J2EE vs. Microsoft .NET

Part I  A Brief History

Java™ was announced to the world on May 23, 1995, at Sun Microsystems' SunWorld conference. Sun executive John Gage, joined by Marc Andreessen, cofounder and executive vice president of Netscape Communications, announced that Java would be incorporated into the Netscape Navigator browser.

Java was based on the idea that the same software could run on many different kinds of computers, consumer gadgets, and other devices on a network. The big promise of Java was that it would allow applications to run on different types of computers without having to be rewritten. Java applications would run on "virtual machines" that were written for each platform.

The combination of Netscape and Java would be a powerful new set of technologies. Netscape browsers could run on many different types of computers to enable users to access World Wide Web sites, and Java virtual machines would run in the browsers to enable different types of computers to access and run Java applications.

Java was originally developed as an operating system for cable TV devices, but was re-purposed as an Internet technology after the explosive rise of the World Wide Web. Said Java creator James Gosling: "All the stuff we had wanted to do, in generalities, fit perfectly with the way applications were written, delivered, and used on the Internet. It was just an incredible accident. And it was patently obvious that the Internet and Java were a match made in heaven. So that's what we did."2

Gosling described the "Oak" language that eventually became Java as "a new small, safe, secure, distributed, robust, interpreted, garbage collected, multi-threaded, architecture neutral, high-performance, dynamic programming language."3

By running on a software layer above the operating system, Java would make applications portable across many platforms. This promise of "Write Once, Run Anywhere" had enormous value for software developers who previously had to write different versions of the same application to run on various operating systems (Unix, Windows, Macintosh, etc.).

---

1 The original system was an entertainment device controller called Star 7, developed by a Sun development group called the "Green Team." The controller was able to control a wide range of entertainment platforms and appliances while displaying animation. It ran on a new processor-independent language created by Green Team member James Gosling, called "Oak," named after the tree outside his window.


3 Ibid.
At the same time, Java and Netscape comprised a new set of "middleware" that threatened Microsoft's monopoly on the desktop. By freeing applications from any single underlying platform, middleware eliminated the need to write applications directly to Microsoft Windows. Java threatened to loosen Microsoft's grip on the application programming interfaces (APIs) through which it controlled development on the Windows platform. This threat would ultimately lead to two antitrust trials against Microsoft.

**Microsoft's War on Java**

Microsoft licensed Java from Sun in March 1996 and created a Java virtual machine to run on the Windows platform, as well as a Java development tool called Visual J++. However, as two U.S. district courts and an appeals court determined, Microsoft's Java implementation was a nonstandard version aimed at thwarting the success of Java.

In March 1997, Sun filed a $1 billion antitrust lawsuit against Microsoft in California. The suit claimed that Microsoft violated federal copyright laws and broke two California laws: trade restraint and unfair competition. In November 1998, U.S. District Court judge Ronald Whyte ruled that Sun was likely to prevail on the merits of its licensing case against Microsoft, and he granted Sun's request for a preliminary injunction.

The preliminary injunction required Microsoft to modify all its software products that shipped with Java technologies to pass Sun's Java compatibility test suite. Microsoft settled the suit with Sun for $20 million in January 2001. In addition, the license agreement between Sun and Microsoft was terminated, and Microsoft was banned from the unauthorized use of the "Java Compatible" trademark. Microsoft could continue to use an outdated version of Java within its products for seven years.

A week after Judge Whyte's ruling that Sun was likely to prevail in its suit, the U.S. Department of Justice and 20 U.S. states filed a separate antitrust suit against Microsoft in a U.S. District Court in Washington, D.C. In a decision issued in April 2000, Judge Thomas Penfield Jackson found that Microsoft "maintained its monopoly power by anticompetitive means and attempted to monopolize the Web browser market."  

In June 2001, The U.S. Court of Appeals for the D.C. Circuit affirmed certain of Judge Jackson's findings that Microsoft had "behaved anticompetitively," and that "these actions contributed to the maintenance of its monopoly power."

In its "Findings of Fact" released in November 1999, the U.S. District Court for the District of Columbia found that "Microsoft was apprehensive that the APIs exposed by middleware technologies would attract so much developer interest, and would become so numerous and

---

4 Conclusions of Law, United States District Court for the District of Columbia v. Microsoft Corporation, Civil Action No. 98-1232 (TPJ).
varied, that there would arise a substantial and growing number of full-featured applications that relied largely, or even wholly, on middleware APIs."

The court found that "Microsoft focused its antipathy on two incarnations of middleware that, working together, had the potential to weaken the applications barrier severely without the assistance of any other middleware. These were Netscape’s Web browser and Sun’s implementation of the Java technologies."7

In his conclusions of law for the U.S. antitrust trial, Judge Jackson wrote: "As part of its grand strategy to protect the applications barrier, Microsoft employed an array of tactics designed to maximize the difficulty with which applications written in Java could be ported from Windows to other platforms, and vice versa. The first of these measures was the creation of a Java implementation for Windows that undermined portability and was incompatible with other implementations."8

Moreover, the court found, "Microsoft also deliberately designed its Java development tools so that developers who were opting for portability over performance would nevertheless unwittingly write Java applications that would run only on Windows."9

The U.S. Court of Appeals for the D.C. Circuit found that, "Microsoft documents confirm that Microsoft intended to deceive Java developers, and predicted that the effect of its actions would be to generate Windows-dependent Java applications that their developers believed would be cross-platform; these documents also indicate that Microsoft's ultimate objective was to thwart Java's threat to Microsoft's monopoly in the market for operating systems. One Microsoft document, for example, states as a strategic goal: ‘“Kill cross-platform Java by grow[ing] the polluted Java market.”’10

The appeals court also found that "Microsoft undertook a number of anticompetitive actions that seriously reduced the distribution of Netscape Navigator." The ruling found that, "Because Microsoft's agreements foreclosed a substantial portion of the field for JVM distribution and because, in so doing, they protected Microsoft's monopoly from a middleware threat, they are anticompetitive."11

The appeals court affirmed that Microsoft threatened Intel to prevent Intel from building a virtual machine to support standard Java. In its ruling, the court found that, "Microsoft does not deny the facts found by the District Court, nor does it offer any procompetitive justification for pressuring

---

7 Ibid.
8 Conclusions of Law, United States District Court for the District of Columbia v. Microsoft Corporation, Civil Action No. 98-1232 (TPJ).
9 Ibid.
10 Conclusions of Law, United States District Court for the District of Columbia v. Microsoft Corporation, Civil Action No. 98-1232 (TPJ).
11 Ibid.
Intel not to support cross-platform Java. Microsoft lamely characterizes its threat to Intel as 'advice.' The District Court, however, found that Microsoft's 'advice' to Intel to stop aiding cross-platform Java was backed by the threat of retaliation, and this conclusion is supported by the evidence cited above. Therefore we affirm the conclusion that Microsoft's threats to Intel were exclusionary, in violation of § 2 of the Sherman Act.

C# and .NET Are Born
While engaged in these legal struggles over Java, reports began circulating that Microsoft was developing a Java-like language as an alternative to Java. In January 2000, Microsoft announced its Next-Generation Windows Services (NGWS) strategy centered on XML Web services. In June 2000, Microsoft renamed the strategy Microsoft .NET and announced a new programming language called C#.

Said Gartner Group: "C# was first known publicly in March 1999 as a project code-named “C++ Object-Oriented Language” (COOL). At the time, Microsoft was positioning COOL as a next-generation C++ language that would provide Java-like features, such as dynamic architecture, garbage collection and security, via an extended implementation of the C++ language that was tightly integrated with Microsoft’s own technology. Since most programming languages are similar, whether one is derived from C++, Java or both is difficult to prove. C#, announced as part of .NET, is essentially COOL.”

Said eWeek: "Microsoft originally developed C# under the code name "Cool," and prior to release of the product, claimed that Cool was merely a better version of Microsoft's C++ language. Privately, Microsoft told developers that Cool was meant to be Microsoft's answer to Sun's Java.”

"C# is Java by another name," said Steve Mills, general manager of IBM's software division. "Microsoft has its own unique programming model with Visual Basic. But it's not designed to be a scalable, multi-user system like Java, and C# is the alternative to Java.”

Java Essentials

The term “Java” refers to four main elements:
• A Java programming language with which developers can write applications.
• A set of programs written in Java that expose APIs on which developers writing in Java can rely. These programs are called the “Java class libraries.”
• The Java compiler, which translates the code written by the developer into Java “bytecode.”

Programs called Java Virtual Machines (JVMs), which translate Java bytecode into instructions comprehensible to the underlying operating system. If the Java class libraries and a JVM are present on a computer, the machine is said to carry a Java runtime environment.

The original Java platform was extended into specialized forms for enterprise applications and small devices. The current Java 2 platform has three versions:

- J2ME (Micro Edition) for handheld devices.
- J2SE (Standard Edition) for all major operating systems.
- J2EE (Enterprise Edition) for Web application servers.

All three versions share a common Java language specification, and there are no hard boundaries between editions. J2ME makes some standard J2SE APIs optional, and J2EE adds enterprise APIs to the J2SE base. J2EE is a component-based environment for developing and running multi-tier business applications. IBM's WebSphere platform is based on the J2EE standard.

Part II  A J2EE vs. .NET Comparison

J2EE and .NET Fundamentals

J2EE and Microsoft .NET aim to simplify the development of Web applications by providing a set of standardized, modular components and services. By having standard components and services at their disposal, developers can concentrate on business needs ("business logic") rather than programming more fundamental services. Using visual tools that simplify application development, this type of programming is often called Rapid Application Development (RAD).

J2EE and .NET each were created to enable programmers to build "tiered" Web applications. A three-tiered application consists of a client tier, server tier, and database tier. In a typical application, a user will click on a link on a Web page and a request will be transmitted to a Web application server. In response to the request, the application server will access data from a backend database, create a Web page, and present the page to the client. J2EE and .NET provide services to facilitate Web-based transactions in which business processes can be triggered and data recorded in databases. Automated workflow and human workflow may be part of the Web application.

Web sites that provide flight information are an example of a three-tier Web application. The request from the client is processed by the application server; the requested flight information is fetched from a backend database; a Web page is created; and the page is transmitted to the client and presented to the user. Another example is an e-commerce Web site in which merchandise is presented, shopping baskets filled, and orders taken and fulfilled.

Multi-Platform vs. Windows-Only

A key difference between Java and .NET is that J2EE is an open standard that runs on multiple platforms, while .NET is Microsoft proprietary and runs only on Windows. J2EE products are
available from numerous vendors, while .NET technology is available only from Microsoft. It is important to note that J2EE is not a product, rather a standard to which products are written.

Another difference between J2EE and .NET is that J2EE is written in a single language, Java, while .NET supports multiple languages—the chief ones being Microsoft's Visual Basic.NET, C++, .NET, and C#. Other languages can be supported by .NET if they are rewritten to run in the .NET environment.

J2EE and .NET each provide libraries of components. The J2EE component library comprises the Java core API, which includes Enterprise JavaBeans. .NET provides class libraries that contain .NET Managed Components. .NET also utilizes the previous generation of COM+ serviced components in the Windows operating system.

In the J2EE model, particular operations of each tier take place in separate "containers." A container is a piece of the runtime that handles tasks in a consistent manner—such as retrieving data and creating a Web page. .NET does not have containers, rather there are multiple ways in which operations can be coded or handled.

In the .NET Framework, "managed code" is all code that is executed by the .NET Common Language Runtime. Managed code also can access and interoperate with unmanaged code.

J2EE and .NET each have a set of components, services, and features that provide a standard way of performing tasks such as accessing databases, scripting Web pages, handling messages, and connecting to remote resources. A comparison of the key J2EE and .NET features and services are summarized in the table below.

<table>
<thead>
<tr>
<th>Service or Feature</th>
<th>Microsoft .NET</th>
<th>J2EE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>C#, VB.NET, C++.NET, other modified languages</td>
<td>Java</td>
</tr>
<tr>
<td>Operating System</td>
<td>Windows</td>
<td>Multiple</td>
</tr>
<tr>
<td>Runtime</td>
<td>CLR</td>
<td>JVM</td>
</tr>
<tr>
<td>Server Components</td>
<td>.NET, COM+ serviced</td>
<td>EJBs</td>
</tr>
<tr>
<td>Client/GUI Components</td>
<td>.NET class</td>
<td>JavaBeans</td>
</tr>
<tr>
<td>Web Server Scripting</td>
<td>ASP.NET</td>
<td>JSP/Servlet</td>
</tr>
<tr>
<td>Data Access</td>
<td>ADO.NET</td>
<td>JDBC</td>
</tr>
<tr>
<td>Persistent Objects</td>
<td>Business Entity Comp.</td>
<td>EJB Entity Beans</td>
</tr>
<tr>
<td>Message Queueing</td>
<td>Sys. Messaging, MSMQ</td>
<td>JMS on (MQSeries, etc.)</td>
</tr>
<tr>
<td>Asynchronous Invocation</td>
<td>COM+ QC</td>
<td>EJB Message Beans</td>
</tr>
<tr>
<td>Remoting</td>
<td>SOAP, HTTP, DCOM</td>
<td>RMI-over-IIOP</td>
</tr>
<tr>
<td>Naming</td>
<td>ADSI</td>
<td>JNDI</td>
</tr>
<tr>
<td>XML</td>
<td>System XML</td>
<td>JAXP</td>
</tr>
<tr>
<td>HTTP Engine</td>
<td>IIS</td>
<td>Application Servers from Multiple Vendors</td>
</tr>
<tr>
<td>Web Services Support</td>
<td>Built-In</td>
<td>Add-On</td>
</tr>
</tbody>
</table>
J2EE and .NET are each based on a virtual machine runtime that sits above an operating system. J2EE uses a Java Virtual Machine (JVM) built to run Java on each individual platform—Unix, Windows, Macintosh, etc. .NET uses a Common Language Runtime (CLR) virtual machine that runs on Windows machines. To support multiple languages, the .NET CLR contains an Intermediate Language (IL) engine. Code and objects written in a language can be compiled into the IL runtime, once an IL compiler is developed for the language.

Corresponding to the IL on the Java side is J2EE "bytecode." Generated by Java compilers, bytecode is an intermediate language that supports object-oriented programming. Bytecode is a language abstraction that allows Java code to run on different operating platforms, as long as the platforms have a Java Virtual Machine to execute bytecode.

In the .NET world, Microsoft calls its language-abstraction layer the Common Intermediate Language (CIL), or simply Intermediate Language. Microsoft's IL is similar to Java bytecode. Any .NET language may be converted to IL to run on the .NET Common Language Runtime.

To enable the CLR to support different languages, Microsoft has provided a Common Language Specification (CLS). The CLS specifies basic rules for language integration in the .NET platform. To enable a language to run on the .NET CLR, compiler vendors must build compilers to the CLS specification.

In J2EE, JavaServer Pages (JSPs) and servlets are used to create dynamic Web pages. Servlets are controllers that coordinate the processes in a Java application: fetching data from databases, building Web pages, and presenting the pages to browsers and client applications. JSPs are programs that run on a Web server and control the building of dynamic Web pages. As Sun says, "In its basic form, a JSP page is simply an HTML Web page that contains additional bits of code that execute application logic to generate dynamic content." When the page is displayed in a user's browser, it will contain both the static HTML content and dynamic information retrieved from the database. Sun says JSPs are a type of servlet, and that JSP pages are compiled into servlets before they are used.

The .NET technology for building dynamic Web pages is ASP.NET, which is an update of Microsoft's Active Server Pages (ASP) technology. ASP is a server-side scripting technology that enables scripts to be executed by the Internet Information Server that is part of the Windows server operating system. Microsoft says the advantages of ASP.NET include the ability to program in more languages, and that "ASP.NET server controls enable an HTML-like style of declarative programming that let you build great pages with far less code than with classic ASP." 

In the J2EE world, the standard way to connect to relational databases is provided by the Java Database Connectivity (JDBC). As Sun says, "JDBC technology is an API (included in both J2SE and J2EE releases) that provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files."\(^{17}\)

For accessing databases on the Microsoft side, .NET employs ADO.NET, an updated version of ActiveX Data Objects (ADO) that was the data access technology in Microsoft's previous client/server model. ADO.NET employs Web service technology (XML and SOAP) to enable data exchange among different systems. As O'Reilly's " .NET Framework Essentials" explains, clients need only employ an XML parser to read standard XML data. "The data producers and consumers need only adhere to the XML schemas to exchange data among themselves."\(^{18}\)

Other Java services for which .NET has comparative technology include:

*The Java Naming and Directory Interface* (JNDI), which allows application components and clients to look up distributed resources, services, and EJB components.

*The Java Message Service* (JMS), which provides a common way for J2EE applications and modules to create, send, receive, and read messages in a distributed environment.

*The JavaMail*, which allows J2EE applications to create, send, receive, and read mail messages. The JavaMail API includes support for the IMAP4, POP3, and SMTP mail protocols. JavaMail sessions are made available through the JNDI API.

*The Java Connectivity Architecture* (JCA), which allows Java enterprise applications to interface with existing non-Java enterprise applications like SAP and Siebel systems.

*Java Management Extensions* (JMX), which manages J2EE servers and applications. The corresponding .NET features are listed in the feature comparison table above. However, as described in Part III below, there also are missing pieces in .NET, as well as differences in how operations are coded and handled in .NET.

**Enterprise JavaBeans**

Enterprise JavaBeans (EJBs), introduced in 1999, are components that contain a Java application's business logic and data. The use of EJBs is not mandatory for Java applications. Successful applications can be built using servlets, JSPs, and standalone Java applications.

EJB processes run in "containers." There are client-side and server-side containers. The container provides the EJB with services such as security, transactions, database connectivity, and lifecycle management.

Portions of an application typically run in a specific container and then hand control to other processes running in other containers. On the server side, there is a Web container and an EJB


\(^{18}\) " .NET Framework Essentials," Thuan Thai, Hoang Q. Lam, O'Reilly,
container. On the client side, there is an applet container that runs in a browser, and an application client container.

The container controls the various EJBs in existence and the resources they are using, such as memory and database connections. Each container will maintain a pool of EJB instances that are ready to be assigned to a client. When a client no longer needs an EJB, the EJB instance will be returned to the pool and all of its resources will be released.

The EJB server is the base set of services on top of which the container runs. EJB servers are delivered as part of a J2EE-compliant application server. An application server (such as IBM's WebSphere Application Server) provides the underlying services required of an EJB server and will host EJB containers.

There are three different types of EJBs: session beans, entity beans, and message-driven beans.

- **Session Beans**—Session EJBs are used for mapping business process flows. There are two sub-types of Session EJBs — stateless and stateful. Session EJBs commonly represent functionality that is created as it is needed.

- **Entity Beans**—Entity EJBs map the data and associated functionality being used in an application. Entity EJBs are usually based on an underlying data store.

- **Message-Driven Beans**—Message-driven EJBs are similar in concept to a Session EJBs, except that message-driven beans are only activated when an asynchronous message arrives.

The EJB 2.1 specification enhances the EJB architecture with support for Web services. Enterprise developers can implement and deploy Web service applications like other server-side applications.

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.

**.NET Platform and Framework**
The **Microsoft .NET Platform** consists of five main components: Windows operating system, Windows Enterprise Servers, .NET Framework, .NET Building Block Services, and Visual Studio.NET toolset.

The **.NET Framework** is Microsoft's Java-like development and execution environment for building and running Web services. It is a set of programming interfaces that are being incorporated into the Visual Studio.NET toolset.

The three major components of the .NET Framework are the Common Language Runtime, Framework Class Library, and ASP.NET. While the Common Language Runtime runs only on Windows, it is potentially portable to any operating system, including Linux and Unix. However, Microsoft says it has no plans to port .NET to other platforms.

The .NET Framework simplifies the creation of Web services by automatically handling many common programming tasks. The Common Language Runtime encompasses the services that are language-independent and provide access to the object management within the NET Framework. The Common Language Runtime services allow a Visual Basic object to talk to a C# object or C++ object. The underlying object services, security services, memory management services, and just-in-time compiling services are all encapsulated in the Common Language Runtime in a language-independent way.

Other elements of .NET include ADO.NET and ASP.NET, described above, and Win Forms and Web Forms, which are user interface component frameworks accessible from Visual Studio.

**.NET Multi-Language Support**
Microsoft promotes .NET's multi-language support capabilities as though various programming languages would simply run on Microsoft's platform unaltered. However, the languages must be rewritten to run on the CLR. As publisher O'Reilly reports: "The IL common language runtime, for example, has some fairly significant hurdles to overcome before it has any real payoff for developers. Each language that wants to integrate with the component runtime has to define a subset/superset of the language that maps cleanly into and out of the IL runtime, and has to define constructs that provide the component metadata that IL requires. Then compilers (x-to-IL and IL-to-x) will have to be developed to both compile language structures (objects, components, etc.) into IL component bytecodes, and also generate language-specific interfaces to existing IL components."^19

The Middleware Company reports that "a combination of languages running in the CLR may lead to a mess of combination spaghetti code that is very difficult to maintain." In addition, it says that, "With combination language code, your developers are unable to share best practices."^20

---


Analysts have been lukewarm about the benefits of .NET's multi-language support. Most agree with O'Reilly's assessment that, "If the Java environment is the goal, people will generally choose to learn Java. I predict that the same will be true of .NET: People will generally choose to learn C# and write .NET components in that language."\(^{21}\)

**Web Services**

IBM and Microsoft have spearheaded the development and adoption of Web services technology, and both J2EE and .NET support Web services. Programming for Web services is built into the .NET model, while Web services are supported as an extension of J2EE.

Web services are applications that can make their data available to other Web sites or applications, regardless of the platform, operating system, or programming language of the application. Microsoft describes the .NET Platform as a "set of programming tools and infrastructure to enable the creation, deployment, management, and aggregation of XML Web services."

Web services enable easier application integration through standardization. As The Middleware Company reported: "Prior to the advent of Web services, enterprise application integration was very difficult due to differences in programming languages and middleware used within organizations. The chances of any two business systems using the same programming language and the same middleware was slim to none, since there has not been a de-facto winner."\(^{22}\)

Through Web services, applications can be reused and linked together. Web services are created as a natural extension of J2EE programming. The IBM WebSphere Studio toolset enables developers to easily turn a server application into a Web service. The toolset also enables client applications to access Web services. Microsoft's Visual Studio.NET toolset also enables programmers to create and invoke Web services on the Windows platform.

In the IBM environment, systems and applications can be made available as services, linked together by Web services, as part of a "Service Oriented Architecture."

**J2EE, .NET and Openness**

While Microsoft tries to portray .NET as an open platform, analysts have recognized that .NET is Microsoft proprietary technology. Said Gartner Group: "The .NET system programming model is Microsoft-proprietary, although pieces have been opened up as part of European Computer Manufacturers Association’s (ECMA’s) C# standardization efforts."\(^{23}\)


Through the partial submission of .NET to a European standards committee, Microsoft has created the "illusion of open standards support" for the .NET platform, said Gartner Group.24 As O'Reilly reports: "The core components of the framework (IL runtime environment, ASP.NET internals, Win Forms and Web Forms) are kept by Microsoft, and Microsoft will be the only provider of complete .NET development and runtime environments."25

On the other hand, J2EE, says O'Reilly, "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."26

Analysts widely agree that J2EE is an open standard while Microsoft .NET is proprietary. "In practical terms," says O'Reilly, "if you build your application on .NET you're wedded to Microsoft for the life of the application—and at their 'mercy' as license costs continue to increase and terms of use become increasingly restrictive."27

Said IDC:

"Through its proprietary, platform-preferential directions of Visual J++, its abject rejection of Java as a viable language for .NET services, and its ultimate development of the C# language as a proprietary alternative to Java, Microsoft continues to be perceived in non-Windows environments as the biggest holdout against open, ubiquitous, cross-platform, and interoperable standards for application development."28

"IDC believes that Microsoft is not motivated to promote any standards that would encourage its partners to bring their value-added Web services to non-Microsoft platforms. On the surface, Microsoft is adopting leading Internet protocols and standards as a means of establishing its credibility as a provider of supposedly open services. In reality, Microsoft has only one primary motivation — to leverage its dominant market presence to obtain a leadership position as a provider of Web services and to derail the Java community and Unix efforts to compete successfully against it."29

Likewise, said Gartner Group: "Gartner believes Microsoft encouraged confusion around the issue of .NET on other platforms because the speculation served the company’s objective to create an environment in which .NET was seen as potentially having characteristics (for example, cross-platform and open) of Java without actually having them."30

26 Ibid.
27 Ibid.
29 Ibid.
Moreover, said Gartner, "Microsoft carefully devised subsets of .NET targeted at academia and standards efforts — such as C# standardization with the European Computer Manufacturers Association (ECMA) — without ever intending to fully open up .NET."  

J2EE’s maturity, openness, and cross-platform capability make it attractive to ISVs, said Gartner Group. "Because J2EE is platform-neutral, applications can be deployed on a variety of different platforms (Windows, Linux, Solaris, HP-UX, AIX, iSeries and zSeries). Thus, ISVs can expand their potential market at a very reasonable cost. Moreover, J2EE is now mainstream, proven technology, widely accepted in the marketplace and supported by software industry leaders. Therefore, ISVs feel comfortable investing in J2EE, because they perceive it as a low-risk and viable platform that will protect their technology investments in the long term."

Likewise, said The Middleware Company: "Unless you can guarantee that every one of your customers will accept a Windows/.NET solution, you are restricting your salespeople from major accounts that may have solutions deployed on Unix or mainframes. This is rarely acceptable at most ISVs or consulting firms."

Part III Advantages of J2EE vs. .NET

Similar But Not Equal
While J2EE and .NET are similar in many ways, there are differences that give J2EE architectural and development advantages. J2EE provides a componentized runtime environment that reduces coding and facilitates more rapid application development. .NET currently has no container concept, and certain functions that are automated within J2EE containers must be hand coded in .NET.

A number of .NET programming limitations and missing capabilities vs. J2EE are summarized in the chart below, which shows how J2EE provides a more complete server programming environment.

<table>
<thead>
<tr>
<th>J2EE</th>
<th>.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Transaction Service (JTS)</td>
<td>Interoperate with COM+ services</td>
</tr>
<tr>
<td>Procedural transactions via JTA</td>
<td>Limited declarative-only capabilities</td>
</tr>
<tr>
<td>Container-Managed Persistence</td>
<td>Program it</td>
</tr>
<tr>
<td>Message-Driven Beans</td>
<td>Build with queued components</td>
</tr>
<tr>
<td>Java Database Connector (JDBC)</td>
<td>Different APIs for each ADO.NET provider</td>
</tr>
<tr>
<td>Java Naming &amp; Directory (JNDI)</td>
<td>Build it</td>
</tr>
</tbody>
</table>

31 Ibid.
32 Ibid.
While J2EE and .NET each provide database connectivity technology--Java Database Connector and ADO.NET, respectively, the .NET model does not have a standard set of APIs for each vendor's database. In the .NET model, there is a specific set of APIs for each vendor, and each database vendor must write its own database driver. This limits the ease with which you can swap databases in a .NET solution. If you change databases, you must rewrite your applications to support the new database APIs. In the J2EE model, there is one set of APIs. Any J2EE application can support any vendor's database that conforms to the Java Database Connector standard.

Another J2EE capability lacking in .NET is "persistence," in which a transactional change will automatically be registered in a database. In J2EE, the container or EJB will track any change that occurs and will update the database. .NET does not have this capability, and requires extra programming. In complex applications, you may have multiple paths, which complicates the effort required to code for the database changes.

.NET also lacks the transactional capabilities of the J2EE Transaction APIs. .NET has nothing comparable to JTA, and must rely on Microsoft's older COM+ services for this type of functionality. The .NET model does give programmers the fine level of control and flexibility that JTA provides. Through tooling, JTA enable programmers to specify database access, security levels, and transactional requirements in ways that .NET is not capable of providing.

.NET has no capability similar to J2EE message-driven beans. When a message arrives at a queue, J2EE creates a message-driven bean and invokes the message. In the .NET model, you must continually sift through a message queue, discover the message, then act on it. In J2EE, the discovery and response process is automated, while in .NET you have to write the code to sift the message queue, discover the message, and respond to the message.

Another J2EE advantage over .NET is the Java Naming and Directory (JNDI). This is a standard service available to tools and vendors for finding resources on a network. .NET has no similar standard way of locating objects on a network. For example, an object could be in a file, in Active Directory, or elsewhere. The .NET system has no automated way of knowing. If you want to locate an object, the means for doing so may vary, depending on how the programmer wrote the code. And if the location of the object changes, a programmer has to make changes. Moreover, changes may be required by several programmers in different locations, depending on how the application was written.

.NET also lacks a counterpart to the J2EE Connection Architecture for writing adapters, or connectors, to applications. In the J2EE model, there is a standard approach for writing J2EE Adapters, and the way you invoke the connector in J2EE is standard. There is no general standard for .NET connectors.
The Java Message Service (JMS) is another feature for which .NET lacks a counterpart. In the Microsoft messaging model, Microsoft Message Queue (MSMQ) only enables messaging between Windows systems. To communicate with a non-Windows system requires a bridge. In the J2EE model, a system like IBM's WebSphere Application Server, using JMS, can exchange messages with IBM WebSphere systems running on different platforms.

J2EE derives advantages from Java tools working in combination with the J2EE foundation. IBM achieves programming advantages through the capabilities of its WebSphere Studio toolset wizards.

In its most recent review of Web application servers *PC Magazine* found that, "The .NET path offers fewer options in building business logic and database components. Microsoft has no official blueprint for business objects comparable to Enterprise JavaBeans (EJB), though it recommends best practices on the Web (www.microsoft.com/resources/practices). The .NET developers have to devise their own component models based on these practices, while a J2EE developer just needs to run a wizard to get EJB."34

**.NET Enterprise Limitations**
While J2EE and Microsoft .NET are similar in nature, each has particular strengths and weaknesses. Analysts generally agree that .NET has strong user-interface tools and is better at client-side development, while Java has advantages on the server side, particularly for enterprise applications. The .NET toolset is generally seen as easier to use for simpler Web service development, while J2EE is seen as superior for complex, heterogeneous enterprise application development.

Gartner Group has consistently said it believes that J2EE will dominate high-end enterprise applications, while .NET will be deployed mostly in departmental and smaller enterprise applications. In a CIO update, Gartner said it "expects that most of Microsoft’s market share will be dominated by small and midsize businesses during the next five years. Larger-scale, national and multinational enterprises are more likely to have significant investments in heterogeneous data center infrastructures (for example, a mixture of mainframe, AS/400, Unix and Win32 systems). Java’s cross-platform support and transportable code will drive significant investments by these larger entities toward J2EE."35

Giga found that .NET is more difficult to customize and that J2EE provides a broader set of programming interfaces. "J2EE and J2EE server products provide a greater range of features to assist in architecting, designing and building a high-end enterprise application," said Giga. "Thus, with Microsoft’s architecture, it takes more developer work to meet high-end demands."36

---


J2EE is four years older than .NET, and is found by analysts to be more mature. Gartner Group reports that .NET "is still in its infancy, whereas J2EE is much more proven in real-life situations."\(^{37}\) Likewise, said Giga, "J2EE is further along the maturity curve than .NET."\(^{38}\)

Another major problem for Microsoft is the incompatibility of its .NET technology with its previous generation of technology. .NET components are radically different than the previous Component Object Model (COM) and COM+ service components that still comprise the core of Microsoft's product families. The .NET strategy calls for Microsoft to replace the old code with .NET code over the next 4-5 years. [INSERT GATES QUOTE].

Microsoft's roadmap calls for a series of rewritten servers as Microsoft replaces its older technology with .NET. Gartner Group advises users to “Take into consideration the discontinuities that are likely to occur through 2010 as Microsoft updates its middleware technology stack.”\(^{39}\)

Because .NET is a project under development over many years, there is heightened risk and instability in the Microsoft platform. This is why Giga said, "Besides instability of the platform itself, the transition to .NET creates a high degree of instability in the development of applications using the platform.”\(^{40}\)

The combination of .NET, COM, and COM+ service components complicates development and administration in Microsoft environments. Multiple toolsets are required to develop and maintain applications. The transition to .NET requires new training and skills for Microsoft developers.

The complications of .NET development will grow more severe as the server replacements in the .NET roadmap progress. Said Giga, "Microsoft .NET will add more APIs and programming models, plus options and decisions for how and when to migrate existing code to .NET, further complicating things.”\(^{41}\)

As Microsoft rewrites its applications, users are required to migrate through new and incompatible versions of Microsoft products. For example, no Windows 2000 applications run on Windows Server 2003 without revision or completely new versions. Microsoft's roadmap calls for complete rewrites of Windows Server, SQL server, and Exchange Server, which will introduce further incompatibility. The new releases of BizTalk Server and SharePoint Server are incompatible with previous releases.

\(^{41}\) Ibid.
One of J2EE's strengths is that a wide variety of tools, products, and applications are available from numerous vendors. Competition among J2EE vendors drives innovation. Microsoft users, on the other hand, must rely on Microsoft for all .NET innovation.

Dennis Sosnoski, a 30-year software developer veteran, in his assessment of J2EE vs. .NET, found that Java has advantages in "stability, reliability, portability, robustness, and scalability." He believes Java's stability stems from the fact that the platform has evolved over last several years, changes are coordinated through the JCP, hundreds of companies and individuals are involved in Java development, and public discussion takes place before standards become final.

Java is more robust and reliable, says Sosnoski, because Java servers are more powerful, most servers include full clustering support, the EJB architecture is designed for distributed use, and transactions and security are built in from the ground up.

---

43 Ibid.