Deciphering .NET for GIS

Mark R. Brûlé

Just what exactly is Microsoft's new .NET platform? And what does it mean for the GIS industry? To find out, read on.

The computer industry has a history of generating new products with a lot of hype surrounding their launch. New ideas are dreamed up all the time, and companies want people to believe their products will revolutionize the way we operate.

According to Microsoft Corporation (www.microsoft.com) Chairman Bill Gates, his company has initiated two changes that have fundamentally altered the way computer users have done business. The first was the move from DOS to Windows in the early 1990s. The second was the announcement of the 32-bit version of the operating system (Windows 95) a few years later. Now, Microsoft is touting its recently unveiled .NET platform as the third such fundamental change through its commercials, Web site, and other marketing material. Yet, the technical details about .NET have been either too vague (from the marketing material) or very obtuse (from several large tomes on the subject), leaving programmers and software developers both hazy on the particulars and weary of delving into the minutiae.

But, once you get your head around it, .NET reveals itself as a very powerful tool which may indeed revolutionize enterprise application development as well as provide a more effective architecture for easier integration of spatial functions with mainstream enterprise IT applications. This overview of .NET should help clarify the platform and, more importantly, supply some insights about how .NET might impact GIS users and the industry.

The concept

The primary goal of the .NET initiative, according to Microsoft, is to simplify the development and integration of applications that can be deployed in a network setting. The platform is built around the concepts of language and platform independence, allowing programs to be deployed on any .NET-enabled machine (similar to Java deployments) and to be written in any mixture of .NET-enabled languages. Unlike Java, .NET offers a more integrated development environment and enhanced server-side technology that simplifies the development of more sophisticated applications.

On an abstract level that description sounds fine, but what does it mean practically in terms of application development? To answer that question, it's important to understand that .NET depends on the concept of Web services.

Web services. Web services are essentially applications that integrate data and computation as deliverable executables over a network. Unlike traditional applications, Web services comprise componentized modules that can be separated out from the application and accessed by other programs. For example, suppose you had developed a module for your GIS to compute the drive time between two addresses, and you wanted to make just this functionality available to other enterprise applications, such as a CIS, without having to interface the entire GIS with

Glossary

ADO: ActiveX Data Objects

ASP: Active Server Pages

CIS: Customer information system

CLR: Common Language Runtime

COM: Component Object Model

NAD: North American Datum

OGC: Open GIS Consortium

OLE: Object Linking and Embedding

SOAP: Simple object access protocol

VB: Visual Basic

XML: Extensible markup language

Mark R. Brûlé is CTO and cofounder of Coherent Networks, Inc. (www.coherentnetworks.com), a firm specializing in providing software-driven solutions and services that address the information-access needs of telecommunications carriers and power utilities. This article is based on a presentation delivered at the 25th Geospatial Information and Technology Association (www.gita.org) conference in Tampa, Florida, March 17–20, 2002.

Courtesy of Microsoft Corporation

FIGURE 1 This schematic illustrates the general concept of .NET and its ability to provide integrated application environments through Web services created from an array of enterprise and application servers.

the CIS. By implementing the drive-time calculation function as a Web service, other applications, like the CIS, could connect to it through a standard interface to take advantage of its capability. And, all the tasks of registering, launching, and executing the service and passing the data are handled through common protocols.

As another, more GIS-oriented example, suppose you need to overlay survey data in NAD 1983 on a map delineating wetlands in State Plane coordinates. The concept of Web services would enable the owner of the survey data to create a server-side application containing not only the data, but the metadata describing the projection as well as the computation routines to transform and convert the data to a specified coordinate system. Thus, when you subscribe to the Web service (again housed on a Web or central server), you enter the projection and coordinate system of your wetlands map and the Web service automatically converts the survey data to your specified coordinate system and overlays the information on your map in the correct location. This eliminates the need for you to perform the coordinate transformation and reprojection and gives you instant access to the data in a readily usable form.

Going further. Sophisticated programmers have been developing simpler Web services for many years using Java, OLE/COM, ActiveX, and other technologies. What .NET does is enable the creation of more sophisticated Web services (like those previously described) in a simple and effective manner. The .NET framework also facilitates Web service development in a language-independent manner by using XML for formatting data and SOAP for communicating over standard HTTP channels to exchange requests and results. The end result is that with .NET you can develop server-side applications that support other Web applications regardless of where the consumers of your application reside on the network. They can be on the same Web server, another server on a local-area network, or anywhere on the Internet. In fact, .NET Web services aren't just intranet/Internet–based solutions. Rich clients supporting XML, SOAP, and HTTP, such as Microsoft Word, can talk to a Web service stored on an application or enterprise server inside a firewall. Of course, Web services can also run on an intranet/Internet server, but the point is that access to .NET services doesn't have to be browser based. Figure 1 provides a conceptual overview of .NET as provided by Microsoft.

As an example, using .NET to create the drive-time calculation Web service described earlier, you would take the original VB or C++ language for the application and deploy it on a Windows server with the .NET framework. Then, in the .NET programming environment, you would code the application as a function that accepts two addresses and returns a time duration. Once that step is complete, the .NET Web services tools at your disposal handle all of the details of encoding and decoding these parameters into XML and SOAP and transmitting the messages to and from the requesting application.

The nitty-gritty

Of course, coding applications and deploying them is much easier described than done. To use .NET, system implementers and software developers will still have to learn the new programming environment and application execution languages on which .NET is based.

Execution environment. For .NET, Microsoft has developed its own execution environment referred to as the CLR. The CLR is a virtual machine, meaning it is a runtime environment implemented on top of a hardware platform that uses a computer's processor to run compiled programs. At this level, the CLR embodies the same concept as Java. Both the CLR and Java provide class libraries for things like windowing capabilities and file system interfaces. The CLR, however, takes the concept a bit further — attempting to abstract the entire operating system in a language-independent manner.

The popularity and widespread acceptance of Java should enable easy adoption of the CLR. Java programmers will still have to learn the intricacies of the CLR's routines and assemblies. For Windows programmers, however, CLR adoption is a natural progression because the CLR uses the same integrated development environment (Microsoft's Visual Studio) that a majority of developers employ to create Windows applications.

Another language. With the .NET framework, Microsoft has also introduced a new programming language, called C# (C-sharp), and enhanced VB, taking some bold steps in terms of language support.

C# is a fully object-oriented language that looks very much like a cross between C++ and VB. Syntactically, similar to Java, C# code resembles C++ and can be characterized (with a very broad brush) as C++ without pointers. In addition, Microsoft has borrowed some syntax from VB (such as Get and Set methods) to give C# a flavor all its own.

The CLR also introduces the concept of *managed code* into the Visual Studio development environment on which it relies. The CLR's managed code provides functions for

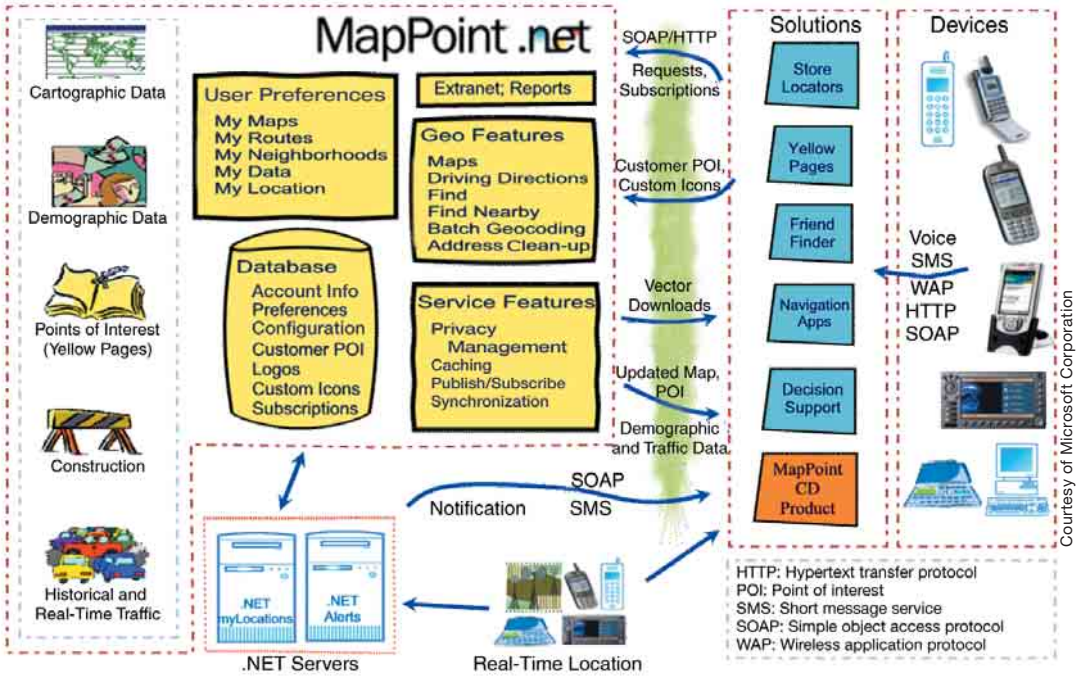

FIGURE 2 MapPoint .NET, the first .NET Web service made available, enables the integration of several MapPoint desktop mapping modules with other applications. This diagram illustrates how MapPoint .NET is able to supply mapping function to connected and SOAP- and XML-capable applications.

allocating memory, referencing it, and cleaning up unused memory blocks. Languages that implement the CLR's managed code paradigm (such as C#, VB, and C++ when used in managed code) have no access to pointers, and all unused memory is cleaned up by the system and not the programmer.

In addition to adding C#, Microsoft has modified VB to enable it to interact with the CLR and leverage its power. The .NET implementation of VB, referred to as Visual Basic.NET, adds the missing components to make VB a full-fledged, object-oriented development language. Code developed in VB.NET, similar to C#, is managed code, and fits well with previous implementations of VB, making it easier to distribute existing desktop applications as Web services in a networked environment.

With all these language enhancements, what Microsoft has effectively accomplished is enabling developers with applications written in C++, VB, and other similar codes to easily translate their programs to the .NET framework for creating new Web services via the CLR.

Other tools. To round out the .NET framework, Microsoft has added an array of development tools to ease the programmer's and Web developer's life. In addition to a set of classes to ease the creation of thin-client Web services, .NET offers another set of classes, known as Windows Forms, for fat-client developers who want to deploy applications with sophisticated user interfaces. Finally, Microsoft has re-invented and extended the functionality of ASP and ADO, renaming them ASP.NET and ADO.NET, respectively.

ASP.NET, which started off as ASP+ but was renamed to fit the .NET style, strives to simplify the development of custom controls and intelligent Web pages. Probably the biggest enhancement is the addition of language-independent development and the elimination of scripting. Thus, with ASP.NET developers can use any .NET language (VB.NET or C#, for example) to specify the behavior of a Web page, which means they can use all of the features of the programming language to develop the code as well as debug it and maintain it. In addition, developers can create variations on a control to add or modify behavior in a way that is particular to the information being presented. In essence, what ASP.NET does is put more desktop-like programming options at the developer's fingertips for creating more sophisticated Web sites.

ADO.NET provides database access to developers. The original ADO supplied users with a COM interface to databases in their network, allowing the developer to take a table out of an existing relational database and perform common operations on it. ADO.NET extends this notion by enabling multiple tables to be stored and manipulated. The result is that one can develop more sophisticated operations because the data can support more application logic before needing to re-query the database. In addition, because an ADO.NET object communicates its information in the form of XML documents (with schema information describing the types of data to be represented) virtually any platform can send and receive information provided it has an XML parser on board.

Ready-made services

The details of .NET can be overwhelming — especially for end users who simply want the benefits of .NET without having to learn complex programming. Luckily, Microsoft has developed a business strategy to enable utilities, government agencies, and other organizations to benefit from .NET without having to enlist a cadre of custom software programmers.

Microsoft's .NET business strategy is to provide Web services in ready-to-integrate modules so that access to Web services is not dependent on having an in-house programmer create them. For example with MapPoint .NET (www.microsoft.com/mappoint/net/), the first .NET Web service made available, Microsoft provides mapping modules for integrating with other applications (see Figure 2). Thus, by setting up a Windows server with a .NET framework and subscribing to a MapPoint .NET service, users can integrate mapping functions with other systems and data, such as a CIS or sales/marketing database. The idea is that MapPoint .NET makes several functions in the MapPoint desktop software available as Web services. These functions include not only map generation, but driving direc-

tions, distance calculations, proximity searches, and other location intelligence.

GIS poised for .NET

Of course MapPoint .NET is just the start of Microsoft's .NET Web services offerings. Since early 2002, the company has been gearing up to offer similar types of Web services based on its other software packages, such as its media players. But it is interesting that Microsoft unveiled MapPoint .NET as the first Web services for the new platform. It may be that the MapPoint team at Microsoft had already been heading in that direction, enabling it to quickly adapt to the .NET platform. But it also seems that the geospatial software industry in general has been heading toward a .NET-type Web services model because of the implications for broadening into markets beyond mapping (for example, location-based services).

OGC's (www.opengis.org) Web Services interoperability initiative and ESRI's (www.esri.com) G.NET are two somewhat parallel developments to Microsoft's MapPoint .NET. Both the OGC and G.NET models seek to promote and develop Web services that, like .NET, combine data and computation as network-deliverable applications that enable interoperability. Of course, OGC's and ESRI's initiatives are geared more specifically to GIS users and mapping and have different goals. G.NET, for instance, focuses on creating Web services for enabling easy integration of GIS data (such as in the example described earlier involving the overlaying of NAD 1983 survey data on a map delineating wetlands in State Plane coordinates). OGC's initiative, on the other hand, seeks to set a standard for how software companies should deliver map-based Web services like MapPoint .NET and G.NET.

Regardless of whether it's .NET or a similar initiative, the concept of Web services will demand a whole new way of thinking about GIS and a new model for doing business. By creating and delivering GIS functions as Web services (from which theoretically one might build a custom GIS), software companies would have to adopt a licensing model for the use of the services and a way to track use of the service. With MapPoint .NET, for instance, Microsoft charges a subscriber fee for a set number of uses of the Web services in a one-year period, and the company monitors usage by hosting the services on its own network.

Will it catch on?

In the end, it is clear that Microsoft has learned from previous products and incorporated those lessons into the .NET framework to launch a powerful platform for advancing a Web services computing model.

The architectural choices that Microsoft has made can be viewed (by the optimist) as forward looking and inspired, or (by the pessimist) as Microsoft once again trying to make the world conform to its proprietary technology. But for the .NET framework to become the revolution that Microsoft envisions, the real test will rest on the proliferation and openness of the CLR. The big win for .NET won't happen until other platforms (such as Linux- or other Unix-based systems) support the CLR. The good news is that Microsoft is actively working in the direction of openness for .NET, cofounding the Web Services Interoperability Organization (www.ws-i.org). The group, comprising some typical anti-Microsoft companies (Oracle, IBM, SAP, and others), is seeking to promote Web service interoperability across platforms, operating systems, and even programming languages.

As for users of GIS technologies, the first use of .NET will probably be on corporate intranets that support Windows servers. The primary benefit of the .NET framework at this point is the ability to share data and applications across an IP network. Most of the enterprise programs used by utilities, for instance, have COM interfaces, which provide IT department personnel with programming access. This COM access makes it relatively easy to place a program on an application server and wrap it up into a .NET component, thus distributing a system across the enterprise without fat-client controls while providing novel ways to use the application.

It all sounds good in theory, but only time will tell if the developers, companies, and users will find enough return on investment to start publishing their data and applications as .NET Web services.