Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 131–140. http://www.scpe.org
ISSN 1895-1767

THE COMPARISON OF *J2EE* AND *.NET* FOR ENTERPRISE INFORMATION SYSTEMS

JONGWOOK WOO*

Abstract. e-Business and Enterprise Information Systems have held the spotlight since Internet and World-Wide-Web came out to the world. The e-Business applications have been evolved from legacy client-server architecture into n-tier architecture, lately even into Enterprise Information Systems. There are two famous approaches to build the e-Business applications, which are *J2EE* and *.NET*. In this paper, e-Business and n-tier architecture are illustrated. Besides, n-tier architecture for Enterprise Information Systems is introduced, which provide the access to the disparate data sources. In addition, *J2EE* and *.NET* are compared for e-Business applications based on many criteria including the methodologies to implement Enterprise Information Systems.

Key words. Integrated Information Systems, Enterprise Information Systems, e-Business, *J2EE*, *.NET*, n-Tier architecture

1. Introduction. e-Business systems have 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 [9]. 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 such as knowledge management. Business-to-consumer, the most common activity, is to provide services to consumers who is out of organizations such as customer resource management, e-Commerce, and web auctions etc. Business-to-business activity is to improve inter-organizational partnerships and relationships such as supply chain integration [8].

The needs of the legacy e-Business systems were simple to maintain functionality and stability on the corporate computing environment. However, the legacy e-Business systems are not sufficient for the current high volume e-Business transactions. People need systems that 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. It partitions systems and software to more flexible blocks that have different roles in order to enable high performance, scalability, and availability to businesses [2]. Section 1 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. *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* fits well to build e-Business systems at both a development and a server site in both Unix (Linux) and Windows operating systems. Besides, the applications of *J2EE* are normally built in Windows operating system and published into servers in 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 the large organizations such as banking and entertainment industries because Unix OS have been more stable than Windows so that it was chosen prior to Windows. Thus, e-Business systems of the server market have been mainly developed in *J2EE* instead of in ASP. Microsoft Corporation might want to compete with Unix systems for the e-Business markets 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—even it is limited for research—but also programming language independent. *.NET* has been popular for several years in the e-Business world and competed with *J2EE*—probably has dominated the small businesses more than *J2EE*.

In this paper, *.NET* and *J2EE*, the most popular e-Business development approaches, are compared in terms of programming language, platform independency, component model, application server, market proof, openness, and Database connectivity including the connectivity to disparare data sources. Since they are the standards to build e-Business systems nowadays, this paper will be useful for people who want to see the de facto distributed computing environment for e-Business systems and who want to select one of approaches. In the paper, Section 2 introduces the e-Business architectures. Section 3 describes the frameworks of *J2EE* and *.NET* in detail. Section 4 compares *J2EE* and *.NET* in terms of several factors including the approach for

*Computer Information Systems Department, California State University, Los Angeles 90032-8530, Los Angeles, CA, USA (jwoo5@calstatela.edu).

information integration. Section 5 illustrates the summary of the comparison based on the analysis in Section 4. Section 6 is the conclusion and culmination of the comparison for integrated information systems.

2. e-Business Architecture.

2.1. n-tier 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 that contain presentation and business logics. We can call it 2-tier architecture as shown in Figure. 2.1. 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. 2.1, 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 or 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 [2].

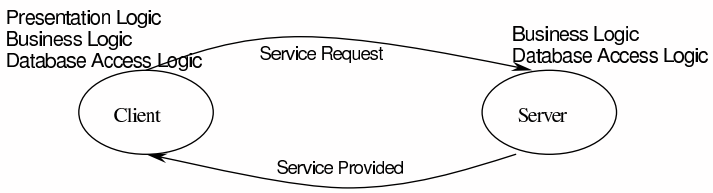

FIG. 2.1. *2-tier Architecture.*

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. 2.2. 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 to 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 the n-tier architecture. It produces the response of the request from the client and provides the response to the client. If the request is related to data access, it will pass the data access request to the back-end database server. The layer 3 is hosted on the back-end database, XML, or other data sources as data access logic. It is to handle the request of data source from the business logic. It has the functions to access data sources such as database, XML, file systems, or EIS (Enterprise Information Systems) etc. 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 [2]. Therefore, n-tier architecture has been the way to implement the e-Business systems lately.

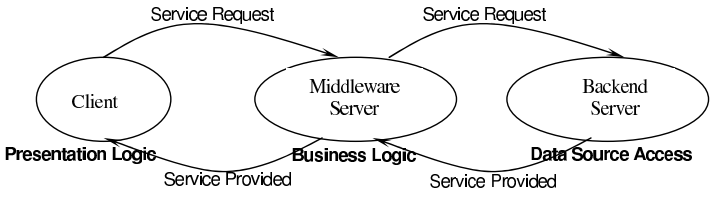

FIG. 2.2. *n-tier Architecture.*

2.2. n-tier Architecture for Enterprise Information Systems. Most of the organizations and companies already have adopted n-tier Architecture for their e-Businesses. Simply, they have the different data sources and their data access methods are different. Thus, each organization's individual solution has made more difficult to share the information among the departments within an organization and among the organizations. However, there has been great need to provide integrated information these days in order to support cooperative works among staffs in agencies and to support their employees and customers. If the different organizations or the different departments of an organization have the integrated information, the integrated information systems will benefit the public.

Integrated Information System can be defined as the system that merges information from the disparate (or heterogeneous) data sources despite differing conceptual, contextual, and typographical representation even in distributed applications. Figure. 2.3 shows the n-tier Architecture with the layer of Information Integration logic that resides on middleware server between Business and Data Source Access logic.

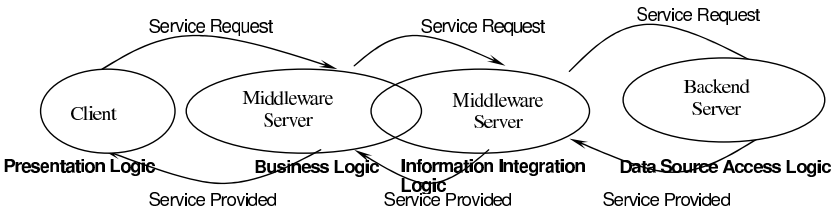

FIG. 2.3. *Information Integration n-tier Architecture.*

3. The *J2EE* and *.NET*. *J2EE* and *.NET* are most popular programming language and framework in order to implement n-tier architecure. This section illustrates the fundamental concepts and frames of *J2EE* and *.NET*.

3.1. *J2EE*. Java platform is composed of APIs (Java Application Programming Interfaces) and JVM (Java Virtual Machine) as shown in Figure. 3.1. Java programs—*J2SE* (Java 2 platform Standard Edition)—are compiled to Java byte codes that are executable on JVM. JVM interprets the 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 Java applications to new platform.

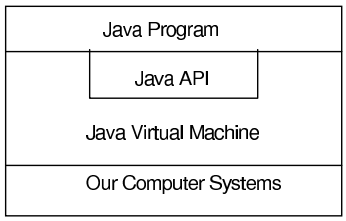

FIG. 3.1. *The Java Platform.*

Java APIs are a set of built-in libraries as byte codes. *J2EE* (Java 2 platform Enterprise Edition) defines the standard APIs for n-tier architecture [10]. *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, PHP and C++ etc. 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 that is the product of Microsoft. Thus, *J2EE* has been the popular method to build e-Business systems in the large scaled market such as bank and entertainment.

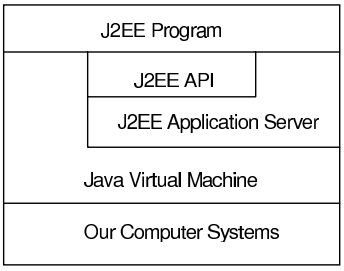

FIG. 3.2. *Application Server for J2EE.*

J2EE is the extended APIs from *J2SE*. 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 *J2SE* such as platform independency and object-oriented language. Besides, *J2EE* supports APIs for enterprise systems: JDBC for database access, EJB (Enterprise JavaBeans), Java Servlets, JSP (JavaServer Pages), XML, Java Mail, and Java Messaging etc. As are *J2SE* codes, *J2EE* source codes are compiled to Java byte codes and run on JVM 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. 3.2. There are many application servers in the market such as BEA WebLogic, IBM WebSphere, ATG Dynamo, RedHat JBoss, Apache TomCat, and Sun One Application server, etc. And, in order to connect databases, JDBC driver is needed for each database. Normally, each database vendor provides its JDBC driver. Sun Microsystems provides the *J2EE* specification for *J2EE* application servers in oder to maintain *write-once-run-anywhere*.

In Nov 2006, Sun Microsystems announced to be open sourcing all of its Java Source Implementations under GPL (General Public License) version 2 licensing used by GNU/Linux Operating System [17]. The platform implementations include Java SE (JDK), Java ME (Mobile & Imbedded), and Java EE. Before that, there are open Java software projects such as GNU Java [18] and Apache Harmony [19]. Since Sun opens Java implementations, the open Java platform can address the new markets for all Java devices more dramatically.

3.2. *.NET.* Microsoft Corporation is the most famous for Windows operating systems in the personal computer market. 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 may win over *J2EE* for e-Business applications in large-scaled markets.

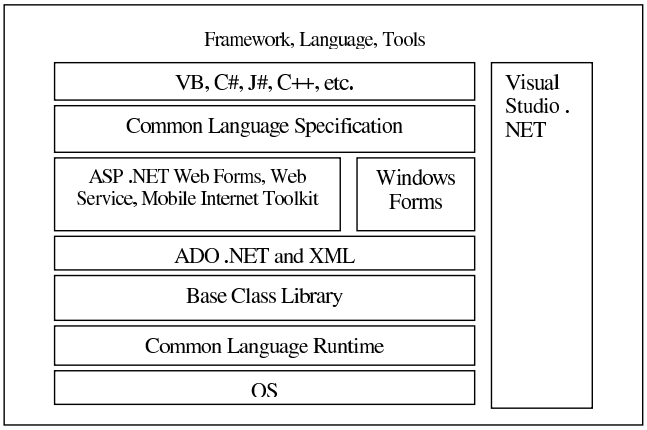

FIG. 3.3. *.NET Framework [Micro].*

Microsoft has focused on its components such as COM (Component Object Model). 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. The component model has been extended in *.NET* framework. Microsoft has produced Windows products integrated with *.NET* framework such as Windows XP and 2003 server etc. *.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. CRL is to translate the MSIL codes to the machine codes as JVM does in Java. Besides, *.NET* framework may accomplish the platform independency as Java does. Even though it only runs on Microsoft Windows system at this moment, Microsoft provides SSCLI (Shared Source Common Language Implementation) to provides platform independency. Even though it is not clear if the platform independency is the target of *.NET*, Microsoft has studied the possible platform independency to build *.NET* framework executable on FreeBSD and Mac OS X 10.2 operating systems [6].

Mono project is originally an open development initiative sponsored by Novell in order to support *.NET* development to *Unix OS*. *Mono* platform provides the necessary software such as compilers and libraries to develop and run *.NET* client and server on any platform. Mono project provides both programming language and platform independency. The platforms to run *Mono* are *Linux*, *BSD*, *Solaris*, *MacOSX*, *Windows*, and *Unix* etc. Multiple languages can be used with *Mono* platform, which are *C#.NET*, *Java*, *VB.NET*, *ASP.NET*, *Python*, *PHP*, and *JavaScript* etc [20].

4. *J2EE* and *.NET* comparison. This section compares *J2EE* and *.NET* in terms of programming language, platform independecy, component model, database connectivity, market, openness, and application server. Besides, they are compared for information integration that receives the most spotlight in the world these days.

4.1. Programming Language. *J2EE* is the enterprise edition of Java. *J2EE* technology and its component model is the extension of *J2SE*. *J2EE* provides simple enterprise development and deployment with the enterprise APIs such as JDBC, JNDI, Servlet, JSP, RMI, EJB, and JMS. The JDBC—we may regard it as Java Database Connectivity—APIs are used to connect a Java code to a data source, that is, database that provides its JDBC driver. The JNDI (Java Naming and Directory Interface) APIs are to register distributed objects and access one of them. The Servlet APIs are to handle HTTP requests and responses between clients and servers such as application and database servers. The JSP is to create dynamic pages as an extended format of Servlet by integrating presentation logic with *HTML* documents. The RMI (Remote Method Invocation) APIs are to execute the methods of the remote objects on networks. The EJB APIs are to build components that simplify the implementation of server site applications such as session controls with Session Bean, data access and mapping logic with Entity Bean, and asynchronous messaging with Message Bean. EJB also can modulate the applications as component. The JMS (Java Messaging Service) APIs are to provide synchronous communications between objects. Besides, since Java is an object-oriented language, the codes written in *J2EE* are easy to extend and to maintain. Therefore, *J2EE* has been a wll-known solution for e-Business systems more than 10 years.

.NET is the product of Microsoft corporation. It is language independent so that the existing *.NET* programming languages such as *C++.NET*, *VisualBasic.NET*, *ASP.NET*, *C♯.NET*, and *J♯.NET* can interoperate each other on Common Runtime Library (CRL) of *.NET* framework. Microsoft's *VisualStudio.NET* supports these languages with each compiler of the languages that supports CRL [3-5]. Therefore, we can simply extend the existing enterprise systems built in one of these languages by using any of those programming languages. Besides, *.NET* languages are object-oriented languages that have the same benefits as *J2EE*. Thus, *.NET* framework is more extensible—in particular, on Windows—than *J2EE* as it is programming language independent and object-oriented.

4.2. Platform Independency. Java is the platform independent language with 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 its JVM installed. JVM converts the byte codes to machine codes of the platform. Almost all platforms have their JVMs to make Java byte codes executable on them. *.NET* framework may have a goal 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 [5]. 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. Therefore, *.NET* framework may achieve the platform independency even though it does not run on most UNIX OSs.

4.3. 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. If a component model is built for a certain function, the component can be salable and integrated with other products. In addition, many components can be developed in modules and run on distributed computing environment. Each component should be registered in a naming server for distributed computing environment. *J2EE* provides component model named EJB. It runs on an EJB application server. The basic idea is to use the built-in applications of EJB application server such as expensive security, transaction, and database integration functions. If a developer purchase an EJB application server, the developer can only focuses on implementing his or her business logic with EJB instead of spending on building those expensive functions. It will save time and cost to develop a product of the organization. EJB application server normally includes JNDI (Java Naming and Directory Interface) server. EJBs are registered to the JNDI server so that an EJB objects registered can be found in the JNDI server whenever they are called in a code.

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 [4]. 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 cannot be seperated from technology and support of Microsoft products.

4.4. Database connection. JDBC technology is an API to access virtually any tabular data source from Java codes. If a data source such as database is linked to JDBC driver, Java codes can access the database. Normally, each database vendor provides its JDBC driver as the database product. 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. In addition to JDBC, 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 or her product.

OLE (Object Linking and Embedding) DB is a standard interface of Microsoft 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. The integrated developing environment (IDE) such as Visual Studio *.NET* normally supports the OLE DB database provider of each database. Since the provider uses certain *ADO.NET* classes to connect a database, the developer can easily establish the database connection application in *.NET*.

4.5. 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, ATG Dynamo, and Oracle application server etc. Besides, there are free application servers such as Apache TomCat and RedHat JBoss. Since there are many vendors that implement application servers, some *J2EE* codes runnable on an application server are not executable on other servers. It violates the motive of Java language. Thus, Sun Microsystems provides J2EE specification to keep the *write-once-run-anywhere* motto. Thus, any *J2EE* application will run on the application server if the vendor follows the direction of the specification when implementing the application server. The server that meets the specification is called the Sun certified J2EE application server

To run *.NET* applications on the legacy Windows OS, *.NET* framework is needed that can be downloaded from the Microsoft Corporation web site [3]. 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. *ASP.NET* 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 some issues in IIS server, for example, security and open source needs, Microsoft provides Cassini that is source-available Web server platform and written entirely in C♯. Thus, a developer can modify the internal functions of Cassini for his or her need and implement the *.NET* compatible application server. Cassini supports *ASP.NET* and other basic functions such as directory browsing on HTTP 1.1. You can demonstrate Cassini on the *.NET* Framework [1].

4.6. Openness. There have been many approaches for Java Open Source. Sun has had an *OpenSolaris* project to develop *SolarisOS* by releasing most of the Solaris source code under the Common Development and Distribution License (*CDDL*) [21]. However, many open source communities critisize that *OpenSolaris* project does not have the true open source community processes. *Sun* provided Java open source for *OpenSolaris* project. And, *GNUJava* project has supported *Java* language with *Java* Compiler and *VM* etc [18]. *Apache* also launched *Harmony* project to support platform independent *JavaSE*5*JDK* under *Apache* license [19]. On Nov 2006, *Sun* announced to open Java source for *SE*, *ME*, and *EE* under *GNUGPL* license [17]. And, many open source communities believe that it can be useful for *Java* world amazingly.

Mono project is to provide open source software for *.NET* on *Unix* platform sponsored by *Novell*. It provides *.NET* compiler and libraries etc. Besides, it is actually both platform and language independent platform even though it needs more studies to be compatible to the platforms and langauges [20].

4.7. Information Integration. There has been great need to integrate and share the information among disparate data sources within the same or among many different organizations. We can define the disparate data sources as databases from differen vendors, file systems, and XML etc. The integrated information system has many demands to satisfy security and reliable requirements, as well as data privacy, quality, and ownership. In addition, it has the complexity of integrating disparate data. Enterprise Information Integration (EII) is one technical approach that addresses integration complexity. EII is the process of using data abstraction to tackle the data access challenges and complexity associated with the disparate data sources in e-Business.

In Java, there have been several approaches to resolve the problem to integrate the disparate data sources for n-tier architecture such as JDO, JAXB, EMF and SDO. With these approaches, Java developers can only focus on the business logic without wasting resources for data management applications. JDO stands for Java Data Objects standardized by JCP (Java Community Process). It provides an API to access data in data sources such as database and file systems etc. EMF (Eclipse Modeling Framework) generates a unifying metamodel based on a data model defined using Java interfaces, XML schema, and UML class diagrams. JAXB stands for Java API for XML Data Binding. It is release by JCP and used to generate Java objects in memory corresponding to XML data [11, 12].

SDO stands for Service Data Objects. It was originally developed as a joint collaboration between BEA and IBM and is now being developed by BEA, IBM, Oracle, SAP, Siebel, Sybase and XCalia etc. SDO is abstracts data in order to utilize multiple disparate data sources, which includes databases, entity EJB components, XML, Web Services, Java Connector Architecture, and JSP pages [11-13]. SDO provides SDO API as JDO. However, SDO is more general than JDO so that SDO can be used for between any tiers on n-tier architecture while JDO is for data access tier only. JDO can be even considered as a data source for SDO. Both SDO and EMF present data representation. SDO is created by EMF code generation and is a facade over EMF as part of EMF project. JAXB only focuses on Java-to-XML binding while SDO takes care of any data source. Thus, SDO has been received many lights as it provides only a single and simple interface to a variety of disparate data. And, it can be also applicable to SOA (Service Oriented Architecture) such as Web Services [12].

Microsoft introduced ActiveX Data Objects (ADO) on the release of VB 5. ADO was built to provide access to disparate data sources on distributed computing, that is, n-tier architecture. *ADO.NET* is the expansion of ADO by using XML. There are proprietaries as *ADO.NET* providers such as Simba Technologies, DataDirect Technologies, and OpenLink Software that present drivers and bridges to other data sources [15]. *ADO.NET* is the product of Microsoft. Java SDO API is JSR (Java Specification Request) 235 that is the request to be Java standard API.

5. Summary. Up to Section 4, we see the approach of *J2EE* and *.NET* to build e-Business applications. It is described how *J2EE* API and *.NET* products are used on n-tier architecture in Figure. 4.1. To build the presentation logic of 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 applicable to build the business functions. And, there is information integration logic between business and data access logics. In *J2EE*, JDO, EMF, and JAXB can

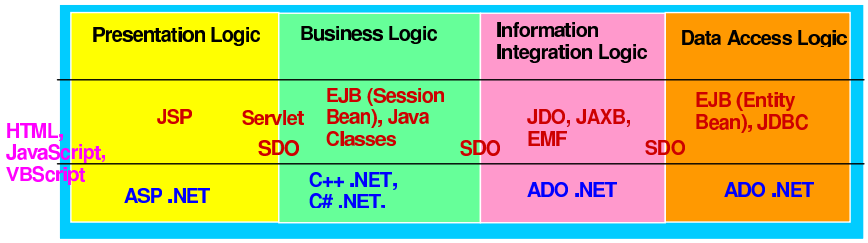

FIG. 4.1. *J2EE and .NET on Enterprise n-tier Architecture.*

be used as *ADO.NET* in *.NET*. For information integration of *J2EE*, SDO can resides on between any tiers. Finally, the developer can implement the database access logic with EJB—especially Entity Bean—and JDBC classes for *J2EE* and *ADO.NET* for *.NET*.

Figure. 5.1 summarizes the comparison between *J2EE* and *.NET* for the criteria of e-Business applications as analyzed in Section 4. The criteria are how to handle dynamic web contents, how to access database, platform independency, possible programming languages to build the applications, to see if there is a component model and if it is proven in the market, how much the cost to use them, how to integrate heterogeneous data sources, how is openness, and performance. In the market, *J2EE* has been proven for more than 10 years and *.NET* has been only for several years. However, *.NET* has been used by many companies and organizations so that it is already proven too. In terms of the cost to build and execute applications, *J2EE* can be less expensive since it is free and there are free application servers to make the *J2EE* codes run, for example, JBoss. But, in *.NET*, people need to buy a *VisualStudio.NET* IDE (Integrated Development Environment) and IIS Web server in order to build solid applications. As the alternative and cheap methodologies to develope *ASP.NET* applications, Cassini as application server and WebMatrix [16] as IDE are not good enough to implement the solid products.

J2EE is platform independent. *.NET* is the Microsoft language independent but not platform independent. However, there is Mono project to make *.NET* code executable on *Unix* platform. *J2EE* community has worked on integrated data source as *ADO.NET* so that *SDO* has come out to the world. In order to get the benefit of open source as *Linux* has done, *Sun* provided open source for *Java*.

For the performance, the Middleware Company presents the report insisting on that *.NET* has better performance on the Pet Store benchmark tuned for *.NET* than *J2EE* on the benchmark [7]. However, since the benchmark is optimized for *.NET* and executed on Windows OS while *J2EE* runs on JVM of Windows OS, the result should be a matter of course. For the better fairness, the performances of *.NET* and *J2EE* applications should be measured with the well optimized benchmark for both *.NET* and *J2EE* on the different platform such as Unix, which is almost impossible at this moment.

	J2EE	.NET
Dynamic Web Content	JSP	ASP.NET
DB Access	JDBC	ADO.NET
Platform Independency	Yes	Yes (Mono project)
Languages	Java	C++, C#, Visual Basic, J#
Component Model	Yes (EJB)	Yes
Market Proven	Yes	yes
Cost of product	Some freeware	No freeware
Integrate Disparate Data Sources	JDO, SDO	ADO.NET
Openness	Java Open Source	Mono
Performance	?	?

FIG. 5.1. *Summary: J2EE and .NET.*

6. Conclusion. As e-Business applications have been implemented, the importance of the information integration among the collaborative groups has been grown. In this paper, n-tier architecture of e-Business is described. Then, Enterprise Information Systems architecture is introduced. The most popular approaches are illustrated to build the applications on n-tier architecture: *J2EE* and *.NET*. *J2EE* is the specification provided by Sun Microsystems. *J2EE* is more flexible because *J2EE* API is free and anyone can implement *J2EE*

application server that meets the specification given by Sun. *.NET* of Microsoft Corporation is the product. Thus, it is only dedicated to Microsoft products. If considering the applications on Windows only, *.NET* is more flexible than *J2EE* because it is programming language independent. *J2EE* and *.NET* are compared in terms of dynamic web content, database connectivity, platform and language independency, component model, market, cost, openness, and heterogeneous data source integration methodologies. However, it is not easy to compare the performance of *J2EE* and *.NET* because *.NET* is not executable on the other platforms yet. The paper should be the useful reference to establish e-Business and Enterprise Information Systems for both profit and non-profit organizations which do not have the technical and architectural ideas for the systems.

Note. Figure. 6.1 is the table for acronyms used in this paper.

Acronym	
ADO	ActiveX Data Object
APIs	Application Programming Interfaces
ASP	Active Server Pages
CDDL	Common Development and Distribution License
COM	Component Object Model
CRL	Common Runtime Language
DCOM	Distributed Component Object Model
EII	Enterprise Information Integration
EJB	Enterprise JavaBeans
EMF	Eclipse Modeling Framework
GPL	General Public License
IDE	Integrated Development Environment
JCP	Java Community Process
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JSR	Java Specification Request
J2EE	Java 2 platform Enterprise Edition
J2SE	Java 2 platform Standard Edition
JVM	Java Virtual Machine
MSIL	Microsoft Intermediate Language
OLE	Object Linking and Embedding
PAL	Platform Adaption Layer
RMI	Remote Method Invocation
SOA	Service Oriented Architecture
SSCLI	Shared Source Common Language Implementation

FIG. 6.1. *Table for Acronyms*

REFERENCES

[1] *Cassini Sample Web Server*, http://www.asp.net/Default.aspx?tabindex=7&tabid=41 Microsoft Corporation, 2003.

[2] *Building a Better e-Business Infrastructure: n-tier Architecture Improves Scalability, Availability and Ease of Integration*, http://www.intel.com/eBusiness/pdf/busstrat/industry/wp012302.pdf Intel e-Business Center White Paper, 2001.

[3] *Overview of .NET Framework*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp *.NET* Framework Developer's Guide, Microsoft Corporation, 2003.

[4] *COM+*, http://www.microsoft.com/com/tech/COMPlus.asp Microsoft Corporation, 2002.

[5] *Got Dot NET (.NET Framework Website)*, http://www.gotdotnet.com/team/lang/ Microsoft Corporation, 2003.

[6] *C♯/JScript/CLI Implementations Shared Source Licensing Program*, http://www.microsoft.com/resources/sharedsource/Licensing/CSharp_JScript_CLI.mspx Microsoft Corporation, June 2003.

[7] *The Petstore Revisited: J2EE vs .NET Application Server Performance Benchmark*, http://www.middleware-company.com/j2eedotnetbench Middleware Company, Oct. 2002.

[8] INDRAN NAICK, LUCA AMATO, JASON K O'BRIEN, JIM NICOLSON, AND TSUTOMU OYA, *Design Considerations: From Client/Server Applications to e-business Applications*, http://www.redbooks.ibm.com/redbooks/SG245503.html Dec 1999.

[9] BRIAN R. SMITH, CHARLES ACKEIFI, THOMAS G. BRADFORD, PRABHAKAR GOPALAN, JENNIFER MAYNARD, AND ABDULAMIR MRYHIJ, *IBM e-business Technology, Solution, and Design Overview*, http://www.redbooks.ibm.com/redbooks/SG246248.html August 2003.

[10] *Java 2 Platform, Enterprise Edition (J2EE)*, http://java.sun.com/j2ee/ Sun Microsystems, Inc, 2003.

[11] C. M. SARACCO, JACQUES LABRIE, AND STEPHEN BRODSKY, *Using Service Data Objects with Enterprise Information Integration Technology*, IBM Developer Works, July 2004.

[12] BERTRAND PORTIER AND FRANK BUDINSKY, *Introduction to Service Data Objects*, IBM Developer Works, Sept 2004.

[13] *Service Data Objects*, Dev2Dev at BEA World, Jan 2006.

[14] Jongwook Woo, *The Comparison of J2EE and .NET for e-Business*, The 2005 International Conference on e-Business, Enterprise Information Systems, e-Government, and Outsourcing, EEE 2005, Las Vegas, Nevada, June 20-23, 2005.
[15] Kirk A. Evans, Ashwin Kamannna, and Joel Mueller, *XML and ASP .NET*, published by New Riders, First Edition, April 2002.
[16] *WebMatrix*, `http://www.asp.net/webmatrix/` 2006 Microsoft Corporation.
[17] *OpenJDK*, `http://www.sun.com/software/opensource/java` 2007 Sun Microsystems, Inc.
[18] *GNU Java*, `http://www.gnu.org/software/java/` GNU Free Software Foundations.
[19] *Harmony*, `http://harmony.apache.org/` 2006 The Apache Software Foundation.
[20] *Mono*, `http://www.mono-project.com` Mono Project.
[21] *OpenSolaris*, `http://opensolaris.org/os/` 2006 Sun Microsystems, Inc.

Edited by: Domenico Talia
Received: May 25, 2006
Accepted: January 20, 2007