Components and the Enterprise

As distributed applications are built from simple components and Internet protocols emerged, a new set of enterprise platform services for component applications will be required. To address enterprise requirements for distributed component architecture without sacrificing rapid development and cost effectiveness, Microsoft is integrating DCOM into the Active Server. The Active Server is a series of technology services that speed deployment of component-based applications for the Internet and corporate intranets. These services include:

- **Transactions**—traditional rollback and recovery for component-based applications in the event of system failure.
- **Queuing**—integration of component communication with reliable store-and-forward queues, which enables component applications to operate on networks that are occasionally unavailable.
- **Server scripting**—easy integration of component applications on the server with HTML-based Internet applications.
- **Legacy access**—integration of component applications with legacy production systems, including mainframe systems running CICS and IMS.

The Active Server technologies use publicly obtainable Internet protocols and are currently available[5].

MICROSOFT .NET FRAMEWORK[6]

Microsoft evolves from the COM-DNA platform to the new .NET platform designed to simplify application development in the highly distributed environment of the Internet. This is a transformation from desktop applications to the distributed GUI-based .NET applications. The .NET Framework affects all of Microsoft's products, from operating systems, servers, and middleware to applications. All these products are capable of handling and processing .NET traffic and leveraging the .NET infrastructure. These products will transform to Windows.NET, Office.NET, MSN.NET, and so forth.

The .NET Framework is aimed to develop a new software platform that is independent of an operating system, components, and applications that are written to run on the virtual machine. The .NET virtual machine is Common Language Runtime (CLR), which can be a platform for applications written in any language, unlike JVM (Java Virtual Machine) which accepts applications written only in Java. Micosoft designed .NET to be a very friendly environment for applications, particularly those written in Visual Basic and C#. The .NET Framework supports any of a variety of languages. It also relies on XML coupled with SOAP (Simple Object Access Protocol) to link components running on distributed .NET platforms.

Since COM is too embedded in Windows, Microsoft is replacing DCOM with XML and SOAP while transforming from Windows to the .NET Framework[7].

The architecture of the .NET Framework is shown in Figure 5-8.

The .NET Framework has four main components:

1. The Common Language Runtime (CLR)
2. The .NET Framework Class Library
3. Communication Protocols
4. Visual Studio .NET

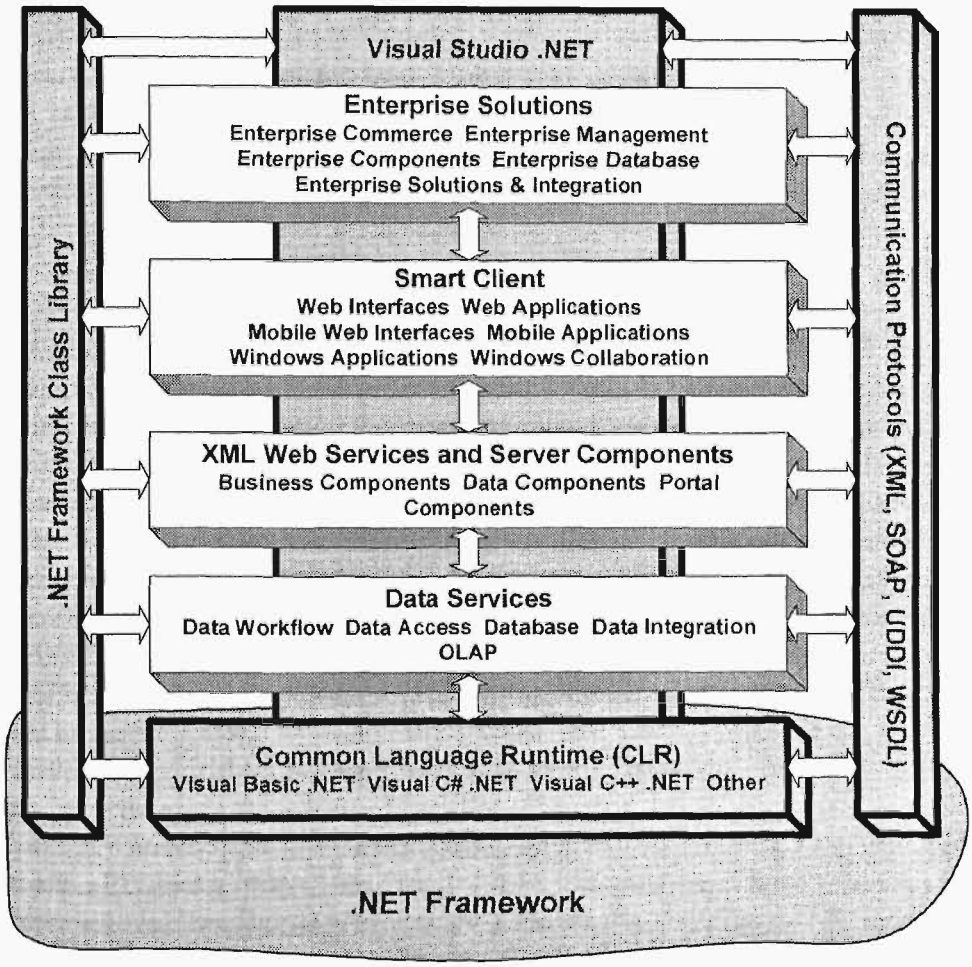
Common Language Runtime (CLR)

The Common Language Runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict safety and accuracy of the code. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code; code that does not target the runtime is known as unmanaged code.

Managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even being used in the same active application.

The runtime enforces security in a way that enables users to trust that although an executable attached to an e-mail can play an animation on screen or sing a song, it cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature-rich. The runtime also enforces code robustness by implementing a strict type- and code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that

Figure 5-8: The Architecture of Microsoft .NET Framework within the Microsoft Software Tools Environment (The Targowski Model)



managed code can consume other managed classes, types, and objects, while strictly enforcing type fidelity and type safety.

The runtime, coupled with the CTS, also accelerates developer productivity. For example, programmers can use their favorite development language, being absolutely assured that they can still take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, thus greatly easing the migration process for existing applications.

Although the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLL's. The runtime is designed to enhance performance. A feature called Just-In-Time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Internet Information Services (IIS) and Microsoft® SQL Server. (www.microsoft.com).

.NET Framework Class Library

The .NET Framework class library is a comprehensive, object-oriented collection of reusable classes that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET and Web Services. The class library builds on the object-oriented nature of the runtime, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the learning curve associated with using a new piece of code. In addition, third-party components can integrate seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios.

For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications
- Scripted or hosted applications
- Windows GUI applications (Windows Forms)
- ASP.NET applications (Active Server Pages)
- Web Services
- Windows 2000 and Windows NT services

For example, the Windows Forms classes are a comprehensive set of reusable types that vastly simplify Windows GUI development. If you are writing an ASP.NET application or Web Service, on the other hand, you use different classes, such as the Web Forms classes (www.microsoft.com).

Communication Protocols

The communication protocols form the foundation of the .NET Framework, which develops distributed applications that inter-operate through the Internet; therefore they must do this by complying with a common communication protocol. The current practice with the set of Remote Procedure Call (RPC) technologies has too many limitations that make an application not fully inter-operational. Rather than develop a brand-new protocol to overcome the shortcomings of COM/DCOM and CORBA, Microsoft decided to build the .NET Framework on top of a set of standard, open, XML-based protocol such as Simple Object Access Protocol (SOAP), Universal Discovery, Description and Integration (UDDI), and Web Service Definition Language (WSDL).

This protocol transmits SOAP, UDDI, and WSDL messages as a plain text, so it is platform neutral. There is no need to write a bridge to translate a method call from one Object RPC to another. A SOAP message looks the same on Linux as it does on Windows as it does on AS/400. Since these protocols are open standards, they can be implemented anywhere in the manner that is most appropriate for the platform. On the Microsoft platform,

a SOAP server is targeted at IIS and written in either ASP or ISAPI (www.microsoft.com).

Visual Studio .NET

To succeed in today's business environment, applications must be more scalable, reliable, and flexible than ever before. At the same time, the fast pace of business change means that developers must design and launch enterprise applications in days or weeks rather than months or years. Microsoft Visual Studio is a complete enterprise-class development system that helps developers meet those demands by providing the tools to create powerful, mission-critical applications—quickly and efficiently.

Visual Studio offers a wide range of features and tools designed specifically to support team-based application development efforts, including teams that are geographically dispersed. The distributed Web project model uses HTTP for all authoring operations, so that developers working in different locations can collaborate on the creation of sophisticated Web applications. With a Visual Component Manager, developers can find, track, catalog, and reuse components easily. For better version control, Microsoft Visual SourceSafe® provides complete code source control and file-locking features for team development projects from any tool within the Visual Studio development suite.

Developers can use a familiar, shared development environment and the programming languages they already know. Pre-built components, programming wizards, and the ability to reuse components written in any language can cut the development cycle time significantly. IntelliSense®-based code completion enables developers to produce accurate code more quickly. Finally, powerful end-to-end, cross-language debugging support, coupled with cross-language debugging, helps development teams get applications up and running more rapidly.

Increasingly, all business is e-business. Enterprises look to the Internet as an essential medium not only for communications but also for commerce and operational efficiency. The Visual Studio delivers highly scalable, data-driven websites and applications. Its wide range of database programming and design tools utilize the Microsoft Universal Data Access technology. Developers can use powerful point-and-click database diagrams and graphical tools for creating tables, relationships, stored procedures, and database functions for Microsoft SQL Server™ and Oracle databases. ActiveX® Data Objects

enable easy access to information from all industry-leading data sources, including Microsoft SQL Server, Microsoft Access, Microsoft FoxPro®, Oracle, and IBM mainframe and AS/400 databases. The Microsoft Data Engine ensures full compatibility with large SQL Server databases.

Developers can also choose the programming language they know best—and the language that is best suited to the solution, including Microsoft Visual Basic®, Visual C++®, Visual J++®, and Visual FoxPro®.

Thin-client, HTML-based front ends make it easy to deploy the results to any desktops running virtually any operating system (www.microsoft.com).

Client Application Development in .NET

Client applications are the closest to a traditional style of application in Windows-based programming. These are the types of applications that bring up Windows or Forms on the desktop and which you use to perform a task. Client applications include applications such as word-processors and spreadsheets, as well as custom business applications such as data-entry tools, reporting tools, and so on. Client applications usually employ windows, menus, buttons, and other GUI elements, and they likely access local resources such as the file system and peripherals such as printers.

Another kind of client application is the traditional ActiveX control (now replaced by the managed Windows Forms control) deployed over the Internet as a Web page. These types of applications are much like other client applications, in that they are executed natively, have access to local resources, and include graphical elements.

In the past, developers created such applications using C/C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft® Visual Basic®. The .NET Framework incorporates aspects of existing products into a single, consistent development environment that simplifies the development of client applications.

The Windows Forms classes contained in the .NET Framework are designed to be used for GUI development. You can easily create command windows, buttons, menus, toolbars, and other screen elements with the flexibility necessary to accommodate shifting business needs.

For example, the visual attributes of forms often need to be modified during the life of an application. Forms are implemented using a window from the underlying operating system. However, some visual changes might not be

supported by the underlying operating system for existing windows. Changes to visual features of that type can require the creation of an entirely new window. In unmanaged applications, such a visual feature would probably be discarded, due to the complexity of updating it. In managed code, however, when you change a Windows Form object, the framework creates a new operating system window object and automatically moves the relevant state from the old window to the new window. This is just one of many examples of the ways in which the framework homogenizes the developer interface, thus making coding simpler and more consistent.

The runtime's built-in security and deployment features can revive the client application in some innovative ways. For example, many applications that once needed to be installed on a user's system can now be deployed through the Web. This is possible because code-access security limits what a piece of software can do, even though the software is running in native machine language. In fact, a single Web-deployed application can comprise various components distributed from any number of different websites. In such a case, object methods provided by one vendor can have different security rights than object methods provided by a second vendor, even though they are used together to create a single application running in a single system process.

Unlike ActiveX controls, Windows Forms controls enjoy semi-trusted access to a user's machine. This means that binary or natively executing code can access some of the resources on the user's system (such as GUI elements and limited file access), without being able to undermine a user's system (www.microsoft.com).

Server Application Development in .NET

Web Services, an important evolution in Web-based technology, are distributed, server-side application components similar to common websites. However, unlike Web-based applications, Web Services components have no UI and are not targeted for browsers such as Internet Explorer and Netscape Navigator. Instead, Web Services consist of reusable software components designed to be consumed by other applications, such as traditional client applications, Web-based applications, or even other Web Services. As a result, Web Services technology is rapidly moving application development and deployment into the highly distributed environment of the Internet.

ASP.NET (Active Server Pages) is the hosting environment that enables developers to use the .NET Framework to target Web Services applications. Both Web Forms and Web Services use Internet Information Server (IIS) as

the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework. ASP.NET is more than the next version of Active Server Pages (ASP); it is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications. While ASP.NET is largely syntax compatible with ASP, it also provides a new programming model and infrastructure that enables a powerful new class of applications.

ASP.NET has been designed to work seamlessly with WYSIWYG HTML editors and other programming tools, including Microsoft Visual Studio.NET. This makes Web development easier, but it also provides the benefits that these tools have to offer, including a GUI that developers can use to drop server controls onto a Web page, as well as fully integrated debugging support.

Developers can choose from two programming models when creating an ASP.NET application, or combine these in any way they see fit:

- Web Forms allows you to build powerful forms-based Web pages. When building these pages, you can use ASP.NET server controls to create common UI elements and program them for common tasks. These controls allow you to rapidly build up a Web Form out of reusable built-in or custom components, simplifying the code of a page.

- A Web service is a way to access server functionality remotely. Using services, businesses can expose programmatic interfaces to their data or business logic, which in turn can be obtained and manipulated by client and server applications. Web services enable the exchange of data in client-server or server-server scenarios, using standards like HTTP and XML messaging to move data across firewalls. Web services are not tied to a particular component technology or object-calling convention. As a result, programs written in any language, using any component model, and running on any operating system can access Web services.

ASP.NET takes advantage of performance enhancements found in the .NET Framework and runtime, and it has also been designed to offer performance improvements over ASP and other Web development platforms. All ASP.NET code is compiled rather than interpreted, which allows early binding, strong typing, and just-in-time (JIT) compiling to native code, to name only a few of its benefits. ASP.NET is also easily factorable, meaning that developers can remove modules (a session module, for instance) that are not relevant to the

application they are developing. ASP.NET configuration settings are stored in XML-based files, which are human readable and writable. Each of your applications can have a distinct configuration file and you can extend the configuration scheme to suit your requirements. ASP.NET provides easy-to-use Application and Session state facilities that are familiar to ASP developers and are readily compatible with all other .NET Framework API's.

If you have used earlier versions of ASP technology, you will immediately notice the improvements in Web Forms. For one thing, you can develop your Web Forms pages in any language that supports the .NET Framework. In addition, your code no longer needs to share the same file with your HTTP text (although it can continue to do so if you prefer that structure). Web Forms pages execute in native machine language because they take full advantage of the runtime like any other managed application. Unmanaged ASP pages were always scripted. In short, ASP.NET pages are faster, more functional, and easier to develop because they interact with the runtime like any managed application.

The .NET Framework also provides a collection of classes and tools to aid in development and consumption of Web Services applications. Web Services are built on standards such as SOAP (a remote procedure-call protocol), XML (an extensible data format), and WSDL (Web Service Description Language). The .NET Framework conforms to these standards to promote interoperability with non-Microsoft solutions.

If you develop and publish your own Web Service, the .NET Framework provides a set of classes that conform to all of the underlying communication standards, such as SOAP, WSDL, and XML. Using those classes enables you to focus on the logic of your service, without worrying about the communications infrastructure required by distributed software development.

Finally, like Web Forms pages in the managed environment, your Web Services will run with the speed of native machine language using the scalable communication of IIS (www.microsoft.com).

XML STANDARD

When not pursuing wholly new application development, organizations can be found attempting to create applications that aggregate several traditional, task-oriented applications into a single, composite application. This sometimes includes integrating applications that exist within the boundaries of a separate entity, such as another company or a service provider. However,

a still greater dilemma arises when attempting to integrate legacy applications built using an assortment of technologies, object models, operating systems and programming languages. How do you make them all work together? The answer is the programmable Web.

XML (eXtensible Markup Language) as an open data description format has given rise to the reality of a programmable Web. Just as TCP/IP provided universal connectivity for the Internet and HTML provided a standardized language to display information on a wide variety of platforms for human consumption, XML provides a standardized language to exchange data for automated consumption. This makes it possible to represent data in a widely accepted format that enables computers to send and receive data in a predictable style that enables programmability that extends beyond closed, controlled systems. XML is liberating because its simplicity and extensibility lets you define just about anything with room to expand later. One of the fundamental building blocks of the programmable Web is Web Services.

XML is for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (e.g., content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all corporate documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents. Among documents one can mention those such as word processing, vector graphics, e-commerce transactions, mathematical equations, object metadata, and so forth.

XML was created so that richly structured documents could be used over the Web. Its predecessor HTML, for the creation of homepages, does not provide a way to define a structured document. The term markup comes from the print profession where electronic documents are "marked up" with tags that tell a computer what to do. It serves two purposes; to determine the formatting and to define a document's structure and meaning. XML is a standardized set of markup tags that conform to a defined syntax (grammar).

Every XML document has logical and physical structures. The former defines a document's framework and the latter contains the actual data that fills a document. In order to publish a document's "data" on a Web it is necessary first to define its logical structure.

The main advantage of XML documents is their ability to be processed by different enterprise-wide applications that are XML ready. XML documents are particularly suitable for e-Document Management Systems, portals, call centers, and so forth.