by GERRY MILLER

.NET vs. J2EE

According to nearly every industry pundit, including my esteemed (though misinformed) colleague from Sun Microsystems, integration of systems is critically important for most enterprises. The ability to quickly assimilate and aggregate large amounts of information from disparate systems can mean the difference between life and death for an organization. Ease of access by customers and seamless supply chain management with business partners are quickly becoming the only distinguishing factors in an increasingly commoditized marketplace.

One of the problems with integrating computer systems is the incredible complexity and associated cost of doing so. Many systems are old and scantily documented; still others are proprietary with no natural hooks into their data. And these are just problems that exist within a company's firewall. Imagine how the complexity increases as an enterprise begins integrating its systems with those of its business partners and customers, with the added security ramifications brought on by Internet communications.

Figure 1. The complexity of integrating systems.

The sheer number of interconnections is another problem. Companies have many systems—the alphabet soup of ERP, HR, CRM, SCM—each with many constituents. The geometric complexity of all these interconnections begins to stagger the imagination. It is no wonder so many integration efforts fail to bring about promised savings or other business benefits.

Web services offer a tidy solution to this integration mess. Rather than having to understand each system's deep underlying data structures, or having to write point-to-point application integration code, companies can simply add a Web services layer around individual systems that exposes necessary information and functionality in a standard way. Integration then becomes an effort to orchestrate business processes by making Web services calls, rather than a massive retooling effort.

At its very root, a Web service is nothing other than a server that listens for and replies with SOAP, generally via HTTP. In practice, a Web service will support WSDL to describe its interfaces, and should also be listed in a UDDI registry. Of course, at this point Web services are deceptively simple. So far the technology industry has not coalesced around Web services standards for security, transactions, state management, and workflow. However, nearly every member of the technology community has a vested interest in developing these standards, so they should come to fruition within a few years.

Most Web services development is being accomplished today using either Microsoft .NET or Sun Microsystems' J2EE specification. This is interesting, considering the undisputable fact that J2EE has no support for Web services. The J2EE specification will not contain any native support for Web services until J2EE 1.4, which Sun delayed recently by six months, so commercial implementations are unlikely before 2004. Developing Web services with J2EE today means using either extensions to J2EE that are not part of the specification, or doing lots of XML parsing in code.

Interestingly enough, despite a common perception to the contrary, J2EE is in fact not an open specification. While Sun contends that J2EE is beholden to the Java Community Process, the JCP really only defines a process for community members to suggest updates and changes to the specification. Final specifications can only be approved by Sun. Sun has more than once publicly committed to turn the technology over to a standards body, and each time has reneged on this promise.

A core part of Microsoft .NET, on the other hand, is comprised of the Common Language Interface (CLI), of which Microsoft officially relinquished control to a true standards body, the European Computer Manufacturing Association (ECMA).

Still, the business world is very large, and there is certainly room for several Web services development tools. In some cases it will be very clear whether Microsoft .NET or J2EE (or some other technology) is appropriate, for example, when a company has existing investments in a particular technology, is experiencing resource constraints, or has limitations based on software it is already using. In most cases, however, Microsoft .NET will offer a considerable advantage based on time-to-market, performance, and overall cost of solution.

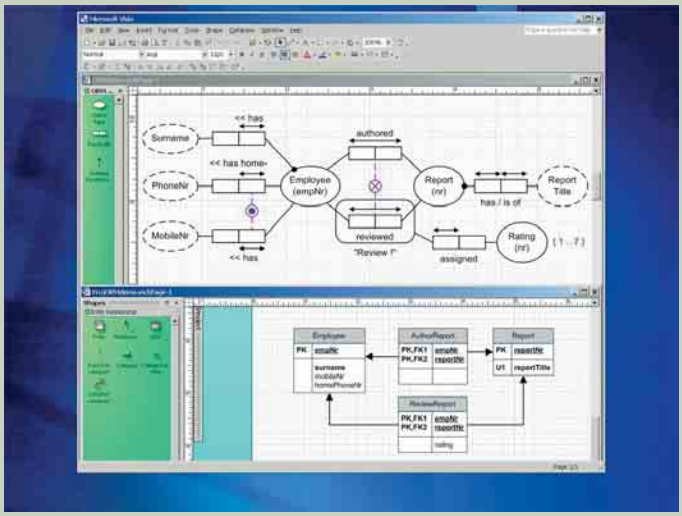

Figure 2. Microsoft Visual Studio .NET design tools for architects.

In Business, the Faster the Better

One of the main goals of application development is to get the best solution possible as quickly as you can. To do this, the development tools must address the needs of enterprise architects, system designers, application developers, and quality assurance testers.

Microsoft Visual Studio .NET, the flagship suite of tools for Microsoft .NET development, is hailed as the best development suite on the market. The product allows enterprises to define templates for consistency across all development projects; it allows architects to use graphical design tools to generate program documentation; it provides developers with a wide choice of languages and features; and it provides testers a rich debugging environment to monitor end-to-end program flow.

Simply put, .NET applications will see the light of day long before their J2EE brethren.

.NET solutions are simply less expensive to build, less expensive to deploy, and less expensive to maintain.

As compelling as Visual Studio .NET is, Microsoft .NET programs do not have to be built with that tool; other vendors such as Borland offer great development suites, and there are even shared source C# compilers available.

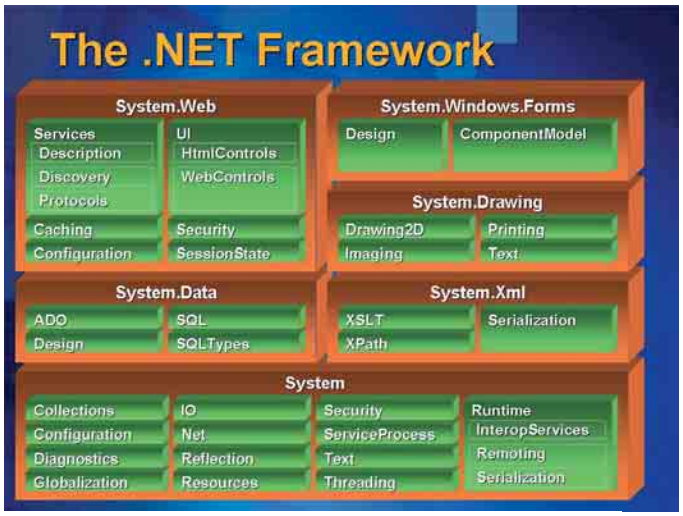

Figure 3. A sample of the .NET Framework

Microsoft .NET programs run inside the Common Language Runtime (CLR), just as J2EE programs run inside the Java Virtual Machine (JVM). The .NET Framework adds a rich library of functionality to the CLR, considerably stronger than the additional capabilities J2EE brings to the JVM. In J2EE, even simple tasks become difficult. For example, EJB performance problems cause most programmers to use bimodal data access, which requires them to write twice as much code.

Because the .NET Framework is so rich, programmers will typically need to write significantly fewer lines of code than they will with J2EE. For example, one of J2EE's most ardent supporters recently concluded that the most optimized Java Pet Store possible requires 14,004 lines of code under J2EE, but only 2,096 using the .NET Framework.

So, what does an 85% reduction in coding mean? It means the application is finished that much quicker. It also means a shorter QA cycle and a more stable and secure product, since bug counts grow with line counts.

Simply put, .NET applications will see the light of day long before their J2EE brethren.

Performance

Much has been made about .NET vs. J2EE performance, so I will briefly mention how much faster .NET solutions will tend to run. Still, overall performance is rarely a critical factor in enterprise systems. More important are figures such as how many users the system can support per server because this translates directly into deployment and maintenance cost.

Despite Sun's claims, The Middleware Company's (TMC) recent Java Pet Shop study is highly relevant to this discussion. TMC is a significant player in the J2EE world, deriving nearly all their revenue from Java and J2EE publishing and consulting. They also run TheServerSide.com, one of the most popular J2EE Web sites. Sun apparently thinks highly of TMC, touting them on the Web; java.sun.com/features/2001/09/server-side.html.

The Java Pet Shop study concluded that Microsoft .NET requires 1/5th the code as J2EE, supports 50-600% more users (depending on the application server), and offers nearly twice the transactions per second on a significantly less expensive platform—all with no errors, while both J2EE application servers studied threw exceptions, and one could not even complete the benchmark. This from a J2EE-biased company that even Sun regards as a definitive source of information!

Even Sun's own data from the JavaOne 2002 conference shows that 86% of J2EE users surveyed had performance concerns. It is telling, too, that there are no J2EE-based applications in the TPC-C benchmarks. For example, IBM's TPC-C submissions use either Microsoft COM+ or IBM's older, non-Java/non-J2EE transaction processing monitor formerly known as Encina.

The J2EE community often counters the performance argument by claiming that J2EE shops trade speed for portability. This is a red herring, as there is no real portability between J2EE application servers. Indeed, even Oracle's Web site admits, "Though in

theory, any J2EE application can be deployed on any J2EE-compliant application server, in practice, this is not strictly true." IBM's Web site contains a white paper over 200 pages in length explaining how to port a J2EE application from BEA WebLogic to WebSphere. The portability argument is truly nothing but fear, uncertainty, and doubt (FUD) and misdirection.

Simply put, .NET applications are faster and support more users than comparable J2EE solutions.

Cost of Solution

The total cost of a solution consists of how much money an enterprise spends to build and then deploy the application. In both cases, Microsoft .NET offers a significant advantage over J2EE.

When building a J2EE solution, programmers have a severely limited range of language choice—they can only use Java. On the other hand, the .NET Framework supports almost 30 languages. In fact, contrary to Sun's claims, Visual Studio .NET supports four languages out of the box (J#, a Java-syntax language, C#, C++, and Visual Basic .NET), giving each language identical support. While C# is a great language for .NET development, the .NET Framework is in no way optimized for C# only. In addition, other vendors offer a wide array of additional languages for .NET development, including JScript, Perl, COBOL, Fortran, Smalltalk, even Eiffel and Mondrian. Clearly, companies will generally not build solutions using, say, 10 languages at once. However, the choice to use whichever language makes sense for a particular project can generate significant savings, because companies can use existing programmers without retraining them for Java. In addition, Visual Basic programmers tend to be more plentiful and less expensive than Java programmers. And, we previously established that once developers begin writing code they will need to write much less code, translating again into significant savings.

Once the application is ready it must be deployed. In the .NET world, this means running it on an Intel-based server with Windows 2000 Server or the upcoming Windows Server 2003. According to TMC, with software costs running approximately $5,990, a fully configured server for the .NET solution will cost around $36,990. Contrast this with the same server running a commercial J2EE application server, which adds between $40,000 and $48,000 per server! Even if an enterprise chooses Linux to eliminate the $5,990 Windows license, it still must add the application server—even today's New Math can't justify eliminating less than $6,000 for an additional $48,000 cost.

Finally, because performance data indicates that J2EE solutions support fewer users than .NET solutions on comparable hardware, the J2EE solutions will generally require more of these more expensive servers.

.NET solutions are simply less expensive to build, less expensive to deploy, and less expensive to maintain.

Conclusion

Web services are clearly critical for the next wave of enterprise computing—integrating disparate business systems for more effective use of information. Companies like Microsoft and Sun Microsystems should be commended for their clear commitment to work together toward a common industry standard for all our customers' benefit.

There should always be more than one choice of development environment for customers, because there will never be one solution always appropriate for everyone in every situation. The world is big enough for both Microsoft .NET and J2EE. Still, for the reasons outlined herein, Microsoft .NET will generally be a better choice for most companies in most situations. C

Gerry Miller (gerrym@microsoft.com) is young enough to have attended a college rave, and old enough to enjoy Paul Anka. He is also Chief Technology Officer for Microsoft's U.S. Central Region.

The opinions expressed herein are the author's own and may not reflect those of his employer.