

The Comparison of J2EE and .NET for e-Business

The Technical Report (hipic-10292003) of High-performance Information Computing Center at California State University, Los Angeles

Jongwook Woo
Computer Information Systems Department,
California State University, Los Angeles
90032-8530, Los Angeles, CA, USA
e-mail : jwoo5@calstatela.edu

Abstract

e-Business has been popular in the world since *Internet* and *World-Wide-Web* came out to the world. Its applications are based on *N-tier architecture*. There are two famous approaches to build the applications, which are *J2EE* and *.NET*. In this paper, *e-Business* and *N-tier architecture* are described. Besides, *J2EE* and *.NET* are compared based on criteria to build *e-Business* application.

Introduction

e-Business system has been popular in the world since *Internet* and *World-Wide-Web* came out. *IBM* defines *e-Business* as the leveraging of network capabilities and technologies in order to achieve and maintain the huge advantages for customers, suppliers, partners, and employees [SABGMM03]. *e-Business* activities can be classified into three categories based on end-users of transactions, normally on the Internet: *Intra-business*, *Business-to-consumer*, and *Business-to-business*. *Intra-business* activity is to share company information and computing resources among employees on the intranet: for example, knowledge management. *Business-to-business* activity is to improve inter-organizational partnerships and relationships: for example, supply chain integration. *Business-to-consumer*, the most common activity, is to provide services to consumers who is out of organizations: for example, customer resource management, *e-Commerce*, and web auctions etc [NAONO99].

The needs of the legacy *e-Business* systems were simple to maintain functionality and stability on the corporate computing environment. However, the needs do not become sufficient for the current high volume *e-Business* transactions. And, businesses need to handle high workloads and changing requirements by applying and adapting applications quickly. Businesses have to improve efficiency by integrating data and applications across the enterprise. Besides, the highest levels of performance and availability must be maintained for the critical businesses. Thus, *N-tier architecture* for *e-Business* system has been presented. In order to enable high performance, scalability, and availability to businesses, it is to partition systems and software to more flexible blocks that have their roles [Intel01]. Section 2 of this paper introduces *N-tier architecture* in detail.

Either Java, especially, *J2EE* (Java 2 Enterprise Edition), or *ASP* (Active Server Pages) has been exclusively used to build server site web systems for *e-Business* market. *J2EE* is the one of editions in *Java* that is a platform independent and object-oriented language - *Java* is the product of *Sun Microsystems*. Thus, *J2EE* is good fit to build *e-Business* systems for both a development and a server site in *Unix (Linux)* and *Windows* operating systems. Besides, the applications of *J2EE* are normally built in *Windows* operating system and published into any operating systems. *Microsoft Corporation* provides *ASP* for *e-Business* systems. *ASP* applications are integrated with the codes in *Visual Basic* or *C++*, etc. given by *Microsoft Corporation* as the products. Therefore, *ASP* applications are developed and published only in *Windows* operating systems.

The *Unix* operating systems have dominated the server market of banks and huge companies, etc. because the *Unix* systems are more stable than *Windows* operating system by *Microsoft Corporation*. Thus, *e-Business* systems of the server market have been mainly developed in *J2EE* instead of in *ASP*. *Microsoft Corporation* might have been threatened or might want to control the *e-Business* market so that it introduced the concept of *.NET* on June 2000. And, *.NET* has been presented to the market in 2002. *.NET* is not only platform independent but also programming language independent. *.NET* is not proven in the *e-Business* world yet but it has the potential power to compete with *J2EE* - probably stronger than *J2EE* in the small business.

In this paper, *.NET* and *J2EE*, the most popular *e-Business* development approaches, are compared in terms of programming language, platform, component, application server, and Database connection. Since they are the standards to build *e-Business* systems nowadays, this paper will be useful for people who want to select one of approaches. In this paper, the section 1 introduces the *N-tier architecture* that is the fundamental architecture of *e-Business* system. The section 2 describes the frameworks of *J2EE* and *.NET* in detail. The section 3 compares *J2EE* and *.NET* in terms of several factors. The section 4 summarizes the comparison. The section 5 is the conclusion.

1. N-tier Architecture (e-Business Architecture)

The traditional *Client-Server* architecture has a mainframe that includes core applications and data. The mainframe is accessed from thick clients that are big applications. We can call it *2-tier* architecture that is shown in Figure 1. The *2-tier* architecture has many loads between client and server because of their tight interoperations for its presentation logic, business logic, and data access logic. As shown in Figure 1, Client has not only the operations of presentation logic but also the part or the full of business and data access logics. This tight interoperation has generated many issues in the current high volume business systems. It is not scalable because it should replace the entire system when its capacity is exceeded. And, it is not flexible because its presentation logic, business logic, and data access logic are tightly coupled. If the developer wants to modify its business logic, he/she should modify the entire logics. Besides, the developer must adapt or modify the business logic when it is integrated with the World-Wide-Web or other applications [Intel01].

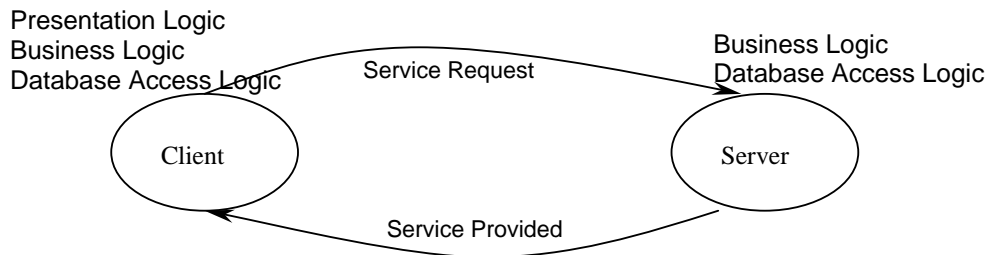


Figure 1. 2-tier Architecture

The *N-Tier* architecture has addressed the issues of the 2-tier architecture and become the solution of the current e-Business systems on Internet and World-Wide-Web. It partitions application functionalities into *N* independent layers, mainly three layers as in Figure 2. Thus, it becomes easier to integrate with the existing business systems. The layer 1 is the presentation logic that is typically hosted on Web server with web browser. The presentation logic is to send the request of client and receive its response from business logic. The response is normally dynamic or static web pages formatted to present the client. The layer 2 is hosted on mid-tier (middleware) server as business logic. It includes the business functions that are the main of the e-Business applications on *N-tier* architecture. It produces the response of the request from the client and provides it to the

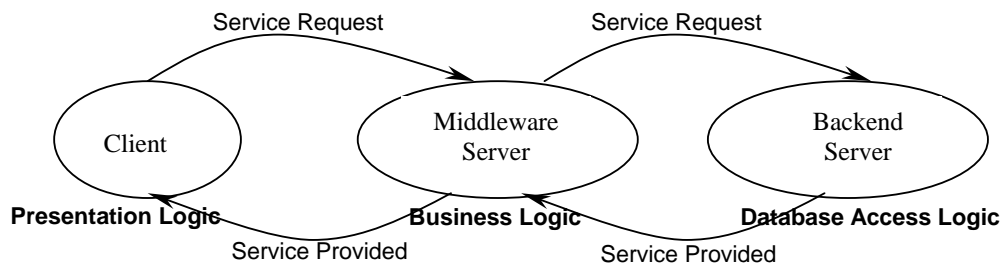


Figure 2. *N-tier* Architecture

client. If the request is related to access data, it will pass the data access request to the back-end database server. The layer 3 is hosted on the back-end database server as database access logics. It is to handle the request of data source from the business logic. It has the functions to access data source, that is, database. Since business logic is separated from presentation logic and database access logic physically, each layer can be scalable and upgradeable independently. And, even if a layer is modified or replaced, the application of other layers do not need to be recreated. Besides, each layer can be implemented with clustered servers for its logic. The clustering enables high-performance computing, availability, and scalability [Intel01]. Therefore, the current e-Business systems are implemented on *N-tier* architecture.

2. The J2EE and .NET

2.1 J2EE

Java platform is composed of Java Application Programming Interfaces (APIs) and Java Virtual Machine (*JVM*) as shown in Figure 3. Java programs are interpreted – compiled – to Java byte codes that are executable on *JVM*. *JVM* interprets byte codes for native operating system of the computer system. In other words, the byte codes are translated to target languages – machine codes – in order to run on the computer system. Thus, Java byte codes can be executable on any operating system if its *JVM* is installed. That is, Java is a platform independent language that reduces the cost to adapt the existing applications to new systems.

Java APIs are a set of built-in libraries as byte codes. Java 2 platform Enterprise Edition (*J2EE*) defines the standard for *N-tier* architecture [Sun03]. *J2EE* has the extended APIs from *J2SE* (Java 2 platform Standard Edition). It is based on the *J2EE* components for modularization and to simplify the development cycle by providing the details of application behaviors. Thus, it enhances a developer to focus on the business logic without implementing the expensive applications such as transaction, security, database management, and naming service, etc. *J2EE* includes the features of Java 2 platform Standard Edition such as platform independency and object-oriented language. Besides, *J2EE* supports for enterprise systems *JDBC* API for database access, *Enterprise JavaBeans (EJB)*, *Java Servlets* API, *JavaServer Pages (JSP)* API, *XML*, *Java Mail* API, and *Java Messaging* API etc. As are *J2SE* codes, *J2EE* source codes are compiled to Java byte codes and run on *JVM* (Java Virtual Machine) that converts Java byte codes to the machine codes. Most operating systems support *JVM* so that a code runs on an operating system should be executable on other operating systems, which meets the policy of write-once-run-anywhere from Sun Microsystems.

In order to execute *J2EE* codes, a *J2EE* application server is needed as well as *JVM* as shown in Figure 4. There are many application servers in the market such as *BEA WebLogic*, *IBM WebSphere*, *ATG Dynamo*, and *Sun One Application server*, etc. And, to connect databases, *JDBC* driver is needed for each database. Normally, each database vendor provides its *JDBC* driver.

J2EE has been popular to implement e-Business applications because it is platform independent and has higher performance comparing to the legacy *CGI* systems with *Perl* and *C++*. Microsoft Corporation's *ASP* is another competitor to build e-Business applications but it is only for Microsoft Windows system with the exclusive *IIS* web server. Thus, *J2EE* has been the most popular tool to build e-Business systems in the market.

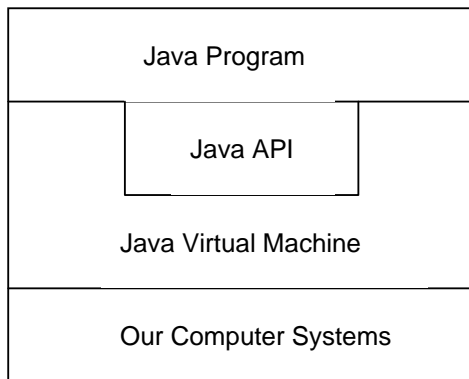


Figure 3. The Java Platform

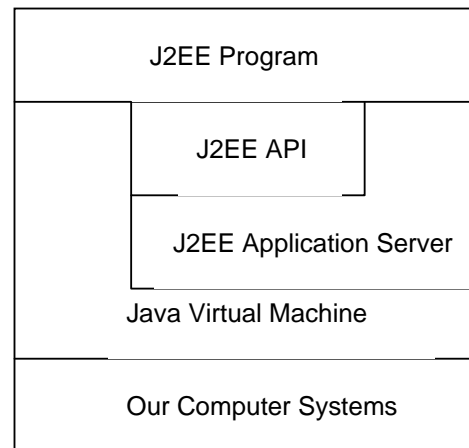


Figure 4. Application Server for J2EE

2.2 .NET

Microsoft Corporation is the most famous for Windows operating systems in the personal computer market. *Microsoft* has not only focused on the Windows server market but also its components such as *COM* (Component Object Model). Thus, they have produced Windows server products such as the current *Windows 2003 server*. Component is similar to object and it is the independent unit that provides a function to a client with an interface of operation, property, and event. If a component is implemented, a developer can sell the component and modularize a code with the number of components. Besides, the components modularized can be used in the distributed computing environment.

Microsoft's ASP (Active Server Page) and languages in *Visual Studio* have been used to build e-Business applications on Internet and *World-Wide-Web*. However, the applications mainly depend on Windows operating system so that *Microsoft* has lost the major portions of server market against *Unix* server systems. It means that *Microsoft* may lose the huge market of e-Business system against *J2EE*. Therefore, *Microsoft* has presented *.NET* solution in June 2000. With *.NET* framework, *Microsoft* can compete with and hopefully win over *J2EE* in both e-Business applications and web services markets.

.NET framework supports multi-language environment. At this moment, *.NET* framework supports *Visual Basic*, *C++*, *C#*, and *J#* languages. Any code written in one of these languages is compiled to a *MSIL* (Microsoft Intermediate Language) code. Then, *CRL* (Common Runtime Language) of *.NET* framework interoperates the *MSIL* codes so that *MSIL* codes in any language can communicate each other. As you can guess, *CRL* is to translate the *MSIL* codes to the machine codes as *JVM* does in *Java*. Besides, *.NET* framework is to accomplish the platform independency as *Java* does. Even though it only runs on *Microsoft Windows* system at this moment, *Microsoft* provides *CLS* (Common

Language Specification) to provides platform independency and has studied to build .NET framework executable on *FreeBSD* and *Mac OS X 10.2* operating systems and finally on any operating systems [MicroJ03].

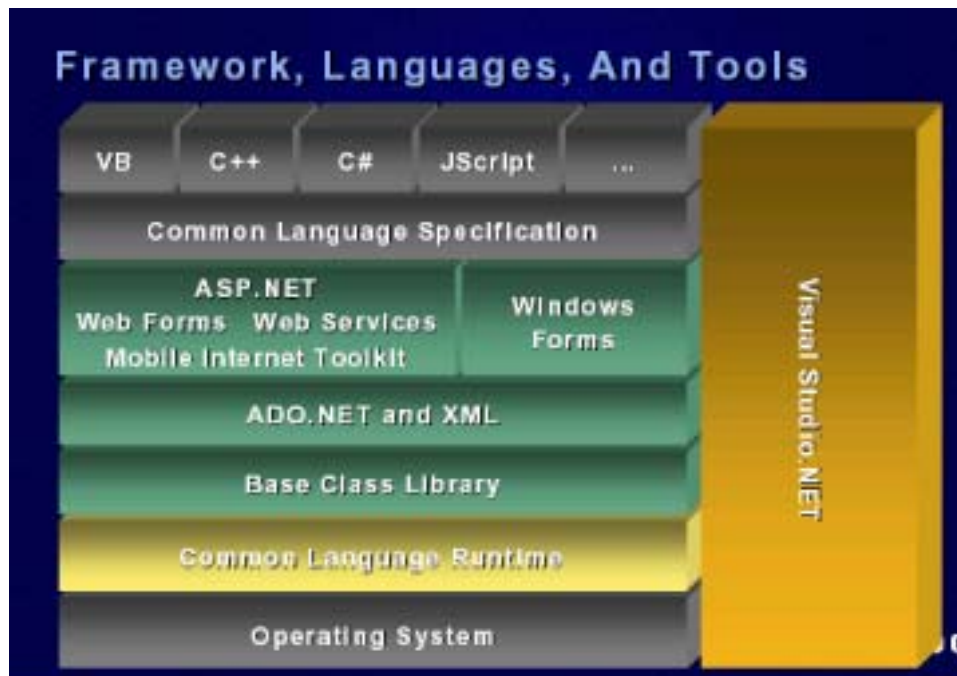


Figure 5. .NET Framework [Micro]

3. J2EE and .NET comparison

3.1 Programming Language

J2EE is the enterprise edition of Java. *J2EE* technology and its component model is the extension of Java standard edition. *J2EE* provides simple enterprise development and deployment with the enterprise *APIs* (*Application Programming Interface*) such as *JDBC*, *JNDI*, *Servlet*, *JSP*, *RMI*, *EJB*, and *JMS*. The *JDBC* (*Java Database Connectivity*) *APIs* are used to connect a Java code to a data source, that is, database. The *JNDI* (*Java Naming and Directory Interface*) *APIs* are to access distributed objects. The *Servlet* *APIs* are to handle *HTTP* request and response between clients and servers such as databases. The *JSPs* (*JavaServer Page*) are to create a dynamic page as a servlet to integrate presentation logic with *html* documents. The *RMI* (*Remote Method Invocation*) *APIs* are to execute the methods of the remote objects on networks. The *EJB* (*Enterprise JavaBean*) *APIs* are to build components that simplify the server site applications such as session controls and data access logics. It also can modulate the applications. The *JMS* (*Java Messaging Service*) *APIs* are to provide communications between objects. Besides, since Java is an object-oriented language, the codes written in *J2EE* are easy to extend and to maintain. Thus, *J2EE* has been a good solution of *e-Business* systems for years.

.NET is for platform independent application as *J2EE*. Besides, the existing programming languages such as *C++*, *Visual Basic*, *ASP*, *C#* and *J#* can interoperate each other in *.NET* framework. These languages have compilers that supports *.NET* framework *Common Runtime Library (CRL)* [Micro03]. Therefore, we can simply extend the existing enterprise systems built in one of these languages by using one of those programming languages. Besides, *.NET* languages are object-oriented languages that have the benefits as *J2EE*. Thus, *.NET* framework is programming language independent and more extensible than *J2EE*.

3.2 Platform Independency

Java is the platform independent language with *Java Virtual Machine (JVM)* provided by *Sun Microsystems*. Java codes in *J2EE* are compiled to Java byte codes as in *J2SE*. The Java byte codes can run on any platform such as *Unix (Linux)* or *Windows* environment, in which the platform has *Java Virtual Machine (JVM)* installed. Almost all platforms have their *JVMs* to make Java byte codes executable on them.

.NET framework is to achieve platform independency. However, it only works on *Windows* environment at this moment. There is the source code named *SSCLI (Shared Source Common Language Implementation)*. It is the working implementation to provide a *Platform Adaption Layer (PAL)* for academics and researchers. *SSCLI* is under a noncommercial shared-source license and it will run on *Microsoft Windows XP*, the *FreeBSD* OS, and *Mac OS X 10.2* [Micro03]. If *SSCLI* is successful, codes on *.NET* framework will be run on *FreeBSD* OS and *Mac OS X 10.2* as well as *Windows* OS. That is, *.NET* framework may achieve the platform independency.

3.3 Component Model

Component in software can be defined as an independent unit to provide an operation with the interfaces such as operation, property, and event. Each component should be registered in a naming server for distributed computing environment. If a component model is built for a certain function, the component can be salable and integrated with other products. Besides, many components can be developed in modules and run on distributed computing environment.

J2EE provides component model named *EJB (Enterprise JavaBeans)*. It runs on an *EJB* application server. Its basic idea is to build the expensive security, transaction, and database integration functions on *EJB* application server. If a developer purchase an *EJB* application server, the developer can only focuses on implementing his business logic with *EJB* instead of spending on building those expensive functions. It will save time and money to develop a product in an organization. *EJB* application server normally has a *JNDI (Java Naming and Directory Interface)* server. *EJBs* are registered to the *JNDI* server so that an *EJB* registered can be found in the *JNDI* server whenever it is needed.

Microsoft Corporation has developed a component model such as *COM (Component Object Model)*. It is a *Microsoft* specification for component interoperability. It has been extended to *DCOM (Distributed Component Object Model)* in 1990s. About 1997,

COM+ plan was announced by *Microsoft*, which is an extension of *COM*. *COM+* builds on *COM*'s integrated services and features. It also makes it easier for developers to create and use software components in any language [Micro02]. *Microsoft Corporation* has applied the existing component concept to *.NET framework*. *.NET framework* is an integral *Windows* component for building and running the software applications and Web services. However, *.NET* components are only registered in the *Windows* registry. Thus, it is dependent on technology and support of *Microsoft* products.

3.4 Database connection

JDBC – you may consider it *Java Database Connectivity* - technology is an API to access virtually any tabular data source from Java codes. If a data source such as Database contains *JDBC* driver, Java codes can access the database. Normally, a database vendor provides the database product with own *JDBC* driver. When a Java code is built for database access application, it needs to refer to classes of *JDBC* API of the *JDBC* driver that is accessible from the code. Besides, an entity bean of *EJB* has database connection interfaces. A developer can easily implement an entity bean that connects a database without building *JDBC* connection logic. Thus, the developer can only focus on implementing business logic so that it will save the cost of his/her product.

OLE (Object Linking and Embedding) DB of *Microsoft* is a standardized interface with which a developer can refer to any data source. It is built in as a part of the *.NET* framework. *ADO (ActiveX Data Object) .NET* is on top of *OLE DB* as another layer. *ADO .NET* is a database object model that is composed of many standard classes to refer to data from any database. If the developing environment has an *OLE DB* database provider of each database to use *ADO .NET* classes, a developer can build database connection applications in *.NET*.

3.5 Application Server

Java codes run on *JVM*. However, *J2EE* codes are not executable on *JVM* alone. It needs an application server that makes the codes executable. *J2EE* codes on an application server are mainly for web applications – you may regard them as e-Business applications. The popular application servers in the market now are *BEA WebLogic*, *IBM WebSphere*, *Macromedia JRun*, *ATG Dynamo*, and *Oracle application server* etc. Besides, there are free application servers such as *Tomcat* and *JBoss*. Since there are many vendors for application servers, some *J2EE* codes runnable on an application server are not executable on another server. It violates the motive of *Java* language. Thus, *Sun Microsystems* forces the *Java* application on an application server to run on other application servers to maintain *write-once-run-anywhere* motto.

To run *.NET* applications on the legacy *Windows* OS, *.NET* framework is needed that can be downloaded from the *Microsoft Corporation* web site [Micro]. Otherwise, we can purchase and install *Windows server 2003* to run *.NET* applications. For web applications, normally, *ASP .NET* is used for a client site – web browser - to access the dynamic functions built in other *.NET* languages at a server site. It only runs on *Microsoft IIS* web server. It means that *Microsoft Corporation* exclusively dominates the *ASP .NET* market

with *IIS* server. The *IIS* server handles both static and dynamic web pages so that we can call it application server. Since there are many issues in *IIS* server, for example, security issue, *Microsoft* provides source-available Web server platform, *Cassini*, written entirely in *C#*. Thus, a developer can modify the internal functions for his/her need. *Cassini* supports *ASP .NET* and other basic functions such as directory browsing on *HTTP 1.1*. You can demonstrate *Cassini* on the *.NET* Framework [Cassini].

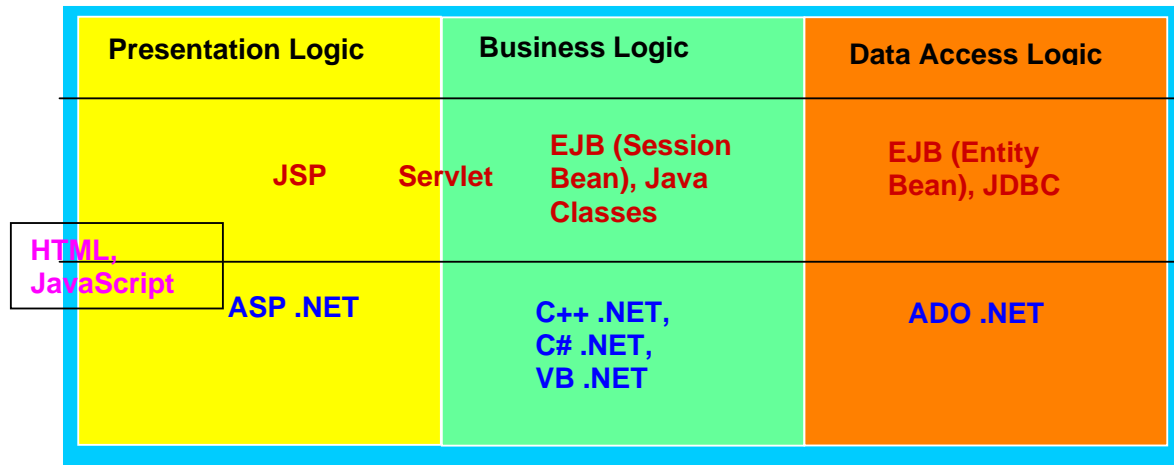


Figure 6. *J2EE* and *.NET* on *N-Tier* Architecture

4. Summary

Up to section 3, we see the approach of *J2EE* and *.NET* to build e-Business applications. We can describe how *J2EE API* and *.NET* products are used on *N-Tier* architecture as in Figure 6. To build the presentation logic of an e-Business application, *JSP* and *Servlet* of *J2EE API* and *ASP .NET* of *.NET* framework can be used. For the business logic, *EJB* - especially *Session Bean* - and standard *Java* classes for *J2EE* and *C++ .NET*, *C# .NET*, and *VB .NET* etc. for *.NET* can be referred. Finally, the developer can implement the database access logic with *EJB* - especially *Entity Bean* - and *JDBC* classes for *J2EE* and *ADO .NET* for *.NET*.

Table 1 is to summarize the comparison between *J2EE* and *.NET* for criteria of e-Business applications. Those are how to handle dynamic web contents, how to access database, platform independency, possible languages to build the applications, to see if there is a component model and if it is proven in the market, and how much the cost to use them. For the market, *J2EE* has been proved for many years but *.NET* has been only for a few years. Thus, even we know the potentiality of *.NET*, it is categorized to “Not yet”. The cost to build and execute applications, *J2EE* is free and there are free application servers to make the *J2EE* codes run. But, in *.NET*, we need to buy a *Visual Studio .NET IDE* (Integrated Development Environment) and *IIS* Web server. For the performance, the Middleware Company presents that *.NET* has better performance on the *Pet Store* benchmark tuned for *.NET* than *J2EE* on the benchmark [Mid02]. However,

since *J2EE* runs on *JVM*, the result should be a matter of course. For the more fairness, the performance of *.NET* applications should be measured on the different platform from *Microsoft OS*.

	J2EE	.NET
Dynamic Web Content	JSP	ASP .NET
DB Access	JDBC	ADO .NET
Platform Independency	Yes	Not Yet (Microsoft OS only)
Languages	Java	C++, C#, Visual Basic, J#
Component Model	Yes (EJB)	Yes
Market Proven	Yes	Not yet, on
Cost of product	Some freeware	No freeware
Performance	?	?

Table 1. Summary: J2EE and .NET

5. Conclusion

In this paper, *N-tier architecture* of *e-Business* applications is described. The most popular approach to build the applications on *N-tier architecture* is *J2EE* and *.NET*. *J2EE* is the specification provided by *Sun Microsystems*. *J2EE* is more opened because anyone can implement *J2EE* application server that meets the specification. *.NET* of Microsoft corporation is the product. Thus, it is only dependable on *Microsoft* products. It is not easy to compare the performance of *J2EE* and *.NET*. However, other criteria for *e-Business* applications are compared for *J2EE* and *.NET*.

Reference

- [Cassini] "Cassini Sample Web Server",
<http://www.asp.net/Default.aspx?tabindex=7&tabid=41>, Microsoft Corporation, 2003
- [Intel01] "Building a Better e-Business Infrastructure: N-tier Architecture Improves Scalability, Availability and Ease of Integration", Intel® e-Business Center White Paper,
<http://www.intel.com/eBusiness/pdf/busstrat/industry/wp012302.pdf>, 2001
- [Micro] "Technology Overview for .NET Framework 1.1", Microsoft Corporation,
<http://msdn.microsoft.com/netframework/productinfo/overview/default.asp>
- [Micro02] "COM+", <http://www.microsoft.com/com/tech/COMPlus.asp>, Microsoft Corporation, 2002
- [Micro03] "Got Dot NET (.NET Framework Website)",
<http://www.gotdotnet.com/team/lang/>, Microsoft Corporation, 2003
- [MicroJ03] "C#/JScript/CLI Implementations Shared Source Licensing Program",
http://www.microsoft.com/resources/sharedsource/Licensing/CSharp_JScript_CLI.msp,
 Microsoft Corporation, June 2003

[Mid02] “The Petstore Revisited: J2EE vs .NET Application Server Performance Benchmark”, <http://www.middleware-company.com/j2eedotnetbench>, Middleware Company, Oct. 2002.

[NAONO99] “Design Considerations: From Client/Server Applications to e-business Applications”, Indran Naick, Luca Amato, Jason K O'Brien, Jim Nicolson, Tsutomu Oya <http://www.redbooks.ibm.com/redbooks/SG245503.html>, Dec 1999

[SABGMM03] “IBM e-business Technology, Solution, and Design Overview”, Brian R. Smith, Charles Acekifi, Thomas G. Bradford, Prabhakar Gopalan, Jennifer Maynard, Abdulmir Mryhij, <http://www.redbooks.ibm.com/redbooks/SG246248.html>, August 2003

[Sun03] “Java 2 Platform, Enterprise Edition (J2EE)”, <http://java.sun.com/j2ee/>, Sun Microsystems, Inc, 2003