1-2004

Move to Component Based Architectures: Introducing Microsoft's .NET Platform into the College Classroom

Meg C. Murray
Kennesaw State University, mcmurray@kennesaw.edu

MOVE TO COMPONENT BASED ARCHITECTURES: INTRODUCING MICROSOFT'S .NET PLATFORM INTO THE COLLEGE CLASSROOM[*]

Dr. Meg Murray
Kennesaw State University
Kennesaw, GA
mcmurray@kennesaw.edu

ABSTRACT

A transformation has been occurring in the architectural model for computer-based application intense software systems. This new model, software-as-a-service, will have a profound impact on the design and development of software for many years to come and as such college level computing curriculums will need to incorporate the concepts and methodologies associated with this new architecture. The platform is built upon a view of interrelated, distributed peer-level software modules and components that work in tandem to achieve specified functional goals. From Microsoft's viewpoint, migration to the new platform requires a radical shift in the software development lifecycle. It is becoming imperative that higher education computing programs take a proactive stance in reviewing their curriculums and making plans to align them with this new paradigm. This paper explores Microsoft's .NET strategy and provides a synopsis of the efforts taken by one Computer Science and Information Systems Department to incorporate .NET into the curriculum and the classroom.

[*] Copyright © 2003 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

INTRODUCTION

A transformation has been occurring in the architectural model for computer-based application intense software systems. In the past, the model espoused the large software application that encapsulated all needed functionality into one centralized system. The focus was on the software application or an integrated suite of related applications as the primary source of functionality. These cohesive application sets are generally implemented, managed and maintained and even purchased by end users from one primary source. A major paradigm shift is on the horizon that is challenging the view of the all-encompassing software application to what, Steve Ballmer, CEO of Microsoft eloquently stated as software that provides a service (Ballmer, 2000). This software-as-a-service model will have a profound impact on the design and development of software for many years to come.

Envisioning software as a service requires a major change in the underlying platform that supports the interoperability of software applications. Spratt (2001) puts it succinctly when he states, "The applications that developers write, the services they wish to provide for their target environment, and the embedded devices that form the end-points of the network they will be deploying, will all work together as a platform to provide access to the information their customers need at anytime and anyplace via the Web" (p. 91). This platform represents the first comprehensive architecture for the post - PC era (Spratt, 2001) and many believe that this model will become the dominant architecture for enterprise and Internet applications (Meyer, 2001).

Both Sun Microsystems and Microsoft have introduced frameworks to support this new component-based, Service Oriented Architecture (SOA). These platforms provide support for software development, deployment, execution and management that facilitate interoperability across servers, development languages, applications and devices. Both Sun's Java2 Enterprise Edition (J2EE) and now their Open Net Environment (Sun ONE) and Microsoft's .Net provide capabilities to achieve this goal but they approach this in different ways. The primary focus of Sun ONE is the definition of an underlying architecture through specifications of a standard platform for hosting and delivering J2EE applications. Microsoft has embraced this change by the introduction of their .NET initiative. Microsoft describes .NET as "an open language platform for enterprise and Web development" (Microsoft, 2002). While the frameworks offer similar services, the cliché commonly used to describe them is that '.Net is Windows-centric and language neutral while J2EE is java-centric and platform neutral.' Both platforms offer viable alternatives, however this paper focuses on the Microsoft .NET environment.

The .Net platform is built upon a view of interrelated, distributed peer-level software modules and components that work in tandem to achieve specified functional goals. From Microsoft's viewpoint, migration to the new platform requires a radical shift in the software development lifecycle. Their model is built upon a three-pronged approach: develop, deploy and execute without constraints. Their plan is to provide the strategies and tools necessary to simplify the development, deployment and execution processes so that interoperability between various heterogeneous computing technologies and software applications can be achieved in as seamless a manner as possible.

This transformation in software development has many implications for college level programs that teach computing technologies. It will require a review of what is currently being done and an assessment of how that can be aligned to this new paradigm. The focus is shifting away from the development structure of one program equals one application to where one application is made up of a collection of software modules. Students will not only be challenged to understand isolated program development, they will be required to understand program interoperability. Students must also be introduced to the new development platforms and their associated development tools. Curriculums will need to be revised to incorporate the theoretical concepts associated with the new Service Oriented Architecture and course instruction will need to be extended to include the new tools into the classroom. It is becoming imperative that higher education computing programs take a proactive stance in this area so that students in the computing sciences will able to be successful participants and contributors to the discipline. Some of the most often cited desired skills for computing jobs is now knowledge of Sun's J2EE and Microsoft's .NET (Swinton, 2003). This paper provides an overview of the .NET framework and discusses the experience of utilizing .NET in the college classroom.

INTRODUCTION TO THE .NET ENVIRONMENT

Microsoft's .NET is not a product, but as Microsoft states, ".NET is a series of Microsoft technologies interconnecting the world of information, people and devices" (Microsoft, 2002). Consequently, .NET is a collection of resources that includes development tools and languages, server software and protocols. .NET also includes device code specifications that allow developers to create interoperable applications that run on a multitude of end user devices such as PCs, PDAs, mobile phones, etc. Other underlying technical components of .NET include Visual Studio.NET, Microsoft's multi-language development tool, C#, a new object oriented programming language, a Common Language Runtime (CLR) component supporting the inclusion of source code written in multiple programming languages and a new version of Active Server Pages. Foundational to the .NET platform is the incorporation of the eXtensible Markup Language (XML) and the Simple Object Access Protocol (SOAP) that serve as mediums for the exchange of data between applications accessible over the Internet. While not all of Microsoft's server products meet the specifications of the .NET architecture, their intention is to migrate all Microsoft products into the .NET structure. A .NET server is scheduled for released in early 2003 and Bill Gates (2001) has stated that .NET will affect every piece of code that Microsoft develops and that all future Microsoft products will be touched by .NET.

The thread that ties .NET technologies together is known as the .NET Framework. The premise of the Framework is to be a common foundation that allows applications, regardless of their source of programming language origin, to access system services in the same way. For example, the Framework supports Internet browser-accessible applications, Web services applications and even Windows based applications. The .NET Framework is based on two components, The Common Language Runtime (CLR) and .NET Framework Classes, that work together to provide a standard set of data types and standard set of code to perform common functions. The .NET Framework, available as a free download from the Microsoft Web site, must be installed on both client and server machines.

DEVELOPMENT

In the past, the model for the development of software systems was focused on developing the all-encompassing software application. The trend is now towards developing more concurrent, more distributed, and more connected applications. This represents many challenges to the software developer. Previous attempts to support this kind of development were characterized by complex development environments and developers often had to use several development tools to create one application. Microsoft has addressed the need to simplify software development with the introduction of their new integrated development environment (IDE) known as Visual Studio .NET. Not only does Visual Studio .NET provide support for the development of source code in three different languages and Web based technologies such as ASP.NET, it also automates the assembly of discrete components into one unified entity. The intent behind the .NET platform (Visual Studio combined with the .NET Framework) is to allow developers to concentrate on the creation of the functional aspects of an application while the IDE handles the background details.

.NET provides development support for source code in three major programming languages: Visual C++, Visual C# (C-Sharp), and Visual Basic .NET. These languages are able to create and execute services that can be distributed over the Internet. Visual Studio .NET supports auto-generation of most code by providing a graphical environment allowing the developer to concentrate on design rather than code writing. Further, Visual Studio .NET features auto-completion (Intelli-sence) of code which can be used to program for any device, including mobile devices. Visual Studio.NET also includes a major overhaul to the Internet-based server-side scripting language, ASP. ASP.NET tools incorporate programmatic support for building smart Web sites. One advantage of the language support provided in Visual Studio .NET is its ability to target the development of XML-based Web Services.

Microsoft has made it easy to create applications using Visual Studio .NET. For instance, Visual Studio .NET provides Web service developers with a whole plethora of tools to create Web services. The developer begins by selecting the project type (ASP Web service, executable, etc) and the programming language to be used. The developer will also name the service and specify the location of the Web server that will run the Web service. The developer then creates the application using the many graphical resource aids and/or wizards provided with the development environment. Upon completion, the developer simply chooses to build the application. The build process not only performs compilation but also creates the auxiliary files needed to successfully implement a Web service (SOAP file, WSDL file, etc.). Visual Studio .NET even provides a Web services registration facility that allows developers to search for or publish Web services through the globally available yellow pages of Web services known as the UDDI.

Visual Studio .NET represents the strongest link in the .NET platform. In the past, if developers wanted to create component-based systems or even Web services, they had to write a good deal of code to 'glue' the parts together. Microsoft has greatly simplified this process while at the same time provided an easy-to-use IDE that allows the developer to focus solely on functional aspects. Visual Studio .NET does facilitate faster development times and enhances application integration.

DEPLOYMENT

Once an application is developed, it must be accessible in order to be used. Distributed applications comprised of a series of functional components greatly increase the complexities involved in their deployment and distribution. Today, deployment typically means copying a set of software components to individual client and server systems on which they will be run and then describing those components to the operating system (in Windows this means making a series of registry entries). A primary goal of the .NET Framework is to simplify this process and .NET does this through an application assembly and packaging scheme. .NET applications are built from assemblies. Assemblies are logical units comprised of a combination of classes, executable files and other needed resources such as image files, etc. Accompanying each assembly is a manifest or metadata that describes the assembly's originator, version identifier, cultural identifier, listing of included files and listing of dependencies on other assemblies. Developers can pick and choose different collections of resources 'a la carte' and combine them as needed to construct applications that serve different purposes. These applications become self-describing units of deployment greatly simplifying the distribution process.

EXECUTION

Assemblies provide a way to package modules into applications for deployment but the real requirement of any application is that it is available when needed. In a truly distributed environment, this means that an application must be able to be executed on a variety of different operating platforms without concern for the source code development language. As such, one of the goals of .NET is to be language-independent (Meyer, 2000). To achieve language-independence, Microsoft has developed the Microsoft Intermediate Language (MSIL). Source code, regardless of its origination language, will be compiled into MSIL. The MSIL is executed in the Common Language Runtime (CLR), part of the .NET Framework. This concept is very similar to the Java Virtual Machine. Applications compiled into the intermediate language will be portable across varying operating system platforms. In addition, the CLR performs resource management tasks such as garbage collection, memory allocation and debugging support. Translation into machine code, required at the time the application is to be executed, will occur in what is referred to as Just-in-Time compilation. At the base level, everything in the .NET platform boils down to assemblies that can be executed on the CLR.

Creating a language neutral platform is a major undertaking. It requires adherence to set of specifications. For a programming language to be eligible for inclusion as a .NET supported language, a set of possibilities and constructions must be specified and included in an agreement called the Common Language Specification. Once this is completed, all that is required is the development of a compiler that translates source code into MSIL. In theory, MSIL increases flexibility; the reality is more limiting. The CLS serves as the common denominator for all .NET supported languages. According to Lauer (2001), this common denominator imposes restrictions such that the compiled code may have little in common with the original, except that the syntax remains the same. To date, Microsoft reports that more than 20 languages have been ported to the CLR and, of course, Microsoft provides full CLR support for C#, VB.NET, C++ and ASP.NET.

.NET represents a major undertaking by Microsoft. Their vision is to provide an all-encompassing platform that provides the underlying commonality needed to bring together disparate technologies and software components. This can only be supported by new ways of viewing software development and as this approach becomes more widely accepted, students within the computing sciences must be introduced to the underlying concepts and tools within their college curriculum. The movement towards more highly integrated and interoperable software components is still very young but implications for the future are immense and college level computing programs must be on the forefront of this movement.

LESSONS LEARNED: INTRODUCING .NET INTO THE CLASSROOM

Microsoft's .NET platform not only represents a major shift in the approach taken to software development and deployment, it also has the potential to change what is done in the college classroom. Migrating to the .NET platform is not trivial. There are many considerations that must be made. These encompass a realm of issues ranging from hardware requirements to the approach taken to teaching code development. Recognizing that component-based development and the .NET platform are being widely adopted in industry, a combined Computer Science and Information Systems Department in a regional state university made the decision to explore this emerging technology and assess what would be the best way to incorporate .NET into the Computer Science and Information Systems programs.

An internal grant was secured to procure a dedicated server to be used for the .NET project. In addition, the University subscribed to the Microsoft Developer Network Academic Alliance program that provides access to Microsoft's programs and developer tools. Through this program, software may also be installed in departmental instructional labs and may also be checked out to students and faculty to be used for research or instructional purposes. The .NET framework and Visual Studio.NET were installed in the computer lab and a server running the Windows 2000 Server operating system and IIS was made accessible for hosting student work.

The incorporation of .NET was designed to be implemented in stages that span three academic semesters. Prior to beginning the project, an upper-level undergraduate student was engaged in an independent study to initially investigate installation and ease of use issues. The focus was specifically on using Visual Studio.NET to develop simple programs including a Visual Basic.NET application and an ASP.NET application. The results of this work showed that there were many desirable features to the .NET platform but also highlighted some problem areas. Applications could be developed quickly using the Visual Studio.NET environment. However, issues arose in the installation of the product and in the completeness of the documentation. While good documentation and resources exist related to the programming languages, such as Visual Studio.NET, limited documentation was found related to the Visual Studio.NET interface and some of the documentation related to ASP.NET and the associated technology of Web Services was out-of-date or incomplete.

Given the preliminary findings, a second independent study program was set up. This time four students were involved. The outcome of this independent study was to thoroughly explore installation requirements, make an assessment of ease of use and to develop sample programs using the primary development environments supported under Visual Studio.NET including Visual Basic.NET, C# and ASP.NET. Further, the Visual Studio.NET development environment was introduced to a group of graduate students studying e-business technologies. The students were asked to develop an e-business solution using ASP.NET. The culmination of this part of the .NET project was the development of a special topics course to be introduced as an upper-level undergraduate course. The experience gained through the previous introduction of .NET into the classroom was insightful. Problems that were not anticipated arose and the opportunity provided a chance to find ways to overcome these problems. As a result of the initial introduction of Visual Studio.NET, it was recognized that gradual implementation into the classroom was necessary to insure a smooth transition to this new environment. Widespread adoption of the .NET platform in those courses deemed appropriate for its use will occur after the special topics course is completed.

One of the first considerations that must be made when moving to the .NET platform is related to available computer resources. There are two areas of concern. One relates to the .NET framework and the other to the Visual Studio.NET integrated development environment. The primary issue with the .NET framework is that it must be installed on the computer used for code development and also on any client machine that will be executing code built using .NET. The framework is available for the most commonly used Windows operating systems and can be downloaded for free from the Microsoft upgrade site. The only caveat is that the .NET framework appears to be modified quite frequently and upgrades must be consistently maintained.

The Visual Studio.NET environment is an extremely rich tool. The environment includes support for three programming languages, Visual Basic.Net, C++.NET and Microsoft's new language, C#. Also included is support for ASP.NET and the development of Web services. Visual Studio.NET has 13 main tool windows that can be incorporated into the development environment and users can create profiles that meet their needs. Visual Studio.NET also includes tools that support additional functions such as an XML editor and what is termed, 'Server Explorer.' The Server Explorer provides automatic access to databases, mail servers, and other system resources making it much easier to incorporate these resources into program code. A default scheme is selected when a specific language is chosen for development and is often best to use this default scheme when first learning the tool otherwise the screen can become cluttered making it difficult to effectively utilize the needed resources. Spending time upfront to explore the different tools offered within this truly integrated development environment is well worth the effort.

Many challenges and issues arose during the installation process. Visual Studio.NET requires a fairly fast processor and uses a fair amount of system resources. For instance, minimum requirements are a 600MHZ processor, 128MB of RAM and a minimum of 4GB hard drive space. A CD or DVD is required to install the software and installation time can extend to well over 2 hours when no problems are encountered. The installation comes on either one DVD or 5 CDs and while it can be distributed over the network, the download time is prohibitive without fast access. Visual Studio.NET is only supported on the Windows 2000, Windows NT and Windows XP operating systems. These requirements are a concern when assessing the resources available in the classroom, student labs and on student owned computers. For instance, while Visual Studio.NET will run on XP home edition, it can only be utilized to develop Windows based applications such as those written in Visual Basic.NET. In order to utilize Internet programming support such as ASP.NET or Web services, the development computer must support the IIS Web server. IIS is supported on XP Professional edition but not on XP home edition. Many students do not have personal systems that support these operating systems. The necessary operating systems can be distributed to students under the MSDN Academic Alliance; however the operating system upgrade is a major undertaking. This restriction means that institutions moving to the .NET platform must be prepared to offer substantial student computer lab support and if distribution to students is an option, one must be prepared to spend the first couple of weeks of the course assisting students in the installation process.

Actual code development using the Visual Studio.NET environment did not present problems beyond those expected when teaching a programming language. (It should be noted for those familiar with earlier versions of Visual BasiC, substantial revisions have been made to Visual Basic.NET to evolve it to a more pure object oriented language.) Developing Windows applications, for the most part, worked as expected. The only issue arose when transporting the code from the development system. The new host or client machine must be running the .NET framework in order to execute the application. Unexpected problems did arise when developing Internet-based applications. In order to support the developer, Visual Studio.NET automatically creates the necessary folder structure for the ASP.NET application utilizing the file structure supported by IIS of the machine on which the application is developed. The common practice is to develop on one system and then transport the finished application to a host Web server. Consequently an ASP.NET application may run on the localhost where it was developed, but due to permissions settings, the same application may not run when copied to another machine. In previous versions of ASP, configuration settings were managed at the IIS level. A new feature of ASP.NET is to associate these settings with individual applications. These settings are contained in a 'Web.config' file. When an ASP.NET application is transferred from one system to another, the configurations settings must be manipulated to match the requirements of the new host system. As one student stated, coding is now only one part of the development process, configuration management becomes just as important. This adds another layer of complexity in the course curriculum. A course incorporating Internet programming using ASP.NET must now include a section on configuration management, an area that previously may have been relegated to a more specialized course on systems administration.

The primary goal of the .NET platform and Visual Studio.NET in particular, is to enhance the process of software development. Software applications have become increasingly complex and continue to be built upon intrinsically difficult technology that changes continuously. Microsoft's .NET initiative is an attempt to address these complexities by providing developers with a way to easily develop and deploy distributed applications. In many ways, .NET achieves its goal but it does not escape the challenges and issues that befall any new technology. The learning curve with .NET is not trivial and, as with all software, Visual Studio.NET has its set of idiosyncrasies that must be identified and dealt with. Widespread adoption of .NET into the college classroom should not be undertaken before sufficient exploration of experimentation with this technology has occurred.

CONCLUSIONS

The software applications of tomorrow will be interrelated, distributed peer-level software modules and components that work in tandem to achieve specified functional goals. These complex software systems encompass a collection of distributed applications, resources and services connected via a network of computing systems. This software as a service model is based upon a build-via-assembly approach and this type of development will require new ways of thinking about hardware and software architectures. These new ways of thinking must begin to permeate what and how computing technologies are taught in the college classroom.

.NET fundamentally changes software development (Meyer, 2000). The model of develop, deploy and execute within an integrated environment will serve as the software development methodology for the service oriented architectural model which represents the next generation of software-based applications. This means that students of computing will be required to learn new processes and utilize new tools as a part of their course of study. And even while the technology is still young and evolving, the time has come to find ways to incorporate these new platforms in to the curriculum and into the classroom.

REFERENCES

Ballmer, S. (2002). Steve Ballmer's Comments to Media at Forum 2000. Available on-line: http://www.microsoft.com/presspass/exec/steve/06-22f2k.asp

Chappell, D. (2002). *Understanding .NET: A Tutorial and Analysis*. Boston: Addison-Wesley.

Douglass, B. P. (2001, January), The evolution of computing, *Software Development Magazine*, 01(1).

Farley, J. (2001, March), "Picking a winner .NET vs. J2EE," , 9(3), pp. 36-50.

Gates, B. (2001, June 14), *Microsoft .NET Today*, Memo to Developer and IT Professionals. Microsoft Corporation.

King, N. (2002, June 28). Visual Studio's (Dot) Net Worth: Not just a new version; a new vision. Intelligent Enterprise. Available on-line: http://www.intelligententerprise.com/020628/511products1_1.shtml

Lauer, C. (2001, January). Introducing Microsoft .NET, Available on-line: http://www.dotnet101.com/articles/art014_dotnet.asp.

Meyer, B. (2000, November), "The Significance of 'dot-NET'," *Software Development*, 8(11), pp. 51-60.

Meyer, B. (2001, August.). .NET is Coming. *Computer* p. 92-97

Microsoft Corporation. (2002, April 04). Defining the Basic Elements of .NET (2002, April 04) Available on-line: http://www.microsoft.com/net/defined/whatis.asp

Microsoft Corporation. MSDN Academic Alliance Program. Available on-line: http://www.msdnaa.net/program

Spratt, D. (2001, April). Leveraging Microsoft .NET and Sun One Technologies in Control Systems Design. *Control Solutions* 74(4). P. 91-92.

Swinton, A. (2003, January 7). .Net may top list of job skills in demand. Available on-line: http://zdnet.com.com/2100-1104-978922.html