to the same platform. Web services are exclusively based on open and commonly accepted Internet protocols, which enable true interoperability. Consumers of the Web Service can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web service interface.

.NET Remoting requires that the client is built using .NET. It needs to be aware of details such as, multiple-object activation, lifecycle management policies and callbacks.

Reliability

.NET gives you the most flexibility to host remote objects in any type of application such as Windows Forms, Windows service, a console application or the ASP.NET worker process. However if the remote objects are hosted in IIS, then it take advantage of the fact that the ASP.NET worker process is both auto-starting and thread safe. The ASP.NET Web service takes advantage of the capabilities provided by IIS.

Summary

Each of these techniques is designed to benefit a different target audience. ASP.NET Web services provide a simple programming model and a wide reach. .NET Remoting provides a more complex programming model and has a much narrower reach.

Communication in COM

COM was initially designed to make object-to-object calls simple, when they were both on the same machine. This made object calls easier but COM by itself did not make distributed applications possible. To enable developers to distribute their COM objects, Microsoft released Distributed COM (DCOM) in 1996. This allowed one COM object to call another COM object on a different machine.

COM's support for distributed applications is based upon an interprocess mechanism named Remote Procedure Call (RPC). RPC is an industry standard that is established on many platforms. Microsoft enhanced RPC with object-oriented extensions to accommodate COM and the resulting implementation on the Windows platform is known as Object RPC (ORPC). With this technique COM gained the ability to serve objects from across the network.

DCOM

DCOM extends the Component Object Model (COM) to support communication among objects on different computers, on a local area network (LAN), a wide area network (WAN), or even the Internet.

Architecture

DCOM is an RPC (Remote Procedure Call) protocol used for communication between distributed components. A standard procedure call is that the client makes a request to a "proxy" class on the local machine which then delegated the call invisibly to a remote "stub" class installed on a remote machine and the results followed the reverse path from the stub class to the proxy class to the client.

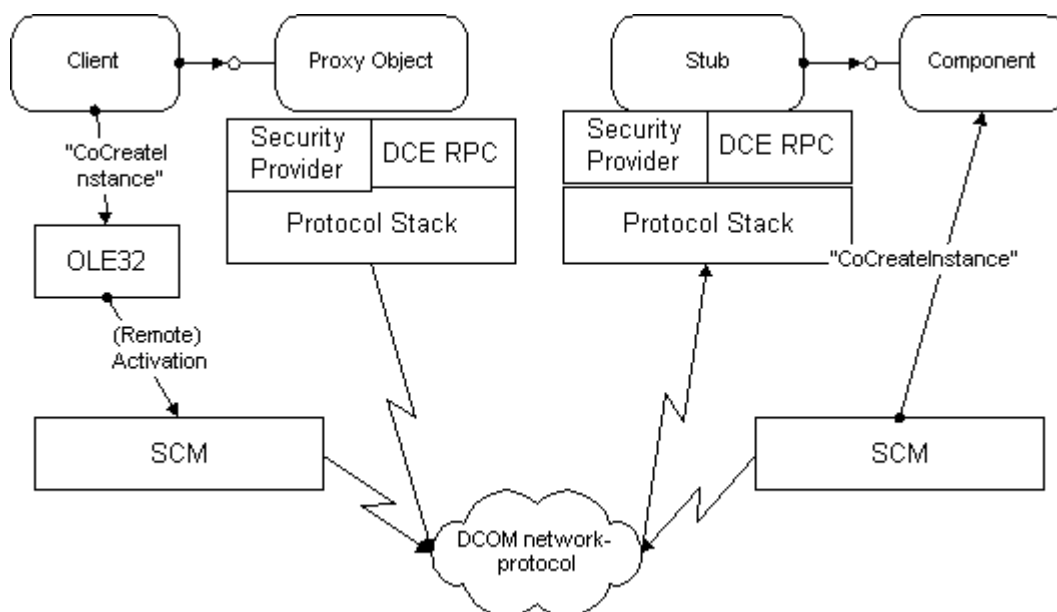


Figure 6 – The architecture of DCOM (Horstmann Markus et al., 1997).

In figure 6, we can identify five different components in the DCOM architecture:

1. The object instantiation component, where clients call "create instance" methods such as CoCreateInstance to create a new instance on the component side.

2. The high-level data transferring between clients and components component, which corresponds to the "Proxy Object/Stub" pair in the diagram.
3. The security-provider component.
4. The DCE RPC component.
5. The low-level protocol stack and DCOM network-protocol component.

Creating objects

One of the most basic requirements of a distributed system is the ability to create components. In the COM world, object classes are named with globally unique identifiers (GUIDs). When GUIDs are used to refer to particular classes of objects, they are called Class IDs. These Class IDs are nothing more than fairly large integers (128 bits) that provide a collision free, decentralized namespace for object classes.

DCOM relies on COM's object creation process to create new instances of remote components. Clients will use COM's "create instance" functions such as CoCreateInstanceEx, CoGetInstanceFromFile and CoGetObject, to create a new instance of a component. In order to be able to create a remote object, the COM libraries need to know the network name of the server. There are two ways to specify RemoteServerName for an object: Through the registry or by passing by a parameter to one of the "create instance" methods. The advantage of the first method is that clients are not even aware that they are working with a remote object, location transparency. The drawback of this approach is that the server name not van be specified on the fly. Every time the component is moved to a different server, the RemoteServerName value in the registry must be changed, either manually by the user or programmatically. The second method allows the server name to be specified on the fly and is the solution for applications that require explicit run-time control over the server that a client connects to.

Marshalling

When functions within one process call each other, they pass the parameters through the process's stack. In a distributed environment, clients and servers are on different machines, and thus have different stacks. COM provides mechanisms for marshalling and unmarshalling method parameters that build on the RPC. Infrastructure defined as part of the DCE. In order to perform marshalling/unmarshalling, COM must know the methods signature and the data types. These are specified in an Interface Definition Language (IDL) file. An IDL file is like a C-file, with some special keywords. It supports data types defined in the programming language C such as int, long, etc.

An IDL file will be compiled by using a special IDL compiler such as the Microsoft IDL compiler (MIDL) to produce C-source files. These C-files are compiled again to produce a proxy/stub pair that handles marshalling/unmarshalling. The proxy resides on the client's side, and the stub resides on the server side. The proxy/stub pair for an interface is specified in the registry. When COM needs to find the proxy/stub combination for a particular interface, it simply looks up the Interface Id (IID) under the HKEY_CLASSES_ROOT\Interfaces key in the system registry and reads the ProxyStubClsid key.

Security

DCOM provides support for authentication, cryptography, and access control by integrating closely with the Microsoft Windows NT security system. DCOM applications deal with four fundamental aspects of security:

1. **Access security:** Protect objects against unauthorized access. DCOM implements access control on a per-process level. Existing components that belong to an application will have the same security context as the application. This allows the developers to ignore the security details on the components themselves (if the objects do not need to control the access security themselves), and configure the security context once at the application level.
2. **Launch security:** Is there to maintain control of object creation. Since all COM objects can be accessible thru DCOM, it is important to prevent unauthorized users from creating instances of these objects. COM handles this by performing special security validations on object activation. If a new object is to be created, COM validate if the caller has the right properties, these are stored in the registry.
3. **Security identity:** Objects often access sensitive files and databases that should not be accessible to all clients. The obvious approach to solve this problem is to check the identity of each client that wishes to connect to the object, and then have the object assume the identity of the client. Actions performed by the object such as network access, file access, connection to database are limited by the client's privilege. This works fine for applications with a small group of users. However, applications with a large number of users, the mentioned approach can impose problems. All resources that are potentially used by an object must be configured to have exactly the right set of privileges. This managing process can be simplified by defining user groups.
4. Another problem is that many Internet applications do not assign dedicated user accounts for every user. Assigning objects a security identity of their own makes this kind of application secure.
5. **Connection policy:** As the distance becomes longer between clients and objects, messages between the clients and the objects become less secure. Connection policy defines the security protection for the messages. DCOM offers two choices for data security: integrity and privacy. Message integrity means that the messages cannot be altered. This implies that each data packet contains authentication information. Message privacy guarantees that third parties will not intercept the messages. This is done by encryption, which implies message integrity as well as authentication information for each data package.

Comparison .NET versus COM

COM, DCOM and COM+ are based on the COM architecture (VB6, ATL, C++, and MFC), while .NET Remoting and Web services are based on the new .NET Frameworks technology.

Security

The security model for .NET Remoting systems has changed quite a bit from the complex and highly configurable model of DCOM. For the initial release of the .NET Framework, the best way to implement a secure remoting system is to host the remote server inside IIS like the Web service. The best part of hosting inside IIS is that you can use its strong security features without changing the client's code or the server's code. This means you can secure a remoting system just by changing the hosting environment to IIS and passing identifications (user name, password, and optionally, the domain) by setting a client configuration option.

Interoperation

DCOM has their biggest drawbacks in the Internet connected world because DCOM depends on proprietary binary protocol, which is not supported by all objects. DCOM have their own proprietary wire format that is usually designed with performance in mind. A few years ago, interoperability wasn't nearly as important as staking out territory and possibly achieving vendor or technology lock in. Some third party "bridge" implementations have attempted to help the locked in organizations talk to the other side. But none of these solutions are as seamless and easy to use as having the interoperability support baked into the various distributed application technologies.

Another disadvantage of DCOM is the usage of a port other than the default HTTP port 80, resulting in that a different port has to be kept open at all times in order for the components to communicate, which is a major security hazard. This drawback resulting in that DCOM can not work easily across firewalls.

Web services are distributed software components similar to COM/DCOM. However it is the accessible through standard web protocols that differs the models apart. Web services can be consumed by any application that understands how to parse an XML-formatted stream transmitted through HTTP channels. While COM-aware clients only understand COM components.

Performance

If you configure a .NET Remoting application for optimal performance, the speed will be comparable to that of DCOM, which is very fast. Of course, when configured this way, .NET Remoting applications have the same interoperability constraints as the DCOM applications they're comparable with. Fortunately, configuring .NET Remoting applications for maximum performance versus interoperability is as easy as specifying a couple of entries in a configuration file.

Lifetime management

DCOM relies on frequent pinging of clients to manage remote object lifetime, where .NET Remoting relies on simple and more efficient leasing mechanisms to maintain object lifetime.

Conclusion

Today when developing a distributed system there are several techniques to use. In this reports second part we have discussed the communication models in .NET and COM. The concentration has been on the three categories stated in the "Which problems do we have to solve" chapter. Until the arrival of .NET Frameworks, Microsoft's primary protocol for intelligent inter-process communication across systems was Distributed COM. The new .NET architecture makes the distributed systems easier to develop, offers more flexibility, interoperability and a better security compared to the previous model. The choice in Microsoft's distributed world is now which .NET Framework model to choose when developing a system; the .NET Remoting or the ASP.NET Web services.

References

- Chappell David, (1998), *COM+ Redux*,
http://www.chappellassoc.com/articles/article_COM_Redux.html
- Dhawan Priya, (2002), *Performance Comparison: .NET Remoting vs. Asp.NET Web Services*,
<http://msdn.microsoft.com/webservices/building/architecture/default.aspx?pull=/library/en-us/dnbda/html/bdadotnetarch14.asp>
- Dhawan Priya, Ewald Tim, (2002), *ASP.NET Web Services or .NET Remoting: How to Choose*,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconchoosingcommunicationoptionsinnet.asp>
- Guy Eddon and Henry Eddon , (1999), *Inside Microsoft COM+ Base Services*.
- B.Harvey, S.Robinsson, J.Templeman, K.Watson, (2000), *C# Programming With the Public Beta*, ISBN: 1-861004-87-7
- Hawkins Jonathan, Obermeyer Piet, (2001), *Microsoft .NET Remoting: A Technical Overview*,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp>
- Horstmann Markus, Kirtland Mary, (1997), *DCOM Architecture*,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cosssdk/html/pgprogrammingoverview_9kjb.asp
- Lam Hoang, Thai L. Thuan, (2002), *.NET Framework Essentials*, 2nd Edition, O'Reilly
- McLean Scott, Naftel James, Williams Kim, (2003), *Microsoft .NET Remoting*, Microsoft Press, Washington.
- Microsoft Corporation, 2002, *Load Balancing COM+ Components*,
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/acs/evaluate/lb-cmpnt.asp>
- Microsoft Corporation, *Redeployment of COM+ Load Balancing (CLB)*, (1999),
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomser/html/complusload.asp>
- Microsoft Corporation, *Technology Overview, What is ASP.NET?*,
<http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx>
- Mojica Jose,(2001), *COM+ Programming, with Visual Basic*, ISBN: 1-56592-840-7
- Parihar Mridula, (2002),*ASP.NET Bible*, ISBN: 0-7645-4816-6
- Pattison Ted, (1998), *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0* ISBN: 1-57231-961-5
- Templeman Julian, Vitter David, (2002), *Visual Studio .NET – The .NET Framework Black Book*, The Coriolis Group, Arizona