

**Distributed Architectures And Web Services: Using .NET platform for  
building an e-commerce application**

By

Elie M. Jurascovitch

A Thesis

Submitted in Partial Fulfillment of  
the Requirements for the Degree of  
Master of Science in Computer Science

Department of Computer Science

Faculty of Natural and Applied Sciences

Notre Dame University – Louaize

Zouk Mosbeh, Lebanon

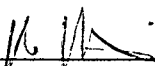
Fall 2002

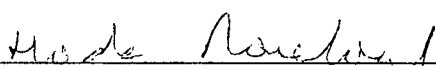
**Distributed Architectures And Web Services: Using .NET Platform for  
Building an E-Commerce Application**

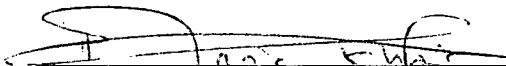
by

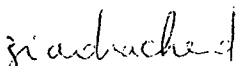
**Elie M. Jurascovitch**

Approved:

  
\_\_\_\_\_  
Khaldoun El-Khaldi: Assistant professor of Computer Science.  
Advisor.

  
\_\_\_\_\_  
Hoda Maalouf: Assistant professor of Computer Science and Chairperson.  
Member of Committee.

  
\_\_\_\_\_  
Mary Khair: Assistant professor of Computer Science.  
Member of Committee.

  
\_\_\_\_\_  
Ziad Rached: Assistant professor of Mathematics.  
Member of Committee.

Date of thesis defense: Monday, November 18, 2002

# Table of Contents

LIST OF FIGURES	VI
LIST OF TABLES	VI
LIST OF ABBREVIATIONS	VII
ACKNOWLEDGEMENTS	VIII
CHAPTER 1 INTRODUCTION	2
CHAPTER 2 THE FIRST GENERATION CLIENT-SERVER	4
CHAPTER 3 THREE-TIER ARCHITECTURE	7
CHAPTER 4 THREE-TIER ARCHITECTURE	10
CHAPTER 5 COMMUNICATION MODES BETWEEN DISTRIBUTED COMPONENTS	12
CHAPTER 6 INTERDEPENDENCE OF COMPONENTS	15
CHAPTER 7 DISTRIBUTED ARCHITECTURES	17
7.1 Windows Dynamic interNet Architecture (DNA)	17
7.1.1 Presentation Services	20
7.1.2 Business Services	20
7.1.3 Data Services	21
7.2 the Java J2EE Architecture	21
7.2.1 Client Components	22
7.2.2 Thin Clients	24
7.2.3 Web Components	25
7.2.3.1 Business Components	25
	iii

7.2.4	Enterprise Information System Tier	26
7.3	Windows DNA vs. Java J2EE	27
<b>CHAPTER 8 SERVICE-ORIENTED DISTRIBUTED ARCHITECTURE</b>		<b>28</b>
8.1	Business and Technical Services	28
8.2	XML and Web services	30
8.3	"Loosely coupled" interfaces	33
<b>CHAPTER 9 J2EE VS. MICROSOFT.NET: A COMPARISON OF BUILDING XML-BASED WEB SERVICES</b>		<b>35</b>
9.1	Introduction	35
9.2	Building Web services with technologies that have gained the most acceptance	36
9.3	The J2EE and Microsoft.NET approach to Web Services	37
9.4	J2EE	38
9.4.1	Java: The foundation for J2EE	38
9.4.2	J2EE and Web Services	38
9.5	Microsoft's .NET Platform	40
9.5.1	The .NET Framework	42
9.5.2	.NET Servers	42
9.6	Understanding J2EE and .NET by analogy	43
9.7	Comparative Analysis	43
9.7.1	Single-Vendor Solution	43
9.7.2	Support for Existing Systems	44
9.7.3	Language Support	45
9.7.4	Portability	47
9.7.5	Web Services Support	49
9.7.6	Tools	50
9.7.7	Scalability	51
9.8	Conclusions	51
<b>CHAPTER 10 DEVELOPMENT OF AN E-COMMERCE APPLICATION USING .NET ARCHITECTURE</b>		<b>53</b>

<b>10.1</b>	<b>Jurasco Style Store Application Overview</b>	<b>53</b>
<b>10.2</b>	<b>Designing the Sample Application</b>	<b>55</b>
<b>10.3</b>	<b>Functional Walkthrough</b>	<b>56</b>
<b>10.4</b>	<b>logical architecture</b>	<b>65</b>
<b>10.5</b>	<b>Architecture of the Application</b>	<b>66</b>
10.5.1	Database	66
10.5.2	Middle-Tier	67
10.5.3	Presentation-Tier	69
<b>10.6</b>	<b>Building an XML Web Service</b>	<b>70</b>
10.6.1	Web Services in .NET	70
10.6.2	Testing the Web Service	70
<b>10.7</b>	<b>Mobile Device Support</b>	<b>73</b>
<b>CHAPTER 11 CONCLUSION</b>		<b>80</b>
<b>BIBLIOGRAPHY</b>		<b>81</b>

## List of Figures

Figure 1 First generation client-server architecture .....	5
Figure 2 Key elements of the object approach.....	8
Figure 3 Three-tier architecture .....	11
Figure 4 Modes of communication between distributed components .....	14
Figure 5 Windows DNA static view .....	20
Figure 6 Multi-tiered applications .....	22
Figure 7 Server communications .....	24
Figure 8 Web tier and J2EE application .....	25
Figure 9 Business and EIS tiers .....	26
Figure 10 Example of an XML document.....	32
Figure 11 Architecture based on distributed objects – WSDL, WSUI, and SOAP .....	33
Figure 12 Developing Web services with J2EE .....	39
Figure 13 Developing Web services with Microsoft.NET .....	41
Figure 14 Jurasco Style Store business.....	54
Figure 15 Application Use Case diagram.....	56
Figure 16 Mobile information Web page.....	73
Figure 17 log in mobile forms .....	74
Figure 18 Purchase mobile form.....	75
Figure 19 Payment mobile form .....	77
Figure 20 Testing the mobile support using Openwave SDK.....	79

## List of Tables

Table 1 Windows DNA Products and Services.....	19
Table 2 Windows DNA J2EE comparison .....	27
Table 3 Analogies between J2EE and .NET .....	43

## List of Abbreviations

<b>ADO</b>	Active Data Objects
<b>CASE</b>	Computer Aided Software Engineering
<b>DCE</b>	Distributed Computing Architecture
<b>DIS</b>	Distributed Information Systems
<b>DOM</b>	Document Object Model
<b>ERP</b>	Enterprise Resource Planning
<b>HTML</b>	Hypertext Markup Language
<b>IIOP</b>	Inter-ORB Protocol
<b>J2EE</b>	Java 2 Enterprise Edition
<b>JDBC</b>	Java Database Connectivity
<b>JRE</b>	Java Runtime Environment
<b>MIL</b>	Microsoft Intermediate Language
<b>MSIL</b>	Microsoft Intermediate Language
<b>MTS</b>	Microsoft Transaction Server
<b>OSF</b>	Open Software Foundation
<b>PDA</b>	Personal Digital Assistant
<b>RDBMS</b>	Relational Data Base Management Systems
<b>RMI</b>	Remote Method Invocation
<b>RPC</b>	Remote Procedure Call
<b>SOAP</b>	Simple Object Access Protocol
<b>SQL</b>	Structured Query Language
<b>UDDI</b>	Universal Description, Discovery, and Integration
<b>W3C</b>	World Wide Web Consortium
<b>WSDL</b>	Web Services Description Language
<b>WSUI</b>	Web Services User Interface
<b>XML</b>	eXtensible Markup Language

## Acknowledgements

A word of thanks goes to Dr. khaldoun El-Khaldi for his encouragement and support. His enthusiasm and involvement implanted in me perseverance and dedication to my research.

I would like to thank my friends for being a real backup to me when I needed them.

Thanks to Toufic Mady for sharing with me some ideas for the implementation of some web services.

A special thank you goes to every member of my family for their support encouragement and trust.



## **Abstract**

The purpose of this thesis is to study distributed applications in the context of heterogeneous clients (Web browser, mobile) and business-to-business (B2B) integration. Two distributed architectures, leaders of the distributed applications market are presented and compared: Microsoft's .NET, and Sun's Java Enterprise Java Beans (J2EE). After this comparative study, we chose .NET to develop an e-commerce application (Jurasco Style electronic shop) in order to outline the major application architectural issues and solutions offered by .NET technology. Our study also shows how the .NET Jurasco Style Shop application can be easily extended to include support for XML Web Services (based on the SOAP and UDDI standards) as well as support for mobile devices.

# CHAPTER 1

## Introduction

Technologies are constantly evolving and to try to effectively meet the needs of companies' new operating modes and organizations, Information System Departments must now refer to distributed systems, three-tier architecture, increasingly user-friendly business services and user interfaces: thin clients equipped with simple browsers, PDA (Personal Digital Assistant), cell phones, multifunction teller machines and, in the near future, pens and other interface peripherals incorporating wireless communication systems.

In addition, like its markets, its customers and the products that it offers, a company continually evolves, adapts and redefines itself. Its computer equipment should not be considered as a brake to the implementation of effective operational changes. And finally, it is no longer conceivable to set up a lasting information system that is compartmentalized and incapable of exchanging data or even processes with other companies.

These multiple requirements (integration of technical innovations, adaptation of the company and communication with the outside) result in a simple and incisive conclusion: now more than ever, a company's computer system must be able to evolve rapidly and take into account all of the strategic, tactical and functional constraints that it faces.

Experts in the field (analysts, architects and technologists) all proclaim that the solution must necessarily include the implementation of a service-oriented information system. They all consider that only a service-oriented information system can contribute to making the company more competitive, more profitable and more accessible than ever before.

This thesis presents a study of the evolution of distributed architectures towards service-oriented architecture. After a recall of the key principles of distributed environments, placed in their historical context, it describes the two major architectural models based on distributed services (standards of service-oriented distributed architectures): SUN's J2EE (Java 2 Enterprise Edition) and Microsoft's

.NET. After a comparison of the two models, an e-commerce application (Jurasco Style electronic shop) is developed using the .NET platform. The application includes support for XML Web Services (based on the SOAP and UDDI standards) as well as support for mobile devices.

## CHAPTER 2

### The first generation client-server

Several models have seen the light of day since the eighties. The first and doubtless most famous is client-server architecture. It distinguishes the specific roles (layers of architecture) played by the distributed components: presentation (user interface), application logic (processing), and data management. Each layer makes up a logical and independent processing unit.

In its simplest version, the computer workstation (PC, Mac, etc.) called the "client" is responsible for presentation and global application logic. A server manages data access. There is no surprise in this, as the first generation client-server architecture corresponds to the advent of personal computers and relational databases in companies. The growth of RDBMS (Relational Data Base Management Systems) and the associated SQL (Structured Query Language) largely contributed to this success.

The model was gradually enriched and produced in multiple versions, taking advantage of the increasing maturity of distributed architectures: multitask operating systems for workstations, development of network protocols, implementation of the object model, theories of cooperative processing, data distribution models, etc. In its most sophisticated version, the client-server architecture installs the user interface on the client station and optionally a part of the functional logic. A system for the management of local data can also be part of the architecture.

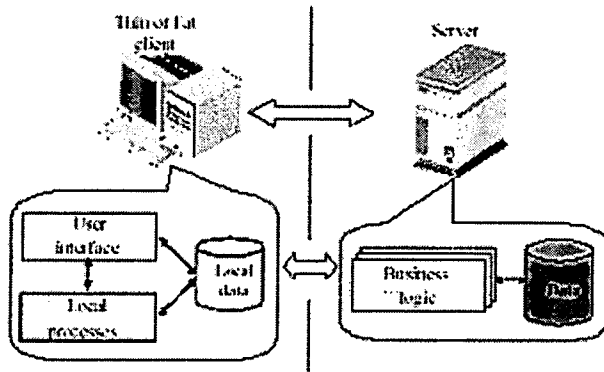


Figure 1 First generation client-server architecture

One or several remote programs (called servers) implement business processing and give access to the shared data, and may even be managed by several RDBMS. We then refer to shared data (appearance of shared relational databases) and distributed processing (with the standardization of remote program-calling mechanisms, including in particular the RPC (Remote Procedure Call) model).

But the idea of a transaction distributed between several machines (for example between the workstation and a server or between several interconnected servers) was not yet mature. In this field, standards were lacking despite the appearance of some interesting proposals of standardized distributed architecture models (such as DCE – Distributed Computing Architecture from the OSF – Open Software Foundation).

The pressure from the final users was partly responsible for the real success encountered by the client-server paradigm at the end of the eighties. The craze that it generated was a result of the user-friendliness (at least theoretical) of the man-machine graphic interface (compared to the awkwardness of screens operating in character mode) and was also explained by the "multi-window" possibilities (still in the development stage) proposed by the microcomputer operating systems (Windows 3 in particular).

An important pitfall was however rapidly discovered: client-server architecture required the explicit installation of application components on each user's workstation. The administration and maintenance of the technical and application stock could become nightmarish and very costly.

The results were however very positive. The client-server model favored the growth of new paradigms for the development and production of user-friendly applications. Associated with the Web and the strong development of networks,

it was the basis of the promising distributed services architectures that were gradually appearing.

## CHAPTER 3

### The object reconciles data and processing

It was at the end of the eighties that the object-oriented development model met with considerable success. This model, whose theoretical bases were outlined at the end of the sixties, simplified the reuse of components and proved to be well adapted to the development of applications in client-server architecture and equipped with a graphic interface.

The object model introduced three key concepts:

- **Class.** This corresponds to an autonomous software unit that "encapsulates" data (called class attributes) and processing (called class methods). In this way, a class resembles a mold (some people refer to a plant) allowing the fabrication of computer structures that model the behavior and the functioning of management objects. Each class contains (encapsulates) a data model (all of the attributes) and a set of processing programs (the methods) that can be applied to the data.

Certain attributes and certain methods may be reserved for internal use, within the object. We can now refer to the interface of a class. It is made up of all the public methods (optionally public attributes) of the class, meaning resources of the class that are accessible by a developer from an application program. The other resources are naturally private.

- **Objects.** These are technical structures created from their mold (the objects are called the "instances" of a class). The attributes (data) of an object possess values that can be modified thanks to the methods associated with the class of which the object is an instance (strong encapsulation). Certain object systems and languages allow the direct manipulation of public attributes from an application program (we call this a weak encapsulation model).

Figure 2 summarizes the three characteristic elements of objects; X its identity (each object is unique), Y the values of its attributes, Z the methods that can be applied to it.

- **A class organization chart.** This chart, called an inheritance graph, enables factorization of the data and processing shared by several classes. Indeed, each class "inherits" attributes and methods from its parent class (simple inheritance) or from its different parents (multiple inheritance, capable of leading to inheritance conflicts). When we look at the attributes and methods encapsulated in the classes located in the upper level of the inheritance graph, we discover the general properties that are characteristic of the objects.

Logically, only the final classes (the inheritance graphs) are destined to be instantiated and to produce objects (instantiation classes). The intermediary level classes are used to factorize knowledge (abstraction classes).

These definitions reveal the duality of the encapsulation concept. Firstly, it designates the fact of building classes containing both data and processing. But it also characterizes the possibility of protecting the data (object attribute values, instance of a class) by imposing the calling of appropriate methods to ensure accessing or updating (in strong encapsulation).

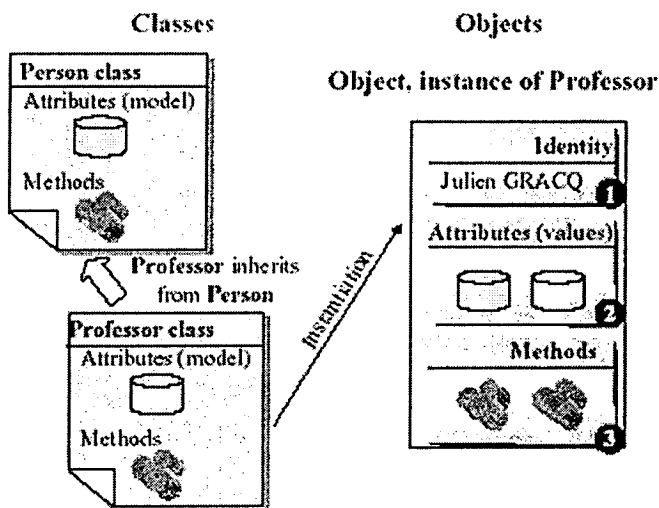


Figure 2 Key elements of the object approach

Many layers and architectural elements of the information system were subject to profound changes linked to the integration of the object approach: the design of operating systems, data management systems, programming languages, software engineering workshops, design and development methods for applications, application architectures, etc. Moreover, as we will see, this model is clearly at the base of service-oriented Distributed Information Systems (DIS).



It should be noted that the mainframe was not lagging behind and also benefited from these innovations, with a slight time lag. This time lag turned out to be sufficient to allow these types of concepts to mature "outside" the more closed and sealed architecture of large proprietary systems.

## CHAPTER 4

### Three-tier architecture

In the mid nineties, with the mainframe still alive and allowing many companies to pursue the development of their activities, the first generation client-server was evolving towards the notion of architecture based on reusable and interoperable components. This time, we can only talk about the client or the server in a contextual manner. A client component is simply the one that sends a service demand to another component, this same "client" being able to act as a "server" in another application context.

Now begins the asking of complex questions inherent to the implementation of a truly distributed architecture, questions that we'll come back to further on:

- How to effectively manage the integrity of data shared between components?
- How to maintain the coherence of this data?
- How to render these types of components truly interoperable?
- How to guarantee the overall security of a distributed system?

The adoption of the "thin client" concept (which happened in stages and cannot be totally assimilated into the appearance of Web browsers) caused the client-server model to evolve: three-tier architecture was born. The first tier is executed on the final user's workstation (PC, Mac, automated teller, PDA, etc.). This station assumes the application's man-machine interface. With the growth of the Web and HTML, the standard configuration of a thin client station henceforth comprised navigation software (browser) capable of interpreting in a dynamic manner (during execution and on-the-fly) the HTML code that it received from another component of the application architecture. The Web server is located in the intermediate level (middle-tier) that also hosts the business components (programs).

The deployment of a new application is thereby considerably simplified: management of the user authorizations relative to the application (paid-up license for the different functions offered by the application) and training of the

new user. No program deployment on the client station is required (excepting, only once, the technical components: browser, communication services, etc). The overall system administration is considerably simplified.

The last tier concerns the mainframe, which was starting to raise serious technical and operational problems of application integration. Although the new applications (front-office, e-Business, etc.) were produced in compliance with the object paradigm, developed using object-oriented languages, in an appropriate software engineering approach, the mainframe was still hosting thousands of batch and transactional programs, "homemade" developments or packages (ERP – Enterprise Resource Planning), often written in PL/1 or in COBOL, or even generated using an integrated CASE (Computer Aided Software Engineering) not necessarily object-oriented (Pacbase for example).

These "closed" programs represented an undeniable functional wealth and there could not reasonably be any question of re-writing them, or even less, simply adapting them to a standardized client-server model.

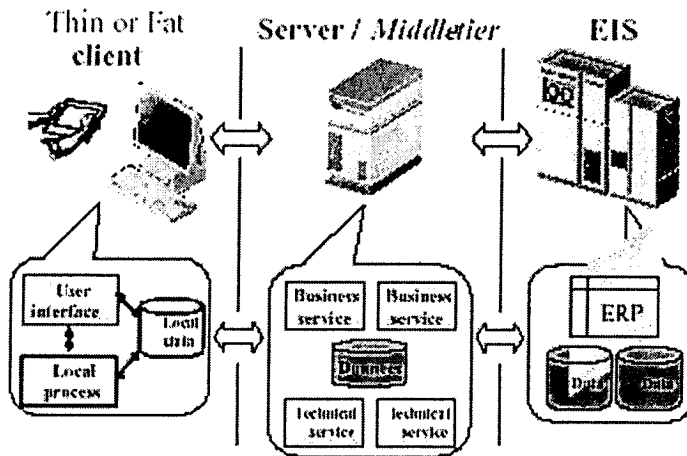


Figure 3 Three-tier architecture

Moreover, mainframes host strategic, reference databases, whose contents are critical for the operation of the company. Large systems should thereby be able to participate in the implementation of distributed architecture. In reality, things are more complex.

Note: generalized client-server architecture does not prohibit the use of "fat clients" in the first tier (the one responsible for the man-machine interface and a part of the application logic), the weaknesses of the classic client-server model thereby reappearing.

## CHAPTER 5

### Communication modes between distributed components

To connect two components within a distributed architecture, different modes of communication and exchange may be used (not including data exchanges performed in store and forward mode, for example through file transfers that may include data transformation operations):

- **Interactive mode.** In this first mode, two distributed components dialogue one with the other, exchange data and messages, generally in a synchronous manner. The establishment of a dialogue requires that the concerned programs be active simultaneously.
- **Request / response mode.** This is a limited version of the interactive mode, involving a single pair of exchanges; a request (query) and its response. The first program (caller, client...) sends a request to the second program (recipient, server...). The caller normally interrupts its execution while waiting for the response. The recipient then performs the requested processing and sends the result to the caller.
- **Message passing mode.** The first component (A) uses a communication channel to send a message (formatted) to the recipient component (B), like a letter sent by a sender for the attention of the recipient (deposited in his mailbox).

The interchange does not block the sender (except if it is waiting on a response in the form of a new message, sent this time by component B using a new communication channel).

- **The use of message queuing.** This is a more sophisticated version of the message-oriented communication mode. This time, the messages posted by component A for the attention of component B are firstly entrusted to an intermediary level of architecture, which in particular ensures their storage in a queue. This middleware guarantees the persistence of exchanges. The recipient may freely consult the queue in order to be informed of the messages addressed to him (like a person picking up their mail in their mail box).

The sender may sometimes choose between posting and porting of messages. In the first case, the communication is performed in an entirely asynchronous manner. In the second, the sender suspends his execution for the time that the sent message has not been delivered to its recipient (pseudo-synchronous communication).

- **Publish & subscribe mode.** Program A uses an intermediary software program to publish messages that could be of interest to other components of the system. For its part, program B calls on the same software in order to subscribe to certain types of messages that it wishes to receive and process.

Numerous message-management modes are proposed by intermediary software (middleware). All of them correspond to specific functional situations. For example, a same message can be deleted (considered as processed) as soon as the appropriate component (subscriber) extracts it from the queue. Conversely, this message may be duplicated, at the sender's request, as many times as required in order to make it available to all of the subscribing components.

Note that functional logic sometimes leads to preventing the applications from managing publication and subscription themselves. In this case, these components dialogue with a centralized hub (in general a message broker) whose behavior is decided by rules defined by an administrator.

This hub is responsible in particular for message transformation (change of format) and routing, according to their type and even their content. Several middleware products now support this mode of exchange, in particular IBM's MQSeries, TIBCO and Microsoft's MSMQ.

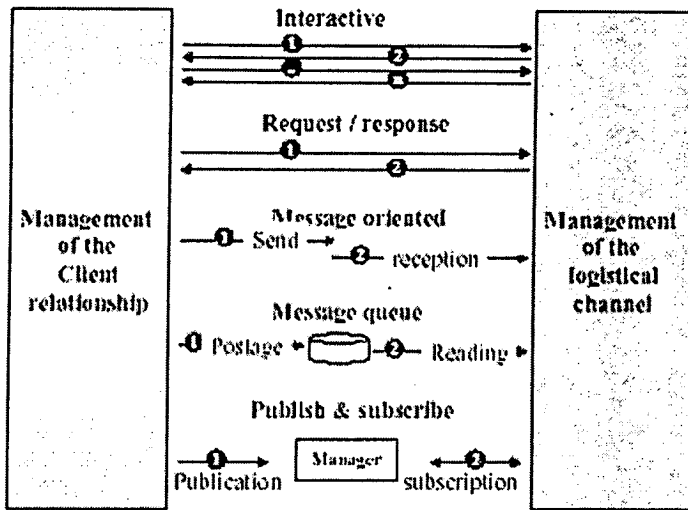


Figure 4 Modes of communication between distributed components

## CHAPTER 6

### Interdependence of components

Intuitively, we perceive that in order to maintain its effectiveness, a distributed architecture must minimize the interdependence between each one of its components: dependencies between components may cause a series of malfunctions, whose causes are often difficult to diagnose. Increasing the independence between components is a complex task, to be taken into account upstream in the cycle of application development or integration.

The dependencies to be taken into account are notably the following:

- **Functional isolation.** Component A calls on component B. The functional developments of B (performed without regression) must under no circumstances affect the behavior of component A.

If the distributed system is well designed, the role of each component is clearly defined and confined. Any change in this role must be controlled and be part of a controlled functional logic.

- **Call methods and returned data.** The signature of a component may evolve during maintenance operations (corrective and evolving) that concern it. It is essential to ensure compatibility of the developments with the potential callers (parameters, data formats, etc.)
- **Localization.** The localization of the called component, the system that hosts it and the resources that it uses must be able to evolve without requiring modification of the caller's behavior.

The object model constitutes a significant step in the search for logical independence between distributed components.

Indeed, encapsulation and the use of public methods follow this same direction by masking the implementation of objects (internal resources, data and processing) from the applications that handle them.

The object approach has a strong influence on system design procedures. The method (meaning the procedures, models and tools) of object design the most widely used throughout the world is UML (Unified Modeling Language) for

which the specification is placed under the control of the OMG [28] (Object Management Group).

The arrival of XML [8] (eXtensible Markup Language) also constitutes real progress. Used within the framework of exchanges between distributed applications, XML and the associated dialects, notably the descriptions of the structure of documents XML – DTD [21] (Document Type Definition) – and more generally the schemas – XSD (XML Schema Definition), allow increased independence between exchanged messages, their technical structure, and the data formats that they contain, etc.

But the company often has a weighty existing structure that has little in common with the object approach or XML. It is therefore difficult to guarantee independence between the various components (often heterogeneous) of the said existing structure, as well as between it and the new applications.



## CHAPTER 7

### Distributed Architectures

A distributed architecture corresponds to a generic description and operation model for a distributed environment. It is therefore a set of principles and rules to be respected in the deployment of a distributed execution environment (including heterogeneous) in order to ensure conformity of the system relative to the specification of the architecture.

The main objectives of this "standardization" are:

- To simplify the apparent operation of the system (to maintain a homogeneous view of a heterogeneous world, to define and respect norms and standards)
- To favor the reuse of software components (functional and technical "bricks")
- To increase system security
- To increase the functional and technical scalability of the system (i.e. to maintain a certain independence of the system relative to the potential technical upgrades of each of its elements).

The quest to develop multi-tier enterprise systems has resulted in considerable industry interest in component architectures [1]. Microsoft's Distributed interNet Architecture (DNA) [2] and Sun's Java 2 Platform Enterprise Edition (J2EE) [3] have both gained strong industry acceptance. These component architectures attempt to define a standard framework for the development of enterprise systems.

#### *7.1 Windows Dynamic interNet Architecture (DNA)*

Microsoft Windows DNA, originally described as the Windows Dynamic interNet Architecture, started out as a set of design guidelines based on Microsoft technologies. These technologies enabled the development, deployment, and management of distributed applications. When DNA was first introduced in 1997, it was described as an architecture for developing Internet

applications. As businesses started taking advantage of Internet and intranet technologies, three-tier applications were being designed with each of the tiers distributed across several machines. Windows DNA both supported and encouraged this move toward distributed applications, and it has evolved into a suite of products, services, and tools that support the creation of distributed applications. Because of this evolution, Microsoft has also dropped the emphasis on Dynamic interNet Architectures and describes Windows DNA as a platform for Web solutions. One important point to remember is, even though the focus is Web solutions, Windows DNA still supports solutions that don't require Web services.

Windows DNA is based on COM [3, 4]. The architecture, as well as most of the products and services that Windows DNA represents, are based on the COM/DCOM programming model. However, Windows DNA isn't only about COM. Windows DNA supports open standards and represents a unified approach for building distributed applications. At the heart of Windows DNA is a three-tier architecture based on a set of design guidelines that use services and products with a common interface. That common interface is COM, which means COM services and products developed up to this point would be supported with Windows DNA. In addition, the investment developers have put into learning COM wouldn't be lost; in fact that knowledge is valuable. Windows DNA makes a lot of sense, is based on existing technologies, supports new technologies, and represents a unified model for building distributed applications.

Table 1 shows an example of some of the products and services, and where they fit into the three-tier architecture. These products represent applications and systems that provide different services, such as managing components and handling HTTP requests.

<b>Product</b>	<b>Services</b>	<b>Tier</b>
Microsoft Access	Database Engine	Data
Microsoft SQL Server	Database Engine	Data
Oracle	Database Engine	Data
Internet Information Server (IIS)	ASP, HTML, Scripting, SSL	Business
Microsoft Exchange Server	MAPI, POP3	Data
Microsoft Transaction Server (MTS)	Distributed Transactions, Resource Management, Component Hosting	Business
Microsoft Message Queue (MSMQ)	Asynchronous Transactions, Messaging	Business
Microsoft Data Access Components (MDAC)	ADO, OLE DB, RDS	Presentation and Business
SNA Server	COM+, Heterogeneous Data	Business
Internet Explorer	HTML, DHTML, Scripting	Presentation
Netscape Navigator	HTML, Scripting	Presentation
Visual Studio	Application Development	Presentation and Business
Windows Operating System	COM/DCOM, COM+, File Services, Active Directory, MSXML	All

**Table 1** Windows DNA Products and Services

The design guidelines Windows DNA proposes are based on using a three-tier architecture. Figure 5 shows a Windows DNA Static View with the tiers labeled according to Windows DNA specifications. Each of the tiers, or layers, in the architecture has different requirements. The following descriptions define those requirements and should help provide a better understanding of how all this fits together.

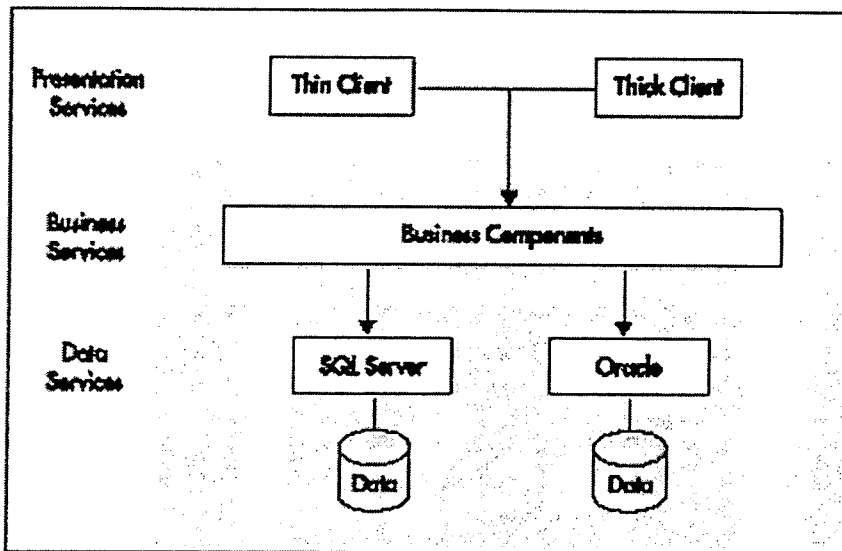


Figure 5 Windows DNA static view

### 7.1.1 Presentation Services

The *Presentation Services layer* represents the user interface. The user interface to a Windows DNA application can be a thin client deployed to a browser, or a thick client deployed as a stand-alone application. The functionality requirements of the Presentation Services deal with providing an interface to the application for users. Thin clients developed for Internet browsers have limited resources compared to stand-alone applications. This may result in different presentation components developed for the same application. Each set of components is focused on managing the user interface and not the business operations behind the interface. Some of the services used at this layer are HTML, DHTML, Scripting, ADO, and RDS.

### 7.1.2 Business Services

The *Business Services layer* is responsible for managing the business operations behind the Presentation Services layer. Business components can reside on the same machine as the presentation components or on a separate machine. This is where COM and DCOM provide benefit. It doesn't matter where you put the component. Applications can link to it at run time using information from the Registry. Business components themselves handle processing requests, implementing business rules and providing a high-level Data Interface layer. It's important to understand that the Data Interface layer here is an abstraction or wrapper to the Data Services layer, which is discussed in the section that follows. Some of the services found at this layer include transaction support, component management, ADO, OLE [6] DB, resource management, and messaging.

For the purpose of Windows DNA, all these services in the middle tier represent Business Services. As previously described, though, several different layers can be within the Business Services tier. This layering is sometimes referred to as an *n-tier design*, which means any number of tiers can be within the design. The main point here is, in the Windows DNA platform, these are all considered part of one tier or layer: Business Services. In alternate platforms, such as .NET, these different tiers can also represent different service layers.

### 7.1.3 Data Services

The *Data Services layer* represents database engines, file systems, and directory services. Database engines can be deployed to a single server or multiple servers. With Windows DNA, the database can also reside on a completely different platform. The Data Service components are responsible for managing the actual data store. With file systems, this means the management of physical data on a hardware medium. Directory Services include the capability to control and manage access to the data. Although some people consider components like ADO and OLE DB as Data Services, they really represent Business Service components that access Data Services in the Windows DNA platform.

## 7.2 the Java J2EE Architecture

The J2EE platform uses a multitiered distributed application model. This means application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on which tier in the multitiered J2EE environment the application component belongs. Figure 6 shows two multitiered J2EE applications divided into the tiers described in the bullet list below. The J2EE application parts shown in Figure 6 are presented in J2EE Application Components.

- Client tier components run on the client machine
- Web tier components run on the J2EE server
- Business tier components run on the J2EE server
- Enterprise information system (EIS) tier software runs on the EIS server

While a J2EE application can consist of the three or four tiers shown in Figure 6, J2EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, J2EE server machine, and the database or legacy machines at the back-end. Three-tiered applications that run in this way extend the standard two-

tiered client and server model by placing a multithreaded application server between the client application and back-end storage.

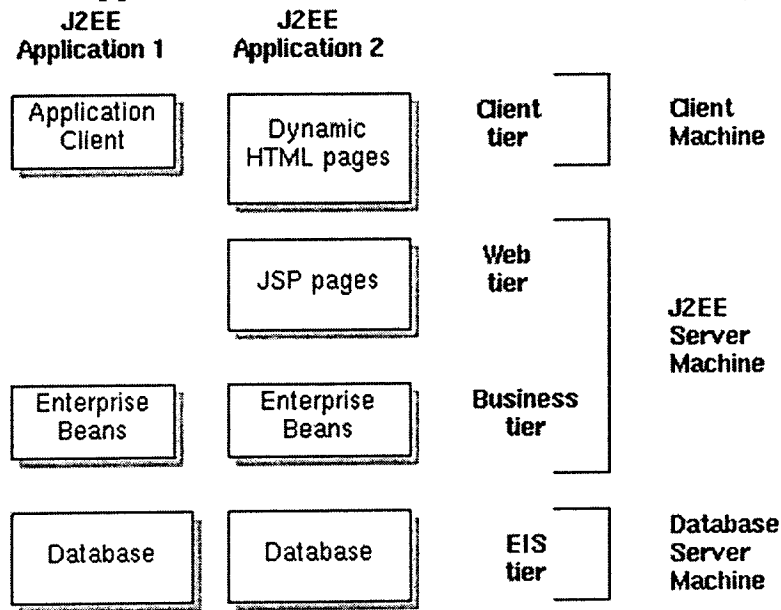


Figure 6 Multi-tiered applications

J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and communicates with other components. The J2EE specification defines the following J2EE components:

- Application clients and applets are client components.
- Java Servlet and JavaServer Pages (JSP) technology components are Web components.
- Enterprise JavaBeans [23] (EJB) components (enterprise beans) are business components.

J2EE components are written in the Java programming language [9] and compiled in the same way as any Java programming language program. The difference when you work with the J2EE platform, is J2EE components are assembled into a J2EE application, verified that they are well-formed and in compliance with the J2EE specification, and deployed to production where they are run and managed by the J2EE server.

### 7.2.1 Client Components

A J2EE application can be Web-based or non-Web-based. An application client executes on the client machine for a non-Web-based J2EE application, and a Web

browser downloads Web pages and applets to the client machine for a Web-based J2EE application.

### **7.2.1.1 Application Clients**

An application client runs on a client machine and provides a way for users to handle tasks such as J2EE system or application administration. It typically has a graphical user interface created from Project Swing or Abstract Window Toolkit (AWT) APIs, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if the J2EE application client requirements warrant it, an application client can open an HTTP connection to establish communication with a servlet running in the Web tier.

### **7.2.1.2 Web Browsers**

The user's Web browser downloads static or dynamic Hypertext Markup Language (HTML), Wireless Markup Language (WML), or Extensible Markup Language (XML) Web pages from the Web tier. Dynamic Web pages are generated by servlets or JSP pages running in the Web tier.

### **7.2.1.3 Applets**

A Web page downloaded from the Web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java VM installed in the Web browser. However, client systems will likely need Java Plug-in and possibly a security policy file so the applet can successfully execute in the Web browser.

JSP pages are the preferred API for creating a Web-based client program because no plug-ins or security policy files are needed on the client systems. Also, JSP pages enable cleaner and more modular application design because they provide a way to separate applications programming from Web page design. This means personnel involved in Web page design do not need to understand Java programming language syntax to do their jobs.

Applets that run in other network-based systems such as handheld devices or car phones can render Wireless Markup Language (WML) pages generated by a JSP page or servlet running on the J2EE server. The WML page is delivered over Wireless Application Protocol (WAP) and the network configuration requires a gateway to translate WAP to HTTP and back again. The gateway translates the WAP request coming from the handheld device to an HTTP request for the J2EE

server, and then translates the HTTP server response and WML page to a WAP server response and WML page for display on the handheld device.

#### 7.2.1.4 JavaBeans Component Architecture

The client tier might also include a component based on the JavaBeans component architecture (JavaBeans component) to manage the data flow between an application client or applet and components running on the J2EE server. JavaBeans components are not considered components by the J2EE specification.

JavaBeans components written for the J2EE platform have instance variables and get and set methods for accessing the data in the instance variables. JavaBeans components used in this way are typically simple in design and implementation, but should conform to the naming and design conventions outlined in the JavaBeans component architecture.

#### 7.2.1.5 J2EE Server Communications

Figure 7 shows the various elements that can make up the client tier. The client communicates with the business tier running on the J2EE server either directly, or as in the case of a client running in a browser, by going through JSP pages or servlets running in the Web tier.

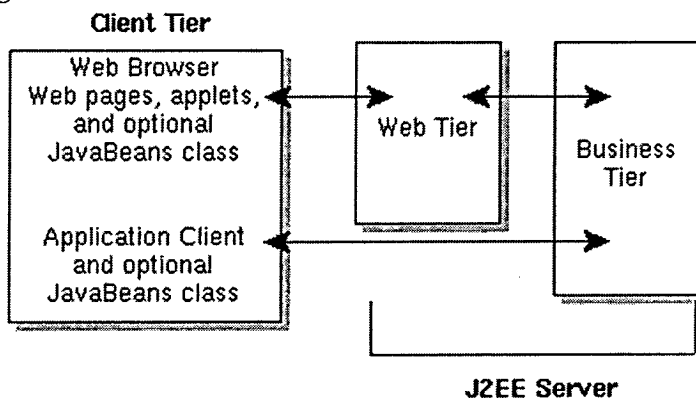


Figure 7 Server communications

#### 7.2.2 Thin Clients

J2EE applications use a thin client. A thin client is a lightweight interface to the application that does not do things like query databases, execute complex business rules, or connect to legacy applications. Heavyweight operations like these are off-loaded to Web or enterprise beans executing on the J2EE server where they can leverage the security, speed, services, and reliability of J2EE server-side technologies.



### 7.2.3 Web Components

J2EE Web components can be either JSP pages or servlets. Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that contain static content and snippets of Java programming language code to generate dynamic content. When a JSP page loads, a background servlet executes the code snippets and returns a response.

Static HTML pages and applets are bundled with Web components during application assembly, but are not considered Web components by the J2EE specification. Server-side utility classes can also be bundled with Web components, and like HTML pages, are not considered Web components.

Like the client tier and as shown in Figure 8, the Web tier might include a JavaBeans object to manage the user input and send that input to enterprise beans running in the business tier for processing.

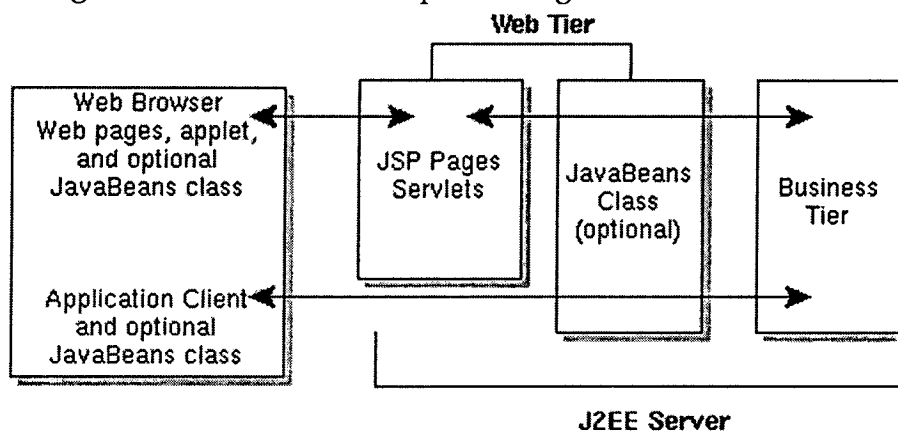


Figure 8 Web tier and J2EE application

#### 7.2.3.1 Business Components

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. Figure 9 shows how an enterprise bean receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans. A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an entity bean represents persistent data stored in one row of a

database table. If the client terminates or if the server shuts down, the underlying services ensure the entity bean data is saved.

A message-driven bean combines features of a session bean and a Java Message Service (JMS) message listener, allowing a business component to receive JMS messages asynchronously. This introduction describes entity beans and session beans.

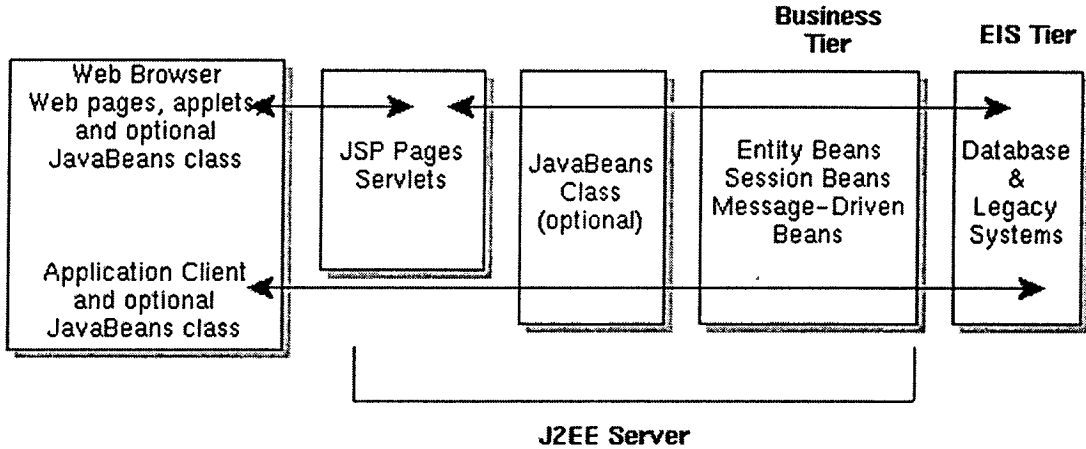


Figure 9 Business and EIS tiers

#### 7.2.4 Enterprise Information System Tier

The enterprise information system tier handles enterprise information system software, and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. J2EE application components might need access to enterprise information systems for database connectivity, for example.

### 7.3 Windows DNA vs. Java J2EE

The following table illustrates the differences in application architecture influenced by the respective DNA and J2EE base technologies.

Service	Windows DNA	J2EE
Operating Systems	Windows CE/95/98/NT/2000	Any operating system
Browser	Internet Explorer	Any browser
Browser Components	ActiveX [6] Controls	Java Applets
Web Server	Internet Information Server	Any Web server
Web Server Components	Active Server Pages	Java Server Pages and Servlets
Application Server	Microsoft Transaction Server	Any EJB Compliant Server (there are about 20 to choose from)
Application Server Components	MTS Components	Enterprise Java Beans
Communication Protocol	DCOM	IIOP
Database Access	ADO and OLE DB	JDBC
Transaction Management	Microsoft Distributed Transaction Coordinator	Any transaction service through JTA (Java Transaction APIs)
Messaging	Microsoft Message Queuing	Any messaging service through JMS (Java Messaging Service)
Security	Windows NT Security Services	Any security service through Java Security Services
Directory	Windows Registry	Any directory service through JNDI

**Table 2** Windows DNA J2EE comparison

## CHAPTER 8

### Service-oriented distributed architecture

This architecture, directly inspired by the object model, is based on the notion of service, be it technical or functional (business).

#### *8.1 Business and Technical Services*

If the generalized client-server model allows the reuse of certain elements of the information system, it is nonetheless based on the concept of tight coupling between components and cannot guarantee total independence between them.

Developers wanting to integrate a component into an application must be aware of its existence, the overall operation and the signature (call modes, entry points, expected parameters and returned data). This type of architecture remains very sensitive to change and makes the different elements of the system more interdependent than truly interoperable.

The limits of the approach are particularly felt in a changing economic context and a market in permanent development requiring the adaptation of the overall operation of the information system.

What's more, new business practices (one-to-one marketing, e-Business, etc.) impose the fact that applications designed independently and in an isolated manner will be communicating. Moreover, the overall effectiveness and relevance of the system depend heavily on the people responsible for its set-up, its maintenance and its daily operation. These observations generate the need for a more flexible architecture, where the components are really independent and autonomous, capable of rapidly deploying new applications. These are the objectives of the transition from a generalized client-server architecture, heavily based on the object model (object-oriented distributed architecture), to a service-oriented distributed architecture.

The principal idea behind service-oriented architecture is that any element of the information system must become a service:

- Clearly identifiable. To identify a service signifies being able to know of the existence, independently of its localization, its operating mode, its call mode, etc.
- Performing a set of perfectly defined tasks. This means understanding the overall operation of the service, meaning knowing the list of tasks that it is capable of processing, its operational execution conditions, the exceptions that it may cause, etc.
- Documented. The documentation must clearly describe how to call on the service. It must clearly specify (and if possible in a unified or even standardized manner) the call methods, the proposed functions and the signature of the service.
- Autonomous and equipped with a controlled security level.
- Reliable in a given context. The reliability of a service naturally depends on its context of use. These must be clearly identified and the service's operating conditions explicitly described.
- Independent relative to other services (even if able to call on some of them).
- Accessible on the network.

As you can see, the service notion is based on many concepts stemming from the object model (for example, the implementation details of a service only interest its developers and not its users).

More precisely, a service must be:

- Encapsulated. Its technical implementation is masked by the methods that it makes available to the other components of the information system (certain methods naturally remain reserved for the private use of the developers of the service, rather like the internal and private resources of an object)
- Available to application developers
- Documented (localization, call method, generated exceptions, signature, etc.)

In short, an application service is a component that may be described, published (made available to other services), localized and called in a remote manner. Note: The notion of a Web Service corresponds to the notion of an application service, with the addition of a constraint on component accessibility from the Internet network.

A service-oriented architecture requires the definition of three key concepts:

- How to publish the existence of a service (identity, description, signature, etc.)?
- How to allow other components to find the services available in the core of a distributed architecture?
- How to allow these same components to bind with a service and to execute the processing that it proposes?

Today, these three questions (however simple) preoccupy many people and are at the core of major debates with important stakes. Certain invariants have been clearly revealed by such studies:

- Service-oriented architectures must make full use of the strengths of the object approach. In particular, services (Web Services and Business Services) must behave like objects. Then developing a new application becomes essentially building an assembly of components, according to an order and to rules that depend entirely and only on the activity and the functional operation of the company.
- The system must provide a group of technical services facilitating the task of developers and guaranteeing total independence between the business services and the technical architecture.
- XML must play a major role in the service orientation. As it is simple, open, offering appropriate specialization abilities and sufficiently widespread, XML is a guarantee of simplicity and scalability. It should in particular facilitate the description of services, communication between services and data exchanges. Beyond the purely "marketing" functions proposed by certain products on the market, the orientation towards XML reveals itself to be promising.

From among the main proposals of architecture models based on distributed services (standards of service-oriented distributed architectures), we'll mention in particular SUN's J2EE (Java 2 Enterprise Edition) and Microsoft's .NET, as well as their different predecessors such as the Object Management Group's CORBA [26] (Common Object Request Broker Architecture), Distributed Smaltalk, Java RMI [16] (Remote Method Invocation) and even Microsoft's COM/DCOM.

## 8.2 *XML and Web services*

A simplified heir to the document management language SGML (ISO compliant since 1986), XML is a tagged text language rather like HTML. But XML is both more structuring and more open than HTML (moreover XHTML proposes an elegant formulation of HTML 4 in XML) and enables the description of

structured contents using strict rules and conventions, which are astonishingly simple.

The first version of the standard (XML 1.0) appeared in June 1996 and specified the rules for constructing a well formed XML document (compliance with XML syntax and its writing conventions) and the compliance conditions of the document (valid document) relative to a model, a standard structure: a DTD (Document Type Definition) or an XSD schema (XML Schema Definition).

These rules are unchanging and common to all of the forms of the language. The constantly increasing number of these forms reflects the numerous fields of use of the language.

XML adapts to a multitude of technical and functional contexts for which specific extensions (essentially predefined sets of tags) were ratified by the W3C (World Wide Web Consortium), the organization responsible for the standardization of XML and its derivatives.

Firstly XLink describes how to add hyperlinks to a XML document. XPointer and XFragments enable the management of pointers between document fragments. The definition language of CSS style sheets, applicable to HTML, is also applicable to XML documents. An extended and exclusively XML version of CSS is proposed with XSL, itself a derivative of XSLT, a powerful language enabling the specification of rules for the transformation, addition or deletion of tags and attributes in a document.

In order to facilitate the use of the language in varied fields, and to avoid the risk of semantic collisions between tags, XML enables the creation of namespaces: collections of reserved names (tags and attributes) associated with a single identifier (an URI). An XML document may contain orders that are of interest to a determined application, which will recognize and interpret the predefined tags of the appropriate namespace.

From this point on, XML permitting the encapsulation of any data structures and application processing, like the object model, there is no longer anything standing in the way of its generalization within the various layers of the information system.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE MainframeOptimist SYSTEM="mo.dtd" >
<Component ID="45001" >
<Call_model>Synchronous</Call_mode>
<Access>API</Access>
<SubSys>CICS</SubSys>
<Method>
<Name>Creation_Account</Name>
<Parameters>
...
</Method>
```

```
<Method>  
<Name>Closing_Account</Name>  
<Parameters>  
...  
</ Component >
```

Figure 10 Example of an XML document

To assist developers in the processing of the XML flow, the DOM (Document Object Model) standardizes a set of document manipulation functions from a programming language (C, C++, Java, VB, etc.): analysis of XML structures (parser), syntax control, validation of a document relative to a DTD or a XSD schema, extraction of document fragments, manipulation of nodes, etc.

A rival proposal called SAX [24] (Simple API for XML) was produced by independent Java developers and defined a specific API for this language.

XPath enables navigation in a XML flow represented as a tree structure where the nodes are tags. As concerns the transactional manipulation of XML data (in an RDBMS), several proposals aiming to define a query language based on the contents of documents have appeared: XQL, QXML, etc.

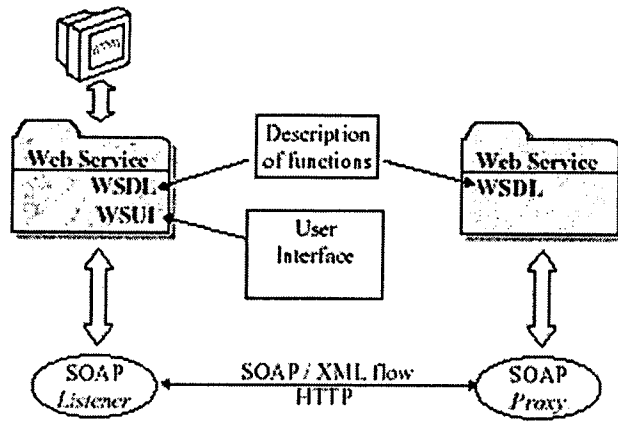
Recently, XMLQuery (recommendation proposal dated June 7 2001) attempts to group the main versions of the query language dialects within a single consistent standard and also to extend these dialects to data management whose structure may be of an arbitrary complexity.

Moreover, the big RDBMS (DB2, Oracle, SQL Server) have progressively integrated XML type data and document manipulation functions (external to or inside the same engine). Finally a new generation of native XML data managers have already appeared (Software AG's Tamino notably).

XML plays an essential role in the deployment of a service-oriented architecture. Each service must in fact list and describe the functions that it proposes, rather like IDL (Interface Definition Language) files in OMG's CORBA (Common Object Request Broker Architecture). This type of description can easily be done in XML.

This is moreover the proposal of a several working groups that are trying to define the generic implementation of an architecture based on Web services.





**Figure 11** Architecture based on distributed objects – WSDL, WSUI, and SOAP

In this field, WSDL (Web Services Description Language) and WSUI (Web Services User Interface) standards have newly been created. According to these proposals, a Web service proposes functions, answers to a group of outside stimuli (actions of the final user, specific events, etc.) and if required generates a data flow (XML or not). The proposed functions are described within XML documents, in a format defined by the WSDL standard.

The WSUI proposal is focused on the standardized definition of a generic user interface for Web services. Like WSDL, WSUI takes full advantage of XML. Finally, the SOAP [5] protocol (Simple Object Access Protocol) conveys messages between different Web services (clients and servers) via HTTP. Naturally, these messages are written in XML.

### 8.3 "Loosely coupled" interfaces

Web services are considered "loosely coupled" because they are extremely abstract, unlike their predecessors CORBA and COM/DCOM [29]. The user need not know much about a Web service in order to use one. This is because a Web service will describe itself telling you how to use it, where to find it, etc. Even more remarkable, the architecture of a Web service hides the "plumbing" of the service, including the language it is written in, the platform it runs on, the type of network, O/S, and hardware it runs on, etc.

In contrast, a "tightly coupled" interface requires a tremendous amount of prior knowledge about a service in order to use it. This is because it is tightly tied to a particular platform, O/S, or language.

The advantage of a "loosely coupled" approach is flexibility and convenience. You can change every single piece of the plumbing (O/S, platform, language, and

hardware) without the consumer of the service knowing or caring. You can even add capabilities. This is because the interface never changes and therefore applications and services calling the service do not have to change either. A consumer of the services need only access the capabilities that apply to her, thus reducing the need to "upgrade" to features she does not need.

## CHAPTER 9

# J2EE vs. Microsoft.NET: A comparison of building XML-based Web services

Although both J2EE and .NET cover a great deal of technologies and standards, we will focus specifically on building server-side systems as Web services using these architectures (for example, we will not be mentioning Jini or Office XP).

The first half of this chapter is background information about Web services, J2EE, and .NET. The 2nd half is the comparison.

### *9.1 Introduction*

Prior to the advent of Web services, enterprise application integration was very difficult due to differences in programming languages and middleware used within organizations. The chances of any two business systems using the same programming language and the same middleware were slim to none, since there has not been a de-facto winner. These 'component wars' spelled headaches for integration efforts, and resulted in a plethora of custom adapters, one-off integrations, and integration 'middlemen'. In short, interoperability was cumbersome and painful.

With Web services, any application can be integrated so long as it is Internet-enabled. The foundation of Web services is XML messaging over standard Web protocols such as HTTP. This is a very lightweight communication mechanism that any programming language, middleware, or platform can participate in, easing interoperability greatly. These industry standards enjoy widespread industry acceptance, making them very low-risk technologies for corporations to adopt. With Web services, you can integrate two businesses, departments, or applications quickly and cost-effectively.

The vision for Web services predicts that services will register themselves in public or private business registries. Those Web services will fully describe themselves, including interface structure, business requirements, business processes, and terms and conditions for use. Consumers of those services read these descriptions to understand the abilities of those Web services. Web services

will be smart, in that once a service has been invoked, it will spontaneously invoke other services to accomplish the task and to give users a completely personal, customized experience. In order for these services to dynamically interact, they need to share information about the user's identity, or context information. That context information should only need to be typed in once, and then made available at the user's discretion to selected Web services.

## 9.2 *Building Web services with technologies that have gained the most acceptance*

Now that we've seen the general philosophy behind Web services, let's look at how to build and use a Web service. Web services are in reality simply XML-based interfaces to business, application, and system services, and are really old technologies wearing a new hat. The following technologies that have gained the most industry acceptance, and is one possible way to perform Web services:

- A provider creates, assembles, and deploys a Web service using the programming language, middleware, and platform of the provider's own choice.
- The provider defines the Web service in WSDL (the Web Services Description Language). A WSDL document describes a Web service to others.
- The provider registers the service in UDDI [10] (Universal Description, Discovery, and Integration) registries. UDDI enables developers to publish Web services and that enables their software to search for services offered by others.
- A prospective user finds the service by searching a UDDI registry.
- The user's application binds to the Web service and invokes the service's operations using SOAP (the Simple Object Access Protocol). SOAP offers an XML format for representing parameters and return values over HTTP. It is the communications protocol that all Web services use.

Note that the above technologies are only sufficient for simple Web services. Extended business exchanges require an agreed-upon structure for business transactions, multi-request transactions, schemas, and document flow. These application requirements often stretch the limits of a purely SOAP based implementation. This is the motivation for ebXML, which is a suite of XML specifications and related processes and behavior designed to provide an e-infrastructure for B2B collaboration and integration.

Note that the above approach is but one way of making Web services work. There are other choices as well, but we feel that these technologies are the most important and will achieve the widest industry adoption. Because of this, in reality, we really haven't reached complete consensus on building Web services, and there are still a lot of issues to be resolved. For example, there is vendor disagreement on SOAP extensions, ebXML, and service flow descriptions. The good news is that:

- For once, all major players, including Sun and Microsoft, generally agree that SOAP, WSDL, and UDDI are good things and that they (or their standard derivatives) will provide a foundation for the future.
- All the vendors are working together to establish Web services standards, and a foundation is emerging.

### *9.3 The J2EE and Microsoft.NET approach to Web Services*

If you want to build a usable Web services system, there is more than meets the eye. Your Web services must be reliable, highly available, fault-tolerant, scalable, and must perform at acceptable levels. These needs are no different than the needs of any other enterprise application.

J2EE and .NET are evolutions of existing application server technology used to build such enterprise applications. The earlier versions of these technologies have historically not been used to build Web services. Now that Web services has arrived, both camps are repositioning their solutions as platforms that you can also use to build Web services.

The shared vision between both 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.

This paradigm allows you to specialize in your proficiencies. If you were a financial services firm, for example, you'd have proficiency in financial services, but likely very little proficiency in Web services plumbing compared to a specialist such as Sun, IBM, BEA, Oracle, or Microsoft. By purchasing the container off-the-shelf, you won't need to be an expert at plumbing to build a financial services-based Web service. Rather you just need to understand their business problem at hand, and leave the Web service plumbing to the container.

With that said, let's take a look at the details of each vision.

## 9.4 J2EE

The Java 2 Platform, Enterprise Edition (J2EE) was designed to simplify complex problems with the development, deployment, and management of multi-tier enterprise solutions. J2EE is an industry standard, and is the result of a large industry initiative led by Sun Microsystems.

It's important for you to realize that J2EE is a standard, not a product. You cannot "download" J2EE. Rather you download a set of Adobe Acrobat PDF files which describe agreements between applications and the containers in which they run. So long as both sides obey the J2EE contracts, applications can be deployed in a variety of container environments.

The J2EE camp's goal is to give customers choice of vendor products and tools, and to encourage best-of-breed products to emerge through competition. The only way this would ever happen is if the industry as a whole were bought-into J2EE. To secure buy-in, Sun collaborated with other vendors of eBusiness platforms, such as BEA, IBM, and Oracle, in defining J2EE. Sun then initiated the Java Community Process (JCP) to solicit new ideas to improve J2EE over time. The reason Sun did this is because they had to do so to achieve success--the best way to secure buy-in to an idea is to involve others in defining that idea.

### 9.4.1 Java: The foundation for J2EE

The J2EE architecture is based on the Java programming language. What's exciting about Java is that it enables organizations to write their code once, and deploy that code onto any platform. The process is as follows:

1. Developers write source code in Java.
2. The Java code is compiled into **bytecode**, which is a cross-platform intermediary, halfway between source code and machine language.
3. When the code is ready to run, the Java Runtime Environment (JRE) interprets this bytecode and executes it at run-time.

J2EE is an application of Java. Your J2EE components are transformed into bytecode and executed by a JRE at runtime. Even the containers are typically written in Java.

### 9.4.2 J2EE and Web Services

J2EE has historically been an architecture for building server-side deployments [1] in the Java programming language. It can be used to build traditional Web sites [2], software components, or packaged applications. J2EE has recently been extended to include support for building XML-based Web services as well. These

Web services can interoperate with other Web services that may or may not have been written to the J2EE standard.

J2EE Web services development model is shown in Figure 12.

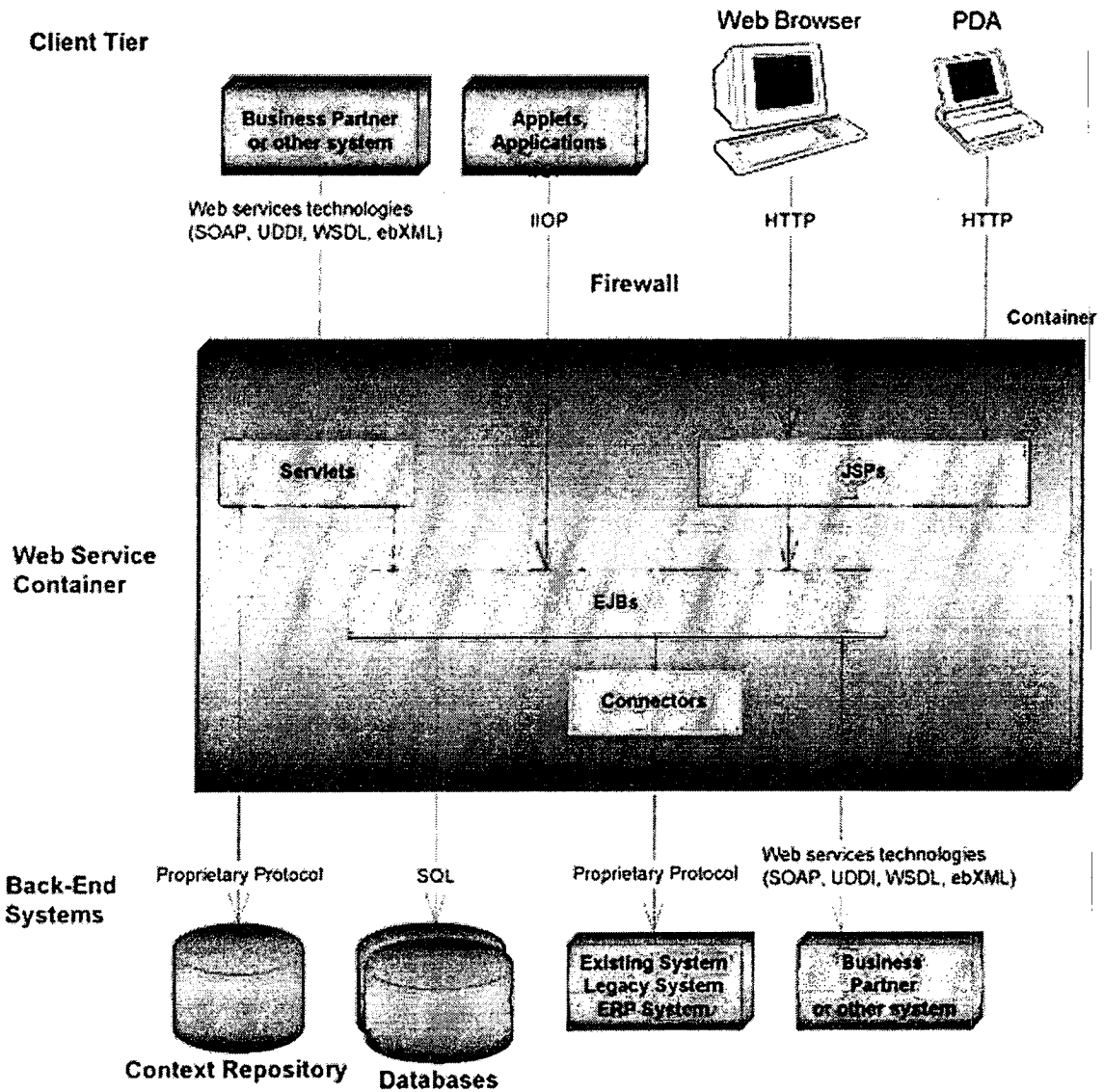


Figure 12 Developing Web services with J2EE

Briefly, Figure 12 is explained as follows:

J2EE application is hosted within a container, which provides qualities of service necessary for enterprise applications, such as transactions, security, and persistence services.

The business layer performs business processing and data logic. In large-scale J2EE applications, business logic is built using Enterprise JavaBeans (EJB)

components. This layer performs business processing and data logic. It connects to databases using Java Database Connectivity (JDBC) or SQL/J, or existing systems using the Java Connector Architecture (JCA). It can also connect to business partners using Web services technologies (SOAP, UDDI, WSDL, ebXML) through the Java APIs for XML (the JAX APIs).

Business partners can connect with J2EE applications through Web services technologies (SOAP, UDDI, WSDL, ebXML). A servlet, which is a request/response oriented Java object, can accept Web service requests from business partners. The servlet uses the JAX APIs to perform Web services operations. Shared context services will be standardized in the future through shared context standards that will be included with J2EE.

Traditional 'thick' clients such as applets or applications connect directly to the EJB layer through the Internet Inter-ORB Protocol (IIOP) rather than Web services, since generally the thick clients are written by the same organization that authored J2EE application, and therefore there is no need for XML-based Web service collaboration.

Web browsers and wireless devices connect to JavaServer Pages (JSPs) which render user interfaces in HTML, XHTML, or WML.

## ***9.5 Microsoft's .NET Platform***

Microsoft.NET is product suite that enables organizations to build smart, enterprise-class Web services. Note the important difference: .NET is a product strategy, whereas J2EE is a standard to which products are written.

Microsoft.NET is largely a rewrite of Windows DNA, which was Microsoft's previous platform for developing enterprise applications. Windows DNA includes many proven technologies that are in production today, including Microsoft Transaction Server (MTS) and COM+ [11], Microsoft Message Queue (MSMQ), and the Microsoft SQL Server database. The new .NET Framework replaces these technologies, and includes a Web services layer as well as improved language support.



The developer model for building Web services with Microsoft.NET is shown in Figure 13.

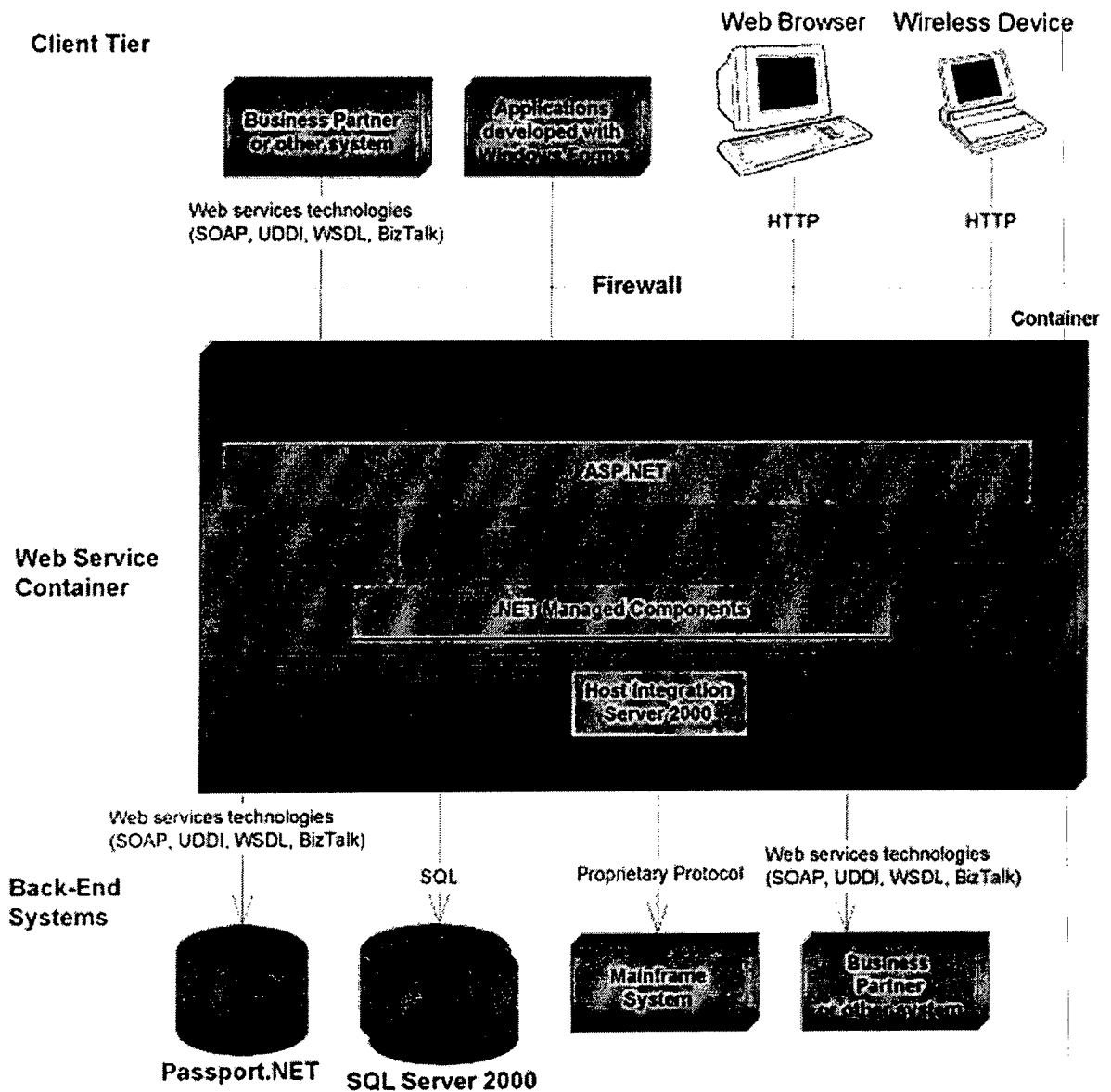


Figure 13 Developing Web services with Microsoft.NET

Briefly, Figure 13 is explained as follows:

The .NET application is hosted within a container, which provides qualities of service necessary for enterprise applications, such as transactions, security, and messaging services.

The business layer of the .NET application is built using .NET managed components. This layer performs business processing and data logic. It connects to databases using Active Data Objects (ADO.NET) and existing systems using

services provided by Microsoft Host Integration Server 2000, such as the COM Transaction Integrator (COM TI). It can also connect to business partners using Web services technologies (SOAP, UDDI, WSDL).

Business partners can connect with the .NET application through Web services technologies (SOAP, UDDI, WSDL, BizTalk).

Traditional 'thick' clients, Web browsers, wireless devices connect to Active Server Pages (ASP.NET) which render user interfaces in HTML, XHTML, or WML. Heavyweight user interfaces are built using Windows Forms.

### 9.5.1 The .NET Framework

Microsoft.NET offers language-independence and language-interoperability. This is one of the most intriguing and fundamental aspects of the .NET platform. A single .NET component can be written, for example, partially in VB.NET, the .NET version of Visual Basic, and C#, Microsoft's new object-oriented programming language.

How does this work? First, source code is translated into Microsoft Intermediate Language, sometimes abbreviated MSIL, sometimes IL. This IL code is language-neutral, and is analogous to Java bytecode.

The IL code then needs to be interpreted and translated into a native executable. The .NET Framework includes the Common Language Runtime (CLR), analogous to the Java Runtime Environment (JRE), which achieves this goal. The CLR is Microsoft's intermediary between .NET developers' source code and the underlying hardware, and all .NET code ultimately runs within the CLR.

This CLR provides many exciting features not available in earlier versions of Windows DNA, such as automatic garbage collection, exception handling, cross-language inheritance, debugging, and "side-by-side" execution of different versions of the same .NET component.

### 9.5.2 .NET Servers

The .NET platform includes the following .NET Enterprise Servers. Many of these are repackagings of existing products under a common marketing term:

- SQL Server 2000 is Microsoft's relational database.
- Exchange 2000 Server is a messaging and collaboration platform useful in developing and running core business services and is now tightly integrated with Windows 2000.

- Commerce Server 2000 offers you quicker and less complicated development and deployment of customizable online e-commerce solutions.
- Application Center Server 2000. Application Center Server 2000 lets you manage clustered servers.
- Host Integration Server 2000. Host Integration Server 2000 gives you access to selected legacy systems running on other platforms (primarily IBM-based).
- Internet Security and Acceleration (ISA) Server 2000 offers firewall and Web caching capabilities.
- BizTalk Server 2000 is Microsoft's XML-based collaborative e-business solution for integrating applications, trading partners and business processes via the Internet.

## 9.6 Understanding J2EE and .NET by analogy

To help you understand both models, we offer analogies between J2EE and .NET technologies in Table 3. This table only showcases the similarities--we will get to the differences in a few moments.

Feature	J2EE	.NET
Type of technology	Standard	Product
Middleware Vendors	30+	Microsoft
Interpreter	JRE	CLR
Dynamic Web Pages	JSP	ASP.NET
Middle-Tier Components	EJB	.NET Managed Components
Database access	JDBC SQL/J	ADO.NET
SOAP, WSDL, UDDI	Yes	Yes
Implicit middleware(load-balancing, etc)	Yes	Yes

**Table 3** Analogies between J2EE and .NET

## 9.7 Comparative Analysis

### 9.7.1 Single-Vendor Solution

When building Web services, in general you should always prefer to have a single-vendor solution. A single vendor solution is usually more reliable, interoperable, and less error-prone than a two-vendor bridged solution.

One of J2EE's strengths is that it has spawned a wide variety of tools, products, and applications in the marketplace, which provide more functionality in total than any one vendor could ever provide. However, this strength is also a weakness. J2EE tools are often-times not interoperable, due to imperfections in portability. This limits your ability to mix and match tools without substantial low-level hacking. With lower-end J2EE implementations, you need to mix and match to get a complete solution, and this is the tradeoff when choosing a less complete package. Larger vendors, such as IBM, Oracle, BEA, and iPlanet, each offer a complete Web services solution.

.NET provides a fairly complete solution from a single vendor--Microsoft. This solution may lack some of the higher end features that J2EE solutions offer, but in general, the complete Web services vision that Microsoft will be providing is equal in scope to that of a larger J2EE vendor.

Another way to look at a single-vendor solution is from a legacy perspective. Many legacy systems are written by J2EE vendors, such as IBM or BEA. J2EE offers a single-vendor solution from the legacy integration perspective, since you can re-use existing relationships with those vendors. A J2EE solution would therefore be a single-vendor solution, since you can stay with that legacy system vendor rather than inject a new vendor such as Microsoft. For users with existing Microsoft-based systems, the reverse argument applies.

### 9.7.2 Support for Existing Systems

Most large corporations have existing code written in a variety of languages, and have a number of legacy systems, such as CICS/COBOL, C++, SAP R/3, and Siebel. It is vital that corporations be given an efficient, rapid path to preserve and reuse these investments. After all, it is likely that businesses will have neither the funds nor the time to reinvent all existing systems. This legacy integration often is one of the most challenging (if not the most challenging) tasks to overcome when building a Web service.

There are several ways to achieve legacy integration using J2EE, including

- The Java Message Service (JMS) to integrate with existing messaging systems
- Web services to integrate with any system
- CORBA for interfacing with code written in other languages that may exist on remote machines.
- JNI for loading native libraries and calling them locally.

But by far, the most important part of the J2EE vision for integration is the J2EE Connector Architecture (JCA). The JCA is a specification for plugging in resource adapters that understand how to communicate with existing systems, such as SAP R/3, CICS/COBOL, Siebel, and so-on. If such adapters are not available, you can write your own adapter. These adapters are reusable in any container that supports the JCA. The major vendors of existing systems are bought into the JCA.

.NET also offers legacy integration through the Host Integration Server 2000. COM Transaction Integrator (COM TI) can be used for collaborating transactions across mainframe systems. Microsoft Message Queue (MSMQ) can integrate with legacy systems built using IBM MQSeries. Finally, BizTalk Server 2000 can be used to integrate with systems based on B2B protocols, such as Electronic Data Interchange (EDI) (the reader should note, however, that BizTalk does not serve as an access point to a proprietary network on which EDI takes place).

In conclusion, we believe that the legacy integration features offered by J2EE are superior to those offered by .NET. The JCA market is producing a marketplace of adapters that will greatly ease enterprise application integration. Integration with packaged applications and legacy systems will become much easier--imagine integrating with a system such as Siebel, Oracle, or SAP without ever leaving the Java programming environment. There is no analog to this in the Microsoft domain; rather, there is limited connectivity to select systems provided off-the-shelf through the Host Integration Server

### **9.7.3 Language Support**

J2EE promotes Java-centric computing, and as such all components deployed into a J2EE deployment (such as EJB components and servlets) must be written in the Java language. To use J2EE, you must commit to coding at least some of your eBusiness systems using the Java programming language. Other languages can be bridged into a J2EE solution through Web services, CORBA, JNI, or the JCA, as previously mentioned. However, these languages cannot be intermixed with Java code. In theory, JVM bytecode is language-neutral, however in practice, this bytecode is only used with Java.

By way of comparison, .NET supports development in any language that Microsoft's tools support due to the new CLR. With the exception of Java, all major languages will be supported. Microsoft has also recently introduced its new C# language which is equivalent (with the exception of portability) to Java and is also available as a programming language within the Visual Studio.NET environment. All languages supported by the CLR are interoperable in that all

such languages, once translated to IL, are now effectively a “common” language. A single .NET component can therefore be written in several languages.

The multiple language support that Microsoft has introduced with the CLR is an exciting innovation for businesses. It is clearly a feature advantage that .NET has over J2EE. But is the CLR a business advantage for you? This is a more interesting discussion. We are a bit concerned that the CLR may represent a poor design choice for you if more than one language is used. This is for the following reasons:

- **Risk.** Many existing systems are internally convoluted. Disrupting such existing systems is a risky proposition, since knowledgeable engineers, original source code, and a general understanding of the existing system are often-times unavailable. The old adage, "if it ain't broke, don't fix it" applies here.
- **Maintainability.** We speculate that a combination of languages running in the CLR may lead to a mess of combination spaghetti code that is very difficult to maintain. If you have an application written in multiple languages, then to fully develop, debug, maintain, and understand that application, you will need experts in different languages. The need to maintain code written in several languages equates to an increase in developer training expenditures which contributes further to an increased total cost of ownership.
- **Knowledge building.** With combination language code, your developers are unable to share best practices. While individual productivity may increase, communication breaks down, and team productivity decreases.
- **Skills transfer.** While developers using different languages may have very quickly coded a .NET system using VB.NET and C#, what happens if the new C# developers leave your organization? You have two choices. Train your VB.NET developers to understand and write code with C#, or hire other C# developers who know nothing about your code base. The resulting lack in productivity equates to a reduced time to market and a higher total cost of ownership.

In most cases, we feel that it's much better design to standardize on a single language, and to treat legacy systems as legacy systems and integrate with them by calling them through legacy APIs, which can be achieved using either J2EE or .NET.

We do feel the CLR still adds significant value. The value is that a new eBusiness application can be written in a single language of choice other than Java. This is

useful for organizations that are not ready to embrace Java. However, again we must provide words of caution.

- It will not be seamless to transition existing developers from their familiar language into productive .NET developers. Procedural languages such as COBOL and VB are being rewritten for .NET to be object-oriented. Teaching developers object-oriented programming is much more of a stepping stone than understanding syntactical rules.
- Languages such as COBOL or VB were never intended to be object-oriented. Legacy code will not seamlessly transition into .NET. The resulting code is forever bound to .NET and can never be taken from its .NET home.
- We question the general wisdom in reinvesting in outdated technologies, such as COBOL, with new eBusiness or Web services initiatives.

In summary, there are pros and cons to both approaches to language support. Use the approach that best suits your business needs, but at least be aware of the consequences of your decision.

#### **9.7.4 Portability**

A key difference between J2EE and .NET is that J2EE is platform-agnostic, running on a variety of hardware and operating systems, such as Win32, UNIX, and Mainframe systems. This portability is an absolute reality today because the Java Runtime Environment (JRE), on which J2EE is based, is available on any platform.

There is a second, more debatable aspect of portability as well. J2EE is a standard, and so it supports a variety of implementations, such as BEA, IBM, and Sun. The danger in an open standard such as J2EE is that if vendors are not held strictly to the standard, application portability is sacrificed. CORBA, for example, did not have any way to enforce that CORBA middleware did indeed comply with the standard, and thus there were numerous problems with portability. In the early days of J2EE there were the same problems.

To help with the situation, Sun has built a J2EE compatibility test suite, which ensures that J2EE platforms comply with the standards. This test suite is critical because it ensures portability of applications. At the time of this writing, there were 18 application server vendors certified as J2EE-compatible. There are a myriad of other vendors as well that are not certified<sup>10</sup>.

Our opinion is that in reality, J2EE portability will never be completely free. It is ridiculous to think that complex enterprise applications can be deployed from

one environment to the next without any effort, because in practice, organizations must occasionally take advantage of vendor-specific features to achieve real-world systems. However--and this is important--portability is exponentially cheaper and easier with J2EE and the compatibility test suite than with proprietary solutions, and that is a fact we stand behind through years of consulting with customers using a variety of J2EE solutions. Over time, as the J2EE compatibility test suite becomes more and more robust, portability will become even easier.

By way of comparison, .NET only runs on Windows, its supported hardware, and the .NET environment. There is no portability at all. It should be noted that there have been hints that additional implementations of .NET will be available for other platforms. However, a question remains - how much of the complete .NET framework will be (or even can be) supplied on other platforms? History has taught us to be skeptical of Microsoft's claims of multiple platform support. Microsoft ported COM to other platforms, but never ported the additional services associated with COM that were necessary to make COM useful. We find it hard to believe that .NET portability will ever become a reality given Microsoft's historically monopolistic stance.

So how important is portability to you? This is the key question businesses must ask themselves. When evaluating the importance of portability, there are three scenarios worth considering.

- If your firm is selling software to other businesses, or if you are a consulting company, and your customers are on a variety of platforms, we recommend specializing in J2EE architecture. Unless you can guarantee that every one of your customers will accept a Windows/.NET solution, you are restricting your salespeople from major accounts that may have solutions deployed on UNIX or mainframes. This is rarely acceptable at most ISVs or consulting firms.
- If, on the other hand, your customers are on the Windows platform, then either J2EE or .NET will suffice, since they both run on Windows. You should then ask your sales force and consultants what middleware your customers are using on that platform, and make your architecture decision from there. It's important to really be proactive and get this information--the more data you have, the better.
- If you host your own solutions, then you control the deployment environment. That enables you to pick J2EE as well as .NET. If you are willing to standardize on the Win32 platform, and live with the advantages and disadvantages of that platform exclusively, then platform



neutrality is irrelevant, and you should consider other factors when deciding on J2EE or .NET. But we offer a word of caution: you can never predict the future. Business goals might change, new vendors might be introduced into the picture, and mergers and acquisitions might happen. All of these may result in a heterogeneous deployment environment. Your applications will not be portable to those platforms in this scenario.

### 9.7.5 Web Services Support

The future of eBusiness collaboration is undoubtedly Web services. For organizations that are pursuing a Web services strategy, or are preparing for the future of Web services, their underlying eBusiness architecture must have strong Web services support.

Today, J2EE supports Web services through the Java API for XML Parsing (JAXP). This API allows developers to perform any Web service operation today through manually parsing XML documents. For example, you can use JAXP to perform operations with SOAP, UDDI, WSDL, and ebXML.

Additional APIs are also under development. These are convenience APIs to help developers perform Web services operations more rapidly, such as connecting to business registries, transforming XML-to-Java and Java-to-XML, parsing WSDL documents, and performing messaging such as with ebXML.

A variety of J2EE-compatible 3rd party tools are available today that enable rapid development of Web services. There are at least sixteen SOAP implementations that support Java. Almost all of these implementations are built on J2EE (servlets or JSP). There are only five UDDI API implementations available, and four of them support Java (IBM UDDI4J, Bowstreet jUDDI, The Mind Electric GLUE, and Idoox WASP). Third-party software vendors such as Tradia ([www.tradia.com](http://www.tradia.com)), CapeClear ([www.capeclear.com](http://www.capeclear.com)) and The Mind Electric ([www.theminelectric.com](http://www.theminelectric.com)) also offer tools for creating Web services.

The preview release of Microsoft.NET also enables organizations to build Web services. The tools that ship with Microsoft.NET also offer rapid application development of Web services, with automatic generation of Web service wrappers to existing systems. You can perform operations using SOAP, UDDI, and SDL (the precursor to WSDL). Visual Studio.NET provides wizards that generate Web services.

Our conclusions from our Web services comparison are as follows.

With J2EE, you can develop and deploy Web services today using JAXP. However, this is not the ideal way to build Web services, since it requires much manual intervention. An alternative is for organizations to leverage 3rd party

libraries to accelerate their development. In the future these libraries will be standardized through the JAX APIs. For now, if you develop Web services rapidly, you'll need to bundle these libraries with your application.

With .NET, you can develop Web services today using the partial release of .NET. However, since this is only a beta implementation, it does not represent a realistic deployment platform. Another issue with .NET is that it does not support true Web services because of a lack of support for ebXML. ebXML is a very important standard for eBusiness collaboration, and is experiencing broad adoption from around the world. Thousands of vendor and non-vendor companies, government institutions, academic and research institutions, trade groups, standards bodies, and other organizations have joined the ebXML community. This includes HL7 (Health Care), OTA (Open Travel Alliance), RosettaNet, OAG (Open Applications Group), GCI (Global Commerce Initiative), and DISA (Data Interchange Standards Association). Undoubtedly, ebXML is going to be an important force in Web services, and we hope that Microsoft chooses to embrace it. Microsoft is still clinging to their BizTalk proprietary framework which has proprietary SOAP extensions. This evidence makes us question Microsoft's true commitment to open and interoperable Web services.

### 9.7.6 Tools

The Sun J2EE Product Portfolio includes Forte, a modular and extensible Java-based IDE that pre-dates both Sun J2EE and .NET. Developers who prefer other IDEs for Java development are free to use WebGain's Visual Café, IBM's VisualAge for Java, Borland's JBuilder, and more. Numerous 3rd party tools and open source-code products are available.

Microsoft has always been a strong tools vendor, and that has not changed. As part of its launch of .NET, Microsoft released a beta version of the Visual Studio.NET integrated development environment. Visual Studio.NET supports all languages supported by earlier releases of Visual Studio - with the notable exception of Java. In its place, the IDE supports C#, Microsoft's new object-oriented programming language, which bears a remarkable resemblance to Java. Visual Studio.NET has some interesting productivity features including Web Forms, a Web-based version of Win Forms, .NET's GUI component set. Visual Studio.NET enables developers to take advantage of .NET's support for cross-language inheritance.

Our conclusion is that Microsoft has the clear win when it comes to tools. While the functionality of the toolset provided by J2EE community as a whole supercedes the functionality of the tools provided by Microsoft, these tools are

not 100% interoperable, because they do not originate from a single vendor. Much more low-level hacking is required to achieve business goals when working with a mixed toolkit, and no single tool is the clear choice, nor does any single tool compare with what Microsoft offers in Visual Studio.NET. Microsoft's single-vendor integration, the ease-of-use, and the super-cool wizards are awesome to have when building Web services.

### 9.7.7 Scalability

Scalability is essential when growing a Web services deployment over time, because one can never predict how new business goals might impact user traffic.

A platform is scalable if an increase in hardware resources results in a corresponding linear increase in supported user load while maintaining the same response time. By this definition, the underlying hardware (Win32, UNIX, or Mainframe) is irrelevant when it comes to scalability, because both J2EE and .NET allow one to add additional machines to increase user load while maintaining the same response time. The major implementations based on J2EE architecture, as well as .NET, provide load-balancing technology that enable a cluster of machines to collaborate and service user load that scales over time.

The significant difference between J2EE and .NET scalability is that since .NET supports Win32 only, a greater number of machines are needed than a comparable J2EE deployment due to processor limitations. This multitude of machines may be difficult for organizations to maintain.

## 9.8 Conclusions

Arguments supporting both platforms

- Regardless of which platform you pick, new developers will need to be trained (Java training for J2EE, OO training for .NET)
- You can build Web services today using both platforms
- Both platforms offer a single-vendor solution.
- The scalability of both solutions are theoretically unlimited.

Arguments for .NET and against J2EE

- .NET has Microsoft's A-team marketing it
- .NET released their Web services story before J2EE did, and thus has some mind-share
- .NET has an awesome tool story with Visual Studio.NET
- .NET has a simpler programming model, enabling rank-and-file developers to be productive without shooting themselves in the foot

- .NET gives you language neutrality when developing new eBusiness applications, whereas J2EE makes you treat other languages as separate applications
- .NET benefits from being strongly interweaved with the underlying operating system

#### Arguments for J2EE and against .NET

- J2EE is being marketed by an entire industry
- J2EE is a proven platform, with a few new Web services APIs. .NET is a rewrite and introduces risk as with any first-generation technology
- Existing J2EE code will translate into a J2EE Web services system without major rewrites. Not true for Windows DNA code ported to .NET.
- J2EE is a more advanced programming model, appropriate for well-trained developers who want to build more advanced object models and take advantage of performance features
- J2EE gives you platform neutrality, including Windows. You also get good (but not free) portability. This isolates you from heterogeneous deployment environments.
- J2EE lets you use any operating system you prefer, such as Windows, UNIX, or mainframe. Developers can use the environment they are most productive in.

## CHAPTER 10

### Development of an e-commerce application using .NET architecture

#### ***10.1 Jurasco Style Store Application Overview***

The application is a typical e-commerce application: an online store enterprise that sells products—clothes—to customers. The application has a Web site through which it presents an interface to customers. The users interact with the application through a user interface mechanism.

While the application handles most tasks automatically, some tasks must be done manually, such as managing inventory and shipping orders. You can consider the entire application as the Jurasco Style Store.

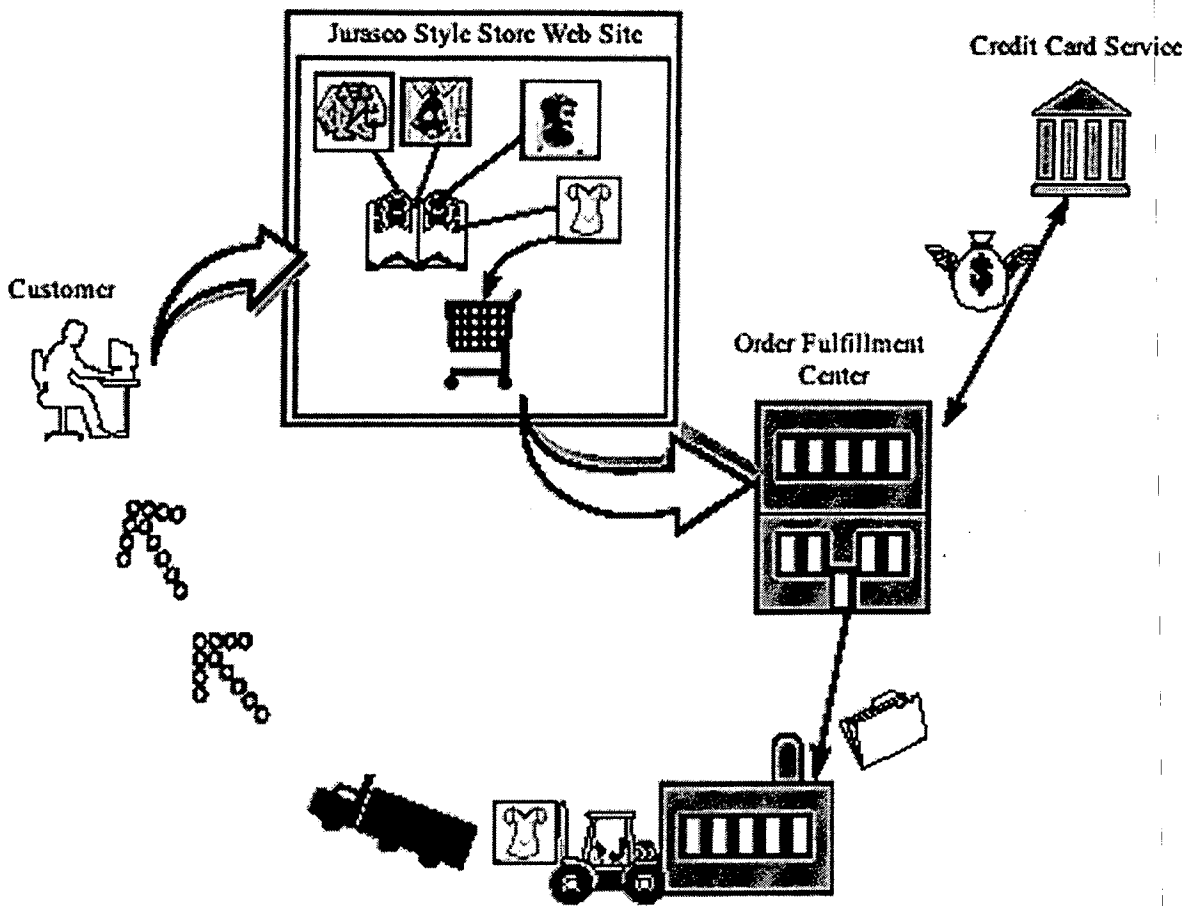


Figure 14 Jurscoo Style Store business

## **10.2 Designing the Sample Application**

Designing an application starts with assessing functional requirements and then determining an optimal software implementation to meet those requirements. There are numerous analysis tools for gathering and assessing application requirements.

Use case analysis is one such tool. Use case analysis identifies the actors in a system and the operations they may perform.

The Jurasco Style store application is a typical e-commerce site. The customer selects items from a catalog, places them in a shopping cart, and, when ready, purchases the shopping cart contents. Prior to a purchase, the application displays the order: the selected items, quantity and price for each item, and the total cost. The customer can revise or update the order. To complete the purchase, the customer provides a billing address, a shipping address, and a credit card number.

Figure 15 shows a high-level use case diagram for the sample application. It shows the potential system actors and their actions:

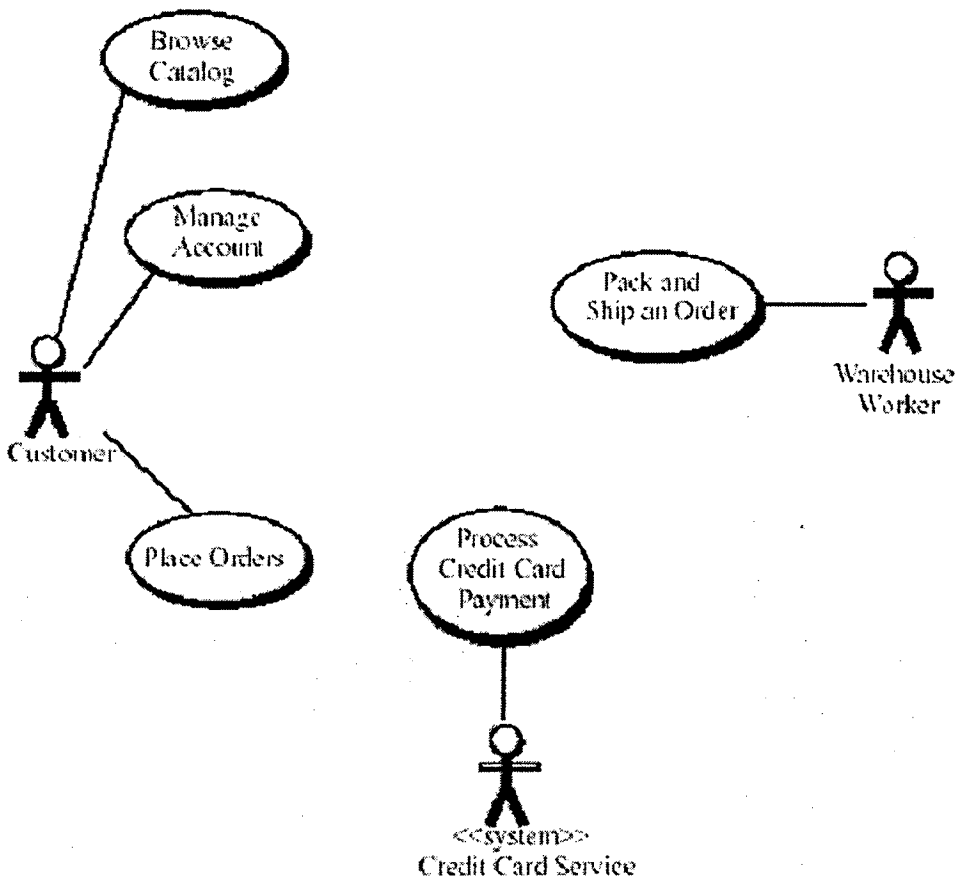


Figure 15 Application Use Case diagram

- A customer shops, places orders, manages his user account
- A bank system processes credit cards.
- A warehouse worker packs and ships orders.

Once you have determined the system’s requirements, you can begin designing the application.

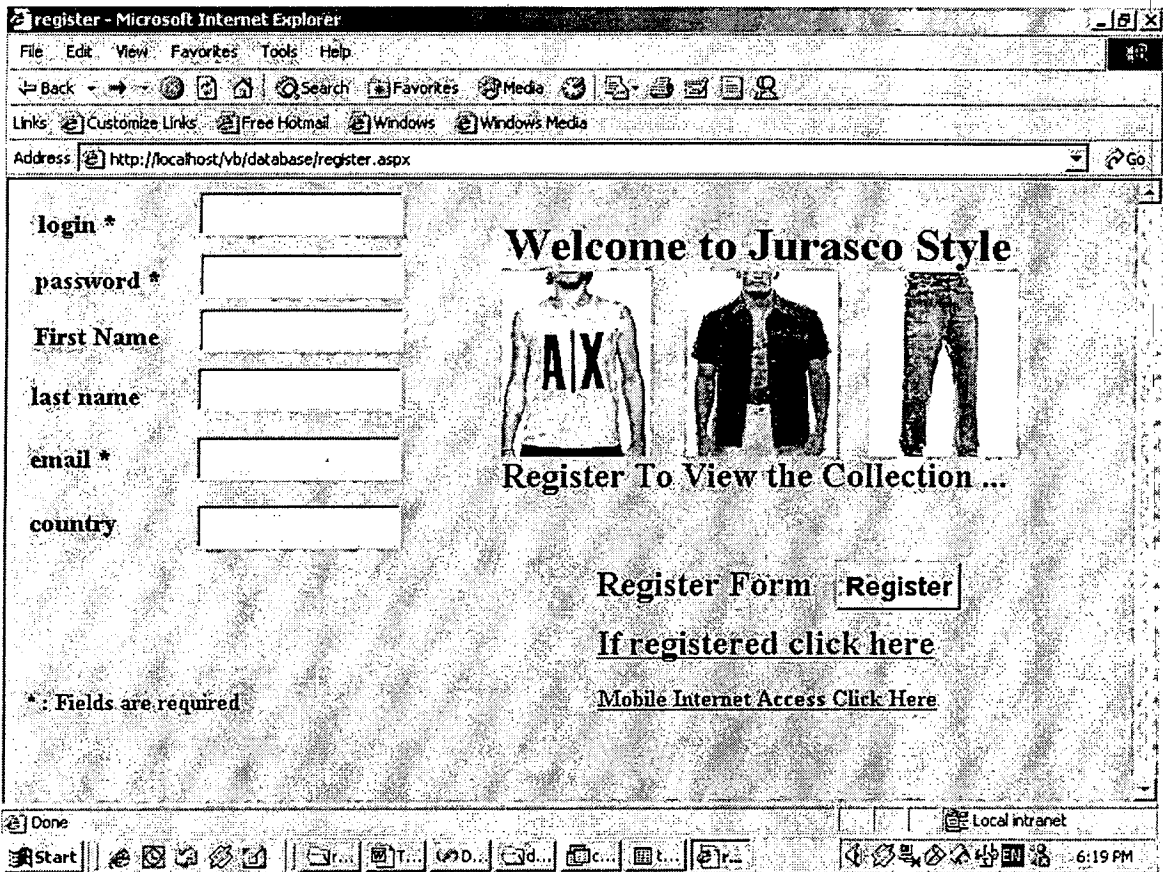
### 10.3 Functional Walkthrough

Jurasco Style Store, is an e-commerce application where customers can buy clothes online. When you start the application you can browse and search for various types of clothes from shirts to jeans.

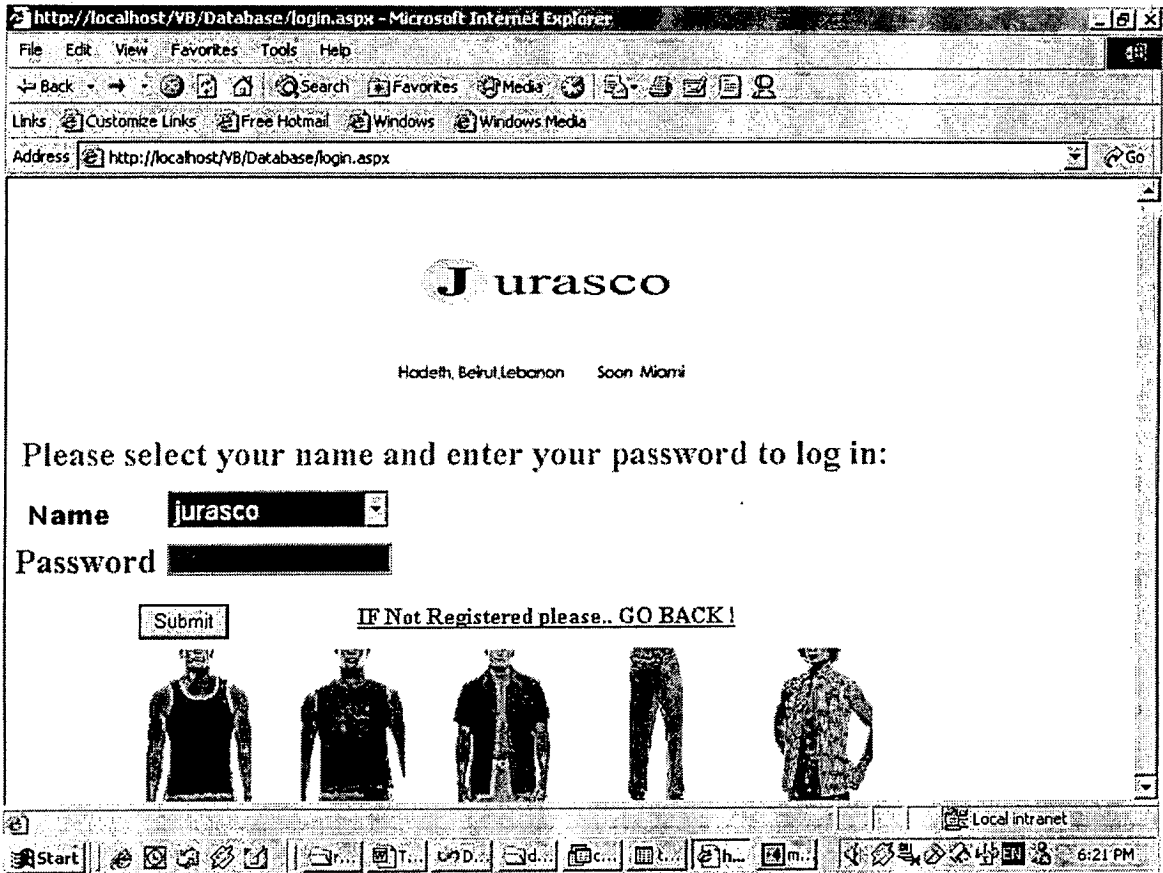
A typical session using Jurasco Style Store is as follows:



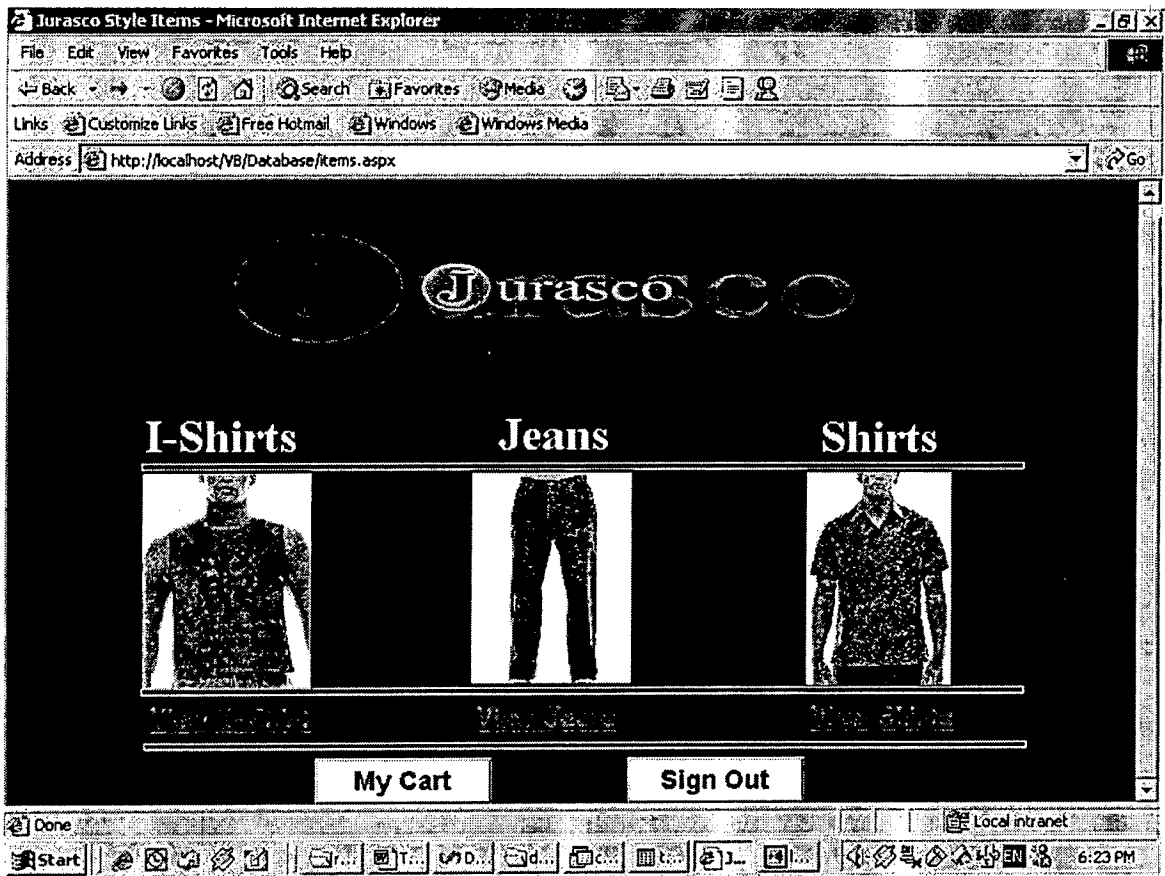
Homepage - This is the main page that loads when the user first starts the application.



Login – The user must enter a user name and password to log in



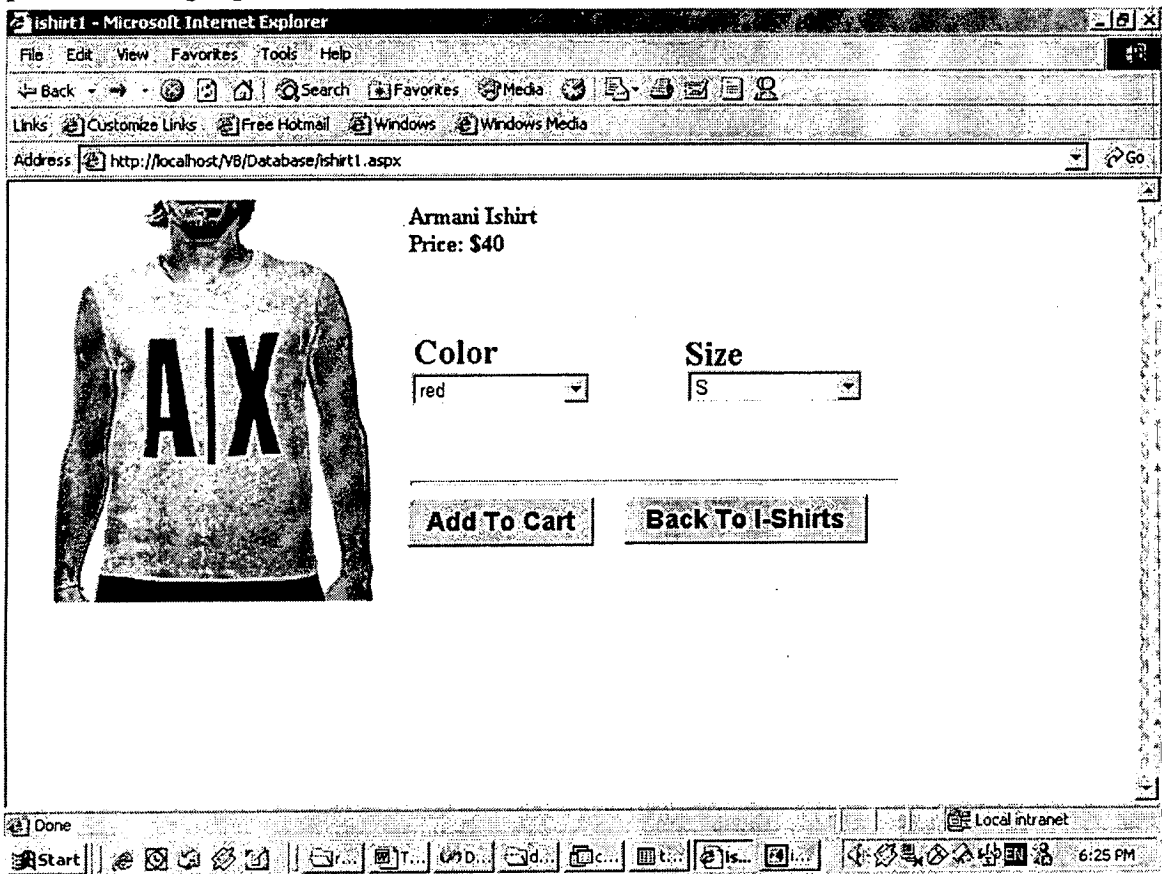
Category View – There are 3 top- level categories: Each category has several products associated to it.



**Products** – When a category is selected within the application, all the products are displayed.



Product Details – Each product will have a detailed view that displays the product image, price, and the available colors and sizes.



**Shopping Cart** – Allows the user to manipulate the shopping cart (add, remove, and update line items).

The screenshot shows a Microsoft Internet Explorer browser window displaying a shopping cart page. The address bar shows the URL: `http://localhost/NB/Database/cart.aspx`. The page content includes a large heading "Your Shopping Cart" and a sub-heading "Welcome To Jurasco.com". Below this is a table listing items in the cart:

Delete	ID	description	CustomerID	ItemID	Quantity	Price
<a href="#">Delete</a>	326	Armani Jeans	1	50020	2	140
<a href="#">Delete</a>	327	Armani Shirt	1	10010	1	45
<a href="#">Delete</a>	328	Armani Shirt	1	10020	1	45

Below the table, a black box displays the "Total Amount: \$ 230". At the bottom of the page, there are two buttons: "checkout" and "Continue Shopping". The browser's status bar at the bottom shows "Done", "Local intranet", and the time "6:27 PM".

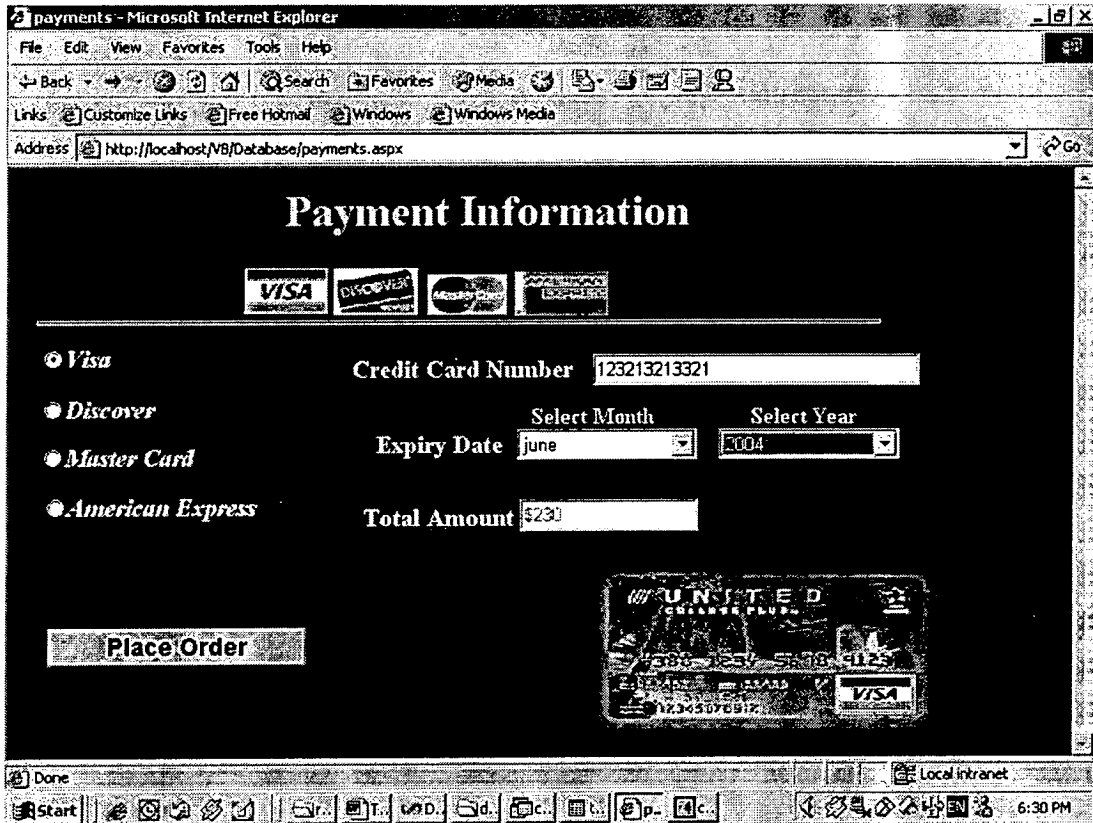
Checkout – The user is asked to give a billing and shipping address.

The screenshot shows a Microsoft Internet Explorer browser window with the following details:

- Address bar: `http://localhost/VB/Database/checkout.aspx`
- Page Title: **Billing and Delivery Address**
- Section: **Billing Address**
- Fields:
  - First Name:
  - Last Name:
  - Country:
  - City, Street, Bld.:
  - Telephone:
- Section: **Shipping Address** (with note: "IF shipping is same as billing address, leave this section empty")
- Fields:
  - First Name:
  - Last Name:
  - Country:
  - City, Street, Bld.:
  - Telephone:
- Buttons: [Payment Type](#) and [Back To Cart](#)

The browser's taskbar at the bottom shows the Start button, several application icons, and the system tray with the time 6:29 PM and 'Local intranet' indicator.

Payment Info – The user enters the credit card information.



Place Order – This is the final step in the order-processing pipeline. The order is now committed to the database at this point.



## 10.4 logical architecture

The overall logical architecture of the .NET Jurasco Style Shop is detailed in Figure 16.

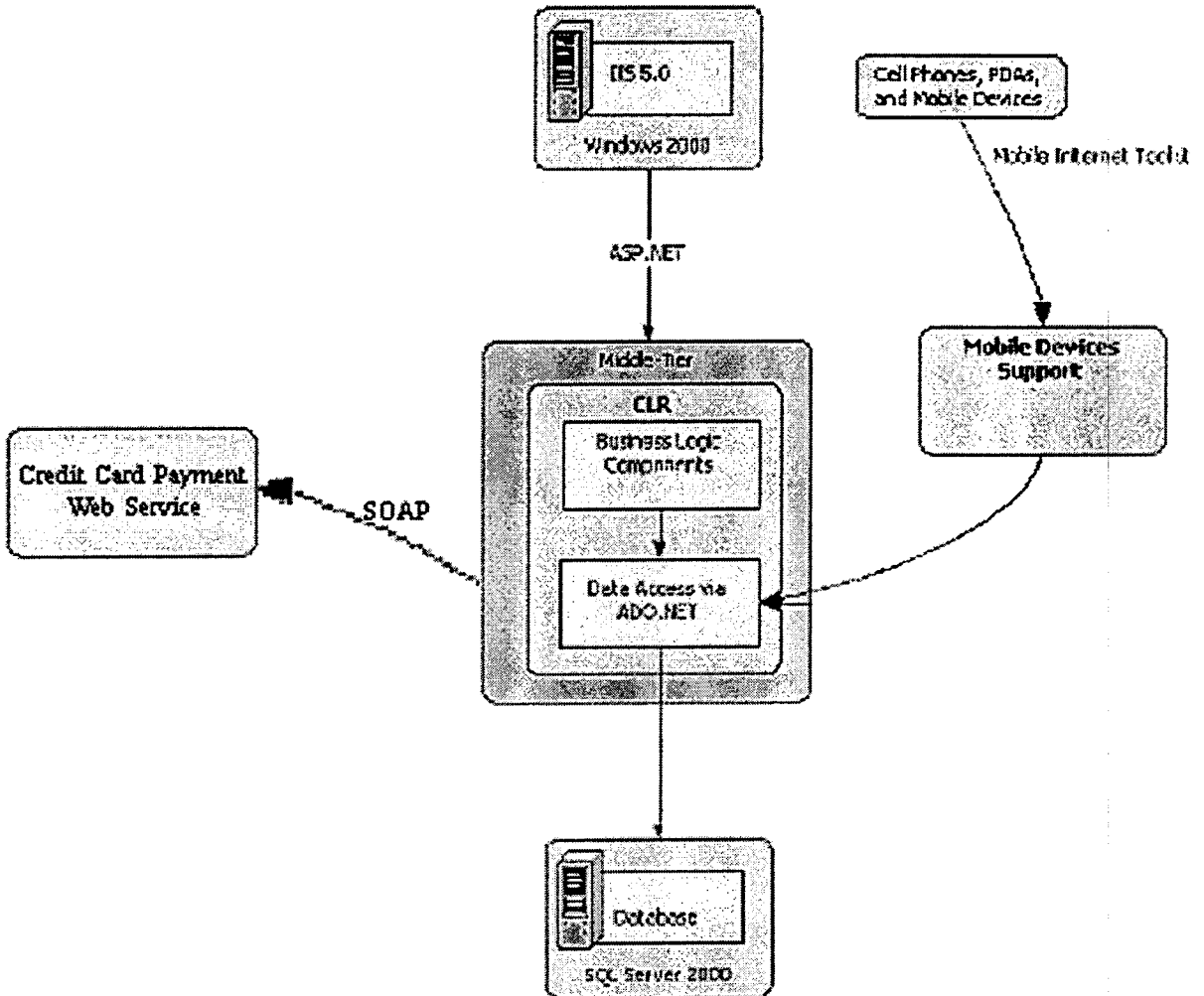


Figure 16 .NET Jurasco Style Shop logical architecture

There are three logical tiers: the presentation tier, the middle tier, and the data tier. The three tiers allow for clean separation of the different aspects of a distributed application. The application represents a complete logical three-tier implementation using .NET, and illustrates coding best-practices for the Microsoft.NET platform.

## 10.5 Architecture of the Application

The previous section addressed the high-level architecture of the application. To gain a better understanding of how the application works, this section will walkthrough a section of the code, demonstrating the interaction between the presentation tier, the middle-tier, and the data tier.

In designing n-tier applications, there are a variety of approaches. In implementing the .NET Jurasco Style Shop, a data-driven approach was used.

### 10.5.1 Database

The database for the Jurasco Style Store is SQL Server 2000. The database has the following overall structure of tables:

Table Name	Purpose
item	Product information.
articles	Article information.
cart	Shopping Cart information.
transactions	The orders placed by the customer. An order contains 1 or more line items.
users	User details.
Line_items	Order details.

Table 4 Database Table Names

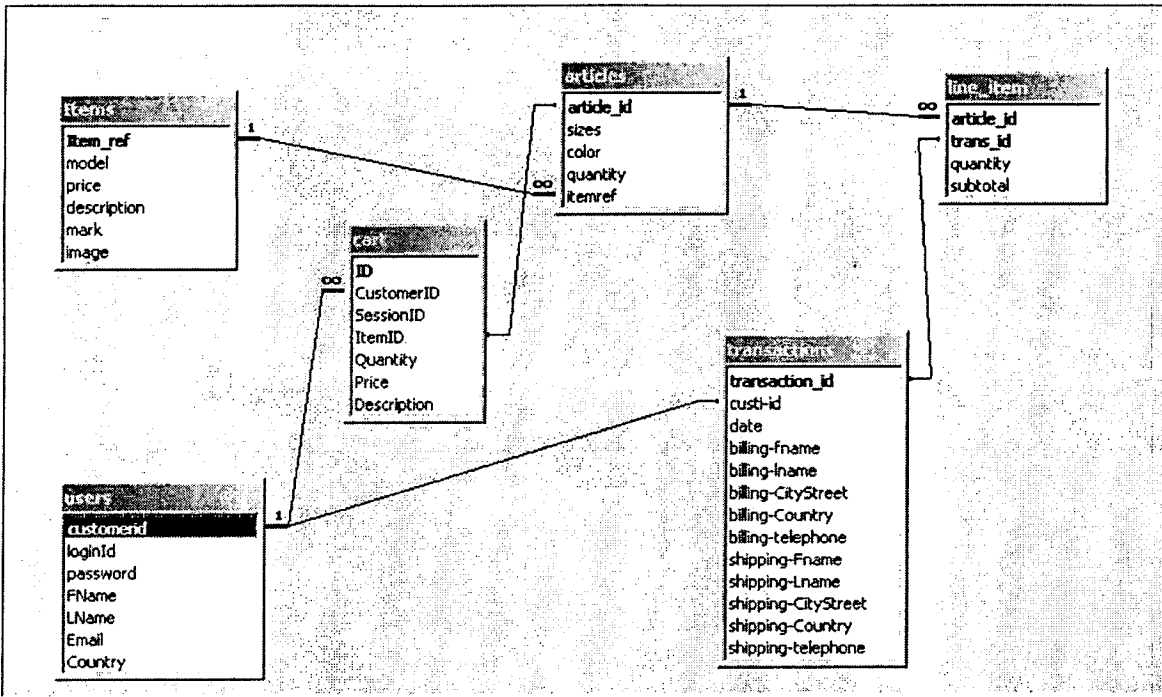


Figure 17 Jurasco Style Shop physical database schema

The complete physical database schema for the .NET Jurasco Style Shop is illustrated in Figure 17.

### 10.5.2 Middle-Tier

The .NET JurascoStyle Shop follows a Web centric architecture. The middle-tier business logic is encapsulated into multiple ASPX pages implemented using VB. The ASPX pages are located in the presentation tier developed in the next section. The only component that we find in the middle tier is the primary key class ItemCode used to calculate the bar code of an item based on its color and size entered by the user from the ASPX page displaying detailed information about a selected product in a category. The code of this class is shown below.

```

Public Class ItemCode
    Private iSize As Integer
    Private iColor As Integer
    Private iCode As Integer
    Public Function GetColorCode(ByVal sColor As String, ByVal sSize As
String, ByVal iClothNumber As Integer) As Integer
        Dim RetCode As Integer

        iCode = iClothNumber * 100
        Select Case LCase(sColor)
            Case "white"
                iColor = 10
            Case "black"
                iColor = 20
            Case "blue"

```

```
        iColor = 30
    Case "green"
        iColor = 40
    Case "red"
        iColor = 50
    Case "grey"
        iColor = 60
    Case Else
        iColor = 0
End select

Select Case LCase(sSize)
    Case "s"
        iSize = 0
    Case "m"
        iSize = 1
    Case "l"
        iSize = 2
    Case "xl"
        iSize = 3
    Case Else
        iSize = 9
End select
RetCode = iCode + iSize + iColor
Return RetCode
End Function
End Class
```

### 10.5.3 Presentation-Tier

The presentation-tier for Juraco Style Shop was written using ASP.NET Web Forms combined with User Controls. The site was created with Visual Studio .NET [20] and therefore uses code-behind where the code for each ASPX page is encapsulated into a separate file.

The configuration is detailed in Table 5.

ASPX page	VB Code-behind	Description
register.aspx	register.aspx.vb	Enables users visiting Jurasco Style Store to register as customers.
login.aspx	login.aspx.vb	Enables a customer to log in.
item.aspx	item.aspx.vb	Displays a list of product categories.
ishirtpage.aspx	ishirtpage.aspx.vb	Displays a list of products in the selected iShirt category.
shirtpage.aspx	shirtpage.aspx.vb	Displays a list of products in the selected shirt category.
jeanspage.aspx	jeanspage.aspx.vb	Displays a list of products in the selected jeans category.
ishirt1.aspx	ishirt1.aspx.vb	Displays detailed information about the particular product iShirt1.
cart.aspx	cart.aspx.vb	Enables customers to view the current state of their shopping cart. They can also remove items.
checkout.aspx	checkout.aspx.vb	Used to get the billing address and the shipping address of the customer.
payments.aspx	payments.aspx.vb	Used to get the credit card information from the customer and validate the purchase.

Table 5 ASPX pages of the Web centric architecture

## ***10.6 Building an XML Web Service***

In this chapter we examine an actual Web service design implemented on an internal business application. Our application is not deployed as a public Web service since an electronic shop is not usually involved in a business-to-business (B2B) communication unless it exposes a particular service to a retail customer business like checking the number of available items in stock. Therefore, and in order to look at some features available with Visual Studio.NET that make it easy to build, deploy, and access Web services, we emulated the Web service of a credit card system used to verify the validity of the credit card information entered by the customer to accomplish a purchase operation in the ASPX page `payment.aspx`.

### **10.6.1 Web Services in .NET**

The .NET framework takes care of all the plumbing, so we don't need to know anything about SOAP or WSDL to build and deploy the Web service.

In a B2B communication, when a Web application needs to perform business operation existing in another Web application, it makes a call into the business tier through the use of a Web service listener. The Web service listener then instantiates a Web service processor that is part of the business tier. The Web service listener is a `.asmx` file deployed to the Web site and the processor is a code behind file written in VB (`.asmx.vb`) or C# (`.asmx.cs`).

### **10.6.2 Testing the Web Service**

Our Web service is called `CCValidator`. The Web service listener is: `ccvalidator.asmx`, and the Web service processor is: `ccvalidator.asmx.vb`. The .NET framework automatically generates a test harness for the Web service. Figure 18 shows a browser with the first page of the test harness created. This page provides two types of links: one to get the WSDL file, or Service Description, and another to test the interface for each method in the Web service.

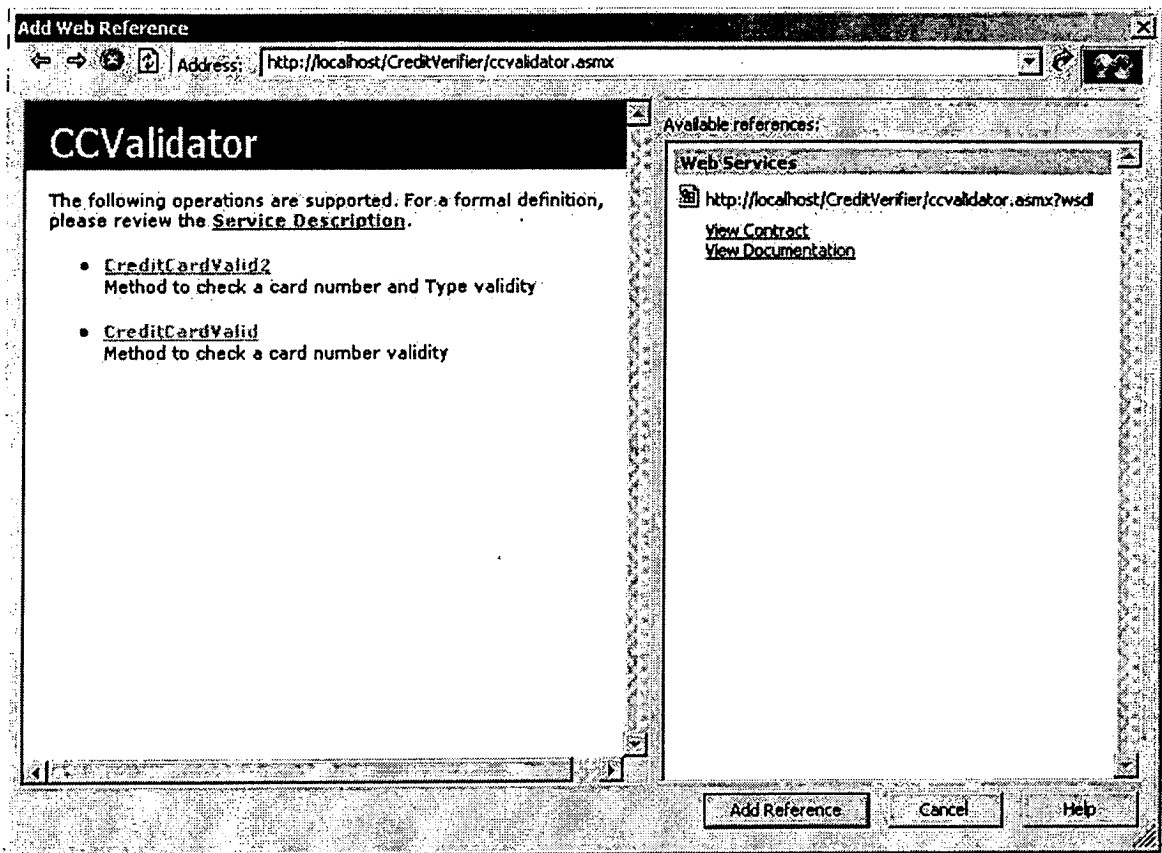


Figure 18 Test harness for CCValidator Web service

The following screen (Figure 19) shows the page displayed when the CreditCardValid link is accessed. From looking at this illustration, you can see in the top of the page the framework created an input box based on what it saw in the schema. When a value is entered, and the Invoke button is pressed, the Web service is called and the result is then displayed in a browser. The bottom of the page contains a section that describes the SOAP message that will be sent.

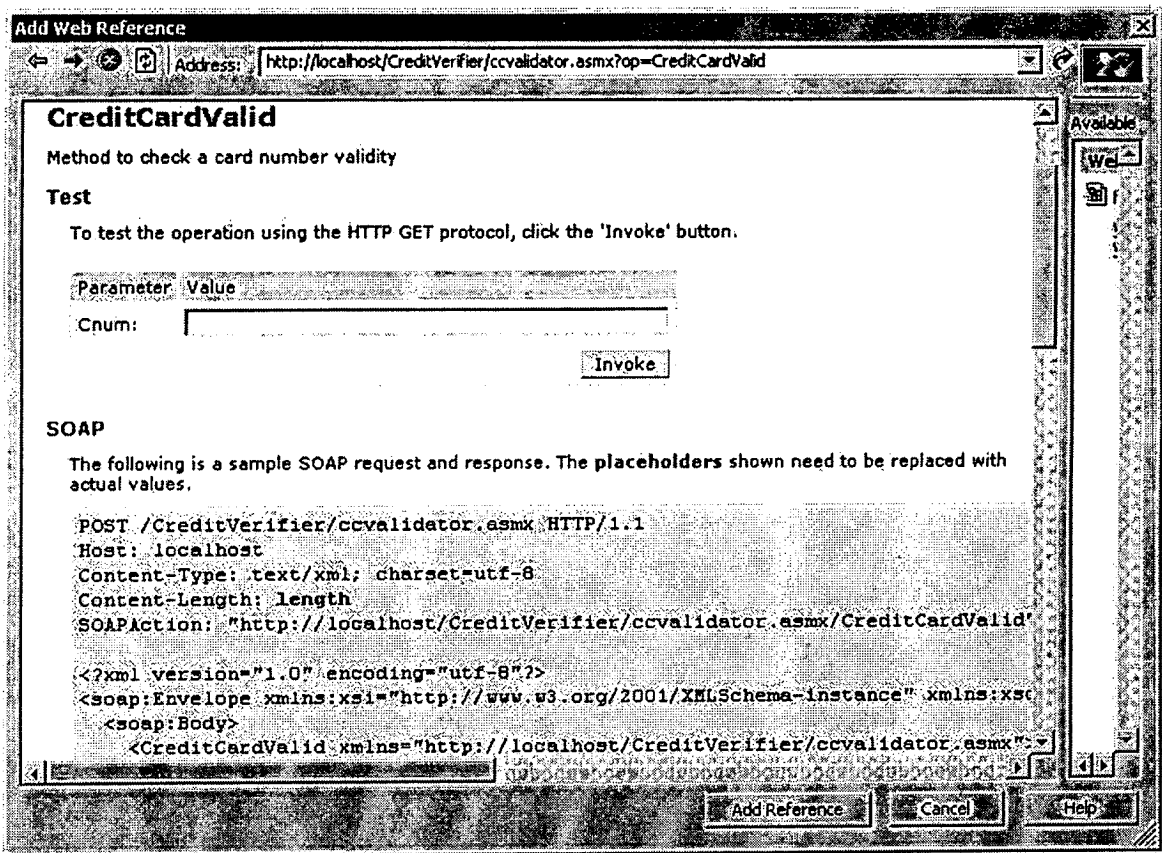


Figure 19 Web service interface



## 10.7 Mobile Device Support

ASP.NET provides some excellent mechanisms to target a variety of Web browsers ranging from previous versions of Internet Explorer to Netscape. One area of growth for Web applications has been extending to browsing with a variety of devices ranging from PocketPCs (Windows CE) running Pocket Internet Explorer to cell phones using WAP browsers. The Microsoft Mobile Internet Toolkit provides a set of assemblies that allow developers to write one code base that can support a multitude of different devices. During the development of the application using .NET, we decided that it would be great if customers could visit our store using a cell phone. The customers should be registered to Jurasco Style store. Figure 20 shows the page presenting the mobile service to the customers of Jurasco Style Store Web site.

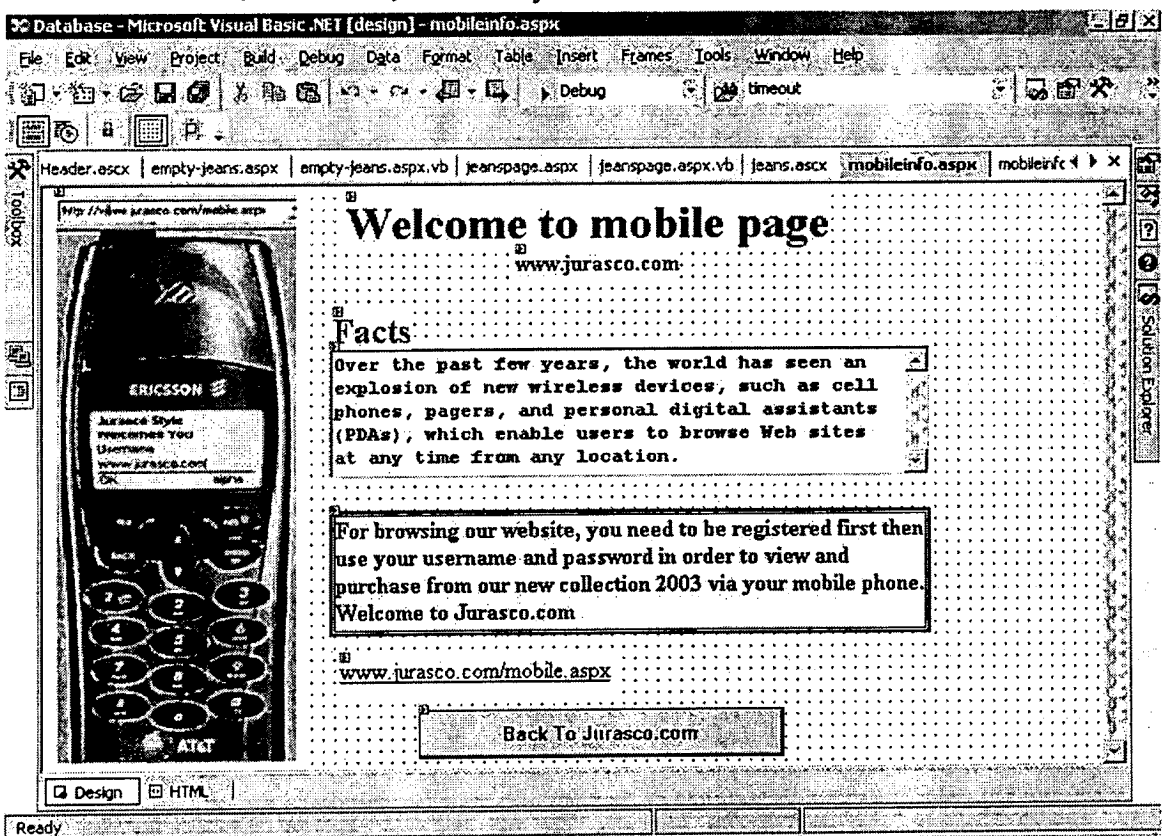


Figure 20 Mobile information Web page

Using the Mobile Internet Toolkit was very straightforward. The toolkit also integrates nicely into Visual Studio .NET making it possible to create the page using the visual design surface as shown in Figures 21, .22, and 23.

Figure 21 shows the log in mobile form. The customer should have already registered in our application otherwise he will not be able to purchase items via a mobile. Thus, a valid user name and password will redirect him to the items view mobile page (figure 23) in order to view and purchase desirable items.

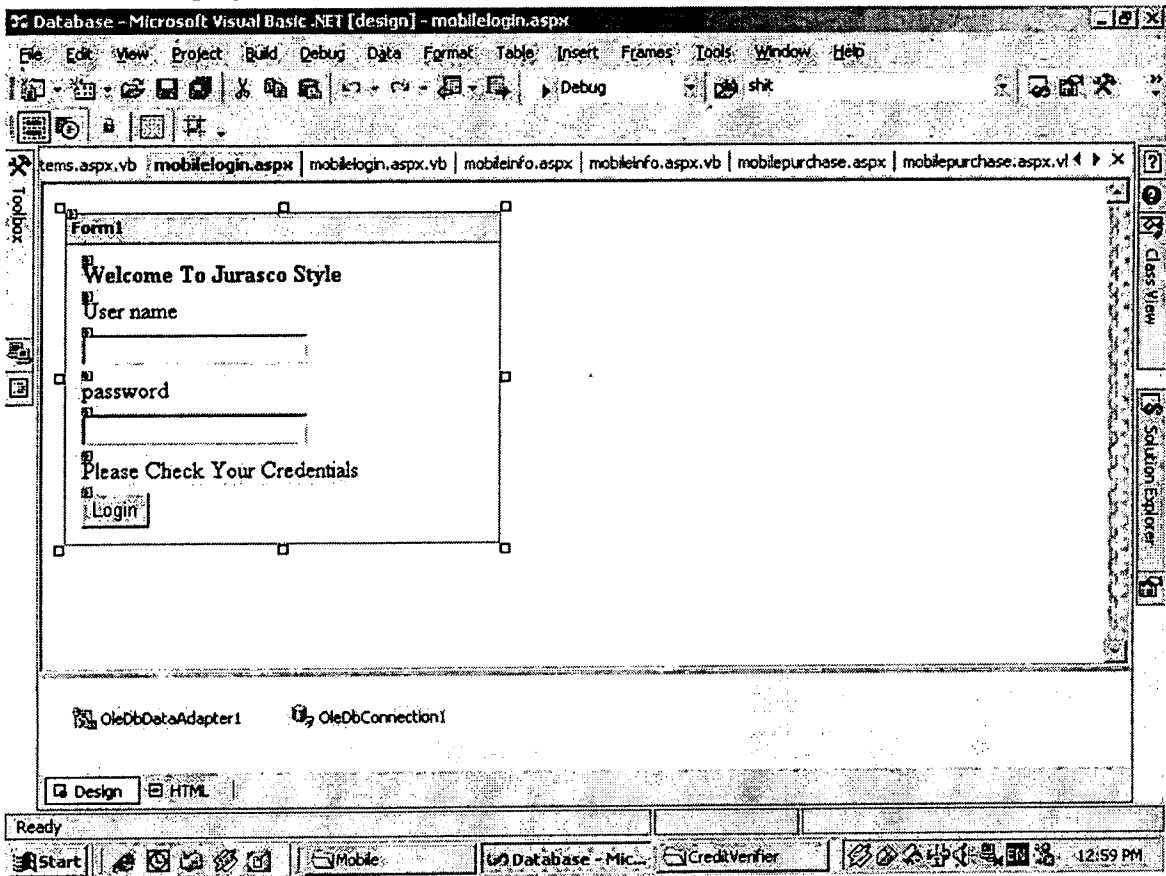


Figure 21 log in mobile forms

The code for the login mobile form mobile form can be seen below:

```

<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="mobilelogin.aspx.vb" Inherits="Database.mobile" %>
<%@ Register TagPrefix="mobile"
Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
%>
<meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
<meta name="vs_targetSchema"
content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
  <mobile:Form id="Form1" runat="server">
    <mobile:Label id="Label1" runat="server" ForeColor="Black"
Font-Bold="True">welcome To Jurasco Style </mobile:Label>
    <mobile:Label id="Label3" runat="server">User
name</mobile:Label>
    <mobile:TextBox id="username"
runat="server"></mobile:TextBox>
    <mobile:Label id="Label4"
runat="server">password</mobile:Label>
  </mobile:Form>
</body>

```

```

<mobile:TextBox id="password"
runat="server"></mobile:TextBox>
    <mobile:Label id="lblerror" runat="server"
ForeColor="#C00000" Visible="False">Please Check Your
Credentials</mobile:Label>
    <mobile:Command id="Command1"
runat="server">Login</mobile:Command>
</mobile:Form>
</body>

```

The button "Login" will verify the customer's credentials and then display the list of available items (Figure 22). The customer will select a particular item, choose the size and color.

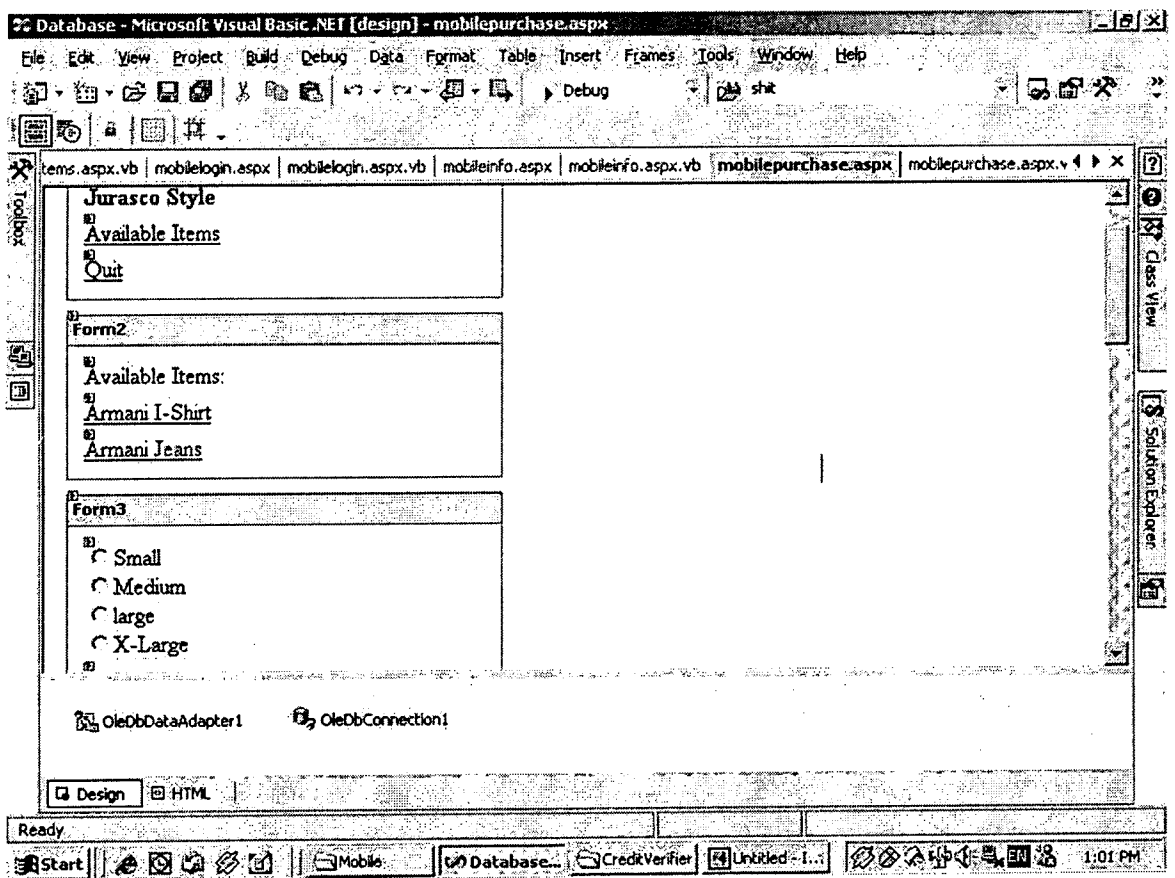


Figure 22 Purchase mobile form

The code of the forms used for purchase is shown below:

```

<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="mobilepurchase.aspx.vb" Inherits="Database.mobilepurchase"
%>
<%@ Register TagPrefix="mobile"
Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
%>
<meta content="Microsoft Visual Studio.NET 7.0" name="GENERATOR">
<meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
<meta content="http://schemas.microsoft.com/Mobile/Page"
name="vs_targetSchema">
<body xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

```

```

<mobile:form id="Form1" runat="server">
  <mobile:Label id="Label1" runat="server" Font-
Bold="True">Jurasco Style</mobile:Label>
  <mobile:Link id="Link1" runat="server"
NavigateUrl="#Form2">Available Items</mobile:Link>
  <mobile:Link id="Link2" runat="server">Quit</mobile:Link>
</mobile:form>
<mobile:form id="Form2" runat="server">
  <mobile:Label id="Label3" runat="server">Available
Items:</mobile:Label>
  <mobile:Link id="Link3" runat="server"
NavigateUrl="#Form3">Armani I-Shirt $40</mobile:Link>
  <mobile:Link id="Link4" runat="server"
NavigateUrl="#Form3">Armani Jeans $70</mobile:Link>
</mobile:form>
<mobile:form id="Form3" runat="server">
  <mobile:SelectionList id="slSize" runat="server"
SelectType="Radio">
    <Item Value="s" Text="Small"></Item>
    <Item Value="m" Text="Medium"></Item>
    <Item Value="l" Text="Large"></Item>
    <Item Value="xl" Text="X-Large"></Item>
  </mobile:SelectionList>
  <mobile:Label id="Label4" runat="server"
DESIGNTIMEDRAGDROP="111">Choose Color</mobile:Label>
  <mobile:SelectionList id="slColor" runat="server"
SelectType="Radio">
    <Item Value="Black" Text="Black"></Item>
    <Item Value="white" Text="white"></Item>
  </mobile:SelectionList>
  <mobile:Label id="notavailable" runat="server" Font-
Bold="True" Visible="False">Out of stock</mobile:Label>
  <mobile:Command id="Command2" runat="server">Place
Order</mobile:Command>
</mobile:form>
</body>

```

Then, the customer will provide a billing address, shipping address, and his credit card information. (figure 23)

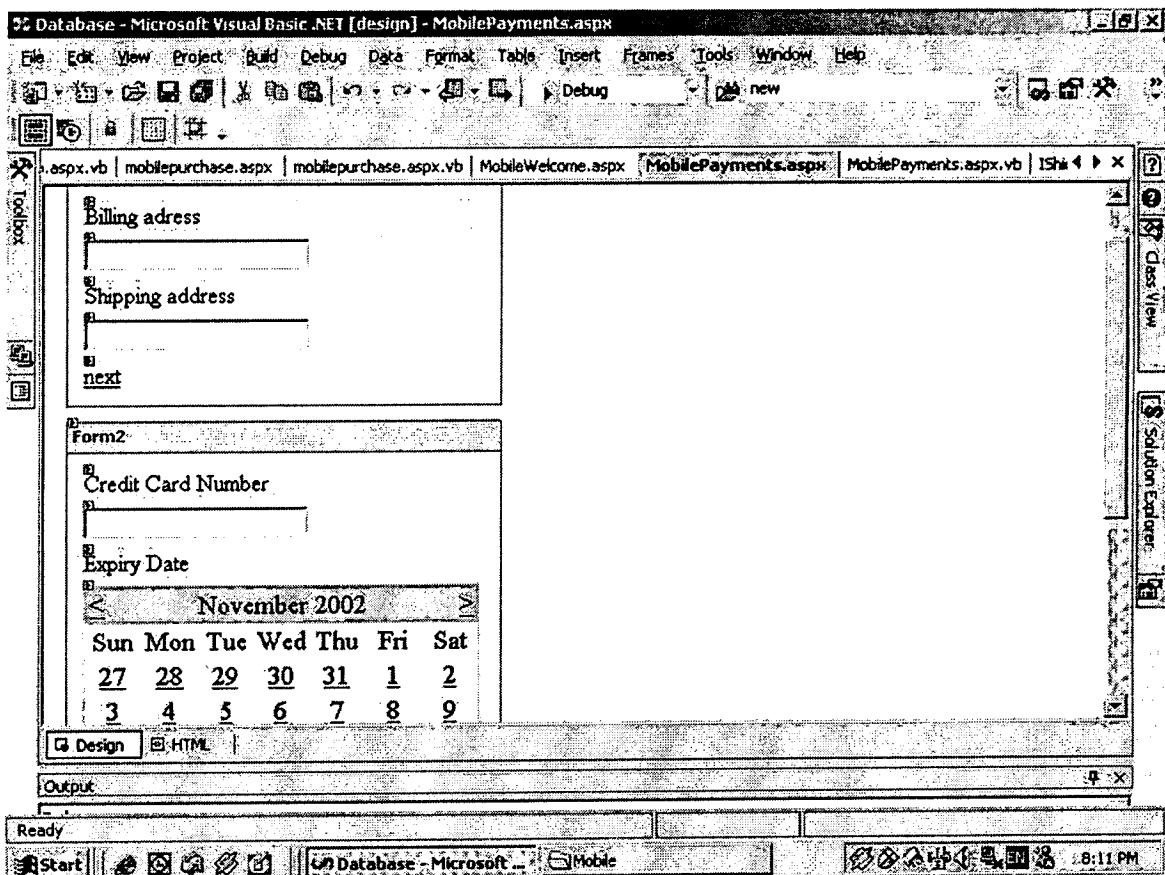


Figure 23 Payment mobile form

The code for mobile payment forms is shown below:

```

<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="MobilePayments.aspx.vb" Inherits="Database.MobilePayments"
%>
<%@ Register TagPrefix="mobile"
Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
%>
<meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
<meta name="vs_targetSchema"
content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
  <mobile:Form id="Form1" runat="server">
    <mobile:Label id="Label1" runat="server">Billing
address</mobile:Label>
    <mobile:TextBox id="TextBox1"
runat="server"></mobile:TextBox>
    <mobile:Label id="Label2" runat="server">Shipping
address</mobile:Label>
    <mobile:TextBox id="TextBox2"
runat="server"></mobile:TextBox>
    <mobile:Link id="Link1" runat="server"
NavigateUrl="#Form2">next</mobile:Link>
  </mobile:Form>
  <mobile:Form id="Form2" runat="server">

```

```

        <mobile:Label id="Label3" runat="server">Credit Card
Number</mobile:Label>
        <mobile:TextBox id="creditbox" runat="server"
MaxLength="12"></mobile:TextBox>
        <mobile:Label id="Label4" runat="server">Expiry
Date</mobile:Label>
        <mobile:Calendar id="calendar1"
runat="server"></mobile:Calendar>
        <mobile:Label id="errorcard" runat="server"
Visible="False">Not Valid Credit Card</mobile:Label>
        <mobile:Label id="error2card" runat="server"
Visible="False">Try Again</mobile:Label>
        <mobile:Command id="Command1"
runat="server">Purchase</mobile:Command>
    </mobile:Form>
</body>

```

The benefit of using the Mobile Internet Kit is realized whenever the developer will need to support a wide range of devices and browsers with a variety of capabilities. To test cell phone support for the mobile page, a great emulator can be found on the developer section Openwave.com. We tested this mobile page with the Openwave SDK version 4.1 (figure 24) which can be downloaded at <http://developer.openwave.com/download/index.html>.



Figure 24 Testing the mobile support using Openwave SDK

# CHAPTER 11

## Conclusion

This thesis has presented an overview of distributed application design and development. Its goal has been to introduce, from an enterprise developer view, the concepts and technologies used in designing applications for distributed platforms, and to give a practical example with a typical enterprise application. Service-oriented architecture has been presented as the natural evolution of distributed architectures. The thesis has shown that this new architecture is not simply a fashion, or the latest marketing discovery providing professionals in the sector with new ammunition. Instead, it has its place in the logical continuity of multiple attempts at processing and data distribution, applications integration, homogenization of the information system, etc. It is aimed at increasing independence between:

- The miscellaneous components of the system (considered as services)
- Each component and the technical architecture implemented.

A relevant comparison has been done between the two choices that businesses have for building XML-based web services: the Java 2 Platform, Enterprise Edition (J2EE), built by Sun Microsystems and other industry players, and Microsoft.NET, built by Microsoft Corporation.

In summary as can be seen from a technological perspective, J2EE and .NET each provide unique solutions to a maturing distributed environment. Posing the question of what new technologies might emerge in the future, only improvements on past experience and adoption of bleeding edge technologies will tell.



## Bibliography

- [1] Armstrong, Eric; Bodo, Stephanie; Carson, Debbie; Fisher, Maydene; Green, Dale; Haase, Kim (2002). *The Java Web Services Tutorial*. Palo Alto, CA: Sun Microsystems.  
<http://java.sun.com/webservices/downloads/webservicestutorial.html>
- [2] Blum, Adam (1996). *Building Business Web Sites*, New York, NY: MIS: Press.
- [3] Box, Don (1998). *Essential COM*. Reading, MA: Addison-Wesley Longmann, Inc.
- [4] Box, Don (2000a) House of COM. *MSJ* 15(1): 87-92.
- [5] Box, Don (2000b) A Young Person's Guide to the Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages. *MSDN Magazine* 15(3): 67-81.
- [6] Chappell, David (1996). *Understanding ActiveX and OLE*. Redmond, WA: Microsoft Press.
- [7] Crouch, Matt J. (2000). *Web Programming with ASP and COM*, Reading, Massachusetts: Addison-Wesley.
- [8] Deadman, Richard (1999). XML as a Distributed Application Protocol. *Java Report* 4(10): 16-21.
- [9] Eckel, Bruce (2000) *Thinking in Java: Second Edition*, Upper Saddle River, NJ: Prentice Hall PTR.
- [10] Ehnebuske, David; Rogers, Dan; Riegen, Claus Von (Ed.). (2001). *UDDI Version 2.0 Data Structure Reference*. [uddi.org](http://www.uddi.org)  
<http://www.uddi.org/pubs/DataStructure-V.200-Open-20010608.pdf>
- [11] Ewald, Tim (2001). COM+ Integration: How .NET Enterprise Services Can Help You Build Distributed Applications. *MSDN Magazine* 16(10): 42-50.
- [12] Fallside, David C. (Ed.). (2000). *XML Schema Part 0: Primer*.  
<http://www.w3.org/TR/2000/WD-xmlschema-0-20000407>
- [13] Gudgin, Martin; Hadley, Marc; Moreau, Jean-Jaques; Nielsen, Henrik Frystyk (Ed.). (2001a). *SOAP Version 1.2 Part 1: Messaging Framework*.  
<http://www.w3.org/TR/2001/WD-soap12-part1-20011217/>

- [14] Gudgin, Martin; Hadley, Marc; Moreau, Jean-Jaques; Nielsen, Henrik Frystyk (Ed.). (2001b). SOAP Version 1.2 Part 2: Adjuncts. <http://www.w3.org/TR/2001/WD-soap12-part2-20011217/>
- [15] Homer, Alex (1999). XML IE5 Programmer's Reference, Birmingham, UK: Wrox Press.
- [16] Juric, Matjaz B.; Rozman, Ivan (2000). Java 2 RMI and IDL Comparison. Java Report 5(2): 36-48.
- [17] Juric, Matjaz B.; Rozman, Ivan (2001). RMI, RMI-IIOP, and IDL Performance Comparison. Java Report 6(4): 26-34.
- [18] Kirtland, Mary (2000). The Programmable Web: Web Services Provide Building Blocks for the Microsoft .NET Framework. MSDN Magazine 15(9): 73-82.
- [19] Linthicum, David S. (1999). XML: It's EAI For the Rest of Us. Enterprise Development 1(13): 12-16.
- [20] Microsoft (2001). Delivering .NET: Visual Studio .NET and the .NET Framework. Microsoft ad 1-9.
- [21] Mikula, Norbert; Levy, Ken (2000). Schemas Take DTDs to the Next Level. XML Magazine 1(1): 81-82. 163
- [22] Mitra, Nilo (Ed.). (2001). SOAP Version 1.2 Part 0: Primer. <http://www.w3c.org/TR/2001/WD-soap12-part0-20011217>
- [23] Monson-Haefel, Richard (2001). Enterprise JavaBeans, Third Edition. Sebastopol, CA: O'Reilly & Associates Inc.
- [24] Musayev, Eldar A. (2001). SAX2: A Simple API for XML. Dr. Dobbs Journal #321: 130-133.
- [25] Naughton, Patrick (1996). The Java Handbook, Berkeley, California, Osborne McGrawHill.
- [26] Olson, Mike (1999). Introduction to CORBA, Part 1: CORBA basics to get you started. [http://www.linuxworld.com/linuxworld/lw-1999-09/lw-09-corba\\_1\\_p.html](http://www.linuxworld.com/linuxworld/lw-1999-09/lw-09-corba_1_p.html)
- [27] OMG (2000). The Common Object Request Broker Architecture Specification v2.4. <http://cgi.omg.org/cgi-bin/doc?formal/00-10-01.pdf>
- [28] OMG (2001). About the Object Management Group. <http://www.omg.org/gettingstarted/gettingstartedindex.htm>
- [29] Raj, Gopalan Suresh (1998). A Detailed Comparison of CORBA, DCOM and Java/RMI. <http://gsraj.tripod.com/misc/compare.html>

- [30] Rosen, Michael; Curtis, David (1998). Integrating CORBA and COM Applications. New York, NY: John Wiley & Sons Inc.
- [31] Scribner, Kennard; Stiver, Mark C. (2000). Understanding SOAP, Indianapolis, IN: Sams Publishing. 165
- [32] Seshadri, Govind (1999). Remote Object Activation . Java Report 4(19): 60-68.
- [33] Shohoud, Yasser (2001). Getting the Web Services You Need. XML Magazine. .  
<http://www.fawcette.com/Archives/premier/mgznarch/xml/2001/06jun01/ys103/ys0103.asp>
- [34] Skonnard, Aaron (2000). SOAP: The Simple Access Protocol. Microsoft Internet Developer 5(1): 24-33.
- [35] Tapang, Carlos C. (2001). Web Services Description Language (WSDL) Explained. Microsoft.  
<http://msdn.microsoft.com/library/en-us/dnwebsrv/html/wsdlexplained.asp>
- [36] Thai, Thuan L. (1999). Learning DCOM, Sebastpol, CA: O'Reilly & Associates Inc.
- [37] Tsai, Wei-Tek (1999). Verification and Validation of Knowledge-Based Systems. IEEE Transactions on Knowledge and Data Engineering 11(1) 202-211. uddi.org (2000). UDDI Technical White Paper.  
<http://www.uddi.org/pubs/Iru UDDI Technical White Paper.pdf>
- [38] Vinoski, Steve (1993). Distributed Object Computing with CORBA.  
<http://www.cs.wustl/~schmidt/PDF/docwc.pdf>