Microsoft .NET

By

Robert Turner
Justin Mah
Peter Luong

Table of Contents:

Overview

What is .NET?

Microsoft's definition of .NET:

> "Microsoft® .NET is the Microsoft XML Web services platform, which will significantly change how people interact with applications and devices via the Web."

This is not very enlightening. At best Microsoft tells us that .NET is it's new XML platform. Delving deeper into .NET however, reveals a more enlighten idea of .NET really is: a concept. It is the concept of a true distributed computing environment. Any "smart device" anywhere can connect to the internet accesses your own information at any time. .NET is an Internet-based platform of Next Generation Windows Services accessed through smart devices. Specifically,

- .NET is a new Internet and Web based infrastructure
- .NET delivers software as Web Services
- .NET is a framework for universal services
- .NET is a server centric computing model
- .NET will run in any browser on any platform
- .NET supports dynamically reconfiguration.
- .NET is built on HTTP, XML, SOAP (standard format for web services), & UDDI (standard search format to discover web services)

.NET and Smart Devices

Microsoft aims for .NET to be used with smart devices. A smart device is any piece of hardware that contains a CPU and a connection to the internet, E.G. Personal computers, cell phones, tablets, PDA's, even the Microsoft X-Box (and theoretically the Sony Playstation as well). Perhaps the best way to understand the concept of .NET is to look at the way smart devices are supposed to behave:

- **Smart about you.**
  A smart device uses your .NET identity, profile, and data to simplify your experience. It is smart about your presence, tailoring notifications in response to your presence or absence.
- **Smart about the network.**
  A smart device is responsive to bandwidth constraints, provides support for both online and offline use of applications, and understands which services are available.
- **Smart about information.**
  A smart device can be used anywhere anytime! Your information is accessible to

you no matter where you are, as long as you have a smart device and remember your login and password.

- **Smart about other devices.**
  A smart device discovers and announces other smart devices, knows how to provide smart services. Think USB on a software scale.
- **Smart about software and services.**
  A smart device presents applications and data optimally for form factor; enables input methods and connectivity appropriate for great end-user interaction.

What do you need to run .NET?

All you need is a smart deice, and a .NET server. It's that simple. Specifically, on the client end, Microsoft is aiming at needing only a web browser (Internet Explorer) and active connection to the internet. On the server side, .NET was built with the current standards in mind: ASP, ADO, and SQL. As well, they hope to show off IIS.

What are Individuals Benefits to .NET?

Individual benefits are incredible! Think of it – being able to access your computer desktop from home, work, or anywhere you want with a cell phone or PDA – and having them all run appropriately the same way – it's like living in the future! Microsoft terms this as the **Personal and integrated experience** that .NET has to offer.

On top of that, if everything works like it's supposed to, **users can experience the same .NET through smart devices**. Each of these devices generates a different interface for the .NET experience. Each appropriate to the type of device you are using.

As well, **Privacy** has been a very important issue to Microsoft during the development of .NET, and they have gone to great lengths to make sure everything is secure and protected.

What are Business Benefits to .NET?

Since .NET is Microsoft's next generation XML platform, it features **XML Integration for easy transactions.** Weather it's between partners in business to business transactions, between customers through web servers & smart devices, or within organizations getting specific views of files – a single XML sheet is all that's needed. The XML translators do the rest!

If Microsoft is right, and the benefits of using smart devices are as large as they claim then **XML Web services** offers a huge incentive to businesses. Since uses will no longer be forced through a PC, you could get much better targeted advertising on

something like a smart PDA (say advertising extra PDA pens). And, if .NET catches on, we are sure to see more of these smart devices taking new forms we haven't even though of yet.

Savings in development costs are also touted to be quite high. Development should be much more rapid, implying cheaper development costs, and faster time to market!

What are Developers Benefits to .NET?

Through .NET's common language runtime any developer should be able to use his favorite language (or combination of languages) to quickly get the project done. This **Rapid development** can done through recompiled existing code (involving only minor changes to incorporate .NET procedures), or through **easily finding available XML Web Services**. If you don't have time to build pieces of your project, you can buy pieces of your applications rather than build everything from scratch. This also implies that someone will be selling XML Web Services, meaning more jobs for independent developers.

Why should you learn .NET?

With the abundance of XML products quickly entering the market, why should you choose to develop with .NET rather then something else? Well, there are several reasons.

The first is that **.NET is backed by Microsoft**. They will not give up no matter how bad there technology looks and will do everything in there power to crush any and all competition in order to monopolize the XML product market.

The second is that Microsoft has already built a **very strong base for .NET** XML platform: the .NET passport integrated into Hotmail, the MSN browser, and MSN Messenger. 75% of all Microsoft Windows based platforms out there are running at least one of the three. If you want an idea of just how big the .NET passport has already gotten, try their web page:

http://www.passport.com/Directory/Default.asp?PPDir=C&lc=1033

Thirdly, the **.NET software development kit is free**. There is a tremendous amount of software available already, not to mention lots of info and help available on the web.

Fourthly, and finally, .NET development will be big business. This is where the **big bucks** are going to be!

Why should you avoid .NET

At the moment, **.NET is unproven**. While it looks very good on paper, there has yet to be any powerful .NET programs released (beyond the Microsoft trio of .NET Passport, MSN Messenger, and MSN Explorer). Since .NET is meant to run through a browser there is **no guarantee it will be cross browser compatible**. In theory, it should be cross browser compatible, but in theory Java is cross platform compatible, and we all know how that turned out. The most important unproven aspect of .NET however, is **privacy**. Although Microsoft promises it, there is always a risk theft…

.NET is meant to run on a network, which means the network must be up, must not crash, and must not overload. Any breakdown would have very harsh repercussions, as users would have many problems with each and every one of their smart devices.

As well, the costs **may be prohibitively expensive**. At the moment, .NET is meant to run on a Microsoft Windows 2000 server. While these days a Windows 2000 server is not very expensive, it is compared to one running a free copy of Apache.

Finally, the full concept of .NET represents a threat to all existing programming languages - executables, C++, and Java too must die! Since Microsoft is trying to escape platform specific programs with .NET they want everything made to run on a standard browser! Guess which one…

.NET Framework

Introduction to the .NET Framework

The .NET framework is the underlying architecture of Microsoft's software development suite. It is responsible for compiling any supported code into a common language, handling communication with web clients and smart devices by using the lowest common denominator language through HTTP and XML, handling memory, security, processes and threads, as well as providing a rich set of services based and built on the existing operating system that the framework resides on.

Three major components make up that which is called the .NET framework. They are the common language specification layer, the unified core and presentation classes, and the common language runtime (CLR) layer. These components work together to make up the .NET framework, and are illustrated on the next page:

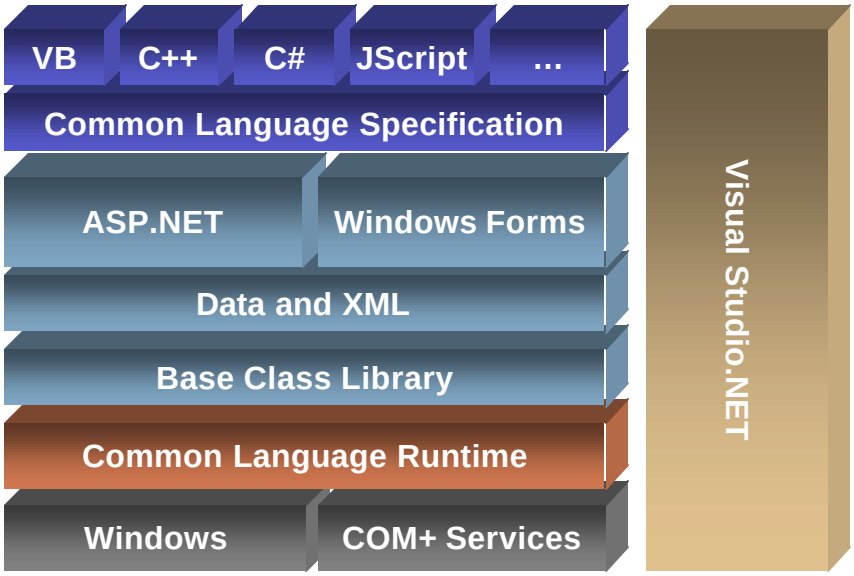

Figure X: The .NET Framework

The middle layer in the figure above, which is not as explicitly obvious, are the unified core and presentation classes. It includes ASP.NET for web development, Web Services for providing specific web services to clients, and Windows Forms for creating and developing Windows-based and/or smart client applications. These classes provide the XML support, networking and data access that allow any web service or application to talk to any device that has internet capabilities. They provide so much functionality in their libraries, such as shopping cart functionality or the ability to create tree-view objects that software developers and businesses can smile. It will greatly reduce a project's time-to-market, as well as a developer's code, making the application reliable and robust.

The common language specification layer allows developers to create their code in any of the major or supported languages, which effectively reduces software language training and allows a choice of languages to choose from to develop code based on a language's strengths and weaknesses. The layer converts the source code that it was developed in and converts it into an intermediate language. At this point, no matter what language you use to develop the software, it becomes a 'common language', meaning that it is indistinguishable which language the code was originally from. This language is called the Microsoft Intermediate Language (MSIL or IL for short). Now the compilation job is easier because it only needs to deal with the IL. The IL source code is compiled using the Just-In-Time (JIT) compiler that is specific to the platform and operating system that the framework resides on. The whole process is illustrated below:

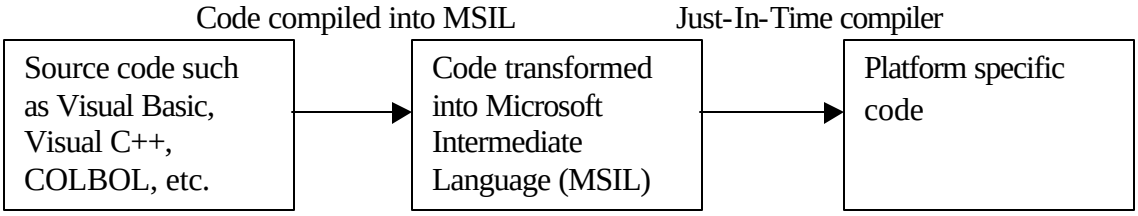

Figure X: Common Language Specification Layer

The common language runtime (CLR) is able to talk to this managed code. The CLR is the layer that handles memory management, security, and the execution of applications, threads and processes. Some say that the CLR is an operating system, but rather, it only provides a rich set of services and features that are built on top of the existing operating system that it resides on. The memory manager now has a sophisticated garbage collection feature, similar in idea to Java's garbage collector. It keeps track of all memory being used, and is able to detect when allocated memory is no longer in use or is lost, and is able to free up that memory. Normally on a Windows-based machine, memory is managed very poorly, and there is constant memory leakage. If the leakage continues for an extended period of time, eventually there will be no available memory left and the system will crash. Managing memory properly is extremely important especially if the computer is running a web service in which the computer is required to be on for days and weeks. The memory manager in the CLR provides extremely effective control over memory and prevents leaks from occurring. The CLR also controls security of applications. In order to do so, applications can or cannot perform certain tasks based on things such as who is running the application, who authored it, where it came from, what it is trying to do, and many other such criteria.

With all this functionality, ease of development and robustness, it is obvious that a lot of work has been dumped on Microsoft to build such complex software. The amount of work on their .NET framework shows, because it comes delivered on 5 CDs!

.NET and Web Services

Web services make up one of the cornerstones of Microsoft's .NET strategy. Essentially, applications will be broken up to simpler and smaller components that are reusable. These components are what Microsoft defines as Web Services that any can access through the network. Basically a Web Service is a dedicated application running on a server. Therefore, applications can be created that use several Web Services from different sources. One of the benefits of this architecture is that the Client application does not need to be updated constantly as the bulk of the work is performed from the Web Service. Therefore, it is up to the Web Service provider to keep these services up to date.

Web Services "expose" their functionality to clients using a XML document called a WSDL (Web Services Description Language). Inside of this XML document are interfaces for calling and invoking procedures defined by the Web Service. The structure

for the communication for sending and receiving SOAP requests are shown in the WSDL document. Any client can receive this information by adding a "?WSDL" to the Web Service URL such as shown:

http://www.webservices.com/Services/WebService1.asmx?WSDL

The architecture that Microsoft has devised is a simple communication stack involving several industry standardized protocols. Locating services is done through the UDDI (Universal Discovery Description and Integration) protocol. One of the protocols that will be discussed in more detail is the SOAP protocol. SOAP is a simplified communication protocol similar to CORBA and DCOM (Another Microsoft initiative). Initially, Microsoft and several other companies set out to push SOAP as an industry standard. SOAP defines the XML format of the exchange of messages between the client and Web Service.

Data exchange is done through the markup language XML which essentially describes the structure of data. Unlike HTML, which details how the information is presented, XML encapsulates data inside the tags (<BOOK>Introduction to .NET</BOOK>).

```
Public Class MathService : Inherits WebService

   <WebMethod()> Public Function RRSP(Income As System.Single) As
System.Single

        If (Income > 75000)
                Return (13500)
        Else
                Return(Income * 0.18)
   End Function

End Class
```

An example of a Web Service class is shown above

ASP.NET

What is ASP.NET? A lot of web developers are familiar with the scripting language ASP that allows the generation of dynamic pages. ASP.NET is the next generation of server side scripting on the Microsoft platform. Some of the benefits of ASP.NET include increased performance increase, separation of content and code, full Visual Basic support, and simplified programming model.

One of the downsides of traditional ASP was that it was a pure scripting language. Every time a client accessed the ASP page, the script is executed and the corresponding HTML output is generated. ASP.NET resolves this issue by compiling the ASP.NET script into MSIL (Microsoft Intermediate Language) when the page is access the first

time. Upon every other request to the ASP.NET page, the compiled MSIL version is executed instead. This leads to better performance as the compiled code is already optimized. Benchmarks have been conducted and they show that ASP.NET code executes 3-4 X faster than ASP and JSP.

ASP.NET now has support for full Visual Basic as oppose to VBScript in ASP. Web objects can be created at run time at the server side using the "runat = server" attribute. With the support of the .NET framework, objects such as datalists, datatables, and dropdown lists have methods and function calls. Therefore, there is less code in ASP.NET than ASP to perform common tasks. To serve up a ASP.NET page simply place the ".aspx" file into the Microsoft Internet Information Server directory and ensure the framework is installed. The Microsoft .NET framework allows both ASP.NET and ASP to run cooperatively. However, the ASP script will not enjoy the optimizations of ASP.NET pages.

```
<html>
   <head>
      <link rel="stylesheet" href="Sample.css">
   </head>

   <body>

       <center>

       <<h3> Car: <asp:textbox id="Car" runat="server"/>

           Manufacturer:  <asp:dropdownlist id="Make" runat=server>
                        <asp:listitem >Honda</asp:listitem>
                        <asp:listitem >Nissan</asp:listitem>
                        <asp:listitem >Toyota</asp:listitem>
                    </asp:dropdownlist>
           </h3>

           <asp:button text="Lookup" OnClick="SubmitBtn_Click"
runat="server"/>

           <p>
```

An Example of ASP.NET code is shown below.

Visual Studio .NET

Microsoft has developed Visual Studio .NET to allow developers to deploy .NET applications and services more easily and efficiently. All of Microsoft's development environments have been synchronized into one development product. Now VB developers will be accessing the same tools and features of Visual C++ developers. The Web Services and ASP.NET pages can all be written in a simple application such as

Notepad. However, Visual Studio simplifies the development by offering integrated MSDN (Microsoft Developers Network) help, Intellisense feature, etc.

Web development projects can now be developed using a broad range of languages. The compiled code once compiled as MSIL is standardized and can interact with each other. Visual Studio supports the development of more than 20 languages using the Command Language Runtime. However, only C++, C, C#, Visual Basic, and Jscript Command Language Runtime Compilers are included with the Visual Studio .NET.