J2EE and Microsoft .NET

An Oracle White Paper
April 2002

ORACLE

Table of Contents

"Two major application platforms now dominate the enterprise application development market: Java 2 Platform, Enterprise Edition (J2EE) and the Microsoft platform.

The decision about which platform to use is a business decision, but technology factors can have significant business impacts."

Giga, September 2001

THE LANDSCAPE

The Internet build out through the 1990s created a new set of requirements for business applications. Browser based application delivery has been proven to reduce costs and to grow business opportunities. Partners now expect direct access to, and integration with, back-office systems. Employees demand access to their business applications both in the office and on the road. Customers assume products and services are available 24 hours a day.

New terminology has emerged to describe the types of applications that organizations now require. Portals are the new entry point into all business applications. Content management systems automate the production of web sites. Business intelligence is used to personalize applications to specific users. And, Web services are the latest channel of application delivery that includes browser, voice and wireless.

Technology vendors and organizations have raced to adapt their infrastructures to work in this new world. At the same time that this change in business requirements occurred, two major application development models emerged: Java 2 Enterprise Edition (J2EE) and .NET. J2EE is the server side application model for Java applications. .NET is the latest evolution of Microsoft's server side application model.

These two application models were designed to take advantage of new opportunities revealed by the Internet. However, while founded upon similar architectural principles, they are built from two different philosophical backgrounds -- J2EE targets broad adoption based on open standards; .NET targets broad adoption based on the adoption of a single operating system, Windows.

Among technologists this has resulted in an ongoing debate: J2EE versus .NET. Senior management, however, has not overlooked the debate. Internet-based applications have been proven to reduce costs and create competitive advantage. Astute managers realize that they must also understand these issues in order to make effective information technology investments.

INTRODUCTION

Both J2EE and .NET come loaded with preconceptions that make the decision difficult to understand and to evaluate. Which is a product versus a specification? Can they coexist or are they compatible? Which is more mature and proven? Does one offer more than the other?

These questions need to be mapped into the context of every organizational environment. Will the decision be made within a homogeneous or heterogeneous infrastructure? What are the skill levels of the technical resources? Does the organization build or buy applications? How aggressively does the organization adopt new technology? How long are systems kept operational?

"Considerable confusion surrounds .NET, largely due to Microsoft's rather indiscriminate use of the term.

It is important, therefore, that when enterprise managers consider .NET, they first consider all possible meanings of this term, and then answer the questions in the appropriate context."

Gartner, April 2001

The choice between these two architectures is not one to be taken lightly. For short-term tactical projects either application architecture is likely to suffice. For longer-term, strategic architectures, the choice must be made more carefully. Choosing one over the other can have significant future business costs and benefits. These must be fully understood before a decision for J2EE or .NET is made.

LEVELING THE PLAYING FIELD

To make a valid comparison between these two architectures, a common basis has to be established. This comes in two areas:

- **Product versus Specification**
  J2EE is a specification with multiple product implementations and .NET is a mixture of product and specification. An analysis of J2EE and .NET has to ensure that appropriate product-to-product comparisons and specification-to-specification comparisons are made.
- **Architecture**
  Both J2EE and .NET provide a server side runtime environment and a set of runtime services for applications. At a high level, these appear remarkably similar; however, upon close analysis, they differ in significant ways.

Product Versus Specification

The first thing to understand about the debate is what is being compared. J2EE on its own is a specification. Multiple vendors implement the J2EE specification in application servers, and multiple vendors build product offerings upon those implementations. Oracle, IBM and BEA, for example, provide highly competitive J2EE certified application servers.

J2EE is a part of the broader Java platform, which is broken into three editions: Java 2 Standard Edition (J2SE), Java 2 Enterprise Edition (J2EE) and Java 2 Micro Edition (J2ME). These Java specifications evolve through a standards process called the Java Community Process to which most major technology vendors except Microsoft belong.

.NET is a moniker for many different things from Microsoft. At a programming level, it represents a server-side application model with characteristics similar to J2SE, J2EE and J2ME. What confuses the comparison with J2EE is that Microsoft .NET is also a catchall branding for Microsoft's product offering built on top of this programming framework.

"The shared vision between J2EE and .NET is that there is an incredible amount of 'plumbing' that goes into building web services, such as XML interoperability, load balancing and transactions.

Rather than writing all that plumbing yourself, you can write an application that runs within a container that provides those tricky services for you."

The Middleware Company, June 2001

Comparing the Platforms

A more reasonable basis for comparison is to broaden the comparison to the overall platform rather than attempting to compare individual specifications. This can be done on four levels: Application Model, Server Implementation, Product Offering and Online Services as shown in Table 1.

The application model is the set of application programming interfaces exposed by J2EE and .NET. The server implementation is typically an application server on the J2EE side whereas it is the Windows operating system on the .NET side. The product offering is the product set provided by each vendor or third parties. Online services are hosted versions of the product offering.

Comparison Level	Java	Microsoft
Application Model	**J2EE Specification**	**.NET Framework**
Server Implementation	**Multiple Vendors**	**Microsoft Windows**
Product Offering	Multiple Vendors	Multiple Vendors
Online Services	Multiple Vendors	Multiple Vendors

Table 1: Comparing Platforms

What becomes abundantly clear from this high level comparison is that for J2EE the server implementation is implemented and offered by multiple vendors based on a documented specification. With .NET one vendor, Microsoft, offers a .NET server implementation based on a single vendor framework.

Not so apparent, but equally important, is that the J2EE specification is developed and managed in an open industry committee called the Java Community Process. With .NET, one vendor, Microsoft, manages the core .NET specification.

The issue illustrated here is choice. Choosing .NET represents choosing Microsoft. There is no other choice. Choosing J2EE means choosing a market of vendors who will compete for the initial and ongoing infrastructure business. Vendors in the J2EE market will compete on cost, performance, product depth and other characteristics. Having chosen .NET there is no comparable, ongoing competition.

Architecture

J2EE and .NET are both based on the concept of a 3-tier architecture. The goal is that the construction and delivery of the user interface is separate from the

construction of the business logic, which in turn is separate from the backend data or infrastructure being accessed.

This type of architecture depends upon a middle tier container providing a number of services to enable scalability, reliability and availability. Depending on the implementation, the container typically offers a rich set of services for security, transactions and connectivity. Both J2EE and .NET architectures provide developers with this core infrastructure rather than requiring them to build it.

"Standards are not a vendor thing. They are an end-user and buyer thing.

If you – the user, buyer or manager of technology – can't easily substitute one competitive offering for another, then you should immediately question the benefit of the vendor's claim to be standards compliant."

Tech Update, January 2002

Figure 1 illustrates J2EE within an overall Java architecture. It is broken into 4 major layers:

1. **Presentation and Access.**
   For thin client applications, J2EE has Java ServerPages (JSP) for tag oriented dynamic HTML pages and servlets for programmatic HTML pages. Java Foundation Classses (JFC) are used for rich, complex clients. Web services are used for programmatic access.
2. **Business Logic.**
   J2EE provides an extensive business component model including Session Enterprise JavaBeans for business transactions and Entity Enterprise JavaBeans for persisting transactional data. Message Driven Beans are used for asynchronous business processes.
3. **Connectivity.**
   J2EE has a standard database protocol, Java Database Connectivity (JDBC), for database access. Java Connector Architecture (JCA) is used for host-based application access. Java Messaging Service (JMS) is used for asynchronous business processes. An extensive Java API for XML is provided for mapping between Java and XML protocols such as SOAP.
4. **Runtime.**
   All Java applications use a common runtime, the Java Runtime Engine (JRE) to execute business applications. Java code is compiled to Java byte code and runs on any platform with a JRE.

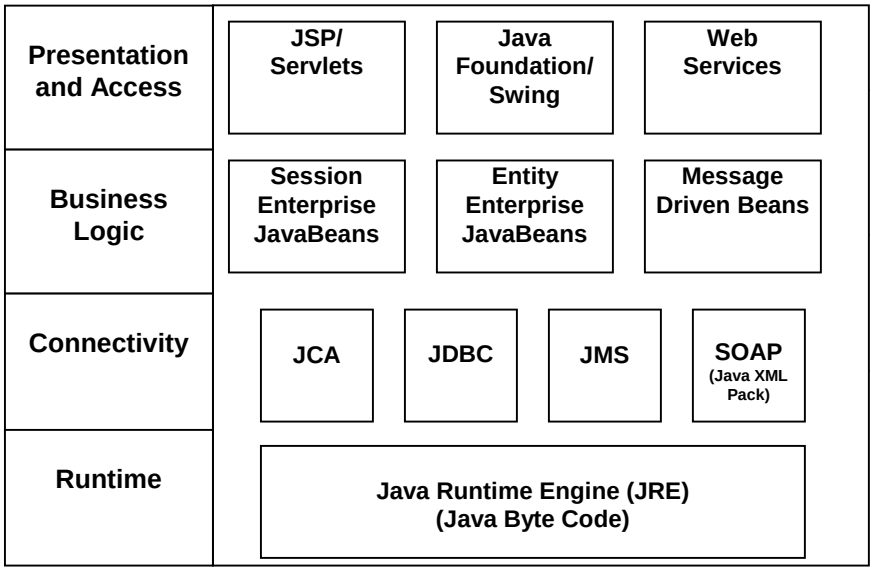

Figure 1: Conceptual Java Architecture

"Microsoft's IL runtime has at least one notable, if improbable, goal: eliminate the programming language as a barrier to entry to the framework.

Java eliminates the platform barrier …"

Jim Farley, Oreilly, 2001

Figure 2 illustrates a conceptual .NET architecture. It too is broken into 4 major layers:

1. **Presentation and Access.**
   For thin client applications, .NET uses ASP. NET for tag oriented dynamic HTML pages. Windows Forms are used for rich, complex clients. Web Services are used for programmatic access
2. **Business Logic.**
   .NET provides two major kinds of business components: .NET Managed Components for synchronous business transactions and COM Queued Components for asynchronous business transactions.
3. **Connectivity.**
   ADO .NET is used for database access. An XML API is provided for mapping .NET Components to XML protocols such as SOAP.
4. **Runtime.**
   All .NET applications use a single runtime engine, the Common Language Runtime (CLR) to execute business applications. Applications can be written in multiple languages and compiled to Intermediate Language byte code and executed in the CLR. The CLR is supported on one platform, Windows.

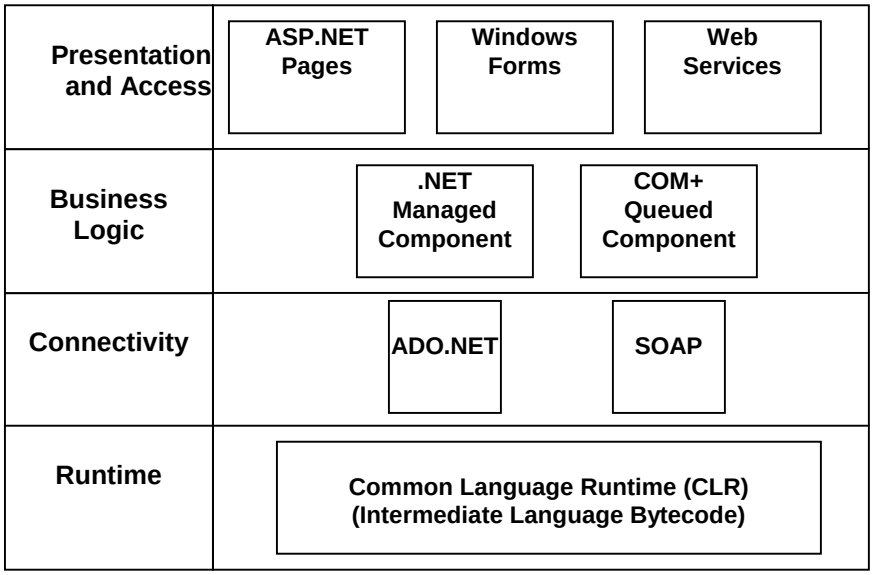

Figure 2: Conceptual .NET Architecture

Comparing the Architectures

At a high level the two architectures look remarkably similar. Both grew out of research in the 1980s and 1990s into multi-tier architectures and provide compelling, high business value implementations of this concept.

However, five major areas differences emerge upon closer inspection: architectural specification, language centricity, architectural depth, and Web services.

"The subset of the .NET Framework submitted to ECMA is just enough to build a C# language compiler and its supporting runtime.

The problem, unfortunately, is that this technology isn't particularly interesting for real-world solutions because it's completely insufficient to build robust, portable .NET applications that you could potentially deploy across platforms. …"

Mark Driver, .NET Magazine, April 2002

Architectural Specification

An analysis of the architectures reveals a critical difference between J2EE and .NET: Every aspect of the J2EE architecture is defined in documented specifications.

Developers merely have to go to the Java Community Process web site (http://www.jcp.org) to find detailed specifications for each component of the architecture. These specifications are used by developers to understand the architecture and by vendors to help them implement the architecture in products.

In addition to the specifications, an important characteristic of the Java Community Process is that each specification has a reference implementation. So, for example, the J2EE specification comes with a stand-alone, vendor independent reference implementation that developers can use to ensure vendor products match the specification.

In contrast, specifications do not exist for the .NET architecture. Rather the .NET architecture is documented within the different Microsoft product offerings. The reference implementation is the Microsoft .NET products themselves.

To counter this proprietary perception, Microsoft has submitted portions of the .NET framework to standards bodies. In particular the programming language C# was recently accepted by European Community Standards Association (ECMA). Microsoft does not appear to want to carry that standardization to the extent of the Java Community Process for its entire platform. Estimates by some industry analysts suggest that this submission represents less than 10% of the .NET framework.[1]

Language Centricity

Another notable difference between the two architectures is the language centricity of each. Microsoft claims over 20 different languages are supported in .NET. In the J2EE environment Java is the programming language and other languages are available through the Java Native Interface API or through connectivity API's like the Java Connector Architecture.

The reality of language use in the Microsoft .NET environment is two major languages are promoted and used for business applications: Visual Basic .NET and C#. With Visual Basic .NET, Microsoft undertook a major re-write of Visual Basic, their most popular language, in an effort to tailor it to .NET. Microsoft also created C# specifically for the .NET platform to give developers a lower level programming language than Visual Basic .NET but not as complex as C++. The

[1] Giga Application Platform and Development Strategies 2002,Florida, March 12, 2002.

result in the Microsoft community has been significant confusion over which language to use.

The Java development community is now recognized to be one of the largest amongst programming languages. IDC, for example, expects the number of Java developers to increase to over 4 million by the year 2004.[2] The popularity of Java is driven not only by the adoption of the J2EE architecture but also the technical education system worldwide where Java is typically used for both introductory and advanced programming classes.

"… Giga Information Group, based in Cambridge, Mass., found that 78 percent of the group viewed J2EE (Java 2 Enterprise Edition) server software as the most effective platform for building and deploying Web services.

Microsoft's .Net, which enables users to build Web services for Windows server operating systems, accounted for 22 percent of the votes …"

InfoWorld, December 2001

At a lower level, the Common Language Runtime engine underlying the multi-language support in .NET, has been subject of considerable debate. It is a compelling technical feat, but is unproven in large-scale deployments. Two areas tend concern architects with the CLR: managed code versus unmanaged code and overall performance, reliability and scalability.

By providing a common runtime for so many different languages, Microsoft has had to abstract away many of the native characteristics and datatypes of the individual languages. Microsoft offers a way around this using a concept of unmanaged code. Unmanaged code gives access to native language capabilities but leaves compatibility with .NET to the developer.

Perhaps more important is concerns with performance, reliability and scalability. Clearly the CLR is a new technology and there is little real-life experience with large-scale .NET deployments. It will take a number of years before a substantial body of experience and best practices are available for organizations deploying the CLR on a large scale.

In contrast, the Java Runtime Engine has over seven years of enterprise level deployments on multiple platforms and continues to see ongoing architectural improvements driven from real-life experience. Proven techniques exist for multiple language support using the Java Native Interface and interoperability standards like the Java Connector Architecture and Java Messaging Service. Performance concerns are not only addressed through well-understood best practices for managing JRE's in a highly scalable, reliable and performance oriented environments, but also by a highly competitive market of JRE providers.

Architectural Depth

A key area that architects will notice about the two architectures is that J2EE has defined a finer grained set of services at the business component level and at the connectivity or interoperability level.

At the business component level, J2EE provides two options, Session Enterprise JavaBeans for business transactions and Entity Enterprise JavaBeans for persisting business data. Session Enterprise JavaBeans support both stateless and stateful

[2] IDC, June 2001

business objects. In contrast, .NET does not provide stateful business objects, nor does it provide a component model for persisting business data.

The argument Microsoft offers for not providing these services is in their opinion most applications do not require them. The counter argument from the J2EE camp is that there are many common business and technical scenarios where these approaches are appropriate and rather than requiring developers to invent a programming paradigm, they should simply be available in the architecture.

"… think about what platform you would choose if you were an ISV or consulting company.

Most likely you would have customers on a variety of platforms, and therefore you'd probably adopt the architecture behind J2EE because you don't want to lock yourself out of deals."

The Middleware Company, June 2001

At the connectivity and interoperability level, J2EE specifications provide a number of APIs including Java Database Connectivity for database, Java Messaging Service for asynchronous connectivity and the Java Connector Architecture for Enterprise Information Systems. Microsoft offers a mix of technologies such as Microsoft Message Queuing (MSMQ) and products like Microsoft Biztalk.

Unlike one vendor interoperability implementation of the Microsoft .NET environment, the existence of standard interoperability API's in the J2EE platform has created a competitive market of vendors offering differentiated implementations and value-add products on top of these specifications. This market is also driven as packaged application vendors like Oracle, SAP and Peoplesoft adopt Java within their core application architectures.

For architects these differences point to capabilities in the J2EE architecture that tend to be more extensively described, documented and implemented than in the Microsoft platform. To some degree this broader connectivity and interoperability capability represents a more diverse set of vendors in the Java Community Process than the participants in the Microsoft product development process.

Web Services

Web services are garnering a significant amount of interest and investment by many organizations. By offering XML-based programmatic access into backend business applications, organizations have a new and cost-effective way to re-use their technology investments.

Both J2EE and .NET provide extensive Web services infrastructure. For J2EE, Web services are simply a natural extension to an already proven architecture using a set of standards called the Java Application Programming Interfaces for XML, also known as the Java XML Pack. For .NET, Web service support is a result of a significant re-write of the underlying Microsoft infrastructure.

Concern exists that Microsoft may try to extend the .NET Web services infrastructure in proprietary ways and lock users into a Microsoft only environment. To counter proprietary tendencies in this area, the Web Services Interoperability Organization (WS-I) has been created as a vendor neutral body to certify interoperability between Web service implementations. Vendors like Oracle, IBM and Microsoft co-founded WS-I to ensure standards compliance in this area.

UNDERSTANDING THE DECISION

The discussion until this point has focused on leveling the playing field for the technical comparison between J2EE and .NET. In addition to such a technical analysis, the decision to use one architecture over another should be looked at from a business and techical risk perspective.

"… Java usage has not merely become more widespread – it has become a mainstream platform for e-business IT efforts.

Eighty percent of respondents reported that their applications development (AD) organizations use Java technology …"
Gartner Summit Survey, February 2001

Business Risks

Organizational Environment

Each organization has an existing environment that must integrate with any new corporate wide architecture. Past experience, existing investments, approach to technology, and other factors play a part in the decision. J2EE and .NET each bring along characteristics that will fit into the environment of some organizations better than others.

A homogenous technical environment will typically incline an organization to one technology or another. If the organization's entire infrastructure is based upon Windows, upgrading to .NET would appear to be a logical choice. Conversely, if an organization is homogeneous in Unix or heterogeneous across multiple operating systems and hardware platforms, the ability for Java applications to run on any platform and operating system make J2EE a logical choice.

Interestingly, even in a Windows centric environment, J2EE still has significant appeal. Every J2EE vendor provides an application server implementation on Windows and Linux because the Intel architecture is seen as a cost-effective hardware platform. Using J2EE in this context gives organizations the flexibility of J2EE while accruing the benefits of low cost Intel hardware. And, if required, the ability to move to different hardware platforms remains.

Vendor Choice

The ability to choose and replace vendors is a high business value characteristic of an architecture. Without competitive offerings, an organization can become locked to a particular supplier of technology with no other option than to re-build the entire architecture.

Clearly in this space the portability aspect of J2EE makes it very attractive in terms of vendor choice. In 2002, over thirty-seven vendors were J2EE licensees and seventeen sold J2EE compatible application servers. These range from free open source application servers like JBoss to enterprise class application servers like Oracle9*i* Application Server. From this a highly competitive Java development tools market has also emerged.

This choice enables organizations to choose vendors that provide the best functionality for specific price points. Further, should an individual vendor fail to meet expectations there is a credible negotiation tactic to move to a competing vendor implementation.

Vendor choice also plays a role in terms of investment into an overall application development and deployment environment. With Microsoft .NET there is one and only one operating system choice for development and deployment: Microsoft Windows. As a result there is one price point to choose from – Microsoft's.

With J2EE, a common scenario is to develop and test on low cost Intel-based machines and seamlessly deploy to multiple hardware and operating systems. Organizations can choose from multiple price points, easily take advantage of existing investments and tailor infrastructure investments directly to requirements.

Completeness of Platform

At the core level both J2EE and .NET are infrastructures upon which business applications are deployed. In many cases a specific vendor will be chosen to provide the infrastructure (J2EE or .NET) and also a product offering for that infrastructure.

Table 2 provides an example deployment to a J2EE infrastructure based upon Oracle9*i* Application Server and a deployment to Microsoft .NET. There are two important issues to notice with this comparison.

1. The comparison between J2EE and .NET architectures extends to product levels. Vendors like Oracle not only compete at the infrastructure level but feature-by-feature with the Microsoft product offering.
2. Platform vendors in the J2EE space also compete at the best-of-breed feature level against the broader market of other J2EE compatible products. This level of competition in the J2EE market keeps vendors like Oracle constantly innovating whereas this competitive drive is not so apparent for single vendor controlled architectures like Microsoft .NET.

Feature	Java	Microsoft
Component Model	J2EE	.NET
Application Server	Oracle9*i* Application Server	Windows
Database	Oracle9*i* Database	SQL Server
Development Tools	Oracle9*i* Developer Suite	Visual Studio.NET
Portal	Oracle9*i*AS Portal	Sharepoint Portal
Content Management	Oracle9*i* Internet File System	Content Manager
Business Intelligence	Oracle9*i* AS Business Intelligence	Not offered
Caching	Oracle9*i*AS Cache	ISA Server
Mobile	Oracle9*i*AS Wireless Edition	Mobile Information Server

Table 2: Oracle9*i* Application Server and Microsoft.NET

"The transition to .NET will create serious discontinuities (e.g. new skills, new architectures and new tools) that will challenge developers through 2004."

Gartner, July 2001

Resource and Skills Availability

The debate of J2EE versus .NET is fundamentally about making a long-term architectural investment and then carrying out an implementation of strategic business applications. An architecture, however, is insufficient without skilled resources to develop, deploy and manage that architecture. Both J2EE and .NET bring significant resources to this area though with different issues to consider.

The J2EE architecture has been in use since 1997 and has a significant following of developers and architects. A recent Gartner study showed that over 80% of IT organizations use Java in their application development organizations.[3] This broad adoption and length of use makes it easier for development organizations to find skilled Java resources.

Microsoft also brings a heritage of a large developer community. However, a major issue for Microsoft is that the recent release of the .NET development environment, Visual Studio .NET, is a significant re-architecting of a popular product. As a result, it can be difficult to find veteran, skilled .NET developers as compared to traditionally skilled Visual Basic developers, Microsoft's largest community of developers.

Contracting skilled 3rd parties to assist in training and implementation can offset lack of availability of in-house skilled developers for J2EE and .NET development. Consulting firms exist that specialize in both J2EE and .NET architectures. The question that needs to be asked of any consulting organization is their commitment to one architecture or another. The broad adoption of J2EE on multiple platforms including Windows may drive consulting firms to choose J2EE for larger market opportunities and maximum flexibility.

The final area of resource availability that needs to be looked at is the availability of partners on the platform chosen. These can be categorized into four categories: independent software vendors, infrastructure partners, system integrators and service providers. A platform offering in J2EE or .NET without depth in these areas will not succeed regardless of the quality of technical implementation. Oracle9*i* Application Server, for example, brings over 900 partners to its J2EE offering.

Technical Risks

Architectural Risks

A decision to deploy a business application must be preceded by a careful analysis of the quality of service available from the underlying J2EE or .NET architecture. Both architectures claim strength in this area, J2EE from a multi-year proven record of successful deployments, and .NET from a re-architecting of the underlying platform.

[3] Gartner, February 2001

Where the largest architectural risk lies with .NET is the dependency on Microsoft as a sole vendor. As reliability, scalability, availability and other quality of service characteristics are added to the .NET platform, Microsoft customers requiring those kinds of capabilities will benefit. However, the Microsoft customers are dependent on Microsoft for those capabilities.

"Microsoft's .NET platform represents a shift from PC-based Windows to the Internet-based Web services world. However, .NET alone cannot serve as the foundation of an enterprise's Internet strategy."
Gartner, November 2001

In contrast in the J2EE market there is a significant amount of differentiation between vendors in quality of service. For example, Oracle9*i* Application Server Release 2 has clustering capabilities at the cache level, at the web server level, the web component level and the Enterprise JavaBean level. This is contrasted with some J2EE development environments where a utility application server is provided but deployment is expected to be in proven environments like Oracle9*i* Application Server.

Platform Maturity

The maturity of an architecture can have a major influence on whether a project is delivered successfully or not. With maturity come best practices, highly skilled resources and an understanding of common problems with application development and deployment.

Here, J2EE would appear to have a significant upper hand. Over five years of enterprise deployments and over 37 different J2EE licensees represents a highly mature platform. To augment this market evidence, J2EE practitioners have also started publishing a series of J2EE best practices, often called the J2EE Design Patterns that document how common design issues are approached from a J2EE perspective.

In contrast, Microsoft .NET is a platform still under development. As of early 2002 the only production portion of the architecture was the Visual Studio .NET development environment. Microsoft has a long-term roll out plan for its corresponding Windows .NET servers over the next several years. As such, while an impressive technical achievement, Microsoft .NET should still be considered an immature platform.

Development

Microsoft has traditionally been seen as a highly successful vendor of development tools. With .NET it has redeveloped its tools offering into Visual Studio .NET, designed to fully exploit the architectural changes in the Microsoft platform. Microsoft's strength in this area has been giving developers a seamlessly integrated development environment.

The explosion of developers in the Java market has created intense competition in the J2EE tools market. Unlike the Microsoft market where Visual Studio .NET dominates, the J2EE market has a wide variety of development tools ranging from open source frameworks through to full-lifecycle integrated development environments.

"The release of Microsoft's new Visual Studio.NET marks the formal launch of .NET. It supports multiple languages, diverse clients, and Web services.

But .NET's steep learning curve means that firms should consider switching to the Java 2 platform instead."

Forrester, February 2002

Larger vendors like Oracle, with Oracle9*i* JDeveloper, are bringing tools to market that not only give integrated, end-to-end J2EE capabilities, but also provide an extensible development platform where open source, 3rd party tools and J2EE frameworks work together seamlessly.

As a result of this intense competition in the tools market it is not clear which of J2EE or .NET has an advantage. As in the server side discussion, the primary issue in development tools appears to be narrowing down to having more choice and price points available on the J2EE platform versus a smaller number of choices available on the .NET platform.

Coexistence

Many organizations will be faced with supporting both a Microsoft .NET and J2EE environments simultaneously either as a long-term co-existence strategy or as short-term transitional environment. While synergies across an integrated platform are not fully taken advantage in this scenario, some J2EE vendors like Oracle have made significant investments to make this strategy successful.

For example, with Oracle9*i*, an architecture aimed at fully taking advantage of J2EE, it has also been designed to integrate seamlessly into the Microsoft .NET environment. Many customers use Oracle9*i* to reduce costs and increase reliability, availability and scalability of the Microsoft .NET environment. Some of the coexistence and interoperability capabilities with Oracle9*i* include:

- Database integration with native ADO .NET, OLE DB .NET and ODBC .NET implementations
- Microsoft .NET transactional integration with Microsoft Transaction Server.
- Ability to proxy Microsoft Internet Information Server to the Apache Web Server of the Oracle9*i* Application Server
- Ability to include ASP .NET content in Oracle9*i* Application Portal
- Ability to cache Microsoft .NET web content using Oracle9*i* Application Server Cache
- Ability to interoperate with Microsoft .NET Web services both at development time with Oracle9*i* JDeveloper and runtime with Oracle9*i* Application Server Web Services.
- Interoperability through standard J2EE API's like the Java Connector Architecture and Java Messaging Service

Web Services

Those considering J2EE and Microsoft .NET often assume that Web services take away the need for choosing one architecture over the other. It is reasonable to

expect, given the focus of Web services is interoperability, that J2EE Web services should be able to easily communicate with .NET Web services.

There is good cause to believe that Web service interoperability will go further than vendor claims with the existence of high profile groups like the Web Services Interoperability Organization. Oracle, for example, does interoperability tests with .NET Web services to ensure customers of Oracle9*i* Application Server can take advantage of Microsoft .NET investments and Microsoft .NET Web services.

However, business applications, whether exposed as Web services or not, still need quality of service, transactions, security and other enterprise class services that are characteristic of the underlying architecture, whether it be J2EE or .NET, not of Web services per se. At a strategic level, Web services are simply another layer on top of the J2EE versus .NET decision and should only be weighed for the practical benefits provided rather than a deciding factor between the two architectures.

ORACLE'S STRATEGIC DIRECTION

Oracle's competitive differentiation has always been to implement best-of-breed products using open industry standards. Since 1979 Oracle has offered its products on multiple platforms and multiple operating systems, letting its customers choose the best price point and functionality rather than dictating only one choice. Open standards, however, are not enough to become the preferred vendor. Oracle has led the industry because it also builds high performance, reliable, available and scalable software using these standards.

The cross platform, multi-operating system and best-of-breed, high performance characteristics of Oracle's heritage are what led Oracle to choose Java and J2EE. In 1995 Oracle committed to Java and in 1997 Oracle began implementation of J2EE standards across its infrastructure.

Oracle has three products that provide a development, deployment and management environment for J2EE applications: Oracle9*i* Application Server, Oracle9*i* Developer Suite and Oracle9*i* Database. In addition, Oracle offers an extensive set of business applications, the Oracle E-Business Suite, covering both Enterprise Resource Planning and Customer Relationship Management built upon this same J2EE infrastructure.

Oracle9*i* Application Server

Oracle9*i* Application Server is a 100% standards based application server that provides a complete and fully integrated platform for running J2EE and Web service applications. Fully compliant with J2EE 1.3 standards, Oracle9*i* Application Server enables developers to take full advantage of the J2EE architecture on multiple hardware and operating system platforms.

The J2EE containers used within the Oracle9*i* Application Server are extremely lightweight and high performance. Benchmarks show that Oracle9*i* Application

Server runs applications twice as fast as comparable J2EE application servers, as shown in Figure 3. Likewise, in comparative benchmarks with Microsoft .NET, Oracle9*i* Application Server provides significantly better performance running equivalent applications as shown in Figure 4.

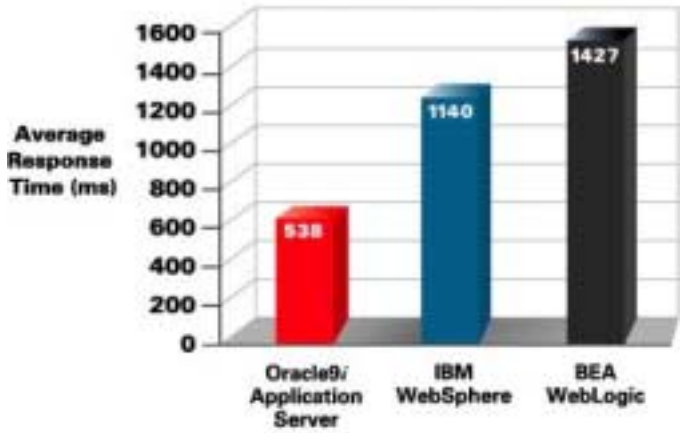

Figure 3: Oracle9*i* Application Server J2EE Benchmark

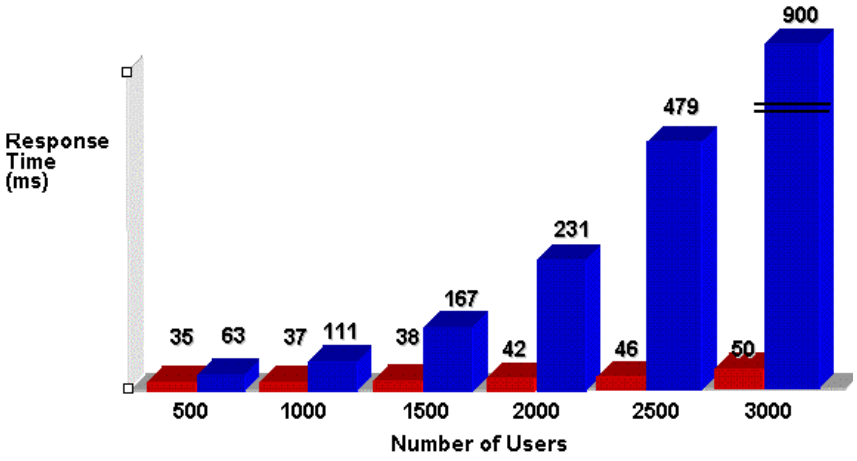

Figure 4: Oracle9*i* Application Server Competitive .NET Benchmark

In the J2EE market, Oracle9*i* Application Server also differentiates based on reliability, availability and scalability. Using advanced clustering techniques and built-in caching technology Oracle9*i* Application Server provides the industry's highest quality of service for a commercial application server.

The J2EE infrastructure of the Oracle9*i* Application Server is the basis for a complete and integrated platform offering. By building portal, wireless, business intelligence, management, security and integration capabilities using J2EE, customers get an unprecedented degree of integration and manageability from the Oracle9*i* Application Server. Figure 5 gives an overview of this broader perspective of the Oracle9*i* Application Server.

Figure 5: Oracle9*i* Application Server Capabilities

Oracle9*i* Developer Suite

"Written from the ground up in Java, Oracle has brought together everything you need for advanced UML modeling, project management, remote debugging, integrated execution and memory profiling, and even 1-click deployment into the embedded Oracle9iAS application server.

Oacle9i JDeveloper is one of the most responsive, complete and best integrated Java development environment that we've seen."

Rick Ross, Java Developer Journal

Oracle9*i* Developer Suite provides the industry's most complete and integrated development suite of tools for rapidly developing J2EE business applications and services. Oracle9*i* Developer Suite combines the power of the three most influential industry standards Java, XML and SQL to create a complete solution for business applications and service-oriented architectures. A key component of the Oracle9i Developer Suite is Oracle9*i* JDeveloper.

Oracle9*i* JDeveloper is an integrated Java development environment that supports the latest J2EE standards. Developing J2EE applications with Oracle9i JDeveloper is easy with a extensive set of wizards for common Java programming tasks, UML modeling, team support, code insight to help the developer writing Java, declarative editing of J2EE deployment descriptors and unique debugging and profiling features.

To accelerate J2EE development, Oracle9*i* JDeveloper also provides a J2EE Design Patterns framework with a visual workbench that developers can use to guide them developing industry best practice applications. The framework automates object relational mapping, user interface presentation binding and business logic development.

For Web services, Oracle9*i* JDeveloper provides the ability to develop Web services based on J2EE components. Java classes, servlets, Enterprise JavaBeans

and database stored procedures can all be published as Web services using SOAP (Simple Object Access Protocol), WSDL (Web Services Definition Language) and UDDI (Universal Description, Discovery and Integration). Web service clients developed with Oracle9*i* JDeveloper are completely interoperable with .NET Web services.

Oracle9*i* Database

The Oracle9*i* Database Server provides a comprehensive data management infrastructure to consolidate and manage data or Internet content from many different fragmented and disparate desktop and legacy data management systems. It is a proven, high performance database that scales to terabytes of data storage.

From a J2EE and .NET perspective the database is the persistence mechanism for business transactions. The Oracle9*i* Database has been optimized to work with both the J2EE and .NET architectures.

For J2EE developers, Oracle9*i* includes high performance JDBC and SQLJ database access drivers. In addition, Java developers can leverage the built-in Java engine in the Oracle9*i* Database to build business logic in stored procedures.

CONCLUSION

Choosing between J2EE and .NET is a strategic enterprise platform decision and should not be treated as a tactical technology decision. There are clearly significant technical issues that must be analyzed but ultimately the decision must be made from understanding the business issues. The future impact of costs, flexibility and risk are significant and often outweigh the initial investment in one architecture or another.

Oracle made a decision to embrace Java in 1995 based on the clear business benefits offered by that platform. The deep integration of Java and J2EE across the industry leading Oracle9*i* Application Server, Oracle9*i* Developer Suite and Oracle9*i* Database represent the fruition of that investment.

Now with the recent emergence of the Microsoft .NET architecture this decision has been further validated. The underlying business and technical values of open standards, choice, portability and interoperability weigh substantially more heavily in favor of J2EE than the one-vendor proprietary approach of Microsoft .NET.

J2EE and Microsoft .NET
Mike Lehmann
April 2002
Oracle

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com