IBM SWG Competitive Technology Office
Somers, NY
August 1, 2002

Why IBM WebSphere is Better than Microsoft .NET

Michael Neubarth
Howard L. Operowsky

This document is composed of two parts. Part 1 is a general discussion of WebSphere's advantages over .NET. It is directed to both non-technical and technical audiences. Part 2 describes specific instances where WebSphere technology is superior to .NET, and is slightly more technical in nature than Part 1.

Part 1: Why WebSphere over .NET on the Server?

Introduction

There are significant reasons to choose WebSphere/J2EE over Microsoft .NET as a foundation for e-business. While IBM supports the choice of .NET as a client, we believe a WebSphere/ J2EE environment is the better choice on the server.

While Microsoft touts .NET as a more productive environment for creating Web applications, the .NET environment is riddled with problems, including missing functions, object incompatibilities, and inflexible coding schemes. These problems can complicate development and administration, degrade network application performance, and break applications.

Microsoft .NET is limited in many respects vs. WebSphere/J2EE.

- *Openness.* The .NET platform is proprietary, while WebSphere/J2EE is an open platform. With J2EE, users have a choice of 22 vendors offering core technology.
- *Completeness*. Microsoft .NET is new and incomplete, while WebSphere/J2EE is mature and proven, and is the preferred platform in enterprises by a three-to-one margin, according to Giga surveys.
- *Compatibility.* Microsoft .NET introduces a new component model and toolset that are incompatible with previous versions, while WebSphere/J2EE toolset and components are uniform.
- *Development.* While the promise of .NET is ease of implementation and administration, there are many .NET coding schemes and requirements that impose burdens on programmers and administrators-- reducing productivity and increasing cost. The WebSphere/J2EE development is consistent, smooth, and more productive.
- *Integration*. .NET integration and interoperability capabilities are limited. WebSphere runs on multiple platforms, supports any client, and has superior integration capabilities.
- *Scalability and Reliability.* Many problems that existed on the Microsoft platform still persist and are even more pronounced in the .NET environment. .NET deployments are not easily scalable, customizable, or portable. WebSphere offers a more open, scalable, secure, reliable, flexible, and productive platform on which to base an enterprise.
- *Security*. Microsoft has a long record of security problems, while IBM has a history of providing bulletproof enterprise systems.

The following sections offer more detailed discussion of the advantages of WebSphere/J2EE vs. Microsoft .NET as the basis for an e-business server infrastructure.

Openness

Openness is a critical criteria for organizations looking to avoid the perils of lock-in and obtain the benefits of flexibility. .NET is a proprietary platform that is available only on Windows, and only from Microsoft. Once locked into the Microsoft platform, users have no choice and must rely on Microsoft for all core technology and any innovations.

"The .NET system programming model is Microsoft-proprietary," said Gartner Group.[1] Likewise, said IDC: "The company's preference for proprietary control of its technology can be found in plans for the desktop, mobile, and server platforms that are steeped in unique and proprietary Microsoft technologies." Furthermore, said IDC, "the entire strategy of Visual Studio .NET is built on Microsoft proprietary technology."[2]

While interoperability is supported through Web Service standards (SOAP, XML, WSDL, etc.), Windows and the .NET Platform are proprietary. The Microsoft .NET development and runtime environments are proprietary, as are other pieces of the core .NET technology (operating system, .NET Framework, .NET toolkit, Web application server, object model).

Said Gartner Group: "Microsoft's openness in establishing industry-wide Web services standards is a Trojan horse. Microsoft needs the industry to invest in and support one common standard middleware format so that the majority of consumers are able to seamlessly access its proprietary Web services applications."[3] Moreover, said Gartner Group, "Success of its proprietary applications is the driving motivation of Microsoft's push for the Web services middleware standards."[4]

Likewise, said IDC: "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."[5]

Completeness

.NET is a work in progress, with the aim of providing an alternative to Java. Much of .NET is futureware that Microsoft has promised to deliver. For example, there is as yet no .NET client or server, and there is little .NET technology in the Microsoft .NET Enterprise Server family. The .NET development environment also lacks many pieces and capabilities that J2EE provides (see Part 2 for details). Key Microsoft products such as "Yukon" (SQL Server), "Longhorn" (Windows server), and "Blackcomb" (Windows server) are still under development and are not slated for release until 2004-2006.

---

[1] Gartner Group, "Weaving the Future: Microsoft's .NET Initiative," D. Smith, M. Driver, C. LeTocq, T. Bittman, W. Andrews, D. Plummer, M. Calvert, April 5, 2001.

[2] Ibid.

[3] Gartner Group, "Microsoft's Web Services: A 'Trojan Horse,' " Y. Natis, D. Smith, October 15, 2001.

[4] Ibid.

[5] IDC.

At its .NET Briefing Day on July 24, 2002 Microsoft executives admitted the company had failed to deliver on its .NET vision. Bill Gates gave Microsoft a “C” for delivering .NET building-block services and for making the software-as-a-service concept a reality. “A lot is still to be done there,” he said. Gates also gave Microsoft an “incomplete” for its efforts to enable “federation,” a secure connection between two or more companies. Gates also gave Microsoft an “incomplete” for its goal of changing users’ computer experience. “That’s still not there, but it’s coming,” he said. The delivery of many promised .NET pieces is still years away, said Gates. “We knew when we did it, it would be a five-, six-year effort,” he said.

Compatibility

At the root of many of the .NET programming difficulties are the three different component models (.NET objects, serviced components, and COM objects), or sub-environments, that now underlie the Microsoft platform. Companies implementing .NET applications must develop, integrate, and support environments comprised of these three different object models. The incompatibility of the .NET component model and toolset with the previous generation of Microsoft technology introduces significant disruptions and stumbling blocks for programmers and system administrators, including version control, deployment, and debugging issues. The conversion required between different components also takes a toll on application performance (up to 12 times longer to make a .NET-to-COM call than a .NET-to-.NET call).

Companies with “legacy” Microsoft applications that are now developing .NET applications must employ two different toolsets (Visual Studio.NET and Visual Studio 6.0). There are a number of problems resulting from the incompatibilities between the old and new Visual Basic and toolsets.

Gartner Group estimates that 60% of Microsoft legacy code must be rewritten to interoperate with .NET. Moreover, a recent Gartner Group study found that the cost of converting old Microsoft applications to .NET would cost 40% to 60% as much as the original cost of creating the applications.

Development

Although .NET is supposed to offer quick, easy programming, this is true mainly for simple applications. For companies wishing to create high-workload distributed transaction processing applications, .NET presents significant problems that make .NET deployments difficult to architect, program, and change. Moreover, .NET programing is hampered by obstacles that make .NET transaction processing applications brittle compared with J2EE applications (these obstacles are described in detail in Part 2).

.NET deployments are in many ways convoluted and inflexible, which makes them more costly to manage. For example, .NET applications are difficult to reconfigure for changing workloads and new functions. .NET tooling for Web services is incomplete, and there is no support for portals or business integration. NET also requires programmers to write code to cover missing elements.

WebSphere has a simple and consistent programming model, and is easier to deploy and manage.

The advantages of WebSphere over .NET are especially pronounced in enterprise transaction applications. WebSphere Studio supports the open-source Eclipse workbench, and WebSphere Studio Application Developer provides all the tools needed for J2EE, XML, Web services, and business integration in one integrated environment.

In comparing the J2EE and .NET architectures, Giga Information Group wrote that “J2EE’s richer features can reduce the effort to build a Web application by perhaps 20 percent to 40 percent (in the coding stage of a project), and Giga has received repeated and consistent confirmation of this from various client bake-offs that have happened in the past year.”[6]

Integration

.NET’s integration technology is limited and often rudimentary. For example, MSMQ, Microsoft’s message queuing system, can only talk to other Windows servers. Microsoft relies on bridges to reach other systems, which requires tedious and complicated manual configuration and duplicate administration tasks. Performance also is degraded.

Microsoft’s integration technology lacks features such as standard reusable service components, and tools to generate service components and service flow. In enterprise system integration scenarios, modifying one system may require changes to all the systems to which it connects. While changes flow automatically in a WebSphere/J2EE environment, there is no automated flow in a .NET environment.

BizTalk, which is Microsoft’s solution for enterprise integration, has several shortcomings. First, BizTalk is an all-purpose tool aimed at three different types of work: enterprise application integration, business process automation, and B2B. While WebSphere provides solutions optimized for each of these tasks, BizTalk’s all-in-one approach results in compromises for each of these categories.

BizTalk is not built as a high-speed switch, so there are serious performance and scalability concerns. Also, BizTalk is a bundled set of tools that are not completely integrated. For example, BizTalk’s development and administration tools are not integrated.

The IBM business integration family is modular, designed to work together, and offers best-of-breed software for each requirement. The acquisition of CrossWorlds has rounded out IBM’s portfolio of business integration software. IBM middleware also supports the widest range of clients, including browsers, wireless devices, Java clients, and Microsoft .Net workstations.

[6] Ibid.

Scalability and Reliability

Analysts such as Gartner, Giga, and IDC agree that Microsoft's scalability, reliability, and availability are limited. .NET's clustering technology is "lower tier," according to IDC. Moreover, .NET imposes limitations and restrictions in the way server clusters can be configured.

Giga Information Group and Gartner Group both see Java as superior to .NET for enterprise applications. Said Giga: "From the technical viewpoint of an enterprise application architect, J2EE provides richer features for high-end application development. .NET will greatly improve Microsoft's offering, but J2EE will still retain the overall technical advantage."[7]

J2EE's strengths, said Giga, "are its architectural, standards-based focus, Java's modularity for large applications, APIs for naming, publish/subscribe messaging and integration, richer scalability and failover features provided by J2EE server products and deeper application platform functionality provided by some of the J2EE vendors (e.g., portal, personalization, wireless, etc.)."[8]

According to Gartner, "Users planning Web services projects will find the new VS.NET useful, but not a breakthrough technology compared to the Java-oriented Web services platforms."[9]

An example of WebSphere's power and flexibility is the eBay site. On eBay, 30,000 lines of code must be refreshed each week, and 210 million page views must be presented each day. Microsoft .NET would not be able to handle the job.

Security

The weak security of Microsoft's products is widely recognized. The problems are so severe that Gartner Group recommended that enterprises with Web applications running on Microsoft's Internet Information Server (IIS) "should start to investigate less-vulnerable Web server products."[10]

Gartner recommended that enterprises hit by both Code Red and Nimda viruses should "immediately investigate alternatives to IIS, including moving Web applications to Web server software from other vendors."[11] In May 2001, Gartner advised enterprises that had not yet made Web server decisions "to weight security heavily and to evaluate other Web server software offerings."[12]

---

[7] Giga Information Group, "J2EE vs. Microsoft and .NET: Enterprise Application Architect's View Favors J2EE," Randy Heffner, Mike Gilpin, September 18, 2001.

[8] Ibid.

[9] Gartner Group, "Microsoft's .NET Aims for the Enterprise, but Starts Small," Y. Natis, March 22, 2002.

[10] Gartner Group, "Nimda Worm Shows You Can't Always Patch Fast Enough," John Pescatore, September 19, 2001.

[11] Ibid.

[12] Gartner Group, "Bugs Cling to IIS in Windows XP Beta," John Pescatore, June 20, 2001.

Microsoft's record of breaches is extensive. In 1999, Microsoft posted advisories warning of 60 software flaws that weakened the security of its products. That total jumped to 100 bugs found in 2000 and about 100 more in 2001 and 2002. Microsoft software powers about 30% of the world's Web sites, and 62% of the sites that have been hacked, according to data collected by Netcraft's Web Server Survey and the Alldas Defacement Archive. Problems continue to surface. In April 2002, CERT issued a warning to the information technology industry about 10 new vulnerabilities affecting IIS.

Said John Schweitzer, chief security officer at Ogilvy & Mather, "Microsoft's NT product line, including Windows 2000 and IIS, has the worst security track record of any operating system in the world. That's why most security professionals reject their products out of hand."

Said Giga analyst Rob Enderle: "The perception that Microsoft products, and Microsoft, carry an unacceptably high security exposure is widespread. That belief was substantially fueled by the successive security exposures and attacks on Microsoft's products and Web site."[13]

WebSphere, on the other hand, is a secure enterprise platform and a highly productive development environment. Gartner Group has advised users to consider IBM WebSphere as an alternative to Microsoft's Internet Information Server.

Why WebSphere?

While .NET has many shortcomings as an enterprise environment, WebSphere excels in the high workload, clustered environments of enterprise customers. WebSphere Portal Server can be used to build business-to-consumer and business-to business portals. In addition, WebSphere can help companies extend their reach with personalized merchandising, cell phone support, voice service, and machine translation.

With WebSphere/J2EE, users avoid all the problems associated with .NET, and obtain the best server infrastructure for Web applications. In summary, the reasons why WebSphere/J2EE beats .NET are:

- Openness
- Interoperability
- Integration
- Flexibility
- Scalability
- Performance
- Developer productivity
- Security

[13]Giga, "Microsoft's Image the Major Hurdle for Its HailStorm Initiative," Rob Enderle, April 3, 2001.

Part 2: Technical Differentiators

This part describes the technical problems inherent in .NET in greater detail, and illustrates why deploying a WebSphere/J2EE solution on the server is a better choice for companies than a .NET solution.

.NET's model for distributed applications is complicated and inconsistent while the WebSphere model is simple and consistent

- **.NET's different object models complicate system administration.**

  .NET objects must still use Windows COM+ Services for transaction processing and other functions. .NET objects which use COM+ Services are called *serviced components* because they must use the .NET "ServicedComponent" class to access the COM+ Services. But because the COM+ Services were invented before .NET, serviced components have different characteristics from "pure" .NET objects and add the following complications to system administration:

  Need To Use Multiple Toolsets: Both system administrators and application developers will need to understand and use two different configuration mechanisms, one for .NET objects and another for serviced components. For example, .NET objects are configured using "AssemblyInfo" files and XML-based "config" files, while serviced components must be registered in the COM+ catalog before they can be used, and are managed using the Component Services administrative tool.

  Proliferation of useless objects caused by .NET Versioning: The default settings for Versioning in Video Studio.NET causes a proliferation of useless serviced components in the COM+ catalog. With Visual Studio's default versioning setting, every build of a serviced component creates a new version of that object, and each new version must be registered in the COM+ catalog. The older versions are not deleted and remain in the catalog until they are removed manually. The only way to correct this problem is to manually change the Version attribute for each assembly that creates serviced components.

  Serviced Components Complicate XCOPY Deployment: With XCOPY deployment applications can be installed simply by copying their files into the appropriate directories. Serviced components in the new application are automatically registered in the COM+ catalog the first time that the application is executed. But only an administrator can add serviced components to the COM+ catalog. If a user without administrative privilege attempts to execute a new application which contains unregistered serviced components, the application will fail when it attempts to add the serviced components to the COM+ catalog. Thus, the system administrator must still install most .NET applications, contrary to the promise of XCOPY deployment.

- **.NET's three different object models complicate application debugging**

  Different debuggers required for .NET and COM objects: For the next several years, there will be very few applications made up of only .NET objects. Instead, most applications will be a combination of both .NET objects and COM objects as users incorporate .NET code into their existing COM-based applications. Visual Studio .NET, however, does not support the older source code; Visual Studio 6 will still be required, in addition to Visual Studio .NET, to create, debug and maintain the hybrid .NET/COM applications. Visual Studio .NET must be used for the .NET source code while Visual Studio 6 must be used for the COM source code. The need for both will be especially vexing when it becomes necessary to switch debuggers *while* debugging a hybrid application. Consider the case of a programmer trying to correct a problem by stepping through the source code using Visual Studio .NET, and who comes to a point in the .NET code where a COM object is called. Normally, the programmer would continue by stepping into the called routine. But he cannot in this case because Visual Studio .NET does not support the older source code. Instead, the programmer must now switch to Visual Studio 6 to step through the COM object. But to do so, he must start Visual Studio 6, attach it to the application being debugged, and put a breakpoint in the COM source code to continue tracing at the correct point. He must put a breakpoint in the .NET code to ensure that he can continue debugging the .NET code when the COM object completes execution. Forgetting one breakpoint, or putting it in the wrong place, could mean missing the error and restarting from the beginning. WebSphere has a single, consistent, object model, and WebSphere programmers are not plagued by compatibility problems such as the one described above.

Serviced Components: Visual Studio .NET behaves in two different ways when debugging serviced components. A serviced component can be executed "in-process", i.e., within its caller's process, or "out-of-process," i.e., within a completely different process. This difference affects a developer's ability to debug applications which use serviced components. If the developer is stepping thorough a program that has an in-process serviced component, she will be able to step into the serviced component when it is called. But she will not be able to step into the serviced component if it is executing out-of-process. When she tries, the debugger will not stop in the serviced component at all. Instead, it will stop at the calling program's next instruction - after the serviced component has completed its processing. In order to step through an out-of-process serviced component, the developer must first get the id of the serviced component's process, attach a second instance of the debugger to that process, and set breakpoints in the serviced component. This additional burden on the developer makes her less productive. Adding more complexity, Microsoft has recommended using the CLR Debugger, which is part of the .NET Framework SDK, to debug out-of-process serviced components, not the Visual Studio.NET debugger.

- **.NET Remoting does not support transactions in distributed applications**

.NET Remoting enables applications on different computers to communicate with one another. But .NET Remoting does not allow distributed applications to participate in the same transaction. The best that can be done with .NET Remoting is to start a new transaction in the called application, which is independent of the caller's transaction. This could lead to incorrect results because the changes made by the remote program can be committed even if the caller decides to abort the transaction. If operations on the remote host must be contained in the same transaction, the programmer must fall back on DCOM, even though .NET is supposed to simplify creation of distributed applications. If the program was already written to use .NET Remoting, code would have to be changed in order to use DCOM. The WebSphere transaction model is simple and consistent. In WebSphere, remote applications are implemented with EJBs and the EJB Container automatically inserts remote EJBs into the caller's transaction whenever necessary.

- **.NET's various mechanisms for implementing distributed applications make development of distributed applications complicated and difficult to implement.**

  .NET provides 3 different mechanisms for implementing distributed applications: web services using ASP.NET, .NET Remoting, and DCOM. Which approach should you choose and when? It's not clear. The developer must understand and weigh various options to properly design distributed applications for .NET. For example, both ASP.NET and .NET Remoting support web services. Web services use the SOAP protocol, but SOAP-serialization in .NET does not guarantee that objects deserialized on the server will be bit-wise identical to the original object on the client. If full .NET type-system fidelity is required then objects must be transmitted using a binary serialization instead of SOAP serialization. In that case, ASP.NET can no longer be used. But the developer can use either .NET Remoting or DCOM. .NET Remoting allows the distributed components to communicate using either HTTP or TCP. But .NET Remoting does not allow distributed components to participate in the same transaction. If the application requires full transaction support for all distributed components, then DCOM must be used. If DCOM is used, then the distributed components must execute on Windows platforms, and the application cannot interoperate with other non-Windows machines as .NET promises. In WebSphere, distributed applications are written in Java and built with web services and/or Enterprise Java Beans (EJBs), and do not have the problems described above.

- **Programmers must sometimes compensate for incorrect .NET code generation by editing low-level, machine-generated code.**

  .NET objects behave differently from COM objects at run time. Therefore, in order for a .NET program to call a COM object, additional code must be created to handle the differences. *COM Interop*, the .NET subsystem responsible for this, attempts to generate the additional code automatically, but sometimes it generates the wrong code, causing the program to fail. When this occurs, the programmer must correct the code created by COM Interop. This process is difficult because the generated code is not written in a standard .NET programming language. Instead, it is written in the low-level Microsoft Intermediate Language which resembles assembly language and which is foreign to virtually all application developers.

The .NET developer needs to write code to cover missing elements; WebSphere provides a more productive development environment for enterprise applications.

- **WebSphere provides advanced functions which are not included in .NET and which are difficult and costly to implement by the enterprise.**

  WebSphere is a mature product, and implements the functionality of the J2EE standard which has been developed by the best software engineers of 22 enterprise software providers. The result is that WebSphere/J2EE products implement advanced functions not available in .NET. These additional functions make J2EE programmers 20-40% more productive than .NET programmers, according to the Giga Information Group[14]. Examples of WebSphere's additional capabilities are:

  Container Managed Persistence: Relational data accessed via **entity Enterprise Java Beans (EJBs)** can be managed by the EJB Container in the WebSphere Application Server. This means that the customer programmers do not have to deal with the intricacies of interacting with the database management system. But WebSphere does more than "simply" read and write data. It also caches the entity beans to make data retrieval more efficient and improve system performance; and safeguards the integrity of data used by multiple concurrent applications by controlling access to the data. In .NET these functions must be implemented by the user**.**

  JDBC: Programmers who need greater control of their relational data can also use JDBC, which provides a vendor-neutral set of APIs for accessing their data. This means that an installation can change database management systems without changing the APIs in their applications. ADO.NET, on the other hand, uses vendor-specific APIs which must all be changed when changing database systems and thus promotes vendor lock-in**.**

  JNDI (Java Naming and Directory Interface): JNDI is a general directory mechanism provided by WebSphere/J2EE which allows applications to exchange data, such as the locations of remote servers, at run time in a consistent manner. .NET does not have this and requires the user to design and implement his own mechanism.

  J2EE/CA (Connection Architecture): J2EE/CA is a general architecture for accessing external enterprise systems, including legacy systems and products such as SAP. Many vendors, such as SAP and PeopleSoft provide J2EE/CA "resource adapters" to simplify access to those systems, much as database vendors provide JDBC drivers. .NET has no such mechanism. The only comparable product is Microsoft's Host Integration Server which provides only limited access to data on legacy mainframe and AS/400 systems. Furthermore, the access is provided by COM objects, not .NET objects, and .NET applications must use COM Interop to access their systems via the Host Integration Server.

[14]Giga Information Group, "J2EE vs. Microsoft and .NET: The Big Application Platform Battle," Sept. 8, 2001.

JMS (Java Messaging Service) **and Message Driven Beans:** JMS and Message Driven Beans allow J2EE applications to interoperate by passing messages. Microsoft's MSMQ and queued components do provide some message passing capabilities but, again, not as much as J2EE. For example, MSMQ cannot interoperate with messaging systems on non-Windows platforms. WebSphere's JMS servers can communicate with each other regardless of the underlying platforms. Nor does MSMQ support the "publish and subscribe" message passing model which allows multiple interested parties, or "subscribers," to receive messages about a specific topic from any number of sources, called "publishers."

Transaction Management: WebSphere/J2EE allows users to manage transactions within their applications in two different ways. First, the WebSphere Application Server can automatically manage transactions affecting entity beans (see "Container Managed Persistence" above) according to the user's specifications. These transactions are sometimes called "declarative transactions" because the user tells WebSphere how the transactions should be managed. Second, the WebSphere Application Server provides APIs which lets programmers explicitly manage transactions within their J2EE applications. These transactions are sometimes called "programmatic transactions." A single application can safely use both declarative and programmatic transactions; the underlying Java Transaction Service ensures that both types work together seamlessly. .NET Enterprise Services only provides declarative transactions; .NET does not provide programmatic transactions. .NET does have "manual" transactions which appear to be similar to J2EE programmatic transactions. But these manual transactions do not interoperate with .NET's declarative transactions, and .NET applications which use both can generate incorrect results. Furthermore, .NET's declarative transactions on Windows 2000 do not implement all the features expected of a transaction manager. To get these additional features, the user must upgrade to Windows XP or the Windows .NET Server which will be available next year.

Personalization: Personalization is the process of gathering and storing information about each visitor to a web site, and using that data on future visits to automatically display information matched to the visitor's interests, roles, and needs. WebSphere Personalization for Multiplatforms, Version 4.0 provides the capabilities to build and deliver personalized intranet, extranet, or Web site pages. Microsoft does not have a comparable product that supports personalization in all web sites. Today, only web sites created with the Microsoft Commerce Server can be personalized.

Service Flows: WebSphere Studio Application Developer Integration Edition increases programmer productivity by providing a tool, called the **Flow Editor**, that allows programmers to create new components from existing application artifacts, such as J2EE/CA adapters, Enterprise JavaBeans components, and Web services, by graphically indicating how they should work together. In particular, the **Flow Editor** represents the existing artifacts with colored shapes and lets the programmer define the sequence and flow of information between the artifacts by drawing lines between the shapes. WSAD-IE takes the finished drawing, which is called a **flow map**, and generates the necessary code files. Visual Studio.NET cannot do this; the .NET programmer would have to write all the code.

Optimistic Concurrency: Optimistic concurrency is a technique for improving the performance of database applications by reducing the amount of time that they restrict access to, or "lock," data within the database. Database locking prevents two users from accessing and changing the same data at the same time. Typically, applications lock the data when they start and unlock the data when they end, forcing other applications which need that same data to wait until the data is available again. Those applications will take longer to complete. Those applications are further blocked by the database system itself which must do more processing to lock and unlock the data. There are times, however, when the application developer knows that it's safe to leave the data unlocked because no other application will change it. But just to be safe, the developer double checks that the data hasn't been changed before writing her changes to the database. This process of confirming at the end that the data hasn't been changed instead of locking the data at the beginning is called "optimistic concurrency" because the developer is optimistic that no one will modify his data while he's using it. Because locks are used much less frequently, system performance improves because there is less application blocking and there is less overhead in the database system for lock management.

.NET's new data access component, ADO.NET, is designed to encourage use of optimistic locking, but then provides very little to help the developer actually use it her application. To use optimistic concurrency in her application, the user must write additional SQL code that checks whether or not the data has been changed, and write an additional method (program) to be called by .NET to inspect the results of the SQL statement and determine whether any data was changed. Optimistic locking is also available in WebSphere. But in WebSphere, the user doesn't have to write any code; the application server automatically handles all the details. WebSphere automatically writes the code that performs the checks, and WebSphere determines for itself whether any data was changed.

.NET deployments are difficult to reconfigure to handle changing workloads and add new functions; changes take more time to complete and are more costly than in WebSphere deployments.

- **.NET deployments must execute on processors running Windows.**

  Limits Ability to Consolidate Systems: Installations are increasingly looking to consolidate their applications on more powerful processors. This simplifies administration, while still meeting the demands of higher workloads, because fewer systems are required. But .NET deployments are limited in their ability to move to more powerful processors because they require Windows which only runs on Pentium-compatible processors. Thus, .NET deployments have fewer options for scaling up and are more likely to install additional processors to handle their increasing workloads, which complicates system administration and places additional burdens on administrators. WebSphere runs on many different processors and supports system consolidation.

  Limits Platform Choices: Because .NET can only execute with Windows, applications must be rewritten to run on other platforms such as Linux or Unix. For that reason, .NET restricts code reuse across heterogeneous systems and further locks customers into the

Windows platform instead of switching to another platform which would better meet their needs. WebSphere runs on many software platforms. Applications written for WebSphere can be used without change on other platforms to meet the changing needs of the enterprise.

- **Complete .NET functionality is not available on all Windows platforms. Users will have to upgrade their systems to use .NET.**

  Not all of .NET's capabilities are available on all versions of Windows. For example, Windows NT does not support ASP.NET, which is one of .NET's major components. Without ASP.NET, for example, developers cannot design web pages visually by simply dropping controls onto web forms. NT users who want to use ASP.NET will have to upgrade to Windows 2000 or a later version of Windows.

  .NET transaction isolation is not fully supported on either Windows NT or Windows 2000. Transaction isolation protects an application from the effects of other applications accessing the same databases at the same time. But this protection is costly and can impact overall performance dramatically. This performance penalty can, however, be reduced by reducing the amount of protection provided by transaction isolation. The .NET Framework defines four different isolation levels, but Windows 2000 only supports the strictest level. .NET applications will unnecessarily suffer the maximum performance penalty when they do not need full transaction isolation. WebSphere has allowed users to choose the appropriate level of protection for several years now.

  These problems highlight the fact that, because .NET is part of the Windows operating system, .NET users will be forced to upgrade to the latest version of Windows to obtain full .NET functionality. WebSphere is not part of any operating system. Installations are not required to change their systems software to use the full extent of WebSphere capabilities.

- **Deployed .NET distributed applications are difficult to maintain because .NET has no automatic location service.**

  J2EE defines a single mechanism, JNDI, for maintaining information about remote object servers. When a remote object type is made known to a client machine, the network address of the remote object's home is stored in the client's JNDI repository. All applications on the client machine locate the remote object's host by using the entries in the JNDI repository. Also, the WebSphere Application Server will automatically update the JNDI repository whenever it gets new information about remote components. There is no such mechanism in .NET. .NET does not use the registry, as DCOM does, to maintain this information. Active Directory provides some of JNDI's features, but it is not required in .NET and is very difficult to install and administer. Therefore, each application owner is left to choose a suitable mechanism such as a configuration or resource file, database table, or hard-coded constant in a program. Given such diversity, it's impossible for an administrator or system tool to upgrade all applications simultaneously when remote object servers are moved or added. Each application owner must change her application individually, and if the application isn't updated by the time the network change occurs, the application will fail.

- **.NET deployments are brittle because developers must know how they will deploy their applications before they start writing code. Changes to the deployment require that the entire application be rebuilt.**

  When an application is built with Visual Studio.NET, the developer loses control of how individual programs are grouped into .exe and .dll files. Visual Studio requires that each program in an application be placed into a logical collection of files, called a *package*, when it is first created. Then when the application is built, Visual Studio creates exactly one .exe or .dll file per package, regardless of the number of individual programs contained in the package. If the application deployer should need to collect the individual programs into different .dll and .exe files, perhaps because of changes to how the application is deployed on different tiers, he or a developer must move source programs between projects, make administrative changes to each project that uses any of the moved programs, and rebuild the entire application. The larger the application, the more likely that the first few rebuild attempts will fail because some necessary administrative changes were missed. This problem does not occur with WebSphere tooling. Applications are deployed by means of *EAR* (Enterprise Archive) files. The deployer always has access to each of the EAR's components and is free to reassemble them into deployable files as he deems best.

- **ADO.NET applications require significant source code rework to change database management systems, thus increasing deployment time and causing vendor lock-in.**

  ADO.NET is .NET's mechanism for accessing relational databases. To obtain the best performance for database applications, programmers must use vendor specific APIs in their source code and vendor specific data controls in their Web Forms and Windows Forms. This application architecture makes it very difficult to change database management systems (DBMS), thus increasing development time and expense, should you find your current DBMS to be inadequate. All source code in the entire application must be modified to replace the current DBMS vendor's product-specific APIs with the new vendor's product-specific API's. All GUIs built with Web Forms and Windows Forms containing data controls must be rebuilt *from scratch* to use the new vendor's data controls. WebSphere supports the JDBC standard which is supported by all DBMS vendors; there is no need to replace any of the JDBC APIs contained in the source code when changing the DBMS. This simplifies migration to a different DBMS, and allows each WebSphere customer to use the DBMS that best meets her needs.

- **Windows cannot load balance all .NET remote objects. Users cannot exploit the full power of their processors.**

  Load balancing directs new requests to the processor that is the least utilized, and thus able to handle the request in the least time. Without load balancing, new work will be directed to a fully loaded system even when another processor is lightly loaded. This work will be forced to wait even though another processor in the system could handle it immediately. .NET Remoting is the new mechanism for managing remote .NET objects. It allows three different kinds of remote objects, but only one type, called "single call" remote objects, can be load balanced. The other two types will be directed to specific processors regardless of how busy those processors are. The problem with single call remote objects, however, is that they cannot maintain state. This means that they cannot remember anything about previous requests from the same user; they start with a clean slate. For example, if a stateless single call object was used to implement an ATM machine, the user would have to type in his PIN every time he responded to a prompt - not just in the beginning. In this case, additional code would have to be written to save the user's PIN to a place that could be accessed by any processor. WebSphere entity EJBs, which are used to model business objects, can both maintain state and be load balanced. Requests involving entity EJBs will be directed to the least utilized processor.

- **Applications may fail because Visual Studio.NET cannot determine which files are needed by legacy components, such as COM .dlls, for correct execution.**

  To deploy an application with Visual Studio.NET, the programmer must create a deployment project which names all the executable files required by the application for proper execution. .NET applications can be built with legacy code, but execution is likely to fail because Visual Studio cannot detect what files are required by the legacy components, and will not add them to the application's deployment project. As a result, Visual Studio will build the application without error, and both the programmer and end-user will be completely unaware that a problem exists until the application fails at run time because necessary files are missing. To correct this problem, the developer must determine **all** the files that are required by the legacy code. This is a difficult and tedious task, especially when the source code is not available, and sometimes requires contact with the author of the legacy code. WebSphere programmers do not face this problem because there is only one object format.