

J2EE vs. .NET XML WEB SERVICES

Arif Bramantoro

Social Informatics Department
 Kyoto University, Kyoto, Japan
 Tel: +81-75-7129412
 E-mail: bramantoro@gmail.com

Abstract— One of the latest innovations for the use of the Internet is Web services. Web services allow applications and Internet-enabled devices to easily communicate each other and combine their functionality to provide services to each other, independent of platform or language. Web services are characterized by SOAP messages used to talk to a Web service, WSDL files that describe a Web service, and the UDDI used to find Web services. The question then becomes, what is the right tools for developing Web services? That question is assessed in this paper through .NET and J2EE technology. A comparative study between these leading technologies is then presented from various sources based on several performance variables.

Category: C: Applied Sciences/Engineering

Keywords: J2EE, .NET, Web services

I. INTRODUCTION

The Web service emerged as the solution for supplying a standard way to retrieve data without proprietary software and hardware. A Web service cornerstone is its capability to transfer data from the provider to the consumer, or requester, using the ubiquitous HTTP protocol; the data format is XML. Formatting the data in XML greatly facilitates raw data's conversion into a renderable format. Extensible Stylesheet Language Transformations (XSLT) performs the transformation rather easily, without complex parsing programs, as Fig. 1 illustrates.

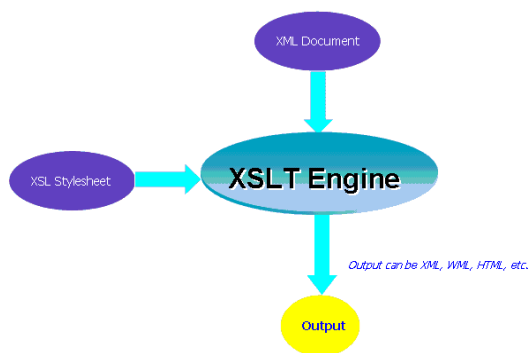


Fig. 1. A conceptual depiction of an XSL transformation

A commissioned whitepaper for Sun defines a Web service as: an application that accepts requests from other systems across the Internet or an intranet, mediated by lightweight, vendor-neutral communications technologies. In "Defining the Basic Elements of .Net," Microsoft describes a Web service as: XML Web services let

applications share data, and -- more powerfully -- invoke capabilities from other applications without regard to how those applications were built, what operating system or platform they run on, and what devices are used to access them.

It can be inferred from previous definitions that both Sun and Microsoft seem to tacitly agree. From a purely intuitive level, a Web service is a service consumed via the Internet. More specifically, a Web service is invoked via HTTP and returns data to the consumer in XML.

Web services signify a shift in software development paradigms. While many companies still debate Web services' viability, companies like Concord EFS reap millions of dollars from their in-production credit card-processing Web service. Web services advocate that we should segment large applications so individual components can exist as Web services. This model would arguably replace the current paradigm: segmentation into dynamic link libraries (DLL) and COM (Component Object Model) components (Wang, H. et al, 2004).

In fact, we can draw a strong analogy between Web services and DLLs. Both centralize related functionality; for example, business rules and database access. However, Web services and DLLs differ significantly. First, you can access Web services through the HTTP protocol, thus allowing any Web client to invoke Web services. This is not the case with DLLs, which are typically invoked from clients within the same intranet. Thus, Web services will guide in a new distributed computing era. Second, Web services return data to the client in XML. DLLs typically return data through programming language-specific data types (Wang, H. et al, 2004)

These differences between Web services and their DLL ancestors result from the major industry trends that preceded Web services:

1. The acceptance of HTTP as a protocol through which users access the Internet
2. The acceptance of XML as the de facto standard for data transfer

These two trends provide the foundations upon which the Web service is built. Fig. 2 illustrates how Know-Can-Do would use Web services. It can be noticed that the absence of leased lines and proprietary data formats.

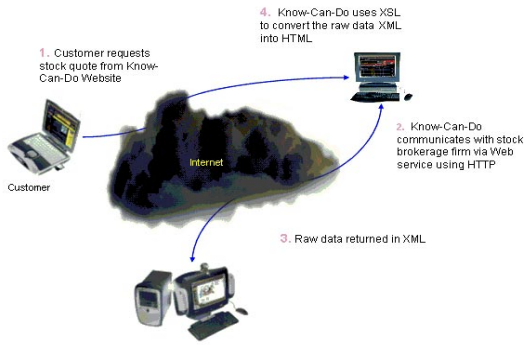


Fig. 2. Case example architected using a Web service approach

II. .NET TECHNOLOGY

Microsoft.NET is product suite that enables organizations to build smart, enterprise-class Web services. Microsoft.NET is largely a rewrite of Windows DNA, which was Microsoft's previous platform for developing enterprise applications. Windows DNA contains many current established technologies, such as Microsoft Transaction Server (MTS) and COM+, Microsoft Message Queue (MSMQ), and the Microsoft SQL Server database. The new .NET Framework replaces these technologies, and includes a Web services layer as well as improved language support. Fig. 3 shows the developer model for building Web services with Microsoft.NET (Miller G., 2003).

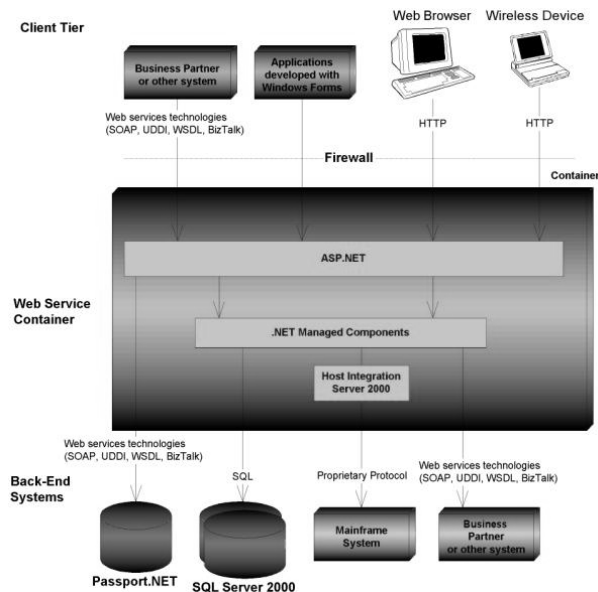


Fig. 3. Developing Web services with Microsoft.NET

The explanation for Fig. 3 is as follows:

- **The .NET application** is hosted within a container, which provides qualities of service necessary for enterprise applications, such as transactions, security, and messaging services.

- **The business layer** of the .NET application is built using .NET managed components. This layer performs business processing and data logic. It connects to databases using Active Data Objects (ADO.NET) and existing systems using services provided by Microsoft Host Integration Server 2000, such as the COM Transaction Integrator (COM TI). It can also connect to business partners using Web services technologies (SOAP, UDDI, WSDL).
- **Business partners** can connect with the .NET application through Web services technologies (SOAP, UDDI, WSDL, BizTalk).
- **Traditional 'thick' clients, Web browsers, wireless devices** connect to Active Server Pages (ASP.NET) which render user interfaces in HTML, XHTML, or WML. Heavyweight user interfaces are built using Windows Forms.

The .NET Framework

Microsoft.NET offers language-independence and language-interoperability. This is one of the most intriguing and fundamental aspects of the .NET platform. A single .NET component can be written, for example, partially in VB.NET, the .NET version of Visual Basic, and C#, Microsoft's new object-oriented programming language.

There are two general steps in .NET processing. Firstly, source code is translated into **Microsoft Intermediate Language**, sometimes abbreviated **MSIL**, sometimes **IL**. This IL code is language-neutral, and is analogous to Java bytecode. Secondly, the IL code then needs to be interpreted and translated into a native executable. The .NET Framework includes the **Common Language Runtime (CLR)**, analogous to the Java Runtime Environment (JRE), which achieves this goal. The CLR is Microsoft's intermediary between .NET developers' source code and the underlying hardware, and all .NET code ultimately runs within the CLR (Murray M., 2003).

This CLR provides many exciting features not available in earlier versions of Windows DNA, such as automatic garbage collection, exception handling, cross-language inheritance, debugging, and "side-by-side" execution of different versions of the same .NET component. A detail framework of .NET technology can be seen in Fig. 4.

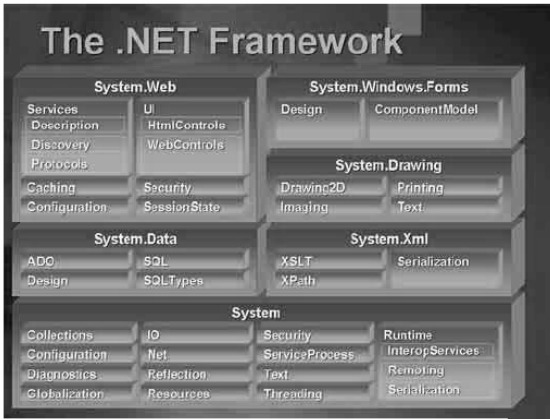


Fig. 4. The .NET framework

III. J2EE TECHNOLOGY

The Java 2 Platform, Enterprise Edition (J2EE) was designed to simplify complex problems with the development, deployment, and management of multi-tier enterprise solutions. J2EE is an industry standard, and is the result of a large industry initiative led by Sun Microsystems.

It is important to note that J2EE is a standard, not a product. J2EE cannot be "downloaded", but only a set of Adobe Acrobat PDF files which describe agreements between applications and the containers in which they run. So long as both sides obey the J2EE contracts, applications can be deployed in a variety of container environments. Thus, the fundamental difference is: .NET is a product strategy, whereas J2EE is a standard to which products are written (Williams J., 2003).

The J2EE camp's goal is to give customers choice of vendor products and tools, and to encourage best-of-breed products to emerge through competition. The only way this would ever happen is if the industry as a whole were bought-into J2EE. To secure buy-in, Sun collaborated with other vendors of eBusiness platforms, such as BEA, IBM, and Oracle, in defining J2EE. Sun then initiated the Java Community Process (JCP) to solicit new ideas to improve J2EE over time. The reason Sun did this is because they had to do so to achieve success--the best way to secure buy-in to an idea is to involve others in defining that idea (Williams J., 2003).

The J2EE architecture is based on the Java programming language. The interesting thing about Java is that it enables organizations to write their code once, and deploy that code onto any platform. The process is as follows:

1. Developers write source code in Java.
2. The Java code is compiled into bytecode, which is a cross-platform intermediary, halfway between source code and machine language.
3. When the code is ready to run, the Java Runtime Environment (JRE) interprets this bytecode and executes it at run-time.

J2EE is an application of Java. J2EE components are transformed into bytecode and executed by a JRE at runtime. Even the containers are typically written in Java.

J2EE and Web Services

J2EE has historically been architecture for building server-side deployments in the Java programming language. It can be used to build traditional Web sites, software components, or packaged applications. J2EE has recently been extended to include support for building XML-based Web services as well. These Web services can interoperate with other Web services that may or may not have been written to the J2EE standard. J2EE Web services development model is shown in Fig. 5.

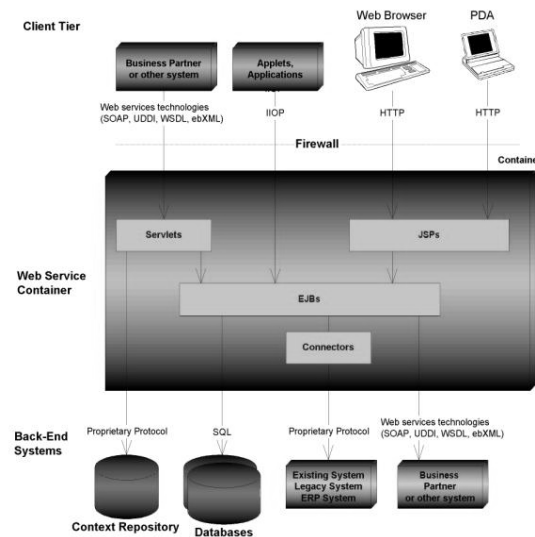


Fig. 5. Developing Web services with J2EE

The explanation for figure 5 is as follows:

J2EE application is hosted within a container, which provides qualities of service necessary for enterprise applications, such as transactions, security, and persistence services.

The business layer performs business processing and data logic. In large-scale J2EE applications, business logic is built using Enterprise JavaBeans (EJB) components. This layer performs business processing and data logic. It connects to databases using Java Database Connectivity (JDBC) or SQL/J, or existing systems using the Java Connector Architecture (JCA). It can also connect to business partners using Web services technologies (SOAP, UDDI, WSDL, ebXML) through the Java APIs for XML (the JAX APIs).

Business partners can connect with J2EE applications through Web services technologies (SOAP, UDDI, WSDL, ebXML). A servlet, which is a request/response oriented Java object, can accept Web service requests from business partners. The servlet uses the JAX APIs to perform Web

services operations. Shared context services will be standardized in the future through shared context standards that will be included with J2EE.

Traditional 'thick' clients such as applets or applications connect directly to the EJB layer through the Internet Inter-ORB Protocol (IIOP) rather than Web services, since generally the thick clients are written by the same organization that authored J2EE application, and therefore there is no need for XML-based Web service collaboration.

Web browsers and wireless devices connect to JavaServer Pages (JSPs) which render user interfaces in HTML, XHTML, or WML.

In addition to the specifications, Sun also ships a reference implementation of J2EE. Developers write applications to this to ensure portability of their components. This implementation should not be used for production, but rather just for testing purposes.

All vendors that offer J2EE platforms provide additional features not found in the standard. Some of them impact portability, such as extended EAI functionality, E-Commerce components, or advanced B2B integration. Other features, such as load-balancing, transparent fail-over, and caching, do not affect portability of application code, because they are implicit services which are provided behind-the-scenes by the container (Williams J., 2003).

IV. THE COMPARISON

To have a better understanding of both models, table 1 shows analogies between J2EE and .NET technologies.

Table 1 Analogies between J2EE and .NET

Feature	J2EE	.NET
Type of technology	Standard	Product
Middleware Vendors	30+	Microsoft
Interpreter	JRE	CLR
Dynamic Web Pages	JSP	ASP.NET
Middle-Tier Components	EJB	.NET Managed Components
Database access	JDBC SQL/J	ADO.NET
SOAP, WSDL, UDDI	Yes	Yes
Implicit middleware	Yes	Yes



As can be seen from table 1, the .NET platform has an array of technologies under its umbrella. Microsoft is ostensibly presenting these as alternatives to other existing platforms, like J2EE and CORBA, in order to attract developers to the Windows platform. On the other hand, the comparison between .NET and J2EE is shown in the following table:

Table 2 The differences between J2EE and .NET

Microsoft .NET	J2EE	Key differentiators
C# programming language	Java programming language	C# and Java both derive from C and C++. Most significant features (e.g., garbage collection, hierarchical namespaces) are present in both. C# borrows some of the component concepts from JavaBeans (properties/attributes, events, etc.), adds some of its own (like metadata tags), but incorporates these features into the syntax differently. Java runs on any platform with a Java VM. C# only runs in Windows for the foreseeable future. C# is implicitly tied into the IL common language runtime (see below), and is run as just-in-time (JIT) compiled bytecodes or compiled entirely into native code. Java code runs as Java Virtual Machine (VT) bytecodes that are either interpreted in the VM or JIT compiled, or can be compiled entirely into native code.
.NET common components (aka the ".NET Framework SDK")	Java core API	High-level .NET components will include support for distributed access using XML and SOAP (see ADO+ below).
Active Server Pages+ (ASP+)	Java ServerPages (JSP)	ASP+ will use Visual Basic, C#, and possibly other languages for code snippets. All get compiled into native code through the common language runtime (as opposed to being interpreted each time, like ASPs). JSPs use Java code (snippets, or JavaBean references), compiled into Java bytecodes (either on-demand or batch-compiled, depending on the JSP implementation).
IL Common Language Runtime	Java Virtual Machine and CORBA IDL and ORB	.NET common language runtime allows code in multiple languages to use a shared set of components, on Windows. Underlies nearly all

		of .NET framework (common components, ASP+, etc.). Java's Virtual Machine spec allows Java bytecodes to run on any platform with a compliant JVM. CORBA allows code in multiple languages to use a shared set of objects, on any platform with an ORB available. Not nearly as tightly integrated into J2EE framework.
Win Forms and Web Forms	Java Swing	Similar Web components (e.g., based on JSP) not available in Java standard platform, some proprietary components available through Java IDEs, etc. Win Forms and Web Forms RAD development supported through the MS Visual Studio IDE - no other IDE support announced at this writing. Swing support available in many Java IDEs and tools.
ADO+ and SOAP-based Web Services	JDBC, EJB, JMS and Java XML Libraries (XML4J, JAXP)	ADO+ is built on the premise of XML data interchange (between remote data objects and layers of multi-tier apps) on top of HTTP (AKA, SOAP). .NET's Web services in general assume SOAP messaging models. EJB, JDBC, etc. leave the data interchange protocol at the developer's discretion, and operate on top of either HTTP, RMI/JRMP or IIOP.

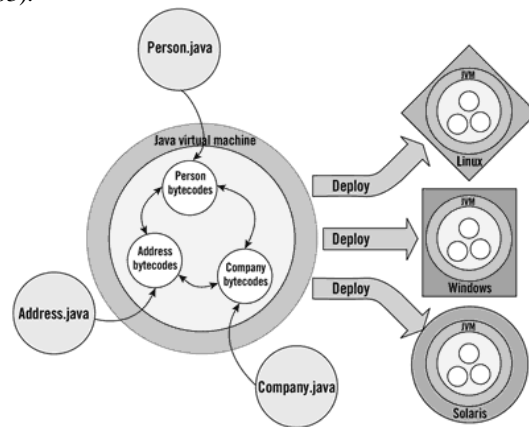
Features in .NET and J2EE offer pretty much the same laundry of list of features, though in different ways. In portability aspect, the .NET core works on Windows only but theoretically supports development in many languages (once sub-/supersets of these languages have been defined and IL compilers have been created for them). Also, Net's SOAP capabilities will allow components on other platforms to exchange data messages with .NET components (Murray M., 2003).

While a few of the elements in .NET, such as SOAP and its discovery and lookup protocols, are provided as public specifications, the core components of the framework (IL runtime environment, ASP+ internals, Win Forms and Web Forms component "contracts", etc.) are kept by Microsoft, and Microsoft will be the only company to provide complete .NET development and runtime environments. There has already been some pressure by the development community for Microsoft to open up these specifications, but this would be counter to Microsoft's standard practices. However, Miller G. (2003) argues that a core part control of .NET (namely CLI) is officially given to the European Computer Manufacturing Association (ECMA) as a true standards body.

J2EE, on the other hand, works on any platform with a compliant Java VM and a compliant set of required platform services (EJB container, JMS service, etc.). All of the specifications that define the J2EE platform are published and reviewed publicly, and numerous vendors offer

compliant products and development environments. But J2EE is a single-language platform. Calls from/to objects in other languages are possible through CORBA, but CORBA support is not a ubiquitous part of the platform. The differences between J2EE and .NET platform can be seen in Fig. 6.

It can be seen clearly from Fig. 6 that .NET is dedicated to Microsoft-centric approach to Web services. .NET operates on a single platform, namely Windows, although it really natively supports multiple languages, which are suspected as a value-added purchase for the .NET development platform and C# is still the chosen language. On the other hand, J2EE is a platform independent solution deployed in a single language, namely java (Williams J., 2003).



Java/J2EE Development Model

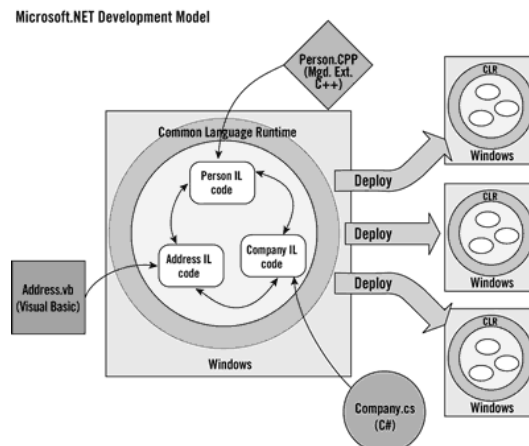


Fig. 6. Comparing Development Models

Web Services Support

The future of eBusiness collaboration is undoubtedly Web services. For organizations that are pursuing a Web services strategy, or are preparing for the future of Web services, their underlying eBusiness architecture must have strong Web services support.

Today, J2EE supports Web services through the Java API for XML Parsing (JAXP). This API allows developers to perform any Web service operation today through manually parsing XML documents. For example, developers can use JAXP to perform operations with SOAP, UDDI, WSDL, and ebXML.

Additional APIs are also under development. These are convenience APIs to help developers perform Web services operations more rapidly, such as connecting to business registries, transforming XML-to-Java and Java-to-XML, parsing WSDL documents, and performing messaging such as with ebXML. However, it is useful to examine the comparison between .NET petshop and Java Pet Store in the lines of code required (Fig. 7).

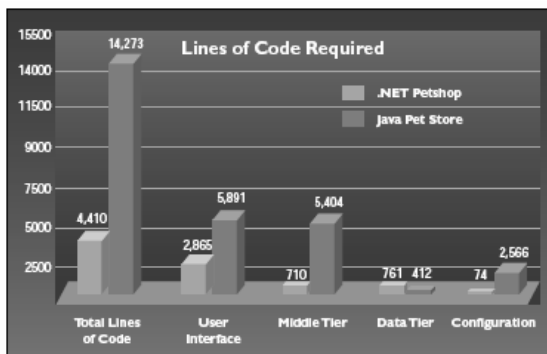


Fig. 7. J2EE vs .NET based on lines of code required

A variety of J2EE-compatible 3rd party tools are available today that enable rapid development of Web services. There are at least sixteen SOAP implementations that support Java. Almost all of these implementations are built on J2EE (servlets or JSP). There are only five UDDI API implementations available, and four of them support Java (IBM UDDI4J, Bowstreet jUDDI, The Mind Electric GLUE, and Idoox WASP). Third-party software vendors such as Tradia (www.tradia.com), CapeClear (www.capeclear.com) and The Mind Electric (www.themindelectric.com) also offer tools for creating Web services (Williams J., 2003).

The preview release of Microsoft.NET also enables organizations to build Web services. The tools that ship with Microsoft.NET also offer rapid application development of Web services, with automatic generation of Web service wrappers to existing systems. Developers can perform operations using SOAP, UDDI, and SDL (the precursor to WSDL). Visual Studio.NET provides wizards that generate Web services.

Performance comparison

All the comparisons presented in this section between J2EE and .NET XML Web Services are taken from various sources. As a performance variables, the benchmarking providers use the throughput and response time. Throughput is an average number of Web service operations executed

per seconds. Response time is an average time taken to process a request.

Since Web Services have been extensively promoted as the new frontier of application servers, The Middleware Company Application Server Platform Baseline Specification includes a description of a Web Service that must be implemented by codebases. The Web Service itself is specified such that it would provide a realistic test case to illustrate application server performance inclusive of base object activation over SOAP, as well as serialization of simple and complex data types/objects into XML. The functionality of the Web Service in the Specification is to take as a single input parameter an OrderId, and then to perform a database lookup of that order, returning an Order object as XML to the calling application. The Order object itself contains both simple data types (string, integers, and decimals) as well as an array (representing individual order detail lineitems) (The Middleware Company Case Study Team, 2003).

In this test, clients send a SOAP request for a random order, of 10,000 possible orders. Each order contained five repeating line items in the returned order object. This test puts high stress on a single application server hosting the Web Service, and is designed to illustrate how well the application server performs as a Web Service host, servicing SOAP requests from thousands of concurrent users (The Middleware Company Case Study Team, 2003).

The database used for each codebase was the one it appeared to perform most reliably and conveniently with, in the Web Application Test, and other informal tests. For both of the J2EE codebases, the Web Services Test was run with Oracle 9i. For the .NET codebase the Web Services Test the chosen database was Microsoft SQL Server 2000; The charts below, Fig. 8 and Fig. 9, show the results of testing the Web service test scripts with increasing user loads for the five codebases against their selected database, and the maximum throughput they achieved.

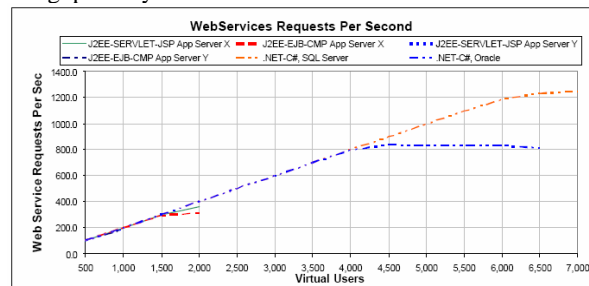


Fig. 8. Showing the throughput, Web service requests per second, increase as user load increase for the various configurations.

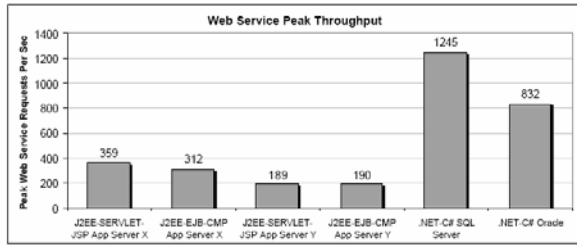


Fig. 9. The maximum throughput achieved by each configuration during the Web service tests.

The results of this test showed that the .NET platform outperformed the fastest J2EE platform, by over 200%. The complete numerical result for this test can be seen in the paper published by The Middleware Company (2002). Furthermore, the previous reports about benchmark test result between Java Pet Store and .NET Petshop implemented together with XML Web service is also presented.

Another comparative study based on the Doculabs-developed Nile @Bench benchmark shows similar results (Fig. 10). Specifically, it discusses the results of benchmarking the Nile application as implemented in Microsoft .NET using C# vs. the latest version of a leading J2EE-based application server product from a top-tier ISV. It also presents detailed performance information comparing the performance of the .NET/C# Nile application to versions implemented in Microsoft Active Server Pages with Visual Basic 6.0 COM+ components, and Microsoft ISAPI using Visual C++ 7.0 ATL Server technology.

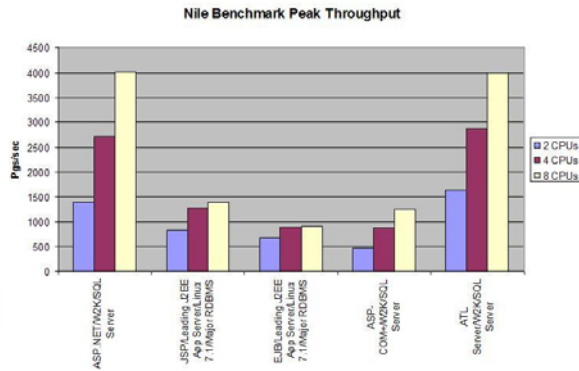


Fig. 10. Benchmark Peak Throughput conducted by Nile

The benchmark shows that the Nile application implemented using Microsoft ASP.NET outperforms the same application implemented using EJBs on a leading J2EE application server by 345% on an 8 CPU system when output caching is enabled for both products. It also shows that the Microsoft.NET version of Nile outperforms the EJB version of Nile on an 8 CPU system by over 421% when output caching is not used (Doculabs-developed Nile @Bench benchmark, 2001).

The newer benchmark test conducted by Sun Microsystems Inc. on June 2004 shows a contrast result.

They consider basic Web service method that is called with different types and sizes of arguments in both platforms and compare its performance. They use WSTest, a Web services test developed at Sun, which is designed to evaluate the performance of Web services based on various types of calls, such as:

- echoVoid – Sends and receives an empty message. This tests the performance of the Web services infrastructure.
- echoStruct – Sends and receives an array of size 20 – each element is a structure composed of one element each of an integer, float and string data type.
- echoList – Sends and receives a linked list of 20 elements – each element is a Struct as defined in echoStruct.
- echoSynthetic – Sends and receives multiple parameters of different types –String, Struct and a byte array of size 1K.

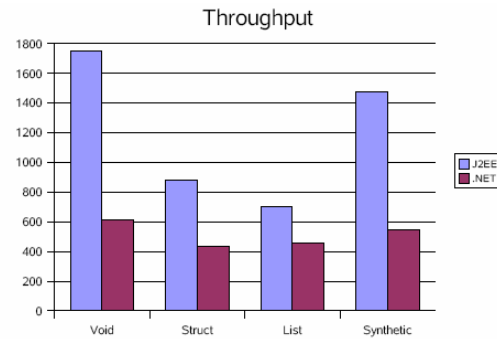


Fig. 11. Benchmark Throughput conducted by Sun

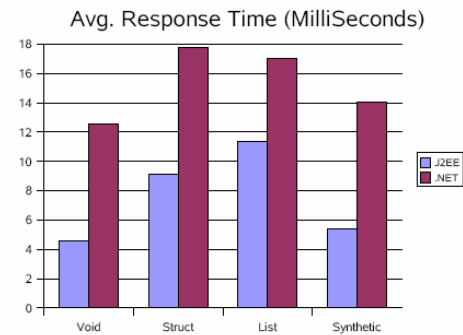


Fig. 12. Benchmark Response Time conducted by Sun

The result of Sun WSTest can be seen in Fig. 11 and 12, which measures throughput and response times. In the basic Web services call, echoVoid, and the most complex one, echoSynthetic, JAX-RPC performs almost 3 times better than .NET. In the other cases, J2EE technology performs almost twice as well as .NET. Moreover, developers consider this supreme performance on the Linux and Solaris platforms as well, because the J2EE platform is entirely portable (Sun Microsystems Inc., 2004).

Like a soap opera, Microsoft published a counter response of Sun WSTest. They replicate the test conducted by Sun in order to prove that Sun publication is not completely right. The result, reported by Microsoft in July 2004, demonstrates the similar result for Java Application. However, the .NET results are revealed to be two or three times better than Sun publication. In Microsoft test, as a whole .NET matched or slightly outweigh J2EE performance. In addition, they argue that in the further test with higher Web service message payloads, it is proved that .NET significantly smashes Java (Microsoft corp., 2004). As an example, it can be seen from Fig. 13 that for echoList test with the listsize of 200, .NET performed 82% better than Java. A complete benchmark test result from Microsoft can be examined in (Microsoft Corp., 2004).

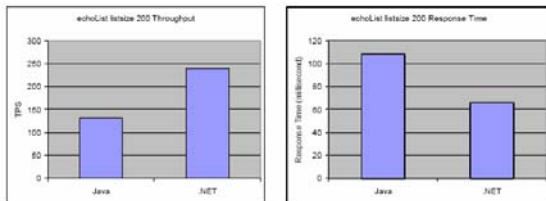


Fig. 13. echoList test conducted by Microsoft

V. CONCLUSION

When navigating the complexities of the .NET vs. J2EE debate, it is easy to get caught up in what seems like the entrenched religious turf war between competing software vendors expounding the virtues of their chosen platforms. The key to cutting through this discussion is to prioritize above all else, business requirements and the functional nature of technology solutions to meet those requirements.

The .NET vs. J2EE question is like choosing a car based on one half of the color spectrum rather than the other. Choosing J2EE up front and then working out how to use it to address user requirements would be like saying users want a purple car, then seeing which manufacturers produce cars in purple. Color may be important to some, but most rational people would agree that performance, reliability, safety, after-sales service and value-for-money are somewhat more important considerations. In the wider scheme of things, then, this debate does not really matter. Both approaches are based on the same technical foundation of Web services. Both are supported by an array of vendors. And both are here to stay.

REFERENCES

- Box, D. et al. (2000). *Simple Object Access Protocol (SOAP) 1.1*. Retrieved 11 August 2004 from <http://www.w3.org/TR/SOAP/>
- Cerami, E. (2002). *Top ten FAQs for Web Services*. Retrieved 11 August 2004 from

<http://www.oreillynet.com/lpt/a/Webservices/2002/02/12/Wobservicefaqs.html>

Doculabs-developed Nile @Bench benchmark (2001), *Microsoft® .NET vs. Sun® Microsystem's J2EETM: The Nile Ecommerce Application Server Benchmark*, October 2001.

Guo, Z., Wang, X., Sun, G. (2003). *Research and Application on Spatial Data Web Service Based on .Net Platform*. IEEE, 2003.

Markatos, D. (November 19, 2002). *Creating and Consuming .NET Web Services in Five Easy Steps*. Retrieved 11 August 2004 from <http://www.dotnetjunkies.com/>

Meyer, B. (2001). *.NET Is Coming*. IEEE, August 2001.

Microsoft Corp. (2004), *Web Services Performance: Comparing Java 2™ Enterprise Edition (J2EETM platform) and the Microsoft® .NET Framework, A Response to Sun Microsystem's Benchmark*, July 2004

Miller, G. (2003). *E-services: The Web services debate, .NET vs. J2EE*. ACM, Vol 46, No. 6, June 2003.

Murray, M. (2003), *Move to Component Based Architecture: Introducing Microsoft .NET Platform into The Collage Classroom*, CCSC, 2003.

Paolucci, M., Sycara, K. (2003). *Autonomous Semantic Web Service*. Carnegie Mellon University IEEE INTERNET COMPUTING, Sept 2003.

Sun Microsystems Inc., (2004), *Web Services Performance Comparing Java™ 2 Enterprise Edition (J2EE™ platform) and .NET Framework*, June 2004

The Middleware Company (2002), *J2EE and .NET Application Server and Web Services Benchmark*, October 2002.

The Middleware Company Case Study Team (2003), *J2EE and .NET (RELOADED) Yet Another Performance Case Study*, June 2003

Wang, H. et al (2004), *Web services: problems and future directions*. Journal of Web Semantics, pp. 309-320, 2004.

Williams, J. (2003). *E-services: The Web services debate, J2EE vs. .NET*. ACM, Vol 46, No. 6, June 2003.