NOTE TO USERS

A Comparative Study of DCOM/CORBA and .NET/J2EE

Zhuofei Zhang

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

July 2004

Your file Votre référence
ISBN: 0-612-94762-9
Our file Notre référence
ISBN: 0-612-94762-9

ABSTRACT

The COM and CORBA technologies are viewed as competing architecture for creating distributed solutions. While the most significant difference between COM and CORBA is their support for different operating system platforms, people must realize that each technology has its own strength that clearly differentiate it from the other. CORBA is considered the industry's leading standard for distributed objects and it's the dominant remoting architecture. COM is primarily a component architecture rather than a remoting architecture.

The decade-old rivalry between Microsoft and Java development communities enters a new phase. Two major application platforms now dominate the enterprise application development market: Java 2 Platform, Enterprise Edition (J2EE) and the Microsoft .NET platform. The decision about which platform to use is a business decision, but technology factors can have significant business impacts.

The paper reviews, examines these candidate architectures, and analyzes the similarities and differences between DCOM and CORBA, EJB and MTS, from a programmer's standpoint and an architectural standpoint. The paper provides an in-depth comparison of the .NET and J2EE platforms, analyzes their key advantages and disadvantages to support decision making for building enterprise solutions.

Acknowledgements

I would like to thank Dr. Peter Grogono for his direction, assistance, and guidance. I would like also to thank Wei Yang, my wife, for her encouragement and assistance.

Table of Contents

Glossary

.NET platform - Microsoft's platform for a new computing model built around XML Web Services.

ActiveX - A set of technologies that enables software components to interact with one another in a networked environment, regardless of the language in which the components were created. ActiveX is used primarily to develop interactive content for the World Wide Web, although it can be used in desktop applications and other programs.

ADO - ActiveX Data Objects. These are COM objects that allow database access.

ADO.NET – It's a successor to ADO.

Apartment - Apartment is one of the important concepts of COM threading. An apartment is a conceptual unit that contains one or more threads running in the same process. There are two type of apartment: single-thread apartments (STAs) and multi-threaded apartments (MTAs). A process never has more than one MTA, but can have many STAs.

ASP - Active Server Pages, a specification for a web page that is dynamically created by the web server and contains both HTML and scripting code. With ASP, programs can be run on a web server in a similar way to CGI scripts, but ASP uses the ActiveX scripting engine to support either VBScript or Jscript.

ASP.NET - The next generation of Microsoft's Active Server Page. It's a technology that allows for the dynamic creation of documents on a Web server when they are requested via HTTP.

Automation - The binding of a client to a server object at run-time. Automation allows a client to bind to a server object without having a type library available at compile time. Automation uses the IDispatch interface.

CLR - Common Language Runtime, a runtime environment that manages the execution of .NET program code, and provides services such as memory and exception management, debugging and profiling, and security. The CLR is a major component of the .NET Framework, and provides much of its functionality by following the rules defined in the Common Type System.

COM - Component Object Model developed by Microsoft. It's Microsoft's 'de-facto' standard. It's primarily a component architecture rather than a remoting architecture. COM+ - COM+ is an extension to Microsoft's Component Object Model. COM+ builds on COM's integrated services and features, making it easier for developers to create and use software components in any language, using any tool.

Coclasses - COM Classes.

CORBA - Common Object Request Broker Architecture from the OMG. It is the industry's leading standard for distributed objects and the dominant remoting architecture.

DCOM - Distributed COM; It's regarded as COM with a longer wire.

DLL - Dynamic Link Library.

DII - Dynamic Invocation Interface; It allows a client to make dynamic invocations on remote CORBA objects.

DSI - Dynamic Skeleton Interface; This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing.

EJB - Enterprise JavaBeans, a specification for a Java-based transaction service. Created by a group of companies led by Sun Microsystems Inc., the initial specification for EJB was released in spring 1998. While EJB originally stood on its own as a spec, since Dec 1999 it has been rolled into J2EE, the family of specs owned by Sun and JCP.

GIOP - General Inter-ORB Protocol, a standard CORBA protocol. IIOP is a TCP/IP-based implementation of GIOP.

GUID - Globally Unique Identifier, a 128-bit integer that uniquely identifies COM coclasses and interfaces.

IDL - Interface Definition Language.

IL - Intermediate Language used by .NET platform.

IIOP - Internet Inter-ORB Protocol, a protocol used for communication between CORBA object request brokers.

IUnknown - The COM interface class from which all other interface classes are derived. This interface allows all COM objects to manage their own lifetime, i.e., to release themselves from memory when they are no longer connected to any clients.
ISV - Independent Software Vender.

J2SE - Java 2 Standard Edition. (JDK+JRE) for common purposes for example on PCs.

J2ME - Java 2 Platform, Micro Edition. (JDK+JRE) for small devices such as cellular phones, palmtops etc.

J2EE - Java 2 Platform, Enterprise Edition, an application server framework from Sun Microsystems for the development of distributed applications. (J2SE + enterprise Java APIs).

JDK - Java Development Kit.

JRE - Java Runtime Environment.

JavaSoft - The Java Software division of Sun Microsystems, Inc.

JAAS - Java Authentication and Authorization Service, the J2EE API for managing security.

Java Applets - A packaging technology for downloading and running Java code on the client side in a browser.

JSP - Java Server Pages, the J2EE technology that, along with Java Servlets, is the programming model for the presentation tier.

Java Servlets - A server side Java program that integrates into a web server for dynamic page content generation.

JAXP - The Java API for XML Processing converts XML into a implementation independent format.

JDBC - Java Database Connectivity. The J2EE API for accessing the data tier.

JMS - Java Message Service, the J2EE API for accessing message queues.

JNDI - Java Naming and Directory Interface, the API for naming and locating specific instances used in J2EE.

JIT - Just-in-time.

JITA - Just-in-time activation, the ability of a COM object to be activated only as needed for executing requests from its client. Objects can be deactivated even while clients hold references to them, allowing otherwise idle server resources to be used more productively.

Marshaling - The act of formatting parameters for transmission through a proxy / stub pair. A proxy marshals data to a remote object, and a stub marshals data to a remote client.

Middleware - Software that mediates between an applications program and a network. It manages the interaction between disparate applications across the heterogeneous computing platforms.

MIDL - Microsoft IDL. The MIDL compiler generates a type library.

MSMQ - Microsoft Message Queue, a COM service that provides for the passing of messages between applications.

MTS - Microsoft Transaction Server, the middleware component model for Windows.

Object Adapter - An ORB component that provides object reference, activation, and state related services to an object implementation.

OLE - Object Linking and Embedding. A standard for linking and embedding documents in other documents. OLE is the evolutionary ancestor of COM.

OLE DB - Object Linking and Embedding for Databases. OLE DB is a COM service that enables a user to access databases. A developer accesses OLE DB services through ADO.

OMG –Object Management Group, an industry group with over six hundred member companies representing computer manufacturers, independent software vendors.

OMA - Object Management Architecture; It defines a broad range of services for building distributed applications. OMA services are divided into three layers named CORBA Services, CORBA Facilities, and Application Objects.

POA - Portable Object Adaptor; It provides the mechanism by which a server process maps CORBA objects to language-specific implementations, or servants.

Proxy - An object that runs in a client's process that acts as a channel for all communication between the client and a remote object. When a client attempts to accesses a server object the proxy intercepts the call and issues an RPC to the real instance of the server object.

RMI - Java Remote Method Invocation.

RMI-IIOP - RMI over IIOP, a version of RMI implemented to use the CORBA IIOP protocol. RMI over IIOP provides interoperability with CORBA objects implemented in any language if all the remote interfaces are originally defined as RMI interfaces.

ROI – Return On Investment.
SSCLI – Microsoft provides Shared Source Common Language Implementation (also known as Rotor). It provides a free, shared-source implementation of Microsoft's Common Language Runtime platform, including source code for C# compilers, as well as for the Common Language Infrastructure (CLI) platform itself. It is the working implementation to provide a Platform Adaption Layer (PAL) for academics and researchers.

SOAP - Simple Object Access Protocol, a communication protocol based on XML that simplifies the complexities of cross-language and cross-platform communication.

Stub - An object that runs in a server's process that acts as a channel for all communication between the server and a remote client.

Type Library - A binary file that describes interfaces, coclasses, and other resources in a COM server.

UUDI - Universal Description, Discovery and Integration specification.

VS.NET – Visual Studio .NET, the programming environment for .NET.

WAS – IBM WebSphere Application Server

Web Services - A Web Service is application or business logic that is accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the World Wide Web. Like components, Web Services represent black-box functionality that can be used and reused without regard to how the service is implemented. Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web.

WSDL - Web Services Description Language

List of Figures

List of Tables

Introduction

Microsoft Component Object Model (COM) and the Object Management Group Common Object Request Broker Architecture (CORBA) are competing architectures for application development. Both approaches apply object technology to the design and use of software components to simplify network programming. DCOM (an extension of COM) and CORBA also provide application interoperability and limited portability in distributed computing environments. Thus, both architectures are intended for enterprise-scale development.

OMG's CORBA is considered the industry's leading standard for distributed objects, and Microsoft's DCOM is the 'de-facto' standard.

Middleware makes the network communications layer completely transparent to the application software and provide robust underlying commerce platform that enables developers to craft high-performing, scalable, maintainable, and multi-user secure commerce systems. Middleware Component Models take a high level approach to building distributed systems. Microsoft Transaction Server (MTS), is based on the Component Object Model (COM), it's the middleware component model for Windows NT.

Enterprise Java Bean (EJB) is a middleware component model for Java and CORBA. It defines a set of specifications and component architecture framework for creating distributed n-tier middleware. Both MTS and EJB target the creation of enterprise level server-side, component-based, transaction-oriented applications.

The decade-old rivalry between Microsoft and Java development communities enters a new phase with the emergence of Web services. Each community is working hard to create the best

support for Web services standards such as the Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). As the companies fight for the hearts, minds, and pocketbooks of current and future developers, Microsoft and the Java vendors differ in the way they're embracing these new technologies. This thesis compares and contrasts their approaches to Web services development and deployment, explain where they converge and where they diverge, and provide guidelines and strategies for deciding how to choose an appropriate platform for building an enterprise solution.

Many IT department are now facing the decision whether to start building application on a J2EE architecture from Sun Microsystems and adopt the Java Community Process (JCP) or the .NET architecture from Microsoft. This is a major decision which will affect companies for many years to come, not only in terms of the efficiency of IT, but might possibly become a key factor for the productivity of the company as a whole.

A company needs to anticipate significant implications, reaching far beyond the core middleware technology. The impact includes operating system choices, development languages and tools, availability of skilled resources and time to market as a whole. This decision will also impact purchasing decision of new packaged applications, interoperability with legacy systems as well as business partners.

This thesis reviews and examines these candidate architectures, and analyzes the similarities and differences between DCOM and CORBA, EJB and MTS, from a programmer's standpoint and an architectural standpoint. The thesis provides an in-depth comparison of the .NET and J2EE platforms, and analyzes their key advantages and disadvantages to support decision making for building enterprise solutions. The thesis covers both theoretical parts and coding parts. The theoretical part focuses on the study of underlying techniques from the Object-Oriented, distributed, component-based, and transaction-based prospective. The coding part will

build some application by using Visual C++, VB, ATL, and IONA Orbix2000 etc. Both researches and sample applications will be concentrated on building enterprise level solutions and reflecting the real world practices with these aspects.

The summary of the thesis work on DCOM/CORBA:

- Reviews and examines these candidate architectures
- Analyzes the similarities and differences
- Compares the two middleware component models — MTS and EJB
- Build sample applications both in COM (using Microsoft IDE tools) and CORBA (Using IONA Orbix2000) to demonstrate the similarities and differences
- Develops the strategy for assessment of the two technologies

The summary of the thesis work on .NET/J2EE:

- Provides in-depth comparisons from both technical and business aspects
- Examine the approaches to building XML-based Web Services in the two platform
- Provide sample architecture of a Web Services application and demonstrate how the two platforms can integrate
- Analyzes their key advantages and disadvantages
- Examine the suitability
- Analyzes the future trends of both platforms
- Provides supports for decision making for building enterprise solutions
- Provides recommendations for choosing the platforms

1 Microsoft's Component-based Technologies

1.1 Overview of COM/DCOM

1.1.1 From C++ to COM

Object-oriented programming (such as C++) is one of the more recent paradigms to enjoy a long and somewhat favorable reception by the software industry [7]. Object-oriented programming became as popular as it did largely because it allowed developers to share code among entirely different projects. While code sharing and reuse is considered a primary benefit of a well-implemented object-oriented design, the percentage of code actually being shared is still small, there are both design-time and runtime obstacles. Many of these obstacles stem from the compilation and linkage model assumed by C++.

Static Linking

Static linking is a way of code reuse, but it has some disadvantages. The first disadvantage to the static-linking approach is redundancy. Every client application, which uses the same library, can share the library's source code for compilation, but they cannot share their target executable image when the machine code is generated. The library code is bound into the client application for every client application. This will immediately cause the waste of user's hard disk, and eventually will waste their virtual memory space when these applications are running at the same time. The second disadvantage is the clients will have to rebuild and reissue their applications to accommodate the new version of the library.

Dynamic Linking

A solution to the problem inherent in static linking is dynamic linking. Dynamic-link libraries (DLLs) are pieces of executable code that exists only once on the user's hard disk. When multiple clients access the code for the library, the OS's loader is smart enough to share the physical memory

pages containing the library's executable code between all client program. The problem of redundancy caused by static linking is overcome. It is also possible to ship a new DLL to the ender user to repair the defects for all client applications. The DLL turns the original C++ class into a field-replaceable and efficient reusable component.

Before COM, Inprise, Microsoft and Symantec all support exporting entire C++ classes from a DLL. However, there is still a downside to exporting C++ class in traditional DLL. When distributing C++ classes as a DLL, one big problem in portability with C++ is lack of standardization at the binary level. The lack of a C++ binary standard makes it impossible to create a vendor-independent component substrate. Another problem is the encapsulation in DLL. Although C++ provides syntax for making members private, protect or public, the semantics don't apply at runtime. The compilation model of C++ requires the client's compiler to have access to all information regarding object layout in order to instantiate an instance of a class. This includes information about the size and order of the object's private and protected data members. When the DLL vendor exports a class from a DLL, it implicitly provides all sorts of non-interface-related information to the client that can vary from compiler to compiler (or even from one version to another of a single compiler).

Assume a new version of DLL, in which a new private member data is added into a specific class, is shipped to the client, the client have already coded and built against the layout of that specific class. Using old client code and new DLL will likely result in a serious program crash. Simply saying, if you're using `new` in your client to allocate objects, you won't be able to change the size of the object (that is, add any data) without recompiling the world.

When a new function is added to the DLL, the clients that want new functionality recompile, the old clients simply break. This is an inherent limitation of virtually all C++ environments.

It might seem that maintaining different names for the DLL would help, provided that clients always load the version of the DLL that they were built against, irrespective of what other versions may be present on the system. Unfortunately, this may cause significant problem in versioning and configuration management.

Java and COM

Java does solve some of these problems that encountered by C++, but it also adds some of its own. The biggest problem is that Java components (e.g. JavaBeans) are only intended to be used by programs written in Java. In general, except for users who are running Windows (Now Microsoft virtual machine allows JavaBeans be used as COM objects, and can therefore be used from any language), Java is a single-language system: JavaBean components can only be used in Java programs, the program written in any other languages can not access JavaBean component.

Java also makes you decide when you write your program whether the component you're using is local (on your machine) or remote (on another machine)—and the methods for using local and remote components are quite different.

Java has a couple of other issues that make it a less-than-ideal answer for all component needs (COM is partially design to serve for this purpose). For instance, it has no really solid way to deal with versioning.

1.1.2 Problem with Traditional Approach

One solution to the challenges of software development is to separate the functionality of a large, monolithic application into smaller components. Traditionally, developers have used libraries of functions as a way of accessing the functionality of a component. These libraries implement functions through an *Application Programming Interface* (API). Reuse of these

functions is as simple as learning the semantics of the API and linking to the library. The following illustration shows how a client can use a component from any vendor by calling it through a standardized API:

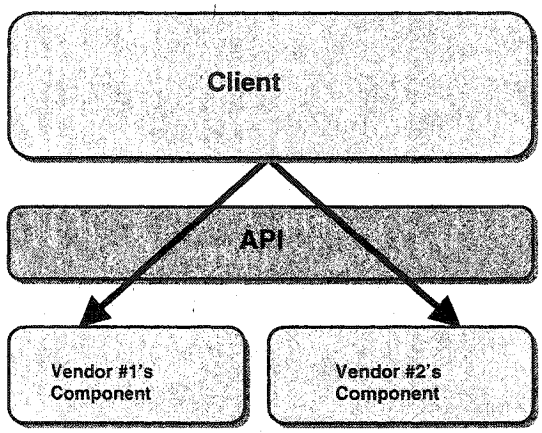


Figure 1-1: Traditional Approach for Software Components

An example of a standardized API is the *Microsoft Open Database Connectivity* (ODBC) API. Many software vendors have implemented the ODBC API in their database drivers. Even with fairly widespread adoption of the ODBC API, interoperability is still challenging because implementation differences can occur from one vendor to another.

The traditional approach of using APIs to access the functionality of a software component has its drawbacks. These drawbacks include evolution of the API, versioning, component communication, and the implementation language. The evolution of an API is a problem for both the API creator and software vendors who want to add value by extending an API. Any changes made to an API by its creator can potentially break existing applications. Changes made to extend the API can result in inconsistent implementations. Advertising and maintaining different versions of the API can also be problematic. As an API creator, how can you force a developer to check for the correct version? Enabling components to communicate with each other is challenging, especially if different developers have created the components. The programming language you use for creating components greatly impacts how the components

will communicate through an API. For example, if you create components in C++ and export classes from a library, it can be challenging to use C or Visual Basic to create the client of the component.

The major goals of COM are language and vendor independence, location transparency, and reduced version problems. COM solve basic interoperability and transparent cross-process interoperability problems — How can developers create their own unique components, yet be assured that these components will interoperate with other components built by different developers? How can we give developers the flexibility to write components to run in-process or cross-process (and eventually cross-network), using one simple programming model?

1.1.3 Component Object Model

COM is a standard (or model) for the interaction of binary objects. An important feature of COM is that objects are precompiled, which means that the implementation language is irrelevant. COM is also an integration technology. Components can be developed independently, and COM provides the standard model for integrating these components. One can think of COM as an enabling technology, rather than a solution in itself.

COM [COM 95] [1] refers to both a specification and implementation developed by Microsoft Corporation, which provides a framework for integrating components. This framework supports interoperability and reusability of distributed objects by allowing developers to build systems by assembling reusable components from different vendors which communicate via COM. By applying COM to build systems of preexisting components, developers hope to reap benefits of maintainability and adaptability.

COM is an object-based programming model that, due to its binary interoperability standard, facilitates the development of software components at different times by different vendors using a

variety of languages, tools, and platforms. Once developed, COM components are easily packaged as reusable building blocks without shipping source code, deployed and integrated into a customer's environment. Corporate application developers can use COM to create new solutions that combine in-house business objects, off-the-shelf objects, and their own custom components.

1.1.4 COM Interface and COM Object

COM is a binary compatibility specification and associated implementation that allows clients to invoke services provided by COM-compliant components (COM objects). COM objects are different from source-code object-oriented programming (OOP) objects such as those defined in C++. Component objects usually have some associated data, but unlike C++ objects, a given component object will never have direct access to another component object in its entirety. Instead, component objects *always* access other component objects through interface pointers. This is a primary architectural feature of the Component Object Model, because it allows COM to completely preserve encapsulation of data and processing, a fundamental requirement of a true component software standard. It also allows for transparent remoting (cross-process or cross-network calling) since all data access is through methods that can exist in a proxy object that forwards the request and vectors back the response.

The services implemented by COM objects are exposed through a set of interfaces that represent the only point of contact between clients and the object, as shown in Figure 1-2,

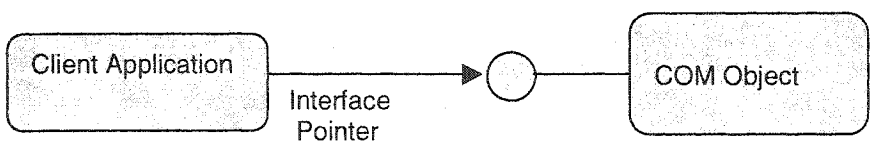


Figure 1-2: Client Using COM Object Through an Interface Pointer [COM 95]

COM defines a binary structure for the interface between the client and the object. This binary structure provides the basis for interoperability between software components written in arbitrary languages. As long as a compiler can reduce language structures down to this binary representation, the implementation language for clients and COM objects does not matter — the point of contact is the run-time binary representation. Thus, COM objects and clients can be coded in any language that supports Microsoft's COM binary structure.

A COM object can support any number of interfaces. An interface provides a grouped collection of related methods. For example, Figure 1-3 depicts a COM object that emulates a cellular phone. IPhone, IStockAlert and IAddressBook are the interfaces of the cellular phone. The IPhone interface can provide the appropriate methods to allow making phone calls. The IStockAlert and IAddressBook interfaces can supply stock alert and address book methods.

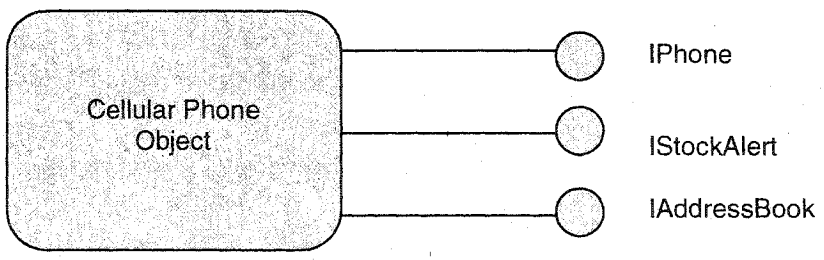


Figure 1-3: Cellular Phone COM object

COM objects and interfaces are specified using Microsoft Interface Definition Language (IDL), an extension of the DCE Interface Definition Language standard. To avoid name collisions, each object and interface must have a unique identifier.

Interfaces are considered logically immutable. Once an interface is defined, it should not be changed — new methods should not be added and existing methods should not be modified. This restriction on the interfaces is not enforced, but it is a rule that component developers should follow. Adhering to this restriction removes the potential for version incompatibility — if an interface never changes, then clients depending on the interface can rely on a consistent set of

services. If new functionality has to be added to a component, it can be exposed through a different interface. For our cellular phone example, we can design an enhanced cellular phone COM object supporting the IStockAlert2 interface that inherits from IStockAlert. IstockAlert2 may expose new functionality.

1.1.5 Binary Standard and Language Independence

For any given platform (hardware and operating system combination), COM defines a standard way to lay out virtual function tables (vtables) in memory, and a standard way to call functions through the vtables (as shown in Figure 1-4). Thus, any language that can call functions via pointers (C, C++, Small Talk®, Ada, and even Basic) can be used to write components that can interoperate with other components written to the same binary standard. The double indirection (the client holds a pointer to a pointer to a vtable) allows for vtable sharing among multiple instances of the same object class. On a system with hundreds of object instances, vtable sharing can reduce memory requirements considerably.

Components can be implemented in a number of different programming languages and used from clients that are written using completely different programming languages. Again, this is because COM, unlike an object-oriented programming language, represents a binary object standard, not a source code standard. This is a fundamental benefit of a component software architecture over object-oriented programming (OOP) languages. Objects defined in an OOP language typically interact only with other objects defined in the same language. This necessarily limits their reuse. At the same time, an OOP language can be used in building COM components, so the two technologies are actually quite complementary. COM can be used to "package" and further encapsulate OOP objects into components for widespread reuse, even within very different programming languages.

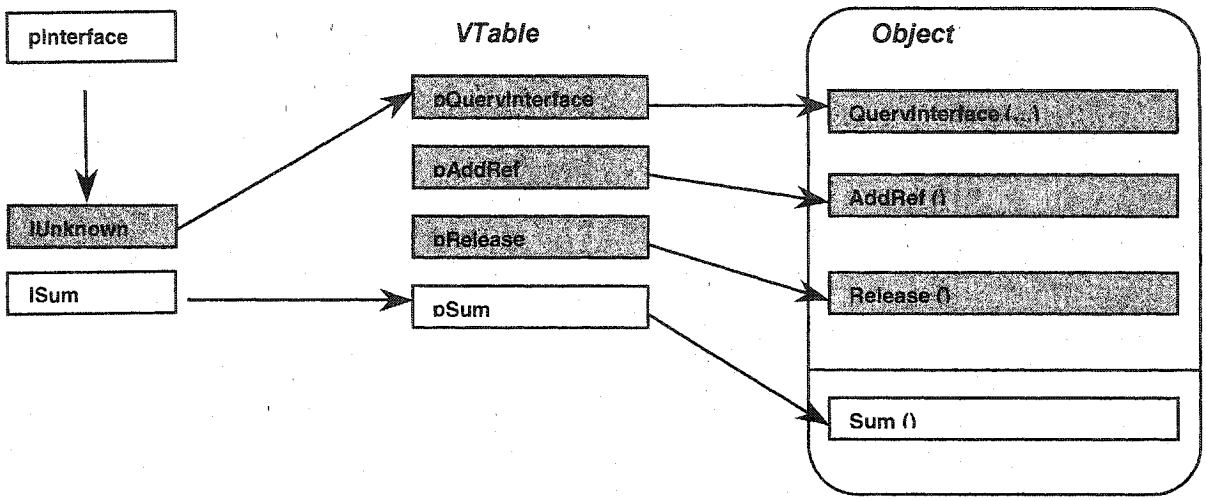


Figure 1- 4: VTable layout in COM

1.1.6 The IUnknown Interface and Versioning

Each COM object must support at least the standard IUnknown interface. IUnknown defines methods that provide the basic building blocks for managing object life cycles and allowing graceful evolution of interfaces supported by an object. IUnknown has three methods: QueryInterface, AddRef, and Release. In C++ syntax, IUnknown is as below, the "interface" is just a macro definition of the key word struct.

```
interface IUnknown {
   virtual   HRESULT   QueryInterface(IID& iid, void** ppvObj) = 0;
   virtual   ULONG   AddRef() = 0;
   virtual   ULONG   Release() = 0;
   }
```

Versioning in COM is implemented using interfaces and IUnknown::QueryInterface. The COM design completely eliminates the need for things like version repositories or central management of component versions.

The QueryInterface method of IUnknown is used by clients to determine whether or not a particular interface, specified as an IID, is supported by an object. Over time, an object may support new interfaces or new versions of the same logical interface, each with its own unique IID. Existing clients can continue using an earlier version of an interface without even being recompiled, and new clients can query for — and take advantage of -- the latest version of the interface. QueryInterface returns a pointer called an interface pointer. Internally, an interface pointer points to a data structure that is dictated by COM's binary interoperability standard. The standard dictates the way interface functions must be called, regardless of differences in the implementation environments of the client and server programs.

The combination of the use of interfaces (immutable, well-defined "functionality sets" that are extruded by components) and QueryInterface (the ability to cheaply determine at run time the capabilities of a specific component object) enable COM to provide an architecture in which components can be dynamically updated, without requiring updates to other reliant components. This is a fundamental strength of COM over other proposed object models. COM solves the versioning/evolution problem where the functionality of objects can change independently of clients of that object without rendering existing clients incompatible. In other words, COM defines a system in which components continue to support the interfaces through which they provided services to older clients, as well as support new and better interfaces through which they can provide services to newer clients. At run time old and new clients can safely coexist with a given component object. Errors can only occur at easily handled times: bind time or during a QueryInterface call. There is no chance for a random crash such as those that occur when an expected method on an object simply does not exist or its parameters have changed.

1.1.7 Reference Counting

COM does not automatically remove a COM object from memory when the COM object is no longer being used. The COM object must provide for its removal programmatically based on its

reference count. Each `AddRef` call increments, and each `Release` call decrements, a counter variable inside the COM object. When the count returns to zero, the interface no longer has any users, and is therefore free to remove itself from memory.

1.1.8 Component Object Library and Transparent Cross-Process Interoperability

Every COM object runs inside a server. A single server can support multiple COM objects. There are three ways in which a client can access COM objects provided by a server (Figure 1-5):

1. In-process server: The client can link directly to a library containing the server. The client and server execute in the same process. Communication is accomplished through function calls.
2. Local Object Proxy: The client can access a server running in a different process but on the same machine through an inter-process communication mechanism. This mechanism is actually a lightweight Remote Procedure Call (RPC).
3. Remote Object Proxy: The client can access a remote server running on another machine. The network communication between client and server is accomplished through DCE RPC. The mechanism supporting access to remote servers is called DCOM.

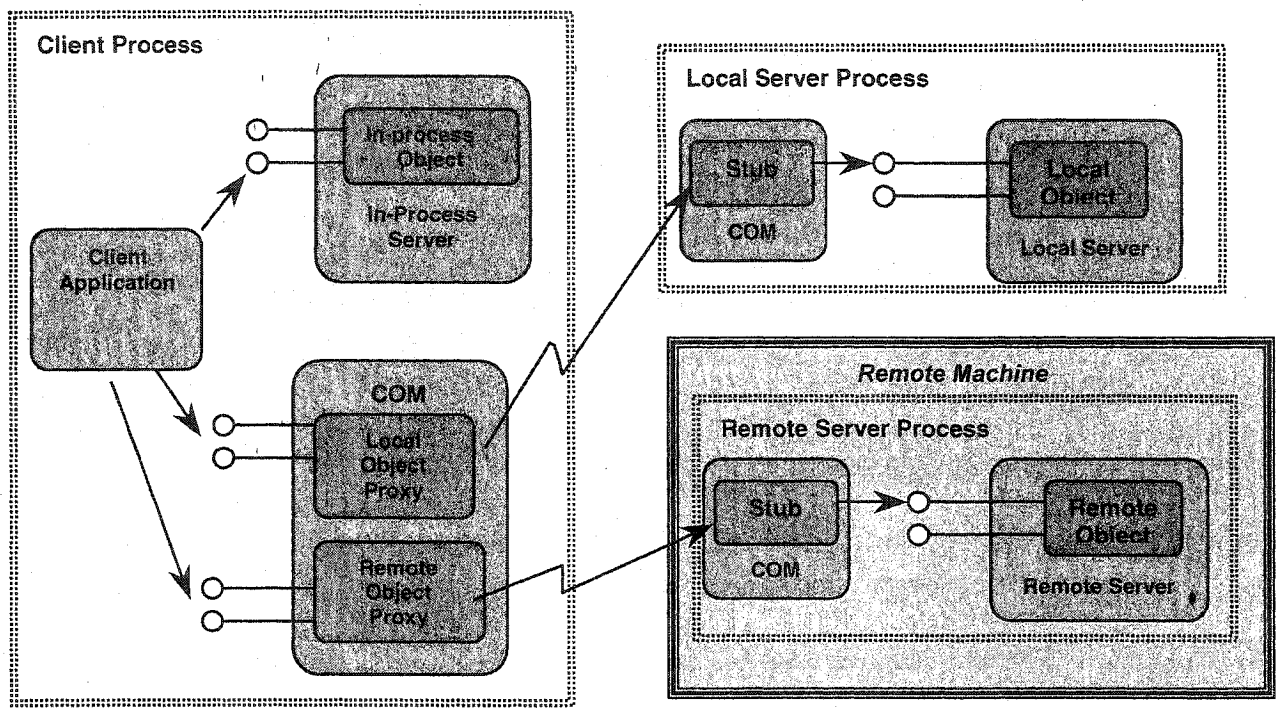


Figure 1- 5: Three Methods for Accessing COM Objects [COM 95]

The Component Object Library is a system component that provides the mechanics of COM. The Component Object Library is the key to providing transparent cross-process interoperability. It encapsulates all the legwork associated with finding and launching components and managing the communication between components. The Component Object Library insulates components from the location differences. This means that component objects can interoperate freely with other component objects running in the same process, in a different process, or across the network. The code used to implement a component object in any of these cases is exactly the same. Thus, when a new Component Object Library is released with support for cross-network interaction, existing component objects will be able to work in a distributed fashion without requiring source-code changes, recompilation, or redistribution to customers.

1.1.9 Cross-Process Communication in COM

If the client and server are in the same process, the sharing of data between the two is simple. However, when the server process is separate from the client process, as in a local server or remote server, COM must format and bundle the data in order to share it. This process of preparing the data is called marshalling. Marshalling is accomplished through a "proxy" object and a "stub" object that handle the cross-process communication details for any particular interface (depicted in Figure 1-5). COM creates the "stub" in the object's server process and has the stub manage the real interface pointer. COM then creates the "proxy" in the client's process, and connects it to the stub. The proxy then supplies the interface pointer to the client.

The client calls the interfaces of the server through the proxy, which marshals the parameters and passes them to the server stub. The stub unmarshals the parameters and makes the actual call inside the server object. When the call completes, the stub marshals return values and passes them to the proxy, which in turn returns them to the client. The same proxy/stub mechanism is used when the client and server are on different machines. However, the internal implementation of marshalling and unmarshalling differs depending on whether the client and server operate on the same machine (COM) or on different machines (DCOM). Given an IDL file, the Microsoft IDL compiler can create default proxy and stub code that performs all necessary marshalling and unmarshalling.

1.1.10 COM Object Creation

All COM objects are registered with a component database. As shown in Figure 1-6, when a client wishes to create and use a COM object:

1. It invokes the COM API to instantiate a new COM object.
2. COM locates the object implementation and initiates a server process for the object.
3. The server process creates the object, and returns an interface pointer at the object.
4. The client can then interact with the newly instantiated COM object through the interface pointer.

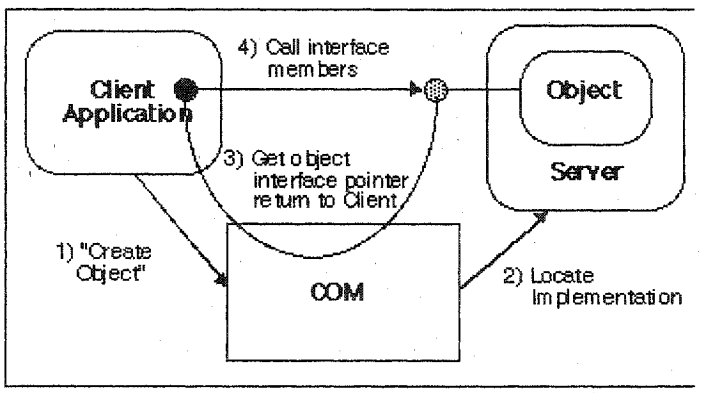


Figure 1- 6: Creating a COM object pointer [COM 95]

1.1.11 DCOM and COM+

Distributed COM [DCOM 97] [2] is an extension to COM that allows network-based component interaction. While COM processes can run on the same machine but in different address spaces, the DCOM extension allows processes to be spread across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the environment.

It is best to consider COM and DCOM as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across heterogeneous networks. While COM and DCOM represent "low-level" technology that allows components to interact, OLE, ActiveX and MTS represent higher-

level application services that are built on top of COM and DCOM. OLE builds on COM to provide services such as object "linking" and "embedding" that is used in the creation of compound documents (documents generated from multiple tool sources). ActiveX extends the basic capabilities to allow components to be embedded in Web sites. MTS expands COM capabilities with enterprise services such as transaction and security to allow Enterprise Information Systems (EIS) to be built using COM components. COM+ is the evolution of COM.

COM+ integrates MTS services and message queuing into COM, and makes COM programming easier through a closer integration with Microsoft languages as Visual Basic, Visual C++, and J++. COM+ will not only add MTS-like quality of service into every COM+ object, but it will hide some of the complexities in COM coding.

2 OMG and JavaSoft's Component-based Technologies

2.1 Overview of CORBA

2.1.1 Distributed Object and Object Request Broker

As networks of computing resources have become prevalent, the concept of distributing related processing among multiple resources has become increasingly viable and desirable. Some applications by their very nature are distributed across multiple computers because of the facts that the data used by the application are distributed, the computation is distributed and the users of the application are distributed. Client/Server architecture is today's most common approach to distributed computing. In Client/Server applications, network clients request information or a service from a server, and that server responds to the client by acting on that request and returning results. A simple Client/Sever application consists of a client layer (also known as tier) and server layer. Enterprise applications are often built upon 3-tier or N-Tier client/server system, in which the middleware "acts as the glue or plumbing between two otherwise separate applications" [37].

Object distribution architectures build upon the middleware concept by encapsulating data within functional interfaces to objects. Like well-designed procedural APIs, implementation details are hidden from the user of the object. Unlike traditional APIs, however, object architectures limit access to the invocation of methods defined for the object. Furthermore, methods are invoked on the objects indirectly, via references to the objects, eliminating the need for local instances of the objects. This near-complete implementation hiding allows distributed object architectures to support location, platform, and programming language transparency. Such transparency is not without its costs, however, which has prompted the designers of some distributed object

architectures to forego some neutrality in exchange for perceived improvements in performance, applicability to specific tasks or ease of use.

Three of the most popular distributed object paradigms are Microsoft's DCOM, OMG's CORBA and JavaSoft's Java Remote Method Invocation (Java RMI). They are all considered as object request broker (ORB) technologies. An object request broker is a middleware technology that manages communication and data exchange between objects. ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects from different vendors that communicate with each other via the ORB. The implementation details of the ORB are generally not important to developers building distributed systems. The developers are only concerned with the object interface details. This form of information hiding enhances system maintainability since the object communication details are hidden from the developers and isolated in the ORB.

ORB technology promotes the goal of object communication across machine, software, and vendor boundaries. The relevant functions of an ORB technology are

- Interface definition
- Location and possible activation of remote objects
- Communication between clients and object

2.1.2 Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a *specification* of a standard architecture for object request brokers (ORBs). A standard architecture allows vendors to develop ORB products that support application portability and interoperability across different programming languages, hardware platforms, operating systems, and ORB implementations.

Using a CORBA-compliant ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call, and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results of the invocation. The client does not have to be aware of where the object is located, its programming language, its operating system or any other aspects that are not part of an object's interface. The "vision" behind CORBA is that distributed systems are conceived and implemented as distributed objects. The interfaces to these objects are described in a high-level, architecture-neutral specification language that also supports object-oriented design abstraction. When combined with the Object Management Architecture, CORBA can result in distributed systems that can be rapidly developed, and can reap the benefits that result from using high-level building blocks provided by CORBA, such as maintainability and adaptability.

The CORBA specification was developed by the Object Management Group (OMG), an industry group with over six hundred member companies representing computer manufacturers, independent software vendors. The OMG was established in 1988, and the initial CORBA specification emerged in 1992. Since then, the CORBA specification has undergone significant revision, with the latest major revision (CORBA v2.6) [3] released in December 2001.

CORBA ORBs are middleware mechanisms, as are all ORBs. CORBA can be thought of as a generalization of remote procedure call (RPC) that includes a number of refinements of RPC, including:

- a more abstract and powerful interface definition language
- direct support for a variety of object-oriented concepts
- a variety of other improvements and generalizations of the more primitive RPC

Figure2-1 illustrates the primary components in the CORBA ORB architecture.

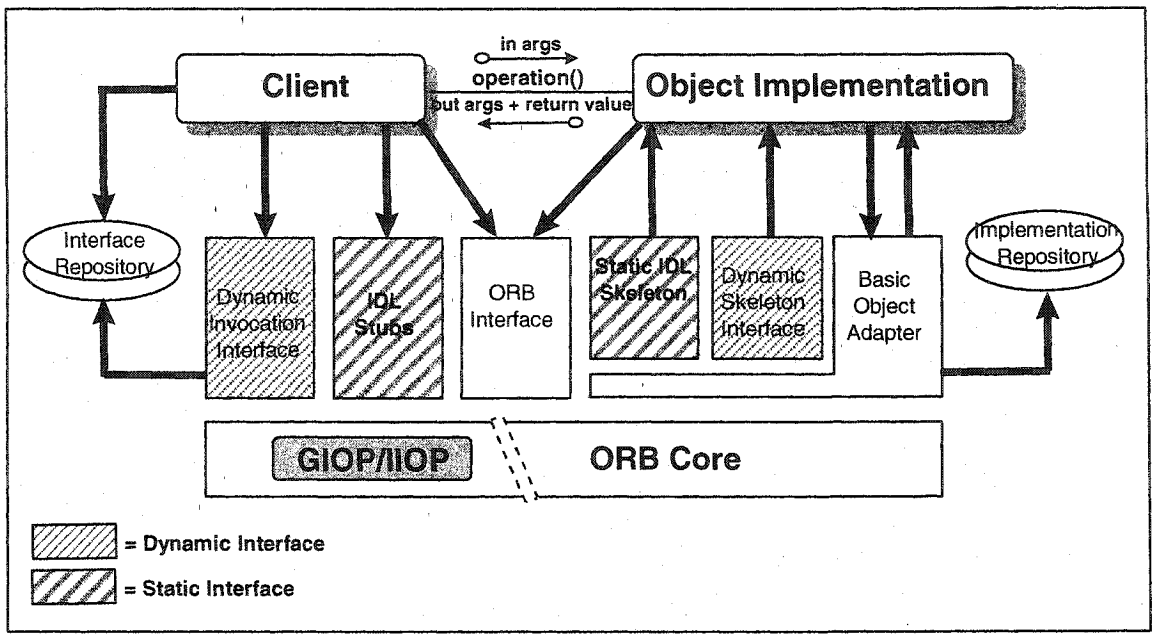


Figure 2-1 The Common Object Request Broker (CORBA) 2.0 Architecture

- ***Object*** — Object, often referred as object implementation or Servant, is a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation.* The Servant is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.
- ***Client*** — Client is the program entity that invokes operations on Servants. Accessing the services of a remote object should be transparent to the caller.
- ***IDL Stubs and Skeletons*** — OMG IDL language compilers and translators also generate client-side stubs and server-side skeletons. Client IDL stubs are used by clients to interface to object services. A client has an IDL stub for each interface it uses on a server. Stubs allow clients to call the methods of remote objects as if they are local. A skeleton is a mechanism that delivers requests to the CORBA object implementation. Skeletons provide static interfaces to each service exported by the server. The stub and skeleton handle the underlying protocol issues such as marshaling and un-marshaling associated with method calls on server objects.

- ***Object Request Broker (ORB)*** — The ORB takes care of all of the details involved in routing a request from client to object, and routing the response to its destination. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. A number of CORBA implementations exist in the market today, including Orbix from IONA Technologies (http://www.iona.com), VisiBroker from Inprise, and JavaIDL from JavaSoft (http://java.sun.com/products/jdk.idl). All CORBA 2.0 (and above) compliant ORBs are able to interoperate via the Internet Inter-ORB Protocol (IIOP). The whole purpose of IIOP is to ensure that the client will be able to communicate with a server written for a different ORB from a different vendor. Orbix ORB is used to build a simple application and illustrate how CORBA works in this thesis.
- ***ORB Interface*** — To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.
- ***Dynamic Invocation Interface (DII)*** — DII allows a client to make dynamic invocations on remote CORBA objects. The client does not have knowledge about an object it wants to invoke at compile time. Once an object is discovered, the client program can obtain a definition of it, issue a parameterized call to it, and receive a reply from it, all without having a type-specific client stub for the remote object.
- ***Dynamic Skeleton Interface (DSI)*** — This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.
- ***Object Adapter*** — The object adapter is an ORB component which provides object reference, activation, and state related services to an object implementation. Object adapters can be specialized to provide support for certain object implementation styles.

- ***Interface Repository*** — A run-time metadata repository of registered IDL-defined interfaces, including their methods/operations and the parameters they require.
- ***Implementation Repository*** — The run-time database storing all of an ORB's registered objects, either activated or available for activation by client requests.

2.1.3 CORBA and the Object Management Architecture (OMA)

CORBA is based on Object Management Architecture. OMA defines a broad range of services for building distributed applications. OMA services are divided into three layers named CORBA Services, CORBA Facilities, and Application Objects as shown in figure 2-2. ORB communication infrastructure is required for applications to access these services. These services are actually definition of different categories of objects in OMA and define a broad range of functionality needed to support distributed applications. The OMA goes far beyond RPC in scope and complexity.

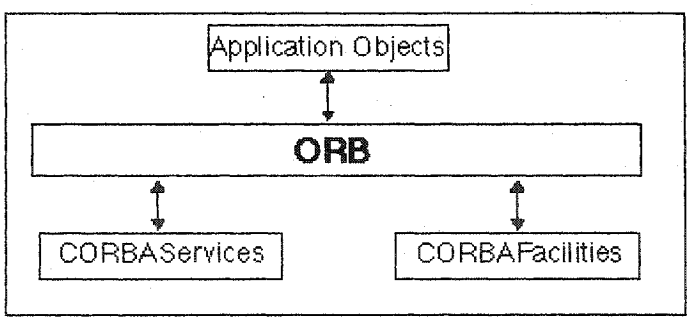


Figure 2-2: Object Management Architecture

The Object Request Broker (ORB) is a communication infrastructure through which applications access these services, and through which objects interact with each other. CORBAServices, CORBAFacilities, and ApplicationObjects define different categories of objects in the OMA; these objects define a range of functionality needed to support the development of distributed software systems.

- ***CORBAServices***—A collection of services that support basic functions for using and implementing objects. Services are always independent of application domains. They are accessed via middleware APIs and are packaged with IDL-specified interfaces. OMG has published standards for sixteen object services. Among them are asynchronous event management, transactions, persistence, externalization, concurrency, naming, relationships, and lifecycle. Table 1 summarizes the purpose of some of these services.
- CORBAFacilities—Include user interface, information management, system management, task management, and a variety of "vertical market" facilities in domains such as manufacturing, distributed simulation, enterprise resource management, and accounting.
- Application Objects—Provide services that are particular to an application or class of applications. These are not a topic for standardization within the OMA, but are usually included in the OMA reference model for completeness, i.e., objects are either application-specific, support common facilities, or are basic services.

Naming Service	Provides the ability to bind a name to an object. Similar to other forms of directory service.
Event Service	Supports asynchronous message-based communication among objects. Supports chaining of event channels, and a variety of producer/consumer roles. A client does not directly invoke an operation on an object in a server. Instead, the client sends an event that can be received by any number of objects. The sender of an event is called a *supplier*; the receivers are called *consumers*. An intermediary *event channel* takes care of forwarding events from suppliers to consumers. Both *push* and *pull* model of event transfer are defined in the CORBA event specification.
Persistence Service	Provides common interfaces for the mechanisms used for retaining and managing the persistent state of objects in a data-store independent manner.
Object Transaction Service	object transaction service (OTS) servers for the common case where only a single resource (database) is involved in a transaction. Applications built against the single resource OTS can easily be reconfigured to use a full-blown OTS when it is available, since the interfaces are identical.
Concurrency Service	Supports concurrent, coordinated access to objects from multiple clients.

Table 1: Overview of CORBA Services

2.1.4 Major Enhancement in CORBA 3.0

CORBA 3.0 has several major new features, including POA, CORBA messaging, and objects by value. POA provides new features that allow applications and their servants to be portable between different ORBs supplied by different vendors. CORBA messaging adds two new asynchronous request techniques: *polling* and *callback*. These new techniques represent a significant advantage for most programming languages because static invocations provide a more natural programming model than the DII. CORBA 3.0 also introduce the feature of using objects by value.

2.1.4.1 Portable Object Adaptor

A portable object adapter, or POA, provides the mechanism by which a server process maps CORBA objects to language-specific implementations, or servants. All interaction with server objects takes place via the POA. Portable Object adapters provide a number of services, including the creation of CORBA objects and their references, dispatching requests to the appropriate servant that provides an implementation for the target object, and activation and deactivation of CORBA objects.

In CORBA 2.0, the only standard object adapter defined by the OMG is called the basic object adapter (BOA), which only provides basic services to allow a variety of CORBA objects to be created. ORB vendors and developers, however, discovered that the BOA is ambiguous and missing some features. This led vendors to develop their own proprietary extensions, which resulted in poor portability between different ORB implementations. The new standard object adapter, the POA, provides new features that allow applications and their servants to be portable between different ORBs supplied by different vendors. This new adapter provides a great deal of flexibility for server implementations.

The POA mediates between the ORB and the server application. Figure 2-3 shows a request from a client to a server.

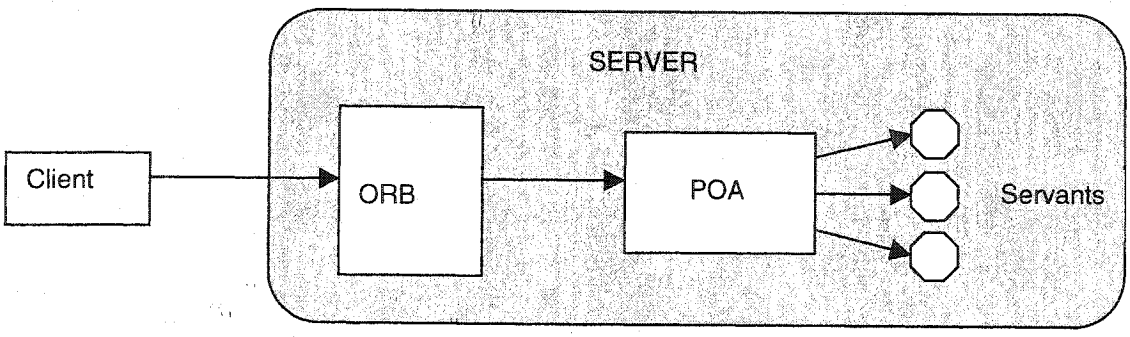


Figure 2-3: Request dispatching based on POA

The client invokes the request using a reference that refers to the target object. The request is then received by the ORB, which will dispatch the request to the POA that hosts the target object. The POA will then dispatch the request to the servant, which subsequently carries the request and sends the results back to the POA, to the ORB, and finally to the client. An application may have multiple POAs, and in order for the ORB to dispatch the request to the right POA, it uses an *object key*, which is an identifier that is part of the request that is kept in the object reference. A part of the object key called the object ID is used by the POA to determine an association between the target object and a servant.

2.1.4.2 CORBA messaging

CORBA 2.0 provides three different techniques for operation invocations:

- ***Synchronous*** - The client invokes an operation, then pauses, waiting for a response.
- ***Deferred synchronous*** - The client invokes an operation then continues processing. It can go back later to either poll or block waiting for a response.
- ***One-way*** - The client invokes an operation, and the ORB provides a guarantee that the request will be delivered. In one-way operation invocations, there is no response.

Synchronous invocation techniques tend to tightly couple clients and servers. This has led many people to criticize CORBA as being unable to cope with large distributed systems. For this reason, a new specification has been adopted by the OMG. This new specification preserves the invocation techniques in CORBA 2.0 and adds two new asynchronous request techniques:

- ***Callback*** - The client supplies an additional object reference with each request invocation. When the response arrives, the ORB uses that object reference to deliver the response back to the client.
- ***Polling*** - The client invokes an operation that immediately returns a *valuetype* that can be used to either poll or wait for the response.

The callback and polling techniques are available for clients using statically typed stubs generated from IDL interfaces. These new techniques represent a significant advantage for most programming languages because static invocations provide a more natural programming model than the DII.

2.1.4.3 Objects by Value

One of the criticisms of CORBA 2.0 is the lack of support for passing objects by value. This has been addressed by adding support for passing objects by value. This has led to the addition of a new construct to the OMG IDL called the valuetype. A valuetype supports both data members and operations, much the same as a Java class definition. When a valuetype is passed as an argument to a remote operation, it will be created as a copy in the receiving address space. The identity of the valuetype copy is separate from the original, so operations on one have no effect on the other. It is important to note that operations invoked on valuetypes are local to the process in which the valuetype exists. This means that valuetype invocations never involve the transmission of requests and replies over the network.

3 Comparisons of CORBA and COM/DCOM

3.1 The Architectures

3.1.1 The Architectural Similarities between DCOM and CORBA

Both technologies support location transparency; use IDL to define the set of services provided by the server objects; generate proxy/stub code to facilitate the marshaling of data to a remote object and marshaling data to a remote client (as shown in Figure 3-1 and Figure 3-2).

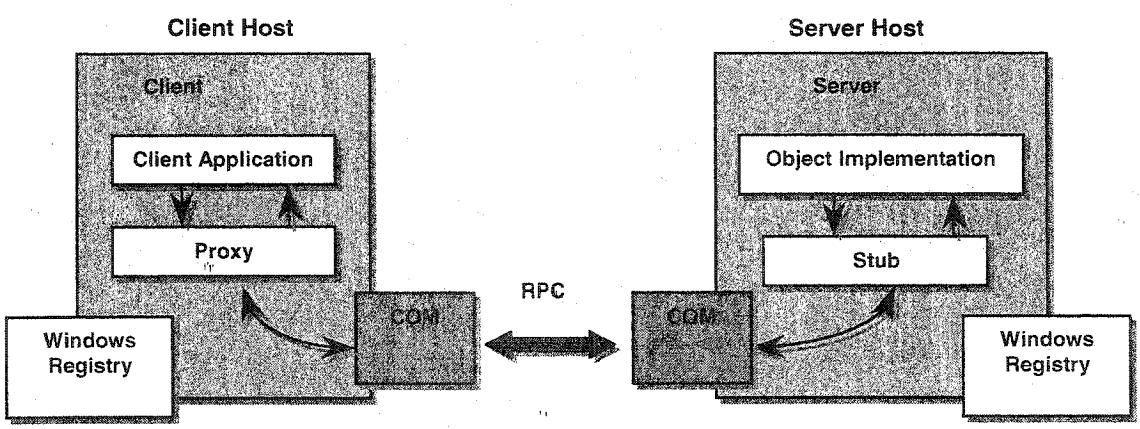


Figure 3-1: DCOM Architecture

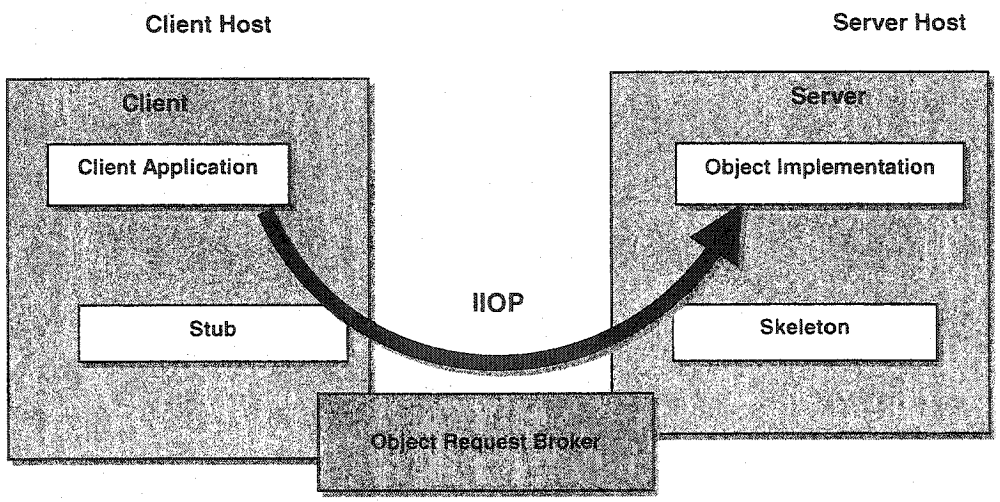


Figure 3-2: DCOM Architecture

3.1.2 COM Component Architecture

COM is the dominant component architecture in use today. It is focused on solving development problem in a desktop environment. The nature of the desktop software development is to create front-end applications with which users interact. COM is primarily a component architecture rather than a remoting architecture. There are three types of COM server:

1. *In-process server*
2. *Local server*
3. *Remote server*

An application that uses a COM component is not required to know what type of server it is using. After a client has obtained a COM object instance handle, client interaction with the COM object instance is the same regardless of server location. This allows for great flexibility when determining how components should be implemented.

The significant weakness of COM as the ideal remoting solution is the platform limitations. COM is primarily based on Windows operating system, with limited support on UNIX and mainframe platforms.

3.1.3 COM and ActiveX

COM is a very mature component architecture that has much strengths. One of the great strengths is in building up ActiveX controls, which are in-process COM components. ActiveX controls, formerly known as OLE controls, are reusable software components that can quickly add specialized functionality to desktop applications, Web sites and development tools. ActiveX controls[36] have become the primary architecture for developing programmable software components for use in a variety of different containers, ranging from software development tools to user productivity tools.

From the application aspects, an ActiveX control is a COM-based object that can draw itself in its own window, respond to events (such as mouse clicks), and be managed through an interface that includes properties and methods similar to those in Automation objects. These controls can be developed for many uses, such as database access, data monitoring, or graphing. In addition, an ActiveX control fully supports Automation, which allows the control to expose writable properties and a set of methods that can be called by the control user.

Huge numbers of third-party ActiveX controls are available in the market that can be used to quickly create sophisticated end-user application in a wide range of client environments.

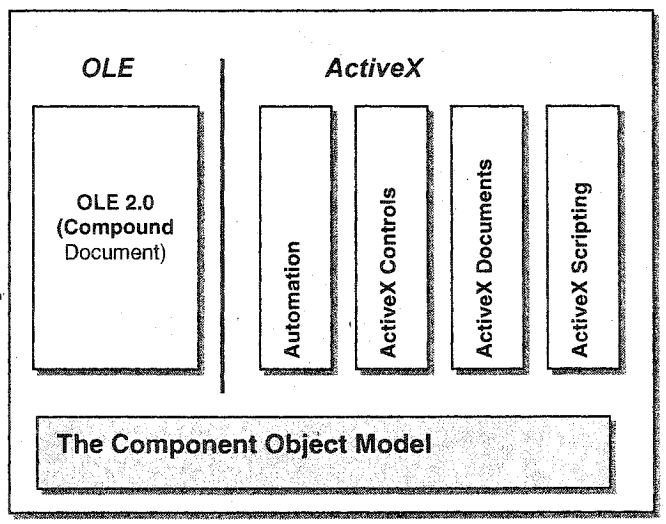


Figure 3-3: COM and ActiveX Controls

Returning to the history of the evolution of COM, Microsoft Object Linking and Embedding (OLE 2.0) was the first technology from Microsoft to be based on COM. OLE was to enable application integration at the compound-document level. For example, a user would be able to embed or link a spreadsheet into a word-processor document. Since OLE was introduced, Microsoft has released a number of additional technologies based on COM. These technologies include OLE Automation and OLE Controls. However, the use of the term OLE in the names OLE Automation and OLE Controls was not quite accurate because these technologies had nothing to do with

linking and embedding. In an attempt to resolve this confusion, Microsoft replaced the term OLE with the new term ActiveX. The only technologies that kept the name OLE were those that actually related to linking and embedding. The following illustration shows the difference between ActiveX and OLE, each of which is built on COM.

3.1.4 CORBA Remoting Architecture

CORBA is the dominant remoting architecture in use today. The CORBA specification defines the foundations of the OMG's object Management Architecture. CORBA provides robust cross-language, cross-platform, and cross-vendor support for creating servers on a wide range of operating system platforms. CORBA is an open, standard solution for distributed object systems. CORBA was intended from the beginning to create a standard for remote method invocation.

The use of CORBA allows for considerable versatility when implementing distributed system. CORBA solutions are available for every common environment and are used to integrate applications written in C, C++, Java, Ada, Smalltalk, and COBOL, running on embedded systems, PCs, UNIX hosts, and mainframes. CORBA objects running in these environments can cooperate seamlessly.

CORBA currently lacks a component model and advanced tool support. Although the CORBA 3.0 specification contains a CORBA Component Model (CCM), it will take time for such a model to appear in various commercial implementations, and to become mature and stabilized in the market.

3.2 CORBA, COM Interface Definition Languages and Data Types

Both COM and CORBA allow for a rich set of data types. This includes support for constants, enumerated types, structures, and arrays in addition to common fundamental types like long and short.

Interface Definition Language (IDL) was originally part of the Open Software Foundation's Distributed Computing Environment (DCE). It described function interfaces for Remote Procedure Calls (RPCs), so that a compiler could generate proxy and stub code that marshaled parameters between machines.

An IDL interface definition typically has the following components:

- Operation definitions
- Definitions of Attribute in CORBA or Property in COM
- Exception definitions (Not in COM)
- Type definitions
- Constant definitions

MIDL is Microsoft's IDL compiler. In addition, Microsoft developed its own Object Definition Language (ODL), which included the dispinterface keyword for specifying IDispatch's logical interfaces. ODL scripts could be compiled by MKTYPLIB into type libraries (.TLB files), which could be used by Automation clients for early binding. By default, if an interface is declared in the IDL file, MIDL will take the declared functions and generate all of the files for an RPC interface. This includes a client proxy, a server stub, and a header file.

Type libraries (.TLB file) were originally binary descriptions of Automation interfaces and the objects that supported them. However, now they have a more general application. Any object that has been written in a language that can understand the semantics of COM can query a type library for information in a language-independent fashion. Type libraries provide a complete description of an object, including its interfaces, methods, and properties. One can think of a type library as a language-independent header file.

COM IDL is rooted in the data type declaration portion of C and C++. It supports all of the standard C++ data types as well as the data definition keywords. More importantly, IDL's data types and definitions are both language-neutral and platform-neutral.

CORBA defines standard mappings from IDL to several programming languages, including C++, Java, and Smalltalk. Each IDL mapping specifies how an IDL interface corresponds to a language-specific implementation. Orbix's IDL compiler uses these mappings to convert IDL definitions to language-specific definitions that conform to the semantics of that language. In Orbix, both client and server program need to include the common IDL file,

3.3 Proxies, Stubs and Skeletons

The COM and CORBA architectures allow developers to treat distributed objects in much the same manner as native objects. The developer may need to address certain timing and error-handling issues, but the syntax for the method invocation is identical in both the native and remote case.

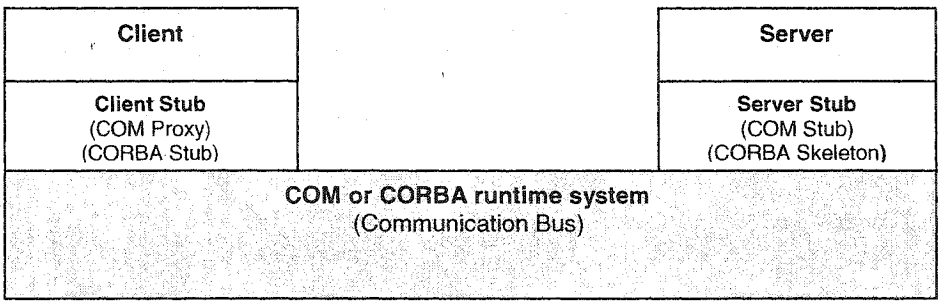


Figure 3-4: Proxies, stubs, and skeleton in COM and CORBA

Both COM and CORBA support location transparency when developing client programs in distributed system. The underlying techniques rely on client-side and server-side mechanisms to manage issues related to remoting. These mechanisms are referred to as *proxies/stubs* in COM and *stubs/skeletons* in CORBA (Figure 3-4). This allows developers to treat distributed objects in much the same manner as native objects. The developer may need to address certain timing and error-handling issues, but the syntax for the method invocation is identical in both the native and

remote case. We will refer to the client-side mechanism as a client stub and the server-side mechanism as a server stub in this paper.

A remote method invocation is implemented as follows:

1) A client invokes a remote method. The remote method is actually invoked in the client stub.
2) The client stub creates a message containing information needed for the remote invocation. (The message creation process is referred to as *marshaling*.)
3) The client stub sends the message to the server stub using the communication bus. (COM or CORBA runtime system.)
4) The server stub receives the message and unpacks it. (The unpacking process is referred to as unmarshalling.)
5) The server stub calls the appropriate server method based on the information provided in the received message.
6) The server stub creates a message based on the outputs of the call to the server method (i.e. the return values and out parameters).
7) The server stub sends the result message to the client stub using the communication bus.
8) The client stub receives the result message, unpacks the message, and return the result to the client.

The client stub, server stub and the runtime system have done the much of the work. The client and server stubs must be created to support the custom interfaces that are used in the system. Hand-coding client and server stubs for every interface would be a tedious and an error-prone task. COM and CORBA solve this problem by providing tools to generate client and server stubs from IDL descriptions.

3.3.1 COM Proxies and stubs

In COM, the proxy and stub are packaged in a single DLL. The DLL is associated with the appropriate interfaces in the windows system registry. The COM runtime system then uses the registry to locate proxy-stub DLLs associated with an interface when marshaling of the interface is required.

Microsoft RPC is a model for programming in a distributed computing environment. The goal of RPC is to provide transparent communication so that the client appears to be directly communicating with the server. Microsoft's implementation of RPC is compatible with the Open Software Foundation (OSF) Distributed Computing Environment (DCE) RPC. The inter-object communication and marshaling detail is as shown in Fig 3-3.

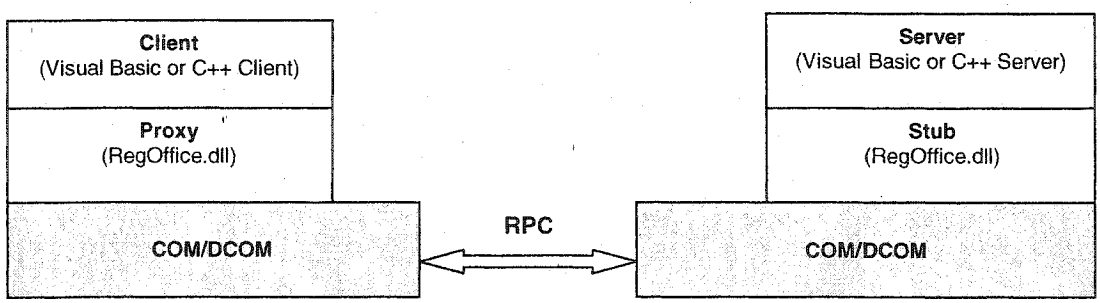


Figure 3-5: COM inter-object communication

The following diagram shows the flow of communication between the components involved. On the client side of the process boundary, the client's method call goes through the proxy and then onto the channel, which is part of the COM library. The channel sends the buffer containing the marshaled parameters to the RPC run-time library, which transmits it across the process boundary. The RPC run time and the COM libraries exist on both sides of the process. The distinction between the channel and the RPC run time is a characteristic of this implementation and is not part of the programming model or the conceptual model for COM client/server objects. COM servers see only the proxy or stub and, indirectly, the channel.

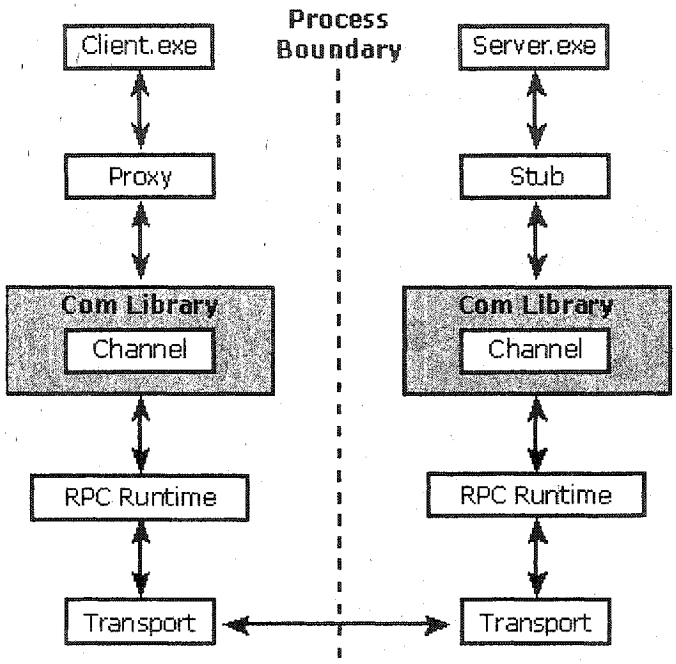


Figure 3-6: Cross-Process communication in COM [2]

3.3.2 CORBA Interfaces and Proxies

When the IDL is compiled, the compiler maps each IDL interface to a client-side proxy class of the same name. Proxy classes implement the client-side call stubs that marshal parameter values and send operation invocations to the correct destination object. When a client invokes on a proxy method that corresponds to an IDL operation, CORBA runtime system conveys the call to the corresponding server object, whether remote or local.

The client application accesses proxy methods only through an object reference. When the client brings an object reference into its address space, the client runtime ORB instantiates a proxy to represent the object. In other words, a proxy acts as a local ambassador for the remote object.

```
//*****************************************************************************
*
//RegOffice.idl
//*****************************************************************************
*

module RegOffice
{

        typedef unsigned long CourseNumber;
        // ...

        interface Registrar
        {
            boolean   CancelCourse(in CourseNumber cnum);
            // ...

        }
}
```

Given this IDL, the IDL compiler generates the following proxy class definition for the client implementation:

```
//*****************************************************************************
*
//RegOffice.h
//*****************************************************************************
*

namespace RegOffice
{
  typedef      CORBA::ULong CourseNumber;
// ...


virtual CORBA::Boolean      CancelCourse( CourseNumber cnum ) = 0;
// ...


}
```

This proxy class demonstrates several characteristics that are true of all proxy classes:

- Member methods derive their names from the corresponding interface operations - in this case, `CancelCourse( )`.
- The proxy class inherits from `CORBA::Object`, so the client can access all the inherited functionality of a CORBA object.
- `RegOffice:: CancelCourse()` and all other member methods are defined as pure virtual, so the client code cannot instantiate the RegOffice proxy class or any other proxy class. Instead, clients can access the RegOffice object only indirectly through object references.

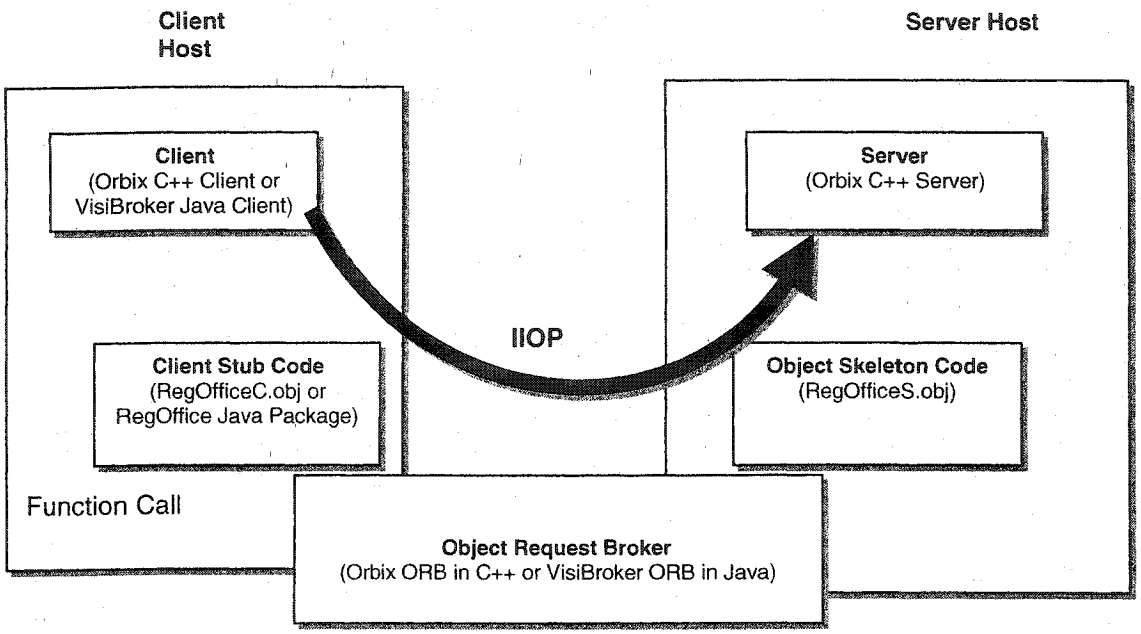


Figure 3-7: Invoking on a CORBA object

OMG IDL language compilers and translators also generate client-side stubs and server-side skeletons. Since they are translated directly from OMG IDL specifications, stubs and skeletons are normally interface-specific. Dispatching through stubs and skeletons is often called *static invocation*. OMG IDL stubs and skeletons are built directly into the client application and the object implementation. Therefore, they both have a complete prior knowledge of the OMG IDL interfaces of the CORBA objects being invoked.

Language mappings usually map operation invocation to the equivalent of a function call in the programming language. For example, given a Factory object reference in C++, the client code to issue a request looks like this:

```
// C++
Factory_var factory_objref;
// Initialize factory_objref using Naming or

// Trading Service (not shown), then issue request

Object_var objref = factory_objref->create();
```

This code makes the invocation of the create operation on the target object appear as a regular C++ member function call. However, what this call is really doing is invoking a stub. Because the stub essentially is a stand-in within the local process for the actual (possibly remote) target object. The stub works directly with the client ORB to *marshal* the request. That is, the stub helps to convert the request from its representation in the programming language to one that is suitable for transmission over the connection to the target object. Once the request arrives at the target object, the server ORB and the skeleton cooperate to *unmarshal* the request (convert it from its transmissible form to a programming language form) and dispatch it to the object. Once the object completes the request, any response is sent back the way it came: through the skeleton, the server ORB, over the connection, and then back through the client ORB and stub, before finally being returned to the client application. Figure 3-7 shows the positions of the stub and skeleton in relation to the client application, the ORB, and the object implementation. This description shows that stubs and skeletons play important roles in connecting the programming language world to the underlying ORB. In this sense they are each a form of the Adapter and Proxy patterns. The stub adapts the function call style of its language mapping to the request invocation mechanism of the ORB. The skeleton adapts the request dispatching mechanism of the ORB to the upcall method form expected by the object implementation.

3.3.3 COM and CORBA Initialization

Initializing the COM library

Any process that uses COM must both initialize and uninitialize COM library. Both Client and Server need to do so. In addition to be a specification, COM also implements some important service in this library. Provided as a set of DLLs and EXEs (e.g. OLE32.DLL and RPCSS.EXE) in Microsoft Windows. The COM library provides:

- A small number of fundamental API functions that facilitate the creation of COM applications, both client and server.

- Implementation-locator services through which COM determines from a unique class identifier which server implements that class and where that server is located.
- Transparent remote procedure calls when an object is running in a local or remote server.
- A standard mechanism to allow an application to control how memory is allocated within its process, particularly memory that needs to be passed between cooperating objects such that it can be freed properly.

To use basic COM services, all COM threads of execution in clients and out-of-process servers must call either CoInitialize or COInitializeEx function before calling any other COM function, and most importantly the threading model is being specified at this time (either apartment-threaded or free-threaded). However, In-process servers do not call the initialization functions, because they are being loaded into a process that has already done so. As a result, in-process servers must set their threading model in the registry under the InprocServer32 key.

```
hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
if(FAILED(hr))
    cout << "CoInitializeEx failed." << endl;
```

Apartment is one of the important concepts of COM threading. An apartment is a conceptual unit that contains one or more threads running in the same process. There are two type of apartment: single-thread apartments (STAs) and multi-threaded apartments (MTAs). MTAs are often referred to as free-threaded apartments. Single threaded apartments only ever contain a single thread. Multi-threaded apartments can contain one or more threads. There can only be zero or one MTAs in a single process. COM calls that are made across apartment boundary need marshaling, and interface pointers from one apartment won't work in another apartment unless they are marshaled first. This marshaling is designed to protect code that has been written with one threading model from being called by code that has been written with an incompatible threading model. This means that the code running in a STA does not need to be written to be thread-safe, the code running in a MTA must be thread-safe.

When an object is run in a STA, access from multiple threads must be synchronized to prevent corruption of the object's state. The developer can feel reassured that the object's state will be protected from multi-threaded access, because no other thread has direct access to the object. All other threads must communicate with the object through a proxy-stub. It's up to COM and the marshaling code to provide synchronization, which is done using the STA's Windows message queue.

UnInitializing the COM library

After using the component, CoUninitialize must be called to close the COM+ library, freeing any resources that it maintains and forcing all RPC connections to close, as shown here:

```
CoUninitialize();
```

Initializing the ORB Runtime

The mechanisms for initializing and shutting down the ORB on a client and a server are the same. Before an application can start any CORBA-related activity, it must initialize the ORB runtime by calling ORB_init(). ORB_init() returns an object reference to the ORB object; this, in turn, lets the client obtain references to other CORBA objects, and make other CORBA-related calls.

C++ Mapping

ORB_init() is defined as follows:

```
namespace CORBA {
// ...
ORB_ptr ORB_init(
int & argc,
char ** aaccv,
const char * orb_identifier = ""
);
// ...
}
```

`ORB_init()` expects a reference to `argc` and a non-constant pointer to `aaccv`. `ORB_init()` scans the passed argument vector for command-line options that start with `-ORB` and removes them.

Shutting Down the ORB

For maximum portability and to ensure against resource leaks, a client or server should always shut down and destroy the ORB at the end of `main()`.

- `shutdown()` stops all server processing, deactivates all POA managers, destroys all POAs, and causes the run() loop to terminate. shutdown() takes a single Boolean argument; if set to true, the call blocks until the shutdown process completes before it returns control to the caller. If set to false, a background thread is created to handle shutdown, and the call returns immediately.
- `destroy()` destroys the ORB object and reclaims all resources associated with it.

3.3.4 Developing a Client

A CORBA client initializes the ORB runtime, handles object references, invokes operations on objects, and handles exceptions that these operations throw.

Before a client application can start any CORBA-related activity, it must initialize the ORB runtime, let the client obtain references to other CORBA objects, and make other CORBA-related calls.

Initialze ORB

```
//**************************************************************************
//Client.cxx
//**************************************************************************
// main() -- the main client program.
//
int main(int argc, char **argv)
{
    int exit_status = 0;
    try
    {
```

```
        CORBA::Object_var tmp_ref;  // For temporary object references.
        // Initialise the ORB.
        // Note: ORB_init will process any -ORB arguments
        // and remove them from argc/argv, so it should
        // be called before any other argument processing.
        //
        global_orb = CORBA::ORB_init(argc, argv);

        // Exercise interface RegOffice::RegistrarFactory.
        //
        tmp_ref = read_reference("c:/temp/RegOffice_RegistrarFactory.ref");
        // ...
}
```

Shut Down ORB

```
//*****************************************************************************
//Client.cxx
//*****************************************************************************
// Ensure that the ORB is properly shutdown and cleaned up.
    //
    try
    {
         global_orb->shutdown(1);
        global_orb->destroy();
    }
```

3.3.5 Developing the Server

```
//*****************************************************************************
//Server.cxx
//*****************************************************************************

// main() -- set up a POA, create and export object references.
//
int main( int argc, char **argv)
{
    int exit_status = 0;              // Return code from main.

    // Variables to hold our servants.
    PortableServer::Servant the_RegOffice_RegistrarFactory = 0;
    try
    {
        CORBA::Object_var tmp_ref;  // For temporary object references.

        // Initialise the ORB and Root POA.
        //
        cout << "Initializing the ORB" << endl;
        global_orb = CORBA::ORB_init(argc, argv);
        // ...
}
```

Shut Down ORB

```
//*****************************************************************************
//Server.cxx
//*****************************************************************************
// Ensure that the ORB is properly shutdown and cleaned up.
    //
```

```
try
{
    global_orb->shutdown(1);
    global_orb->destroy();
}
```

3.4 Object Handles

Object handles are used to reference object instances in a programming language context. To simplify access to distributed objects, object handles referring to COM and CORBA objects need to behave much like their native counterparts. When we use COM and CORBA in a language like Visual Basic, C++, and Java, the syntax for calling an instance method is the same regardless of whether the object instance is native, local, or remote. COM refers to object handles as *interface pointers* where CORBA refers to them as *object references*.

3.4.1 COM Interface Pointer and Reference Counting

When a component object has been created the client receives an interface pointer. This is a pointer to the object's interface, and through this pointer the client can invoke the methods that are described in the object's interface. For all component objects the client gets an interface pointer and using the interface pointer is the only way the client can call the methods of an object.

The client cannot distinguish an in-process object from a local object or from a remote object by examining the pointer. This means that the client programmer treats all objects identically and all requests made to the object's services are made by calling interface member functions. The COM Library provides all the services to transparently make a call, without expecting the programmer to know on which host the object resides.

When a client gets an interface pointer, it has to call a method to tell the component that it has gotten a new user. This is because the component is responsible for keeping track of how many

clients are using it. Later, when a client is finished using a component it calls another method to let the component know it.

3.4.2 CORBA Object References and Reference Counting

When CORBA is used with C++, reference counting directly affects how object references are declared. An object reference type that is appended with _ptr never implicitly affects the reference count of the object that it references. An object reference type that is appended with _var implicitly decrements the reference count of any object it currently references when it's destroyed or assigned to a new object.

The RegOffice client declares TouchScreen as a _var type so that it will automatic released when the TouchScreen session is finished.

```
RegOffice::TouchScreen_var      touchscreen;
touchscreen = Factory->CreateTouchScreen(sid, "dummy");
touchscreen ->Init(sid);
```

CORBA C++ mapping provides the CORBArelease (ptr) function to explicitly decrement the reference count of a CORBA object referenced by ptr. To release the returned pointer, the caller can explicitly call release() or assign the pointer to a _var type.

The CORBA C++ mapping specifies a static _duplicate() method to be used for incrementing the reference count.

When using Java to implement CORBA, there is an important advantage compared to C++ —the garbage collection. Java-based CORBA products rely on the Java runtime system to manage CORBA object reference counting rather than forcing the user to correctly use explicit constructs like those used in C++.

3.4.3 Creating Objects

Creating object instances in languages like C++, Java, and Visual Basic is a simple process: One simply needs to use the `new` operator. In comparison, creating a distributed object instance requires more effort since the object instance is usually created in a different process on a different machine. An abstraction is needed to redirect creation of a distributed object to a remote location.

COM and CORBA both rely on an abstraction called a *factory* to create distributed object instances. A factory is a special type of distributed object whose main purpose is to create other distributed objects. A factory lives within the same server process as the objects that it creates. First, the appropriate factory is located. Then, the factory is used to create the object of interest.

3.4.3.1 COM Class Factory

COM defines a standard interface for factories called IclassFactory.

```
IClassFactory Interface

interface IClassFactory: IUnknown {
     HRESULT __stdcall CreateInstance (IUnknown *pOuterUnk, const IID& iid,
void ** ppv);
     HRESULT __stdcall LockServer (BOOL bLock);

};
```

A typical COM server implements the IClassFactory interface, thereby allowing COM object instances to be created. The creation process for COM objects that rely on IClassFactory is always the same.

- Use the COM object Class ID (CLSID) to obtain and IClassFactory interface pointer to the correct factory.
- Call the IClassFactory::CreateInstance() method to create the COM object instance.
- After the COM object is created, an interface pointer to the newly created object instance is returned, and the factory interface pointer is discarded.

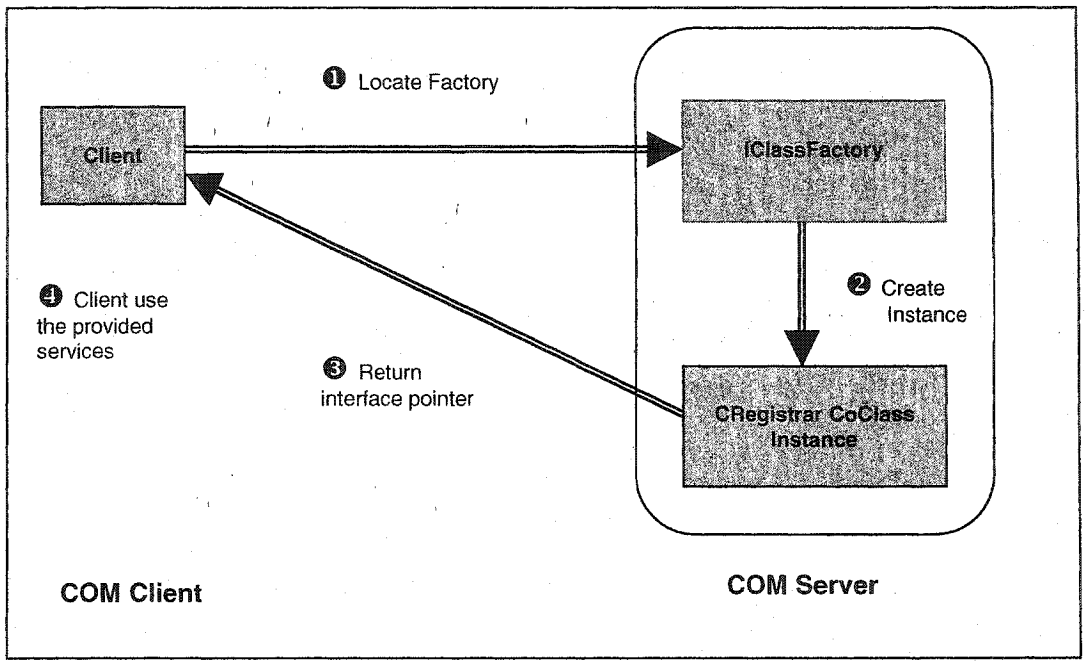


Figure 3-8: Creating COM object

CoCreateInstance is a helper function which creates a class object associated with the class identifier (passed as an argument to CoCreateInstance function), creates an instance of that class identifier and then releases the class object of that specific class identifier. All the components are created with a class factory. CoCreateInstance also uses a class factory behind the scene to create a component.

Explicit use of the factory is required for bulk creation of COM object instance. To make creation of many instances efficient, one needs to first get an interface pointer to the factory and to then create multiple COM object instances using the same factory interface pointer. This eliminates the need to locate a factory for each instance that is created.

3.4.3.2 CORBA Factories

The CORBA notion of factories is somewhat different from that of COM. CORBA does not specify a standard implementation for factories. Instead, CORBA provides for persistent objects (i.e.,

objects that can live beyond the lifetime of a single process). A persistent object provides a useful mechanism for creating a factory. The CORBA clients rely on a stringified interoperable object reference (IOR). A stringified IOR is simply a string of characters that can be used to uniquely identify a CORBA object instance available on the network regardless of vendor or hardware platform.

```
// Create a servant for interface RegOffice::RegistrarFactory.
//
the_RegOffice_RegistrarFactory =
    RegOffice_RegistrarFactoryImpl::_create(my_poa);
oid = my_poa->activate_object(the_RegOffice_RegistrarFactory);
tmp_ref = my_poa->id_to_reference(oid);
write_reference(tmp_ref,"c:/temp/RegOffice_RegistrarFactory.ref");

// Activate the POA Manager and let the ORB process requests.
//
root_poa_manager->activate();
cout << "Waiting for requests..." << endl;
global_orb->run();
```

3.4.4 Destroying Objects

COM and CORBA have very different approaches for destroying distributed object instances. Although both COM and CORBA use reference counting to determine when an object is no longer in use with in a specific process, the similarities end there.

COM supports distributed reference counting and garbage collection where by a server object is destroyed when there are no longer any client referencing it. Supporting such a mechanism is not a simple task. Many issues related to efficiency and reliability must be addressed. COM's built-in management of object destruction is an extremely useful and important feature.

CORBA takes the stance that a server object's reference count should not be affected by the uncontrollable actions of an arbitrary client. In CORBA, server-side reference counts are maintained separately and have no direct relationship to client- side reference counts. The reference count maintained in a CORBA server for a specific instance can be manipulated only

within the context of the server. This means that the responsibility for releasing all references to a server object requires a customized solution rather than a standardized approach.

3.5 Exception and Error Handling

3.5.1 CORBA Exceptions

CORBA specifies an extensible exception capability that maps naturally into languages that have native exceptions, like C++ and Java, and that maps into exception data in languages that do not. It is based on user-defined exception types in CORBA IDL. In practice, this mechanism works well.

An CORBA IDL operation can throw two types of exceptions:

- User-defined exceptions are defined explicitly in your IDL definitions.
- System exceptions are predefined exceptions that all operations can throw.

While IDL operations can throw user-defined and system exceptions, accessor methods for IDL attributes can only throw system-defined exceptions. For User-Defined Exceptions, operations are defined to raise one or more user exceptions to indicate application-specific error conditions. An exception definition can contain multiple data members to convey specific information about the error, if desired. The C++ Mappings for User Exceptions are translated into C++ classes by IDL compiler. In our sample application, the exception ExceedMaxNumOfSeats in idl:

```
interface Registrar
{
        exception ExceedMaxNumOfSeats{};
... ...
};
```

maps to C++ class:

```
class ExceedMaxNumOfSeats: public CORBA::UserException
{
          public:
```

```
            ExceedMaxNumOfSeats();
            void operator=( const ExceedMaxNumOfSeats& );
            static ExceedMaxNumOfSeats*  _downcast( CORBA::Exception* exc  );
            static const ExceedMaxNumOfSeats*  _downcast( const CORBA::Exception*
exc  );
            … …

            virtual void  _raise() const;
            … …

            virtual ~ExceedMaxNumOfSeats();
  };
```

Client code uses standard try and catch blocks to isolate processing logic from exception handling code. If an operation might throw a user exception, its caller should be prepared to handle that exception with an appropriate catch clause. A client often provides a handler for a limited set of anticipated system exceptions. It also must provide a way to handle all other unanticipated system exceptions that might occur. The following code shows how the user defined and system exceptions are handled.

```
try
{
      blnok = registrar->RegisterCourse(cnum, blnmore);
      if(!blnok)
      {
            cerr<<" >>  Failed to register this course. " <<endl;
            cerr<<" >>     1) Invalid course number or"<<endl;
            cerr<<" >>     2) Course already registered or"<<endl;
            cerr<<" >>     3) Exceed the limit of the maxium number of
                              courses allowed"<<endl;
            cerr<<endl;

      }
      else
      {
            cout<<endl;
            cout<<"  >>  Course " <<cnum<<" added in you course list."<<endl;

            if (!blnmore)
            {
                  cout<<endl;
                  cout<<"  >>  You have reached the limit of maxium number of
                               courses allowed, no more course
                               can be added."<<endl;
                  cout<<endl;
            }

      }

}
catch (const RegOffice::Registrar::ExceedMaxNumOfSeats& m)
{
      cerr<<endl;
```

```
        cerr<<"  >> The course is not added, there is no room left
                    for this course. "<<endl;
        cerr<<endl;
}
catch (CORBA::SystemException &ex)
{
        cerr << " system exception." << endl;
        cerr << " " << ex << endl;
}
catch (...)
{
        cerr << " Unspecified error." << endl;
}
```

CORBA system exception can also be evaluated. In Orbix, this is done by two member methods, `complete()` and `minor()`.

3.5.2 COM Exceptions

In COM, the exceptions are not defined in the IDL file, instead COM use HRESULTs to transfer error information. Almost all COM functions and interface methods return a value of the type HRESULT. The HRESULT (for *result handle*) is a way of returning success, warning, and error values. HRESULTs are really not handles to anything; they are only 32-bit values with several fields encoded in the value. A zero result indicates success, and a nonzero result indicates failure. On 16-bit platforms, an HRESULT is generated from a 32-bit value known as a status code, or SCODE. SCODEs are divided into four fields: a severity code, a context field, a facility field, and an error code. The following illustration shows the format of an SCODE on a 16-bit platform; the numbers indicate bit positions:

```
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//  +---+-+-+-----------------------+-------------------------------+
//  |Sev|C|R|     Facility          |               Code            |
//  +---+-+-+-----------------------+-------------------------------+
//
//  where
//
//      Sev - is the severity code
//
//          00 - Success
//          01 - Informational
//          10 - Warning
//          11 - Error
//
//      C - is the Customer code flag
//
//      R - is a reserved bit
```

```
//
//       Facility - is the facility code
//
//       Code - is the facility's status code
```

The reason why COM uses this error handling strategy is because interface methods are virtual, it is not possible for a caller to know the full set of values that may be returned from any one call. One implementation of a method may return five values; another may return eight. In Visual C++ extension of COM, a _com_error object represents an exception condition. It's used to handle smart pointer exceptions. It can be detected by the error-handling wrapper functions in the header files that have been generated by the type library or by one of the COM support classes. The _com_error class encapsulates the HRESULT error code and any associated contextual information provided by the IErrorInfo interface. e.g.

```
HRESULT rc;
try
{
    IMultiObjectPtr pMulti;
    IStreamPtr pStream;
    IPersistFilePtr pFile;

    pMulti = IMultiObjectPtr(CLSID_MultiObject) ;

    rc = pMulti->QueryInterface(IID_IPersistFile,
        (void **) &pFile);
    if (SUCCEEDED(rc))
        MessageBox("The object supports IPersistFile.");

    rc = pMulti->QueryInterface(IID_IStream,
        (void **) &pStream);
    if (SUCCEEDED(rc))
        MessageBox("The object supports IStream.");
}
catch (_com_error & ex)
{
    _bstr_t bstrSource(ex.Source());
    _bstr_t bstrDescription(ex.Description());
    char szTemp[1024];
    CString csMsg = "Error!\n";
    wsprintf(szTemp, "Code = %08lx\n", ex.Error());
    csMsg += szTemp;
    wsprintf(szTemp, "Code meaning = %s\n", ex.ErrorMessage());
    csMsg += szTemp;
    wsprintf(szTemp, "Source = %S\n", (wchar_t*)bstrSource);
    csMsg += szTemp;
    wsprintf(szTemp, "Description = %S\n", (wchar_t*)bstrDescription);
    csMsg += szTemp;
    MessageBox (csMsg);
    … …
}
```

For VB client program, the way to handle error and exceptions is like the following code segment.

```
Public Function StartRegistrar() as Boolean

      on error goto GeneralErrHandler
      Call mobjRegistrar.Init();
      StartRegistrar = true
      Exit Function
   GeneralErrHandler:
      Dim strErr as String
      StrErr = "Error Description:" &  Err.Description &_
               "Error Number: " & Err.Number

      msgbox StrErr

End Function
```

In real applications, the HRESULTs may not be enough to encode all necessary error information, the programs have their own flexibility to handle exceptions based on the programming languages used.

3.6 Microsoft Transaction Server vs. Enterprise JavaBeans

Microsoft Transaction Server (MTS) is the transaction service in the Windows NT operating system. It was first available in 1996. Enterprise JavaBeans (EJB) is a specification for a Java-based transaction service. Created by a group of companies led by Sun Microsystems Inc., the initial specification for EJB was released in spring 1998. While EJB originally stood on its own as a spec, since Dec 1999 it has been rolled into J2EE, the family of specs owned by Sun and JCP. Both Microsoft Transaction Server and Enterprise JavaBeans target the creation of component-based, transaction-oriented applications.

Microsoft Transaction Server is based on the COM. COM+ is the next evolution of COM and Microsoft Transaction Server. COM+ is intended to provide a model that makes it relatively easy to create business applications that work well with MTS. MTS is the middleware component model for Windows, is used for creating scalable, transactional, multi-user and secure enterprise-level server side components. MTS, through the COM infrastructure, handles communication between components using DCOM. This allows MTS to expose its components to Windows

applications from anywhere on the net or the web. Provided that the system is running under Windows, MTS components can be deployed on top of existing transaction processing systems including web servers, database servers, application servers, and traditional transaction processing monitors. MTS is a stateless component model, whose components are always packaged as an in-process DLL. Any in process DLL can be easily imported to MTS without any code change. This can be done through Windows Component Services (previously called MTS Explorer in Windows 98). Windows Component Services allows a system administrator to specify which components on a server will be available through MTS, and the options that will be used (Figure 3-9). The properties of a MTS component can be set through the properties page in Component Services (as shown in Figure 3-10). Since MTS components are composed of COM objects, they can be implemented in a variety of different languages including C++, Java, Visual Basic.

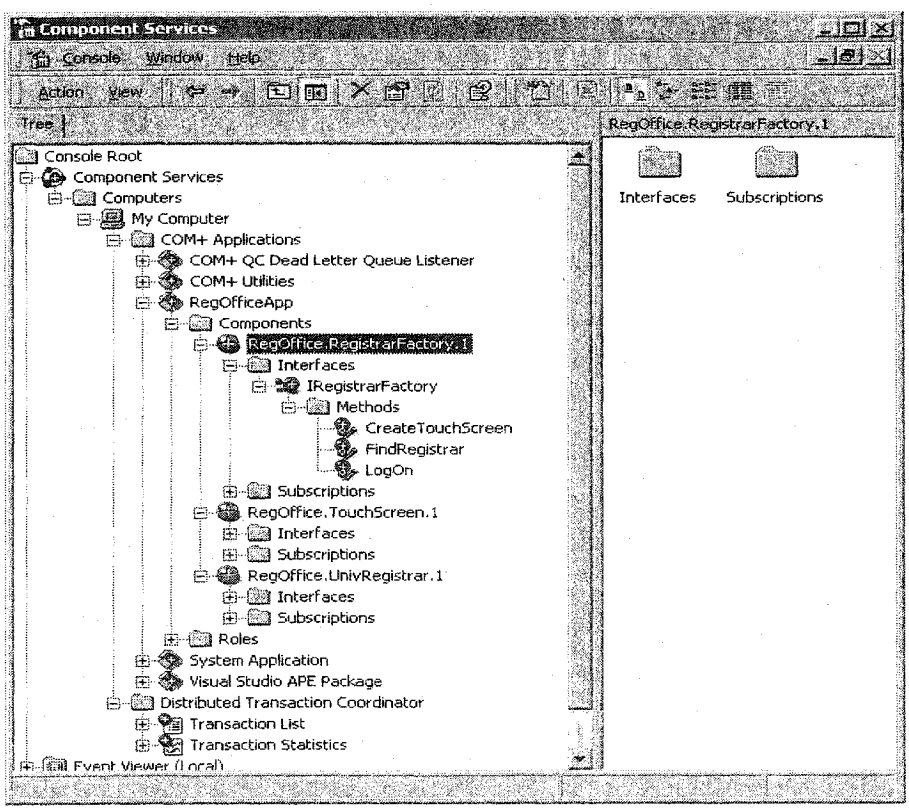


Figure 3-9: A snapshot of Windows Component Services Console

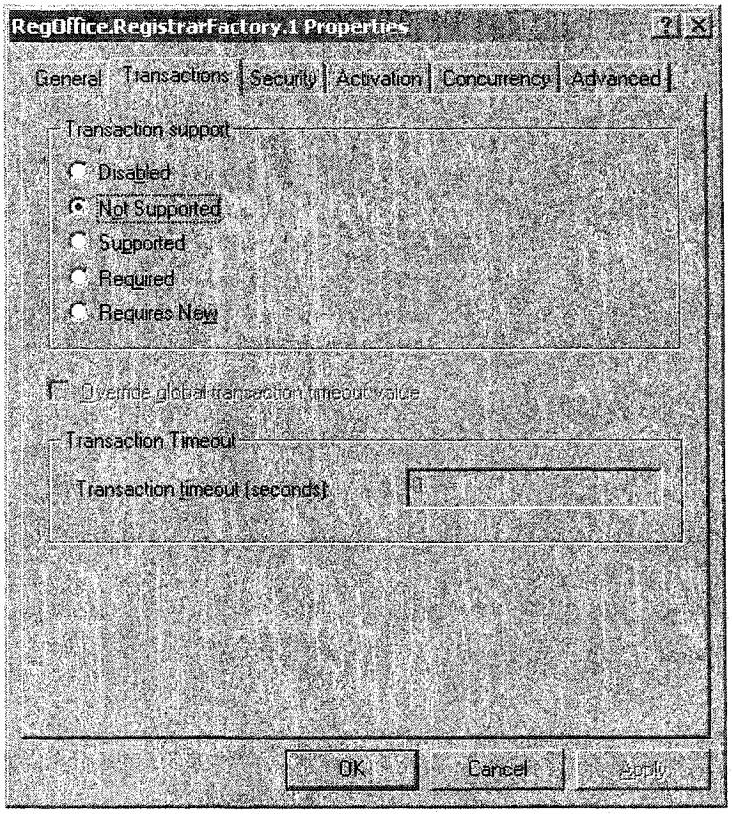


Figure 3-10: Setting the properties of RegistrarFactory component

Enterprise JavaBeans is a middleware component model for Java and CORBA. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology. Most importantly, EJBs can be deployed on top of existing transaction processing systems including web servers, database servers, application servers, and traditional transaction processing monitors. Since EJB components are written in Java, EJBs containing the business logic are platform-independent. EJBs enable developer to work with off-the-shelf components from one vendor, combine them with components from another vendor, and run these components in an application server written by yet another vendor. By using EJB, the developer can write scalable, reliable, and secure applications without having to develop the complex distributed object framework. EJB provides both a stateless and a stateful model. Since EJB is built on top of Java technology, EJB components can only be implemented using the Java Language.

3.6.1 Just In Time Activation and Instance Pooling

In MTS, the actual MTS component is not created until the call from a client reaches the container. Since the component is not running all the time, it does not use up a lot of system resources. As soon as the call comes in from the client, the MTS wrapper process (DLLHost) activates its Instance Management algorithm called JITA. The actual MTS component is created "Just In Time" to service the request from the wrapper. And when the request is serviced and the reply is sent back to the client. When the client calls Release() on the component, the actual MTS component is destroyed. MTS is a stateless component model.

Object Pooling is a service that was documented in MTS but is fully supported only in COM+. The MTS Glossary defines Pooling as "A performance optimization based on using collections of pre-allocated resources, such as objects or database connections." It is also mentioned that pooling results in more efficient resource allocation.

In EJB, the EJB container is responsible for controlling the life cycle of the deployed enterprise bean components. As bean client requests arrive, the EJB container dynamically instantiates, destroys, and reuses beans as appropriate. If a client requests a certain type of bean that does not exist in the memory yet, the EJB container may instantiate a new in-memory instance on behalf of the client. On the other hand, if a bean already exists in the memory, it may not be appropriate to instantiate a new bean. Instead, the EJB container reassigns a bean from one client to another. This is called instance pooling. EJB may or may not be stateless between calls from the client, while MTS is.

3.6.2 Location Transparency

Both MTS and EJB application components are location transparent. It doesn't matter where a component is —on the local machine or a remote machine, in-process or out of process and this

is a feature of the underlying transport mechanism. If this knowledge is unavailable to the client, it makes it possible for a consistent approach to interacting with components. The same exact components can be deployed so that all users can share them. This means that only one set of code needs to be supported for multiple configurations. This is an essential part of reusable components that can be deployed in a wide variety of multi-tier situations. COM and DCOM support MTS components location transparency, since they are all COM components. EJB containers use the Java Remote Method Invocation (RMI) interfaces to specify remote accessibility. Java RMI enables the location transparency of EJB components.

3.6.3 Transaction Support

MTS uses automatic transactions, it allows clients to remain unaware of when transactions begin and end—they never need to make explicit calls to begin or end a transaction. Instead, when a transaction begins depends on the value of that component's transaction attribute. This value can be set to one of four possibilities (also as shown in Figure 3-10):

- ***Not Supported*** - The method does not support transactions.
- ***Supported*** - The method can support a transaction, but it doesn't require one.
- ***Required*** - The method requires a transaction. If the caller doesn't provide one, it will create one.
- ***Requires New*** - The method always requires a new transaction, even if the caller provides one.

Component developers are only required to put two transactional commands into each component. SET COMPLETE indicates that everything is fine. SET ABORT indicates that an error has occurred and the transaction should be rolled back. The default behavior is to commit the transaction. These calls are made by a method in the MTS object itself, not by the client, the client need not be aware that transactions are being used.

EJB supports transactions in a very similar fashion, although it offers two additional options:

- ***TX_NOT_SUPPORTED*** — The method does not support transactions.
- ***TX_SUPPORTS*** — The method can support a transaction, but it doesn't require one.
- ***TX_REQUIRED*** — The method requires a transaction. If the caller doesn't provide one, it will create one.
- ***TX_REQUIRES_NEW*** — The method always requires a new transaction, even if the caller provides one.
- ***TX_MANDATORY*** — The method requires that the caller provide a transaction.
- ***TX_BEAN_MANAGED*** — The bean manages its own transactions, specifying its own transaction demarcation.

EJB also specifies the database transaction isolation level *(read committed, read uncommitted, repeatable read,* or *serializable)* using attributes. EJB transactions are more versatile than MTS. However, flexibility of EJB transaction support also increases complexities and thus can introduce confusion in complex scenarios when client-managed transactions are mixed with container-managed transaction.

MTS also supports client-managed transactions, allowing a client to determine explicitly when a transaction should begin and end. But unlike EJB, MTS builds this service on top of automatic transactions rather than introducing an entirely separate mechanism. This eliminates the possibility of confusion between automatic transactions and client-managed transactions. An MTS object with its transaction attribute set to *Not Supported* can make direct calls on the DTC using OLE transactions. By doing this, an MTS object can demarcate transactions on its own, just like an EJB bean-managed transaction.

3.6.4 Database Connection Pooling

In MTS, database connection pooling is built into the ODBC Resource Dispenser. Any application using ODBC and running on Windows can make use of this facility to reuse database connections. MTS uses ODBC for database access and hence, reuses database connections by making use of this facility built in Windows. Without this database connection pooling, the Just In time Activation (JITA) algorithm used by MTS for Instance Management will not work efficiently. The initial creation of the database connection will be slow, but this happens only once in a while. Object Pooling is a service that was documented in MTS but is fully supported only in COM+.

EJB uses JDBC for database access. Support for Database connection pooling is not available in JDBC until version 2.0. EJB vendors may either need to get JDBC 2.0 compliant or else will be forced to implement their own proprietary database connection pooling algorithms in their products. Although the EJB specification does not formally prescribe it, almost all EJB vendors provide JDBC connection pooling, and this became standardized at JDBC 2.0. And it's now standardized in EJB Connector API standard.

3.6.5 Component Type

MTS components are essentially stateless components, which means objects lose state on transaction boundaries. Although there are many benefits to using stateless MTS components, there are cases where holding state is desirable. The object state can be held across transactions by using the Shared Property Manager or store the state in a database.

EJB components are of two major types—Session Beans and Entity Beans. Session beans represent a business process, whereas entity beans represent permanent business data. Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

A session bean is not persisted across multiple sessions. There are two types of session beans: stateful and stateless. A stateless session bean does not maintain a conversational state for each individual client. Each invocation of a stateless session bean should be considered as a request to a brand new object instance, since any instance-variable state will be lost between invocations. Stateful Session Beans maintain state within the component.

Entity Beans represent persistent data stored in the database. They are stateful and transactional. They are typically shared between several clients.

3.6.6 **Interoperability**

MTS components can be invoked from other machines by clients using DCOM. For communication through HTTP, clients can always communicate with MTS servers through the Microsoft Internet Information Server. DCOM servers are available only for Windows platforms.

EJBs provide platform interoperability and scalability, but you are limited to a single implementation language: Java. Clients from other machines can invoke EJB components using RMI/IIOP protocols. EJB uses CORBA as a backbone but lets Java programmers develop distributed objects without the knowledge base required for CORBA. EJB spec allows the EJB server vendors to use any communication protocol between client and server. Due to these factors, there is system-level and application-level interoperability between products from vendors who choose to implement the EJB/CORBA protocol as the underlying communication protocol. Java Clients using IIOP can go in for Java RMI or CORBA IDL Java mapping. Non-Java clients use IIOP with specific language mapping.

3.7 Building Sample Application in COM

Microsoft Visual Studio Visual C++ 6, Active Temple Library 3.0 and Visual Basic 6 are used to build the Registrar Office sample application in this thesis.

3.7.1 Description of the Sample Project

The sample application provides a user interface for student to interact with the university's course registration system. In the application, the student can browser list of current available courses, course schedules, and related course information. The student can register or cancel courses. The course registration is based on first come first service. The system will ensure that the maximum number of student allowed will not be exceeded, and the student must not re-register any course, which he's previously taken. The results are saved when the student completes the session, and the information can be retrieved and updated at a later time.

The system consists of three logical layers:

- User interface—This is the client layer of the application, which provides a GUI to the user. It is written in Visual Basic 6.
- Business Logic Layer— This layer is responsible of implementing the business logic of the course registration system. It contains three components: Touch Screen, Registrar Factory and Registrar. MDBConnection is created as a special component to handle the data access. The components are implemented as COM in-process servers.
- Database layer—It provides a persistent data storage for system. All the course information, course schedule, course registration, and student information are stored here. Microsoft Access DB is used as the system database.

The IDL describes all the interfaces and methods (services) provided by the servers:

```
// RegOffice.idl : IDL source for RegOffice.dll
//

// This file will be processed by the MIDL tool to
```

```
// produce the type library (RegOffice.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";


typedef long StudentId ;
typedef long CourseId ;

        [
                object,
                uuid(F32B7917-5AA1-4560-A468-8E26FB6A00A3),
                dual,
                helpstring("ITouchScreen Interface"),
                pointer_default(unique)
        ]
        interface ITouchScreen : IDispatch
        {
                [id(1), helpstring("method Init")]
                        HRESULT Init([in] StudentId sid);
                [id(2), helpstring("method GetCourseDetail")]
                        HRESULT GetCourseDetail([in] CourseId cid,
                        [out, retval] VARIANT* CourseDetail);
                [id(3), helpstring("method GetAllAvailableCourses")]
                        HRESULT GetAllAvailableCourses([out] VARIANT *
                        vtCourseNumList);
                [id(4), helpstring("method GetCourseTaken")]
                        HRESULT GetCourseTaken([out] VARIANT * vtCourseNumList);
                [id(5), helpstring("method GetStudentCurrentStatus")]
                        HRESULT GetStudentCurrentStatus([out] BSTR * bstrStatus);
                [id(6), helpstring("method GetStudentDetail")]
                        HRESULT GetStudentDetail([out] VARIANT * vtStudentDetail);
                [id(7), helpstring("method Initialize")]
                        HRESULT Initialize([in] StudentId sid,
                        [in] IUnknown* pIUnk, [out, retval] VARIANT_BOOL *bok);
                [id(8), helpstring("method GetStudentInfo")]
                        HRESULT GetStudentInfo([out] long* plsid,
                        [out, retval] VARIANT* pvtInfo);
        };
        [
                object,
                uuid(D52FC21C-EBB0-44DF-9B4C-193F18CF71FB),
                dual,
                helpstring("IUnivRegistrar Interface"),
                pointer_default(unique)
        ]
        interface IUnivRegistrar : IDispatch
        {
                [id(1), helpstring("method Init")]
                HRESULT Init([in] StudentId sid,
                [out, retval] VARIANT_BOOL * bok);

                [propget, id(2), helpstring("property RegOfficeAddress")]
                HRESULT RegOfficeAddress([out, retval] BSTR *pVal);

                [propput, id(2), helpstring("property RegOfficeAddress")]
                HRESULT RegOfficeAddress([in] BSTR newVal);

                [id(3), helpstring("method GetRegisteredCourses")]
                HRESULT GetRegisteredCourses([out] VARIANT * vtCourseList);

                [id(4), helpstring("method RegisterCourse")]
                HRESULT RegisterCourse([in] CourseId cid,
```

```
            [out] VARIANT_BOOL * bMoreCourseAllowed,
            [out, retval] VARIANT_BOOL *  bok);

            [id(5), helpstring("method CancelCourse")]
            HRESULT CancelCourse([in] CourseId cid,
            [out, retval] VARIANT_BOOL * bok);

            [id(6), helpstring("method CompleteSession")]
            HRESULT CompleteSession([in] VARIANT_BOOL bCommit_trans,
            [out, retval] VARIANT_BOOL * bok);

      };
      [
            object,
            uuid(85F7B7C6-6BFF-4A83-B058-D60EA0EEF6BC),
            dual,
            helpstring("IRegistrarFactory Interface"),
            pointer_default(unique)
      ]
      interface IRegistrarFactory : IDispatch
      {
            [id(1), helpstring("method LogOn")]
            HRESULT LogOn([in] StudentId sid, [in] BSTR bstrPassword,
            [out, retval] VARIANT_BOOL* bok);

            [id(2), helpstring("method FindRegistrar")]
            HRESULT FindRegistrar([in] StudentId sid,
            [in] BSTR bstrPassword,
            [out, retval] IUnivRegistrar ** ppIUnivRegistrar);

            [id(3), helpstring("method CreateTouchScreen")]
            HRESULT CreateTouchScreen([in] StudentId,
            [in] BSTR bstrPassword,
            [out, retval] ITouchScreen ** ppITouchScreen);
      };

[
      uuid(67AC26A6-0AE6-466F-8344-4E492BF2F5FF),
      version(1.0),
      helpstring("RegOffice 1.0 Type Library")
]


//////////////////////////////////////////////////////////////////////////////
/

library REGOFFICELib
{
      importlib("stdole32.tlb");
      importlib("stdole2.tlb");

      [
            uuid(BC4A2448-6911-477B-93AA-AE294B0C5F23),
            helpstring("TouchScreen Class")
      ]
      coclass TouchScreen
      {
            [default] interface ITouchScreen;
      };
      [
            uuid(84CF4D8D-A5CC-4890-93FA-C5D986682881),
            helpstring("UnivRegistrar Class")
      ]
```

```
        coclass UnivRegistrar
        {
                [default] interface IUnivRegistrar;
        };
        [
                uuid(294DCA3B-2BF8-4078-A59C-3F87624D073D),
                helpstring("RegistrarFactory Class")
        ]
        coclass RegistrarFactory
        {
                [default] interface IRegistrarFactory;
        };
};
```

3.7.1.1 The Database Connection component

MDBConnection component is created as a data access tool. It uses Microsoft ActiveX Data Object (ADO) to access data. It provides basic functions to query and update database. It can also explicitly begin database transactions, and complete transactions by either commit or rollback. In the sample application, the database transaction will begin when the user starts adding or canceling the courses. When the user completes his course registration session, all the changes will be reflected in the database by committing the transaction. If the student chooses not to save the changes, the transaction can be rolled back, and nothing has been updated in the database.

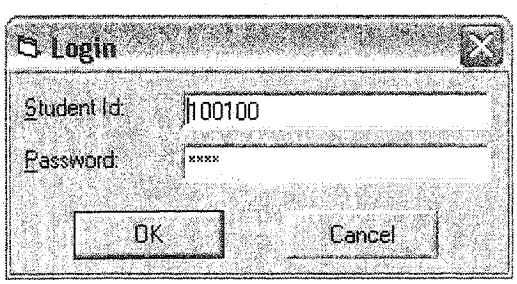
```
// MDBConnection.idl : IDL source for MDBConnection.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (MDBConnection.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
        [
                object,
                uuid(2C37ADB9-BBE5-4EE2-A5BC-D9683A0F8C6F),
                dual,
                helpstring("IMDBCon Interface"),
                pointer_default(unique)
        ]
        interface IMDBCon : IDispatch
        {
                [id(1), helpstring("method Query")]
                HRESULT Query([in] BSTR bstrSQL, [in] short nRetType,
                [out, retval] VARIANT * vtResult);

                [id(2), helpstring("method Execute")]
                HRESULT Execute([in] BSTR strSQL);
```

```
        [id(3), helpstring("method Init")]
        HRESULT Init([in] BSTR strDBSrc, [in] short nDBType,
        [out, retval] VARIANT_BOOL * bOK );

        [id(4), helpstring("method StartTrans")]
        HRESULT StartTrans();

        [id(5), helpstring("method Complete")]
        HRESULT Complete([in] VARIANT_BOOL bCommit);

        [id(6), helpstring("method PutConnectionTimeout")]
        HRESULT PutConnectionTimeout([in] long lConnectTimeOut);

        [id(7), helpstring("method Query2")]
        HRESULT Query2([in] BSTR bstrSQL, [in] short nRetType,
        [out, retval] VARIANT * pvtResult);
    };

[
    uuid(754996EF-8F93-424B-B108-D887E3824F2B),
    version(1.0),
    helpstring("MDBConnection 1.0 Type Library")
]
library MDBCONNECTIONLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(9E9336D6-7B2C-4600-A25C-5970B1871B82),
        helpstring("MDBCon Class")
    ]
    coclass MDBCon
    {
        [default] interface IMDBCon;
    };
};
```

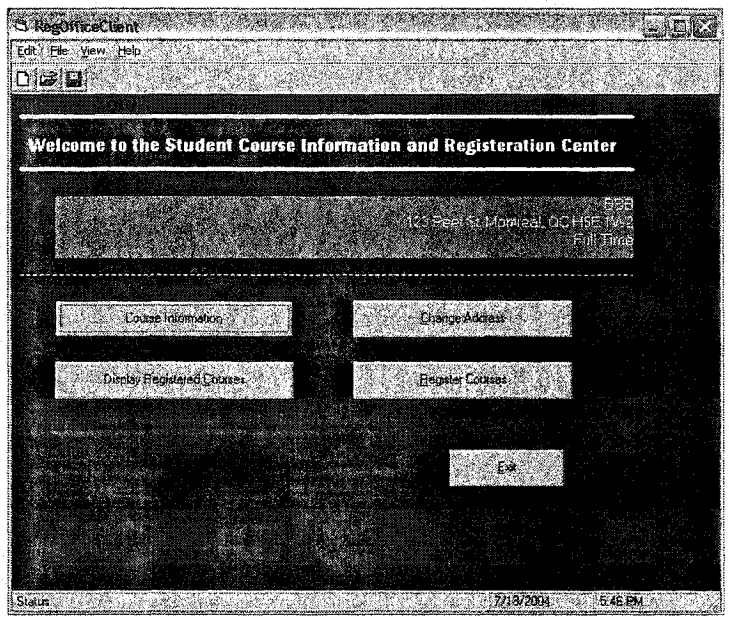
3.7.1.2 The Visual Basic Client

Visual Basic is used to build the client user interface for the course registration system.

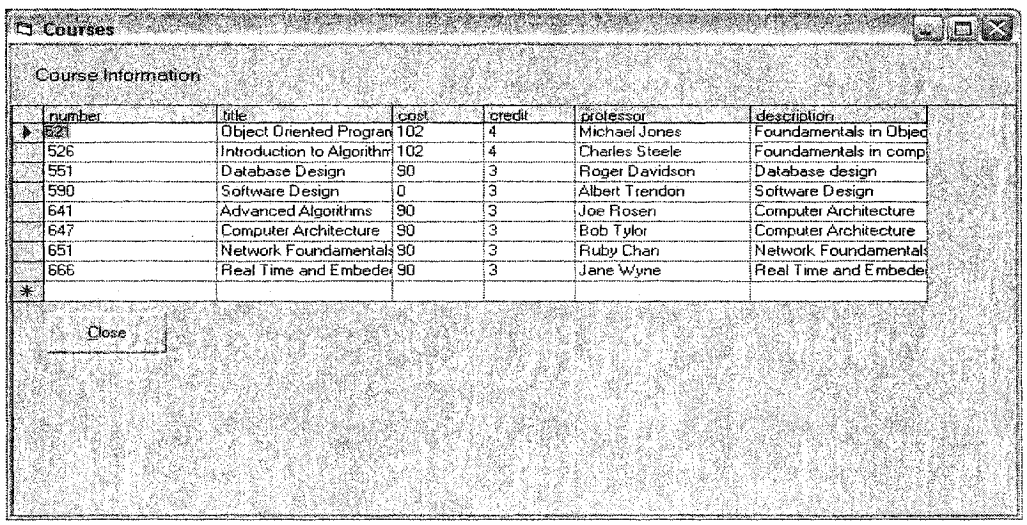
Login screen is used to identify the user and start the application.



The Registration Screen provides the user with welcome information, and entry points for the services, such as viewing course information, change address, display registered courses, and register courses.



The Course Information Screen provides detailed information on course title, credit, instructor and course description etc.



3.8 Build Sample CORBA Application

In this thesis project, I use Orbix2000 from IONA to build the sample CORBA application.

3.8.1 Description of the sample project

The sample application provides a user interface for student to interact with the university's course registration system. In the application, the student can browser list of current available courses, course schedules, and related course information. The student can register or cancel courses. The course registration is based on first come first service. The system will ensure that the maximum number of student allowed will not be exceeded, and the student must not re-register any course, which he's previously taken. The results are saved when the student completes the session, and the information can be retrieved and updated at a later time.

The system consists of three logical layers:

- User interface—It is a DOS console application written C++. This is the client layer of the application, implemented as an Orbix CORBA Client.
- Business Logic Layer—This layer is responsible of implementing the business logic of the course registration system. It contains three components: Touch Screen, Registrar Factory and Registrar. The components are implemented as Orbix CORBA servers.
- Database layer—It provide a persistent data storage for system. All the course information, course schedule, course registration, and student information are stored here. Text files are used as the database.

The CORBA IDL describes all the interfaces and methods (services) provided by the servers:

```
//-------------------------------------------------------------------//
// RegOffice.idl   -- IDL for the Registration Office application
// Declares the following interfaces and their supporting data types:
//   TouchScreen
//   Registrar
//   RegistrarFactory
//-------------------------------------------------------------------
module RegOffice
{
    //---------------------------------------------------------------
    typedef unsigned long CourseNumber;
    typedef sequence<CourseNumber> CourseNumberList;
    //---------------------------------------------------------------
    exception InvalidCourseNumber{};
    exception InvalidStudentId{};
    struct Course
    {
        CourseNumber   course_number;
        double         cost;
        unsigned short credits;
        string         title;
        string         professor;
        string         description;
      unsigned short maxSeats;
   };
    typedef sequence<Course> CourseList;
    //---------------------------------------------------------------
    typedef unsigned long StudentId;
    struct Student
    {
        StudentId          Student_Id;
        string             name;
        string             address;
};
//-----------------------------------------------------------------
interface TouchScreen     //intend to be created by registrar factory
                          //as a return val
{
         void             Init (in StudentId sid);

         //All course offered in the department
         CourseNumberList GetAllAvailableCourses();


         //courses taken by the student in the previous terms,
         //does not count currently registered courses
         CourseNumberList GetCourseTaken();

         Course           GetCourseDetail(in CourseNumber cnum)
                          raises(InvalidCourseNumber);
         string         GetStudentCurrentStatus();
         Student        GetStudentDetail();
         void           LogOffScreen();   //cause the release of the object
reference at server side
    };
//-----------------------------------------------------------------
// The Registrar interface is the main interface that allows
// students to access and update the database.
interface Registrar
```

```
{
        exception ExceedMaxNumOfSeats{};
        readonly attribute string RegOfficeAddress;
        boolean         Init(in StudentId sid);
        CourseList      GetRegisteredCourses();

        //return false if the cnum is invalid or the course has
        //been already registered
        boolean         RegisterCourse(in CourseNumber cnum, out boolean
MoreCourseAllowed )
                        raises (ExceedMaxNumOfSeats);

        boolean         CancelCourse(in CourseNumber cnum);
        boolean         CompleteSession(in boolean commit_trans);

        //cause the release of the object reference at server side
        void             Destroy();

};
    //-------------------------------------------------------------------
    // The "RegistrarFactory" interface finds registrars.
    // There is no standard way of implementing object factory in CORBA
    // Clients only get references to registrar factories -
    // they are not responsible for their lifecycle (thus
    // registrar factories don't have a "destroy" method.
    // There is no way for a client to control the life cycle of this
    //object as we don't intend to do so
interface RegistrarFactory
{
        boolean LogOn(in StudentId sid, in string password);
        //return object ref
         Registrar    find_registrar(in StudentId sid, in string password);
        TouchScreen      CreateTouchScreen(in StudentId sid, in string password);
    };
    //-------------------------------------------------------------------
};
```

3.9 Assessment Strategy in Choosing COM, CORBA for Building Enterprise Solutions

The strategy for objectively assessing the appropriateness of COM or CORBA for building server-side software systems can be achieved by identifying an enterprise domain, legacy systems and platform for new development. Here I provide an example of assessment matrix, which may be used for a decision making for an enterprise solution.

Criterion	COM	CORBA	WEIGHT	COM Rating	CORBA Rating (Vendor A)	CORBA Rating (Vendor B)
Legacy system support	Microsoft and several partners have been able to establish excellent connectivity products from windows to legacy platforms such as MVS and OS/400. The combination of a mature component infrastructure and excellent connectivity makes COM/Windows an excellent candidate for a distributed object gateway to legacy systems. However, COM/Windows is not well suited for a wrapper approach. The greatest weakness of COM has been Microsoft's inability to establish COM on non-windows platforms. Non-windows platforms lack the development tools and advanced services, which are available on the Windows platform.	The greatest strength of CORBA is its proven track record on a wide range of established server platforms. The OMG envisions a heterogeneous environment consisting of many diverse platforms unified by CORBA middleware. Because CORBA focuses on multiple platforms as opposed to a single platform, CORBA can be used to wrap a variety of legacy systems. CORBA's wide platform availability also allows it be used as a gateway to legacy system.	5	15 (3)	20 (4)	15 (3)
Platform	COM has been limited to Windows operating systems. However, Microsoft supports the implementation of COM on other platforms. Software AG developed an implementation on Solaris, for example. However, this product had a number of technical flaws and received a decidedly mixed response from the industry.	Viewed as the object-oriented middleware of choice for enterprises needing to develop/support distributed applications across heterogeneous platforms For example, Iona's Orbix supports Windows NT, many flavors of UNIX (including Solaris, HP-UX, Silicon Graphics IRIX, IBM's AIX and Digital's UNIX), and IBM's MVS.	4	4 (1)	16 (4)	8 (2)
Supports Re-use of existing code	Any COM DLL or EXE can be converted to a DCOM component by modifying the registry. This is a great advantage for creating a N-tier application from an existing 2-tier application.	CORBA's ability to re-use existing code comes from it's capability as a cross-platform technology. CORBA will allow access legacy data and code but not without writing new objects that interact with CORBA.	3	12 (4)	3 (1)	3 (1)
Programming language support	COM is programming-language independent. COM objects can be built in Java, Visual Basic, and Visual C++ and many other languages. Visual Basic and Visual C++ are the 2 most common language in COM development. VB is mostly used for COM client. VC++ is used to build Server side COM components. Microsoft' JVM has been extended to provide intrinsic support for COM client and server implementations. For the most part, the result is a natural mapping of Java to COM objects, but only through Microsoft's JVM. COM for Java will be restricted to Win32 platforms.	CORBA is designed to support many widely used programming languages without requiring their modification. CORBA's language mappings specify how IDL is translated into the target language in a relatively natural way. This makes it possible to use a CORBA implementation with one's choice of language, compiler, development tools, and operating system. The OMG has produced standard CORBA language mappings for Java, C, C++, Smalltalk, Cobol, and Ada95. Some vendors have implemented mappings for other languages. CORBA's Java mapping does not require modifications to the JVM.	5	20 (4)	5 (1)	5 (1)

Essential Services	Microsoft enumerates their COM-related services as "security, lifecycle management, type information, naming, database access, data transfer, registry and asynchronous communications" [COM 95]. COM services can alternatively be understood according to their product packaging. Security, registry, naming, and type information are included in COM or the 32 bit Windows operating systems, at no additional charge. Microsoft Transaction Server includes many services that provide a rich execution environment for COM implementations, allowing developers to simply declare an objects' runtime requirements in contrast to explicit programming. Among these services are transactional guarantees, concurrency control (thread synchronization), instance lifecycle management, database session management, and security. MSMQ is a distributed queuing service providing guaranteed, asynchronous, message delivery. It is accessible through a traditional, non-COM API as well as through COM interfaces. Microsoft's OLE DB provider for AS/400 and VSAM provides access to mainframe data through COM interfaces, the ActiveX Data Object (ADO). The COM Transaction Integrator for CICS and IMS provides access to transactions on those systems. Microsoft's SNA server is an important part of this suite of services for access to mainframe data. COM services are tightly integrated with their operating system services. This integration provides an increasingly flexible and robust environment for object development and execution.	CORBA defines type information and registry as part of basic ORB functionality. OMG has specified fifteen CORBA services. These services are well designed and adhere to a consistent model. The most important exist in at least one implementation, and some have been on the market for years. Vendors can package these services as needed, but most vendor services follow the OMG scheme. The current services include the Naming Service, Event Service, Life Cycle Service, Persistent Object Service, Transaction Service, Concurrency Control Service, Relationship Service, Externalization Service, Query Service, Licensing Service, Property Service, Time Service, Security Service, Object Trader Service and Object Collections Service.	4	12 (3)	12 (3)	8 (2)

Vendor Support	DCOM is a Microsoft (and now the independent ActiveX Consortium) specification AND a reference implementation. Implementations are based on source code provided by Microsoft. Validation tests must be performed before certification. COM enjoys the luxury of being Microsoft's single technological model thereby having no need to consider competing viewpoints	CORBA is a "consortium-approved" specification, but not an actual reference implementation. Implementations are based on a written specification standard which is subject to interpretation and could lead to incompatible interpretations of the standard. Validation tests must be performed before OMG certification. CORBA represents a robust technical review by software industry professionals and, as such, has been optimized/satisfied; could be thought of as being reduced to the least common denominator. Some ORB implementations commonly provide proprietary extensions to the OMG Reference Model, the language bindings, and the inter-ORB protocol.	3	3 (1)	9 (3)	6 (2)
Deploy-ment	COM is included as part of Windows operating systems.	An ORB needs to be deployed on each server and each client.	3	9 (3)	6 (2)	3 (1)
Security	Since COM (ActiveX) objects are binary, there is potential for undetected tampering. COM (ActiveX) objects can get access to perhaps any process running on the client for potential security and/or corruption problems	CORBA's Security service specifies six functions: identification & authentication, authorization & access control, security auditing, security of communication, non-repudiation, security admin.	5	5 (1)	15 (3)	10 (2)
Overall Rating				80	84	58

Table 2: Assessments of Choosing COM, CORBA for building enterprise solutions

From the above assessment, both COM and Vendor A are acceptable chose, whereas CORBA from vendor B is disqualified.

3.10 Conclusion

COM is the dominant component architecture in use today. COM is focused on solving development problems in desktop environment. It therefore makes sense that COM is primarily a component architecture rather than a remoting architecture.

COM is a very mature component architecture that has many strengths. Thousands of third-party ActiveX controls are available that can be used to quickly create sophisticated end-user applications in a wide range of client environments, and there are rich tools that accelerate development of COM based applications.

CORBA is the dominant remoting architecture because it provides robust cross-language, cross-platform, and cross-vendor support for creating servers on wide range of operating system platform.

CORBA was intended from the beginning to create a standard for remote method invocation. The use of CORBA allows for incredible versatility when implementing distributed systems. This versatility is a result of broad language support, diverse platform support, multi-vendor support and freely available CORBA products.

The significant weakness of COM as the ideal remoting solution is the platform limitations. COM is primarily based on Windows operating system, with limited support on UNIX and mainframe platforms.

The key weakness of CORBA is it lacks of a mature component model and advanced tool support. CORBA is often not appropriate for creating client-tier applications because it lacks so many of the important-related advantage offered by COM.

When making decisions in choosing right technologies to build enterprise solutions. An objective strategy of assessments should be made based on the company's real requirements, existing situations, and short term and long term plans.

4 Microsoft .Net

4.1 COM/DCOM and .NET

One of the primary goals of the .NET Framework was to make COM development easier. Dealing with the COM infrastructure was hard one of the hardest things about COM development. Coding COM at the API level such as IUnknown, IDispatch, IconnectionPoint is a complicated process, it represents a massive learning curve. To make COM development easier, the .NET Framework automated virtually all of what developers currently think of as "COM," including reference counting, interface description, and registration. .NET addresses many of the shortfalls of COM, including versioning and deployment problems (DLL Hell), language incompatibilities, reference counting.

One of important goals of .NET is to promote interoperability with existing technologies. .NET interoperability comes in three types:

- Interoperability of .NET code with COM components (called as COM interop)
- Interoperability of COM components with .NET (called .NET interop)
- Interoperability of .NET code with Win32 DLLs (called P/Invoke)

The interoperability features of .NET allow .NET application to work with COM components (backward compatibility), and it also allows COM-based code to use .NET managed components (forward compatibility).

4.2 The .NET Platform Architecture

".NET is Microsoft's platform for a new computing model built around XML Web Services" [47]. .NET platform can be divided into three main areas, built on top of the Windows operating system. The .NET Enterprise Servers deliver a range of services covering database management, messaging, Internet delivery, system integration and so on. The .NET Framework

provides a runtime and a comprehensive class library for .NET applications. Internet-wide authentication and a wide range of basic Web services are provided by .NET My Services (As shown in Figure 4-1).

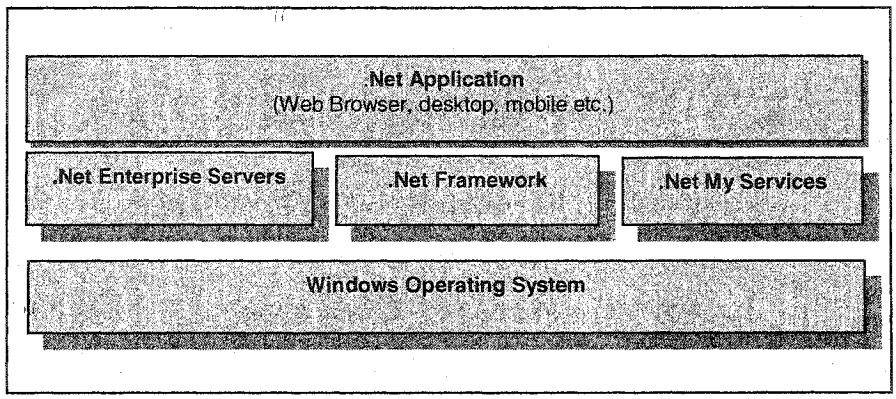


Figure 4-1: The .NET platform

4.3 The .NET Framework

The .NET Framework is .NET's universal application execution environment. It covers all the layers of software development from the low level interaction with the operating system up to client user interfaces. The major components of the .NET framework are shown in Figure 4-2.
The bottom layer is the Common Language Runtime (CLR), which is the heart of the .NET framework. CLR includes Common Type System, which provides common types and a standard interface convention. Common Type System makes cross-language inheritance possible. CRL also does memory management and provides Just-in-Time compiler and so on.

The middle layer is Framework Class Library, which provides standard system services such as ADO.NET and XML. The Framework Class Library is a vast collection of base classes, interfaces, and value types. The Framework Class Library aims at encapsulating the functionality of core system and application services.

The top layer includes' user and program interfaces—Windows Form, Web Forms and Web Services. Windows Forms provide a new way to create standard windows desktop applications. Web Forms provide forms-based UI for Web. Web Services provide a mechanism for programs to communicate over the Internet using Simple Object Access Protocol (SOAP). Web Services are implemented through ASP.NET.

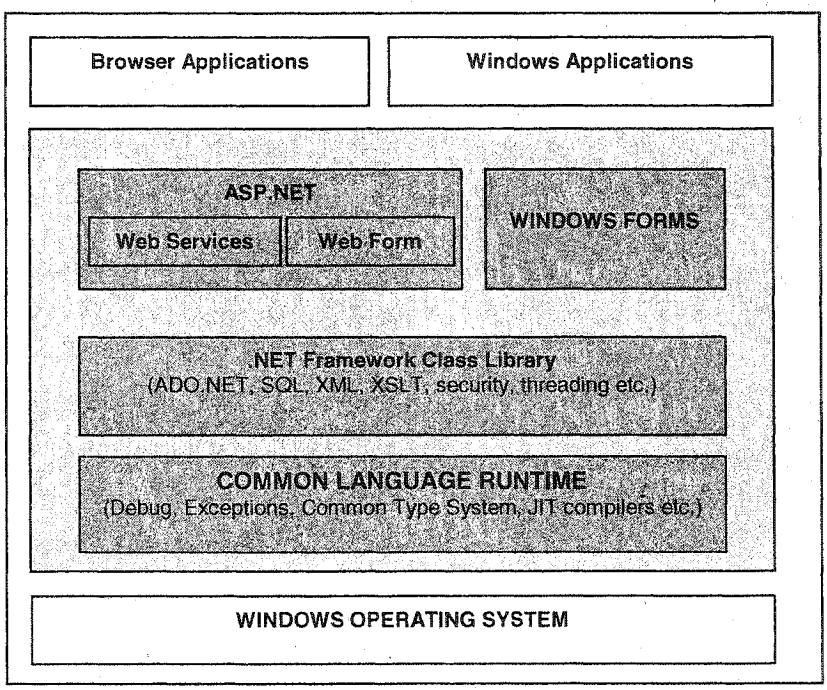


Figure 4-2: Major components of the .NET framework

4.4 Common Language Runtime

The .NET Common Language Runtime is a Windows-specific execution environment, which runs the code and provides services that make the development process easier. CLR is designed to be a language-neutral architecture. .NET supports both managed and unmanaged code. Applications created from managed languages, such as C# and VB.NET, execute under the management of CLR. At development time, .NET source code modules are compiled into the Microsoft Intermediate Language (MS-IL). CRL provides just-in-time (JIT) compilers, which convert IL to native binary machine code. The JIT compilation is done only once at the execution and the results are cached for future use. Figure 4-3 shows the compilation process of .NET

source code from different languages. People often think of CLR as a virtual machine, like Java Virtual Machine. Although both CRL and JVM serve the same goal of supporting source code portability, CLR is not an interpreter. CRL is responsible for compiling .NET applications to native machine code, and then steps aside.

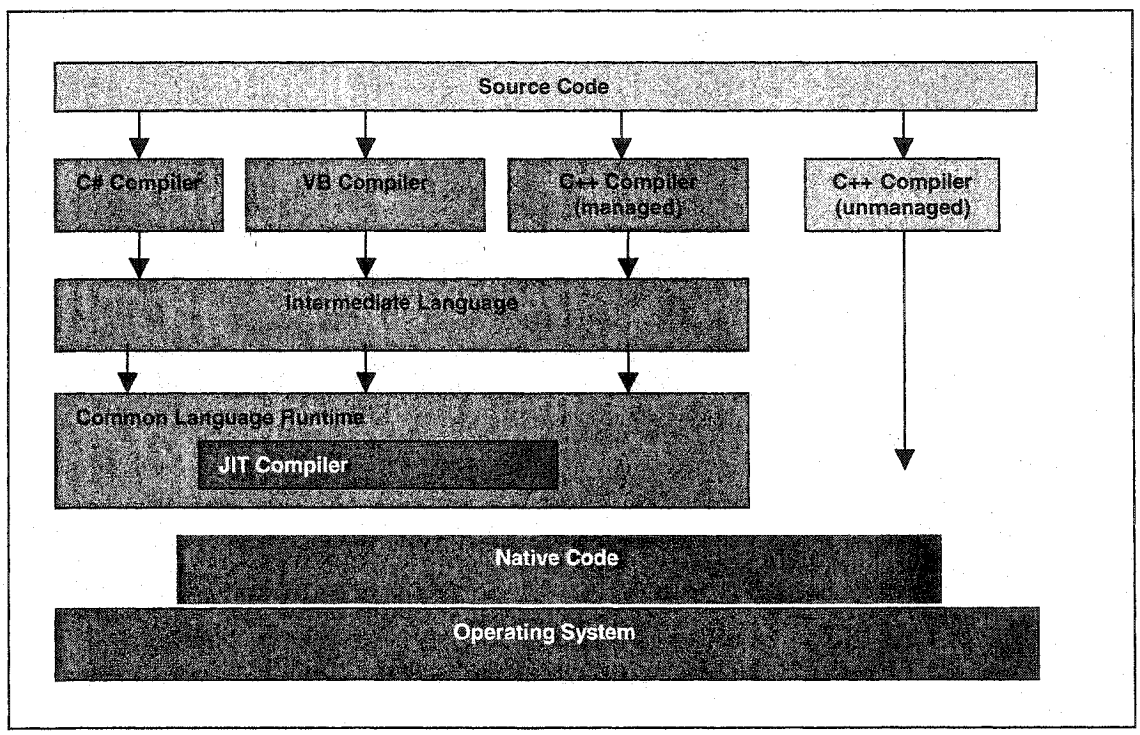


Figure 4-3: Compiling source code into native code in .NET

The CLR provides all the common services required by .NET applications. Some of the key services include language integration, security, memory allocation, process control, thread management, and unified error handling. It also provides development time services to applications being developed within the .NET framework—particularly to facilitate cross-language integration. Some of the key services provided by CLR are:

- **Cross-language integration and interoperability** — the CLR handles exception handling and marshaling in an unified way that is common cross the languages, it also support cross-language inheritance. CLR enables managed code written in one language to seamlessly integrate with code written in another language. The IL facilitates the language interoperability—the source code from different languages will ultimately

compiled to IL and should be interoperable with each other. The Common Language Specification works with Common Type System to ensure language interoperability.

- **Memory management** — the Garbage Collector (GC) is responsible for automatically reclaiming unused memory. The program may explicitly invoke the Garbage Collector by calling GC.Collect.
- **Versioning and deployment support** — The CLR support side by side execution of multiple versions of the same component, or within the same process.

4.5 ASP .NET

ASP.NET is a technology that allows for the dynamic creation of documents on a Web server when they are requested via HTTP. ASP.NET brings a language independent way of creating components and dynamic Web applications that can produce output on any platform or device. There are several server-side components, which have ability to dynamically create controls at runtime. This makes it possible to generate output specifically for a target browser. ASP.NET also provides better integration with XML, separation of HTML from ASP code, easy access to server side .NET services, and uses compiled pages instead of interpreted pages to improve performance.

4.6 ADO.NET

Microsoft has released several different data access technologies over the years that can be used to query and manipulate data. Microsoft has contributed its share of data access technologies to the mix including DAO, RDO, and ADO. ADO.NET is an exceptional and a worthy successor to ADO. It successfully addresses many of the shortcomings found in ADO through its support for working with multiple data sources, navigating data relationships, reading and writing XML data etc. Although the ultimate goal of both ADO and ADO.NET is to provide a data access API that allows data to be selected, inserted, updated, and deleted, there are many differences

between the two technologies. The environment that the two technologies operate in is one of the more important differences. ADO relies on the Component Object Model (COM) whereas ADO.NET relies on a "managed" environment provided by .NET's Common Language Runtime. ADO.NET can work both through Web protocols, using XML (Figure 4-4), or in a more traditional client/server architecture (Figure 4-5).

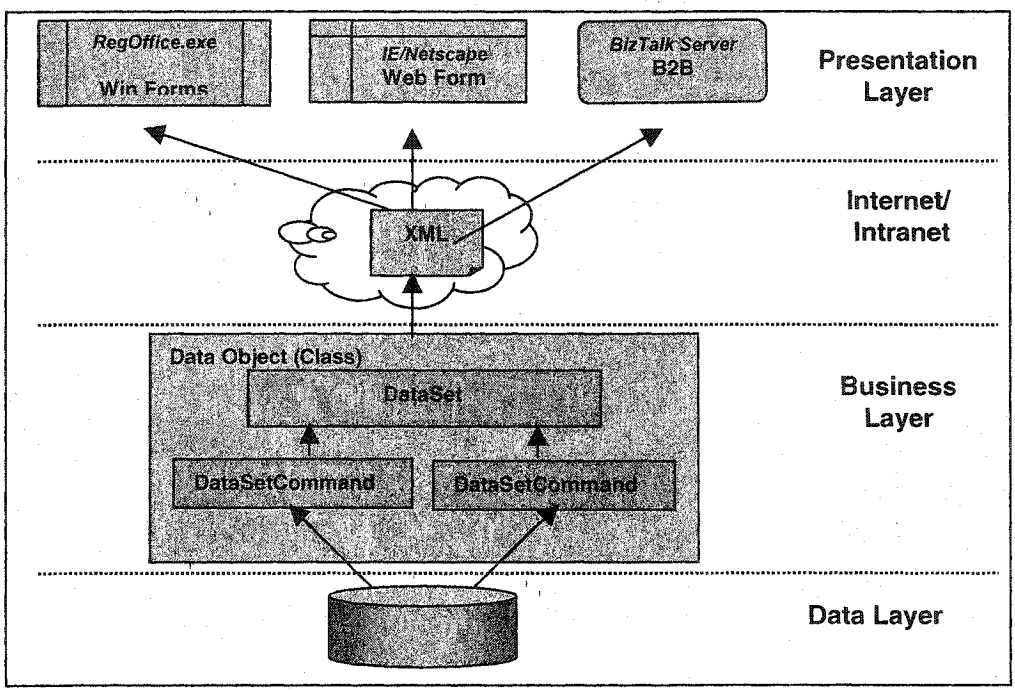


Figure 4-4: ADO.NET works both through Web protocols, using XML

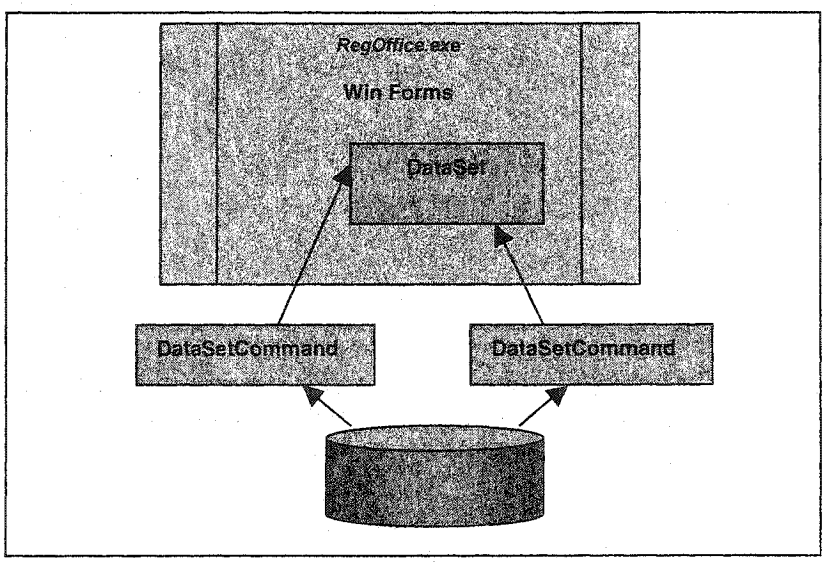


Figure 4-5: ADO.NET works in traditional client/server architecture

4.7 .NET Remoting

.NET Remoting can be used for accessing objects in another application domain. .NET Remoting can always be used whether the two objects live inside a single process, in separate process, or on separate systems. .NET applications work within an application domain. Application Domain is the logical and physical boundary created by the CLR to isolate configuration, security, or stability of one application domain from another. Different applications can run inside the same process but within different application domains. Objects inside the same application domain can interact directly; a proxy is needed in order to access objects in a different domain. The major elements in .NET Remoting includes remote objects, channel, message, formatter, proxy etc. A .NET Remote object has distributed identity. So a reference to the object can be passed to other clients, and they will still access the same object. The proxy knows about the identity of the remote object. .NET Remoting allows creating both stateless and stateful remote object. Single-Call objects don't hold state whereas Client-Activated objects hold state.

5 J2EE Overview

Java™ 2 Platform, Enterprise Edition (J2EE) is Sun's reference standard for enterprise development, first introduced in December 1999, and now at version 1.4. The core technology components of the J2EE platform include servlet, JSP, EJB, JDBC and JMS etc.

5.1 CORBA and J2EE

CORBA is a networking protocol for building distributed applications. It provides the network transparency. Java 2 Platform is designed to take over significant portions of code for thread management, transactions, caching, and database mapping etc. Java provides the implementation transparency.

The Java platform complements CORBA by providing a highly productive implementation environment, and a robust platform. CORBA standards provide the proven, interoperable infrastructure to the Java platform.

There is a CORBA specification (CORBA 3.0) for application server called CCM (CORBA Component Model) that is a mirror image of the EJB spec. CCM is designed to integrate with Java and EJB technology. The CCM allows EJB technology and CORBA Components to be integrated in the same application. EJB and CORBA do not really compete but rather complement each other. EJB requires CORBA interoperability. On the other hand, CORBA does not provide technology and programming models for 3-tier environments. J2EE does this with vast collection of Java APIs.

5.1.1 J2EE Application Components and Containers

The J2EE specifies that a compliant J2EE application server must provide a defined set of containers to house J2EE components. Containers supply a runtime environment for the components. Application Programming Interfaces in J2EE are available to provide communication between components, persistence, service discovery, and so on. J2EE application server vendors provide containers for each type of J2EE components, there are two type of J2EE containers —Web container, and EJB container (as shown in Figure 5-1).

There are three types of components deployed, managed, and executed on a J2EE Server:

- Web Components — A Web components interacts with a Web-based client, such as a Web browser. There are two kinds of web components in J2EE—Servlet components and Java Server Page (JSP) components.
- Web Services — A web component that exposes Servlets and EJB components based on SOAP and HTTP.
- EJB Components — There are three kinds of EJB components—Session beans, Entity beans, and Message-Driven beans.

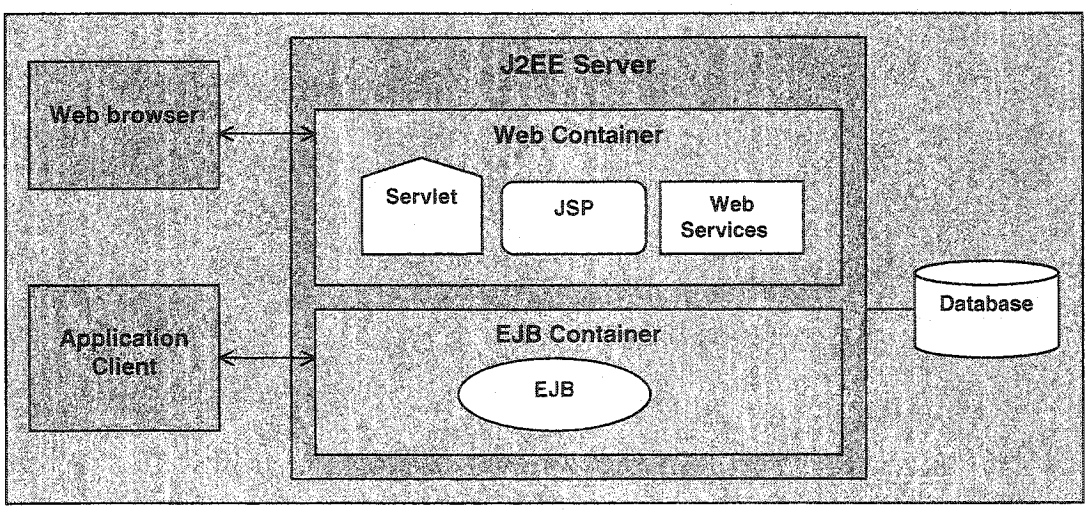


Figure 5-1: A J2EE logical architecture

5.2 J2EE Standard Services

Containers must provide each type of component with a defined set of services:

- Connectivity — Containers must support connectivities to other components and to application clients. One form of required connectivity is using distributed object through Java RMI and CORBA. Internet connectivity must be provided through HTTP.
- Directory Services — J2EE Servers are required to provide naming services in which components and discovered. The Java Naming and Directory Interface (JNDI) provides a way of accessing the services.
- Data Access and Persistence — Data access is provided through the JDBC API. JDBC is a standard API to access data source in a vendor-independent manner.
- Legacy Connectivity — The Java Connector Architecture (JCA) provides J2EE support in integrating enterprise information servers and legacy systems.
- Security — Security is built into the J2EE model. APIs, such as the Java Authentication and Authorization Services (JAAS), assist the J2EE enterprise application in imposing authentication and authorization security checks on users.
- XML support — The Java API for XML Processing (JAXP) converts XML into a implementation independent format. The JAXP supports the parsing of XML documents using Document Object Model (DOM), Simple API for XML (SAX), and the XML Stylesheet Language for Transformation (XSLT).
- Transaction — A J2EE server must provide transaction services for its components. Java Transaction API (JTA) supports transaction management. It allows the container to communicate with the transaction manager, and the transaction manager to communicate with the resource manager.
- Messaging — The Java Message Services (JMS) allows components to send and receive asynchronous messages, typically within an organizational boundary. JavaMail

provides a platform-independent and protocol-independent framework to build mail and messaging applications in Java.

- Web Services — A J2EE server supports for defining and using Web Services through the industry standard protocols such as SOAP, UDDI and WSDL.

Figure 5-2 shows the J2EE architecture with the services available to its containers.

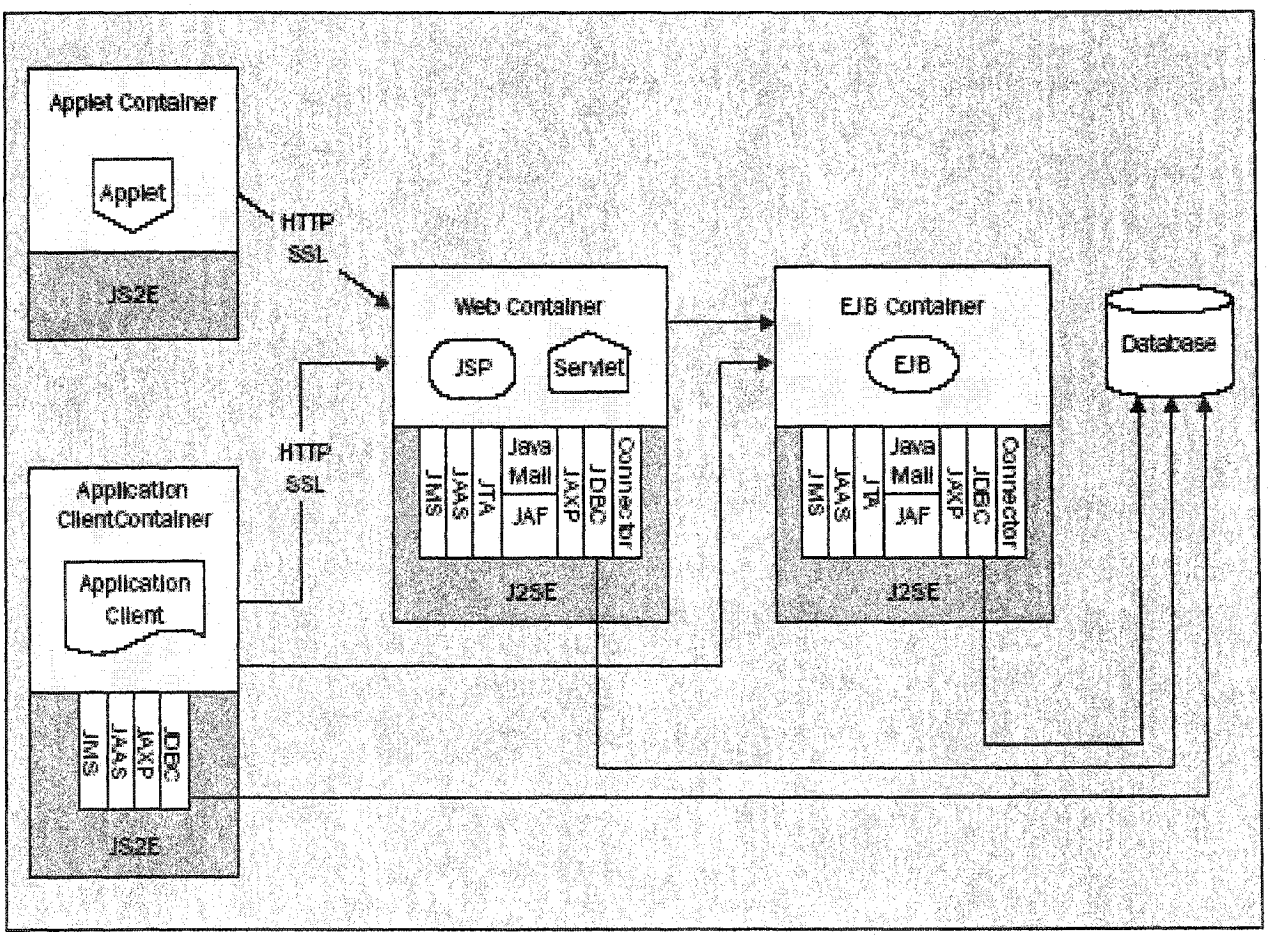


Figure 5-2: The J2EE architecture with the services available to its containers [48]

6 J2EE Vs. NET — A Technical Comparison

6.1 Compare the basic

WAS, IBM's WebSphere, is used as an example of J2EE in the following comparisons.

6.1.1 Platform Independency

J2EE:

Java is the platform independent language with Java Virtual Machine (JVM) provided by Sun Microsystems. Java codes in J2EE are compiled to Java byte codes as in J2SE. The Java byte codes can run on any platform such as Unix, Linux, or Windows environment. Virtually all operating systems have JVM installed.

.NET:

One of the goals of .NET framework is to achieve platform independency. However, it only works on Windows environment at this moment. Microsoft provides Shared Source Common Language Implementation (SSCLI, also known as Rotor). As part of Microsoft's Shared Source initiative, Rotor provides a free, shared-source implementation of Microsoft's Common Language Runtime platform, including source code for C# compilers, as well as for the Common Language Infrastructure (CLI) platform itself. It is the working implementation to provide a Platform Adaption Layer (PAL) for academics and researchers. Rotor is under a noncommercial shared-source license and it now can run on Microsoft Windows XP or Windows 2000, the FreeBSD OS 4.7, and Apple Mac OS X 10.2.[38]. If SSCLI is successful, codes on .NET framework will be run on FreeBSD OS and Mac OS X 10.2 as well as Windows OS. That is, .NET framework may achieve the platform independency.

6.1.2 Language and Runtime Support

J2EE:

Java is at the center of J2EE. All components such as EJB and Servlets that are deployed in J2EE framework are also written in Java. Although JVM byte code is language-neutral, but in practice this byte code can only be used with Java. Java can run on any platform that supports a JVM (Virtually, Java Runtime Environment is available on any platform—Win 32, Unix, and Mainframe). All Java code is compiled into bytecode and interpreted by Java Virtual Machine. Java may also be selectively JIT compiled, depending on the JVM, or fully natively compiled with ancillary products such as JOVE. JOVE is the first optimizing native compiler for large-scale Java applications provided by Instantiations Inc. [39]. JOVE compiles Java byte code directly into Windows-native machine code, making for much faster execution by avoiding the interpretation phase.

.NET:

The .NET framework, based on the new Common Language Runtime (CLR), enables development in any language that is supported by Microsoft's tools. A single .NET component can be written in several languages. CLR also provides seamless integration of code written in various languages. .NET code is initially compiled to an Intermediate Language (MSIL) and then JIT compiled to native code by the CLR at runtime or natively compiled (PreJit) at install time. Code written for the CLR is referred to as managed code.

6.1.3 Class Libraries

J2EE:

The Standard Java Libraries are in the J2SE. J2EE also defines a set of Java APIs (JDBC, JSP, EJB, JMS, JAXP, JCA, JFC, JNDI, etc.). In addition, vendors include their own extensions to J2EE for integration and differentiation in the marketplace, such as WAS JMX (Java Management Extensions).

.NET:

.NET Framework class library includes a hierarchical set of managed classes arranged in namespaces. It provides a consistent programming model and unifies the APIs needed to build solutions. The library includes three key components:

- ASP.NET to help build Web applications and Web services.
- Windows Forms to facilitate smart client user interface development.
- ADO.NET to help connect applications to databases.

6.1.4 Development Tools

J2EE:

WebSphere Studio is now IBM's Java development environment. WebSphere Studio is an open, Eclipse-based, J2EE development environment for Java, Web sites, Web services, XML, and the entire range of IBM middleware.

.NET:

Microsoft Visual Studio .NET. provides an Integrated Development Enviroment(IDE) for .NET programming. VS.NET supports unified debugging, project management, Intellisense, Server Explorer, graphical editor for the Web and forms, class view, dynamic help, add-ins, and scripting. VS.NET can be used to build console applications, Windows Forms, Web Forms, Web Services, Windows Services, and components. VS.NET provides C++, C#, and Visual Basic programmers with a common development environment.

6.2 Core Platform Capabilities

6.2.1 Deployment

J2EE

J2EE deployment is implemented through Java Package, which is a collection of Java classes assembled as a unit and arranged hierarchically. J2EE specification contains few details on application deployment, leaving most up to individual vendors. Each vendor devises its own unique way of deploying J2EE applications. For instances, Server-based deployment uses XML-based deployment descriptors are read and handled differently by each vendor's implementation. WebSphere includes an application assembly tool to handle this process. WAS also features WebSphere Extended Deployment, which is designed to help companies deliver computing resources based on the demands of their business applications.

.NET:

Assemblies are the smallest units of versioning, security, and deployment in the .NET application. Assembly includes built-in PKI (public-key infrastructure) support and metadata to tightly control binding of assemblies at runtime. Server-based deployment can be done by simply copying the relative paths that contain the application. .Net Applications can explicitly share components through shared assemblies. VS.NET includes Windows Installer, which can be used as a deployment tool.

6.2.2 Thin Client

J2EE:

For thin-client development, the servlet and JavaServer Pages (JSP) standards provide ways to build HMTL, WML, XML and other thin interfaces on top of a Java middle tier. Java servlets and JSPs are supported by a number of application server vendors, both commercial (such as BEA WebLogic, iPlanet and IBM WebSphere) and open source.

.NET:

ASP.NET is the .NET successor to Active Server Pages (ASP). Functionally, ASP.NET fills the same role as a thin-client (HMTL, WML or XML) paradigm for applications. ASP.NET uses a new CLR-based framework to replace ISAPI/ASP architecture. That means component references can be added to pages using any CLR-compliant language. ASP.NET pages are compiled into the IL-based CLR runtime, rather than interpreted, as ASP pages are.

6.2.3 Fat Client

J2EE:

For fat GUI clients, J2EE offers the Java Swing API, with a palette of standard JavaBean components that can be assembled programmatically, or pulled into visual GUI design tools like Visual Cafe, Borland JBuilder, IBM WebSphere Studio and so on.

.NET:

Traditionally, Microsoft has offered fat client support through its Microsoft Foundation Classes (MFC) API, but .NET offers a new set of components—Windows Forms, which contain a set of managed classes that allow for the creation of forms-based applications. Windows Forms fill the same functional role as MFC, but they are plugged into the new .NET runtime framework and component model. Windows Forms takes advantage of new Windows 2000 features such as nonrectangular forms. Windows Form includes a managed control architecture to build reusable

visual controls. Windows Forms–based applications can run as standalone applications or can be hosted in Microsoft IE on clients with the CLR installed.

6.2.4 Integration

J2EE:

J2EE Connector (JCA) is a specification based on the IBM Common Connector Framework (CCF). IBM WebSphere contains adapters used to connect to packaged applications such as SAP, Peoplesoft, and JDEdwards. Java Message Service (JMS) also supports application integration. WebSphere MQ (formerly MQSeries) allows J2EE applications to exchange information across different platforms, integrating new and existing business applications. WebSphere MQ "assures reliable delivery of messages, including XML documents and SOAP messages; Connects applications and Web Services; Spans environments such as J2EE and Microsoft .NET; Bridges over 35 platforms"[40].

.NET:

Host Integration Server (HIS 2004 is the newest release) offers a Windows interface to mainframes and legacy applications. HIS 2004 provides a Transaction Integrator(TI) that lets host-distributed applications participate in .NET and COM+ transactions; MSMQ to MQSeries Bridge for integration between heterogeneous applications [41]. Host Integration Server also includes integration with Commerce Server, BizTalk Server. BizTalk Server also supports application-integration components (AIC) that can be built to integrate with packaged applications.

6.2.5 Web Server Processing

J2EE:

HttpServlets provide a request/response model available to Java classes in addition to session-state services. The Response object contains the entire response from the Servlet to the client. The response is usually in HTML but can actually be in any document type such as XML, or even binary such as a JPG or GIF. JSPs often call servlets as the controller that acts as the broker between presentation and business service layers to provide separation between the HTML and Java code. The Servlet engine is included in WAS.

.NET:

ASP.NET requests are initially handled by the HTTP runtime of IIS. ASP.NET ISAPI extension DLL redirects requests to ASP.NET worker process. Requests are process in the "HTTP pipeline" inside ASP.NET worker process. The HTTP Runtime in ASP.NET provides an extensible mechanism by which developers can hook into the processing of resources on the Web server through HTTP Modules. ASP.NET provides integrated compilation support to build source files into DLLs in just-in-time fashion. ASP.NET also supports a request/response model and session-state services. ASP.NET provides a sophisticated caching mechanism for pages and programming code. ASP.NET provides various code-behind techniques promote a separation of code and HTML.

6.2.6 Database Support

J2EE:

JDBC is an API (included in both J2SE and J2EE releases) to access virtually any tabular data source from Java codes. The JDBC 3.0 API is the latest update of the JDBC API. The API is similar to the Win32 implementation of ODBC. Normally, a database vendor provides the database product

with own JDBC drivers. There are currently 209 JDBC enabled drivers listed on Sun's site [43] under 4 different types (as shown in Figure 6-1 and Figure 6-2), ranging from JDBC to ODBC bridges to fully native-protocol Java implementations. Various drivers support connection pooling, distributed transactions, and rowsets, and some are J2EE-certified. WAS includes providers for DB2, MSSQL, Oracle, Informix, and Sybase. When a Java code is built for database access application, it needs to refer to classes of JDBC API of the JDBC driver that is accessible from the code. Besides, an entity bean of EJB has database connection interfaces. A developer can easily implement an entity bean that connects a database without building JDBC connection logic. Thus, the developer can only focus on implementing business logic so that it will save the cost of the product.

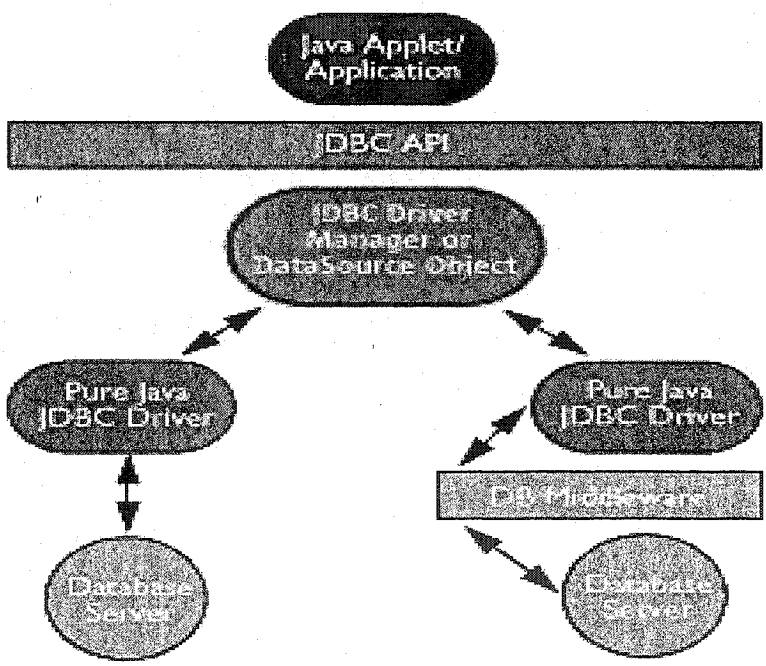


Figure 6-1: Direct-to-Database Pure Java Driver, and Pure Java Driver for Database Middleware [42]

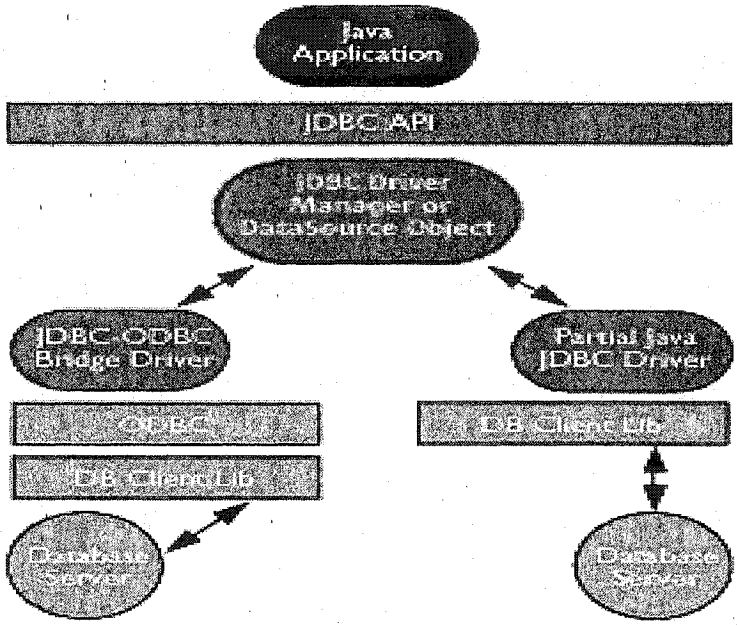


Figure 6-2: JDBC-ODBC, and a native API partly Java technology-enabled driver [42]

.NET:

ADO.NET provides consistent access to data sources such as MS SQL Server, as well as data sources exposed through OLE DB and XML. ADO.NET contains a set of managed classes that provides access to data sources through what are called managed providers. These include the SQL Server .NET Data Provider (which uses Tabular Data Stream to connect directly to SQL Server), OLE DB providers, and ODBC drivers. The managed providers implement connections and connection pooling, commands, transaction processing, and error handling. Figure 6-3 depicts both .NET Framework Data Provider for SQL Server and the .NET Framework Data Provider for OLE DB.

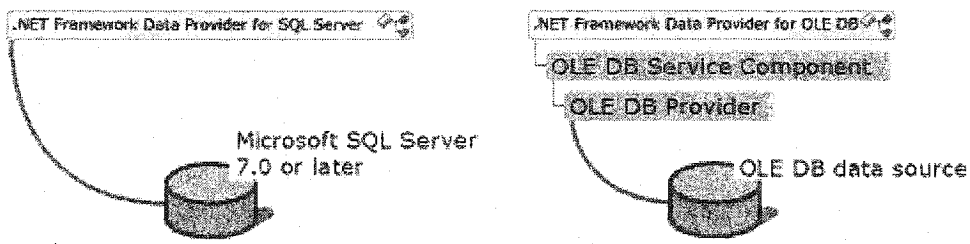


Figure 6-3: .NET Framework data providers [44]

OLE DB was introduced in 1997 as part of Microsoft's Universal Data Access strategies. "To date, there are OLE DB providers for most commercial relational DBMSs including Oracle, Microsoft SQL Server, IBM DB2, Sybase, Informix, CA-Ingres, and ODBC sources" [44].

ADO.NET also includes a middle-tier data cache object called a DataSet that provides an XML-based cache of data that can be used to bind to controls and synchronize changes to a data source through an adapter object implemented by the managed provider.

6.2.7 Messaging

J2EE:

The asynchronous message based communication in J2EE is implemented through Java Messaging Services (JMS). JMS specification v1.0.1 was released by JavaSoft in 1998. JMS is a set of client interfaces to message-oriented middleware (MOM) products that supports both message queues and publish-and-subscribe metaphors. MOM vendors implement JMS providers. Over the years, MOM systems have evolved in a proprietary way. That means each messaging product has its own JMS APIs. This results vendor lock-in since the code is not portable from one to other messaging systems. JMS includes a specification for a new type of EJB, message beans. WebSphere MQ provides an integrated support for the latest JMS interface standard (JMS v1.1), including publish-and-subscribe messaging.

.NET:

In .NET the messaging is handled by System.Messaging namespace. It contains the set of managed classes that wrap the underlying MSMQ infrastructure. The managed classes provide the administrative and client APIs for working with MSMQ servers. Message Queuing (MSMQ), originally released with the Windows NT 4.0 Option Pack. MSMQ is built into Windows 2000 Server and Windows Server 2003.

6.2.8 XML Support

J2EE:

"The Java API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX, and XSLT. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation." [45]. IBM provides a validating XML parser (written in 100% pure Java, and with JAXP support) that easily enables applications to read and write XML data [46].

.NET:

The .NET Framework is built entirely on top of Web standards. It provides several foundation classes in System.Xml namespace to work with XML documents and data. System.Xml namespace contains a set of managed classes that implement the DOM, XSLT, and XPath specifications by the W3C. The ADO.NET architecture uses XML as its native data format. System.Xml contains a class that maps XML documents to the ADO.NET DataSet class to provide XML-based access to relational data.

6.2.9 Remoting

J2EE:

Java Remote Method Invocation (RMI) is a mechanism for invoking methods remotely on other machine. RMI is a programming model and is tightly integrated with the Java language itself. The EJB specification mandates that EJB to client communication uses a more portable and CORBA system interoperable version of RMI, called RMI-IIOP or "RMI over IIOP", rather than standard Java RMI. RMI-IIOP is a standard Java extension developed by Sun, IBM and OMG. RMI over IIOP together provides the programming model and the protocol layer for Java-to-Java and Java to CORBA client communication. Servlets and applets communicate with EJBs via RMI.

.NET:

.NET Remoting is implemented in a set of managed classes. .NET Remoting allows managed code to communicate across application domain boundaries. Application Domain is the logical and physical boundary created by the CLR to isolate configuration, security, or stability of one application domain from another. .NET Remoting supports state management options and can correlate multiple method calls from the same client and supports callbacks..NET Remoting requires the clients be built using .NET. That means it works only in homogenous environment.

6.3 . NET and J2EE 'S Approaches To Web Services

6.3.1 What are web services?

The web services concept offers a compelling new blueprint for implementing communication between applications and services over the Internet. The advocates of web services promise that this blueprint will transform the Internet from a medium that allows people to interact with relatively static, isolated islands of information and processing. It will become a global, pervasive middleware platform that allows software programs to collaborate online in order to carry out tasks for users. This transition will create value-added opportunities, not just for software players, but also for network service providers.

Web services technology is compelling because it provides an implementation-independent mechanism through which applications can interact. Web services technologies prescribe a new way of deploying software in order to make it available for direct program-to-program access across the Internet. They are actually an extension of the concept of component-based software development, together with the power of IP across mobile and fixed communication lines. New, generally agreed standards such as SOAP and UDDI are making it possible.

Much of the confusion (and there is a lot) that exists around web services arises from the fact that there is both a business and technology perspective of the concept:

- from a business perspective, web services are about the delivery of software as a service, whereby software becomes a utility; bought, sold and delivered in a similar manner to electricity or telecommunications
- from a technology perspective, web services are about re-use, and comprise a specific set of technology specifications that are used for packaging software into easily accessible, re-usable components.

Conceptually, Web Services is a combination of two existing successful software technologies – web technology and component technology – so it is unsurprising that it gives rise to these two perspectives. Web services uniquely combines the two technologies to form a lightweight software model that works over the Internet in a widely adopted standard way.

The web services model provides a completely implementation-independent mechanism through which applications can interact. Web services are independent of programming languages, hardware platforms, execution models and program locations. Web services technologies prescribe a new way of deploying software, in order to make it available for direct program-to-program access across the Internet; they make no assumptions about how that software is built. The software that sits behind a web service could be an enterprise Java application or component, a Cobol transaction, a Visual Basic program, or a CORBA-based application.

SOAP, WSDL, UDDI, and XML form the foundation of web services technology. They are aiming to solving the four areas of challenge of Web Services.

- Web Service Description — Describing Web Services using the Web Services Description Language (WSDL).

- Web Service Implementation — Using XML as the common language for Web Service communication. XML is ubiquitous throughout all aspects of Web Services. Web Services can be implemented in any programming language that can read and write XML, and can be deployed on any Web-accessible platform.
- Web Service Publishing, Discovery, and Binding — Publishing Web Services in a registry to be discovered later by interested parties accessing the registry. One type of registry provides a directory service for Web Services providers and their services. This registry provides information categorized by industry-type, product-type, service-location, service binding, etc. One implementation of this registry is based on the Universal Description, Discovery and Integration (UDDI) specification. Another type of registry also acts as a repository where Web Services entities such as business-process schema are stored. A current example of this type of registry is the electronic business XML (ebXML) Registry and Repository.
- Web Service Invocation — Invoking Web Services over an existing Internet protocol such as HTTP or SMTP using the Simple Object Access Protocol (SOAP).

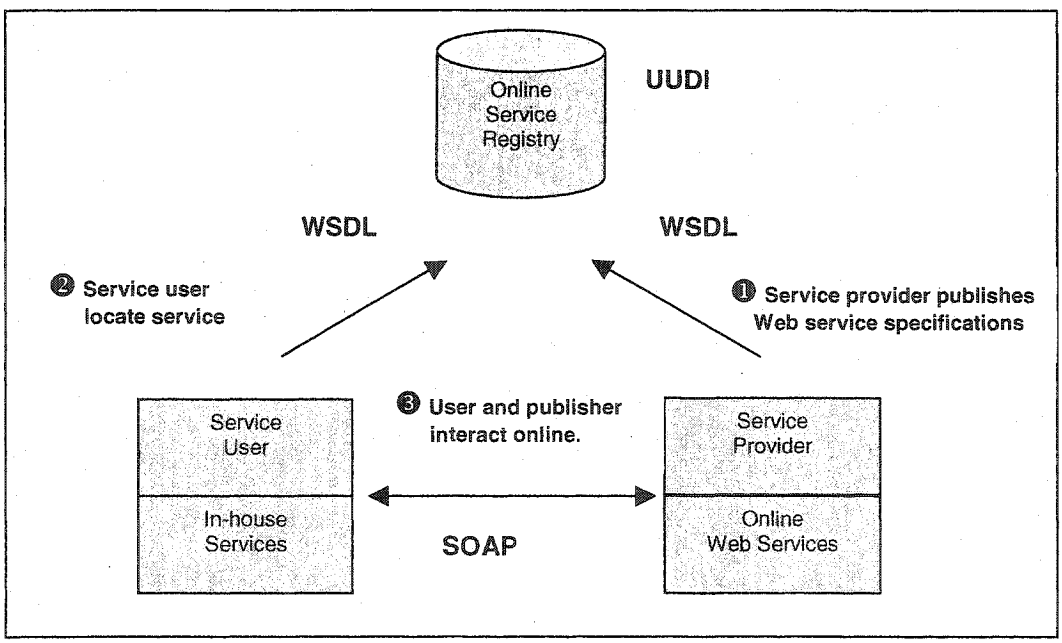


Figure 6-4:How web services technologies work together

The figure 6-4 shows a service provider in the shape of an online web service; they describe the programmatic interfaces using WSDL. They then publish this to the UDDI online service registry, where the service user requiring an online stock-quote service can locate the service. The service provider and user can then establish a direct connection over SOAP.

6.3.2 Web Services Implementations

Both .NET and J2EE support Web services. Microsoft is focusing its implementation within its .NET initiative, whereas Sun Microsystems and other Java vendors are focusing their implementations within the J2EE initiative. Sun Microsystems has published its Open Network Environment Architecture (Sun ONE) as a blueprint for the evolution of Web services. Sun is touting its Java Web Services Developer Pack as its Java 2 Platform, Enterprise Edition (J2EE) toolset for wrapping XML-based Web Services technologies such as SOAP, UDDI, ebXML, and WSDL with Java objects and interfaces. Microsoft has defined its Global XML Architecture, which is aiming to providing additional capabilities to baseline XML Web services specifications. Microsoft is presenting a comprehensive set of development tools to accommodate this new technology. Among these is the SOAP 2.0 Toolkit, which provides a broad range of SOAP support tools. IBM's Web Services Toolkit provides a runtime environment and examples to design, implement, and execute Web Services. The toolkit provides a Web Services architectural blueprint, a private UDDI registry, some sample programs, some utility services, and tools for developing and deploying Web Services.

6.3.3 Microsoft's .NET Web Services

Microsoft is one of the key players when it comes to Web services, and this role is unlikely to diminish in the coming years. Microsoft initiated industry focus on Web services. Microsoft has had a hand in the development of most of the pivotal Web services standards (e.g., SOAP, WSDL, UDDI, WS-Security, WS-Coordination). Furthermore, Microsoft is one of the four

companies around the world providing a UDDI Business Registry (UBR) to facilitate Web service location and identification using UDDI. Microsoft's strategic .NET initiative is highly Web-centric.

ASP .NET is the technology for implementing Web services based on the .NET Framework. ASP.NET Web services process service requests using SOAP over HTTP, as well as HTTP *GET* or *POST*. ASP.NET Web services automatically generate WSDL files for Web services. ASP.NET Web services can be used to implement a Web service listener that accesses a business facade implemented as a COM component or managed class, typically hosted on a .NET server. The .NET Framework development kit also provides tools to generate proxy classes that client applications can use to access Web services.

Web services are stateless. ASP .NET Web services—like any Web services—do not expose the server-side data types to client applications. These are completely hidden inside the Web service. The .NET Web services tools assume a stateless programming model—that is, each incoming request is handled independently. The only state maintained between requests is anything persisted in a data store.

.NET Remoting, however, supports stateful interactions. NET remoting supports a more tightly coupled, object-based programming model between client and server, which provides remote access to server-side objects with full data type fidelity. Clients can also obtain references to server-side objects and control the lifetime of those objects for stateful interaction. If these object lifetime services are used, however, client applications will also need to be implemented using .NET Remoting.

In addition to the features provided by the .NET Framework, Visual Studio .NET provides tools to help build, deploy, and consume Web services. For example, the IDE supports UDDI for locating Web services and understands how to generate client-side proxies from WSDL files. Visual

Studio .NET also includes the Active Template Library (ATL) server, which C++ developers can use to construct Web service listeners that connect to a business facade implemented as a C++ class. ATL Server supports SOAP over HTTP, will automatically generate WSDL files for Web services, and also provides tools to generate C++ proxy classes that client applications can use to access Web services.

Microsoft's vision of Web services is that they will integrate services typically thought of as desktop, or local operating system, services with services accessed over the Web. Therefore .NET supports, or will support, all types of Web services interaction styles. Multiple vendors, such as IONA, is also providing generic Web services, or Web services building blocks.

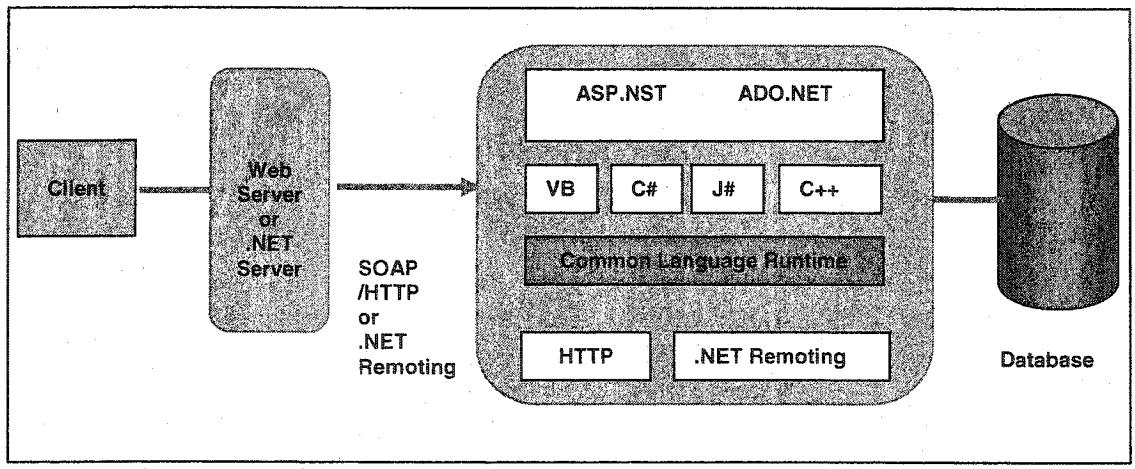


Figure 6-5: The .NET Architecture in Web Services

6.3.4 J2EE and Application Servers

J2EE vendors are supporting Web services. Application server vendors are extending their products with Web services capabilities, in much the same way that many new technologies tend to get incorporated into J2EE, as new APIs.

Web services can be integrated with application servers, such as WebLogic, WebSphere, and Orbix E2A J2EE Edition, to provide access to a variety of back-end systems, including .NET classes and COM objects, Enterprise JavaBeans, CORBA objects, MQSeries, MSMQ, and Java Messaging System queues. The SOAP messages arrive via the Web server integrated with the application server's servlet engine, in which is deployed a Java class representing, or wrapping, the back-end system to be integrated. Web services toolkits, such as IONA's XMLBus and those integrated directly with WebLogic and WebSphere, translate this Java class into a corresponding WSDL file, which can be stored in a UDDI or an ebXML registry. The SOAP client can find the location of the WSDL file from the registry or directly, using its URL, and invoke the Web service.

Figure 6-6 shows how Web services can be integrated with application server environments. J2EE vendors essentially view Web services as another type of application server client. All the requisite qualities of service are defined in J2EE and implemented in application servers.

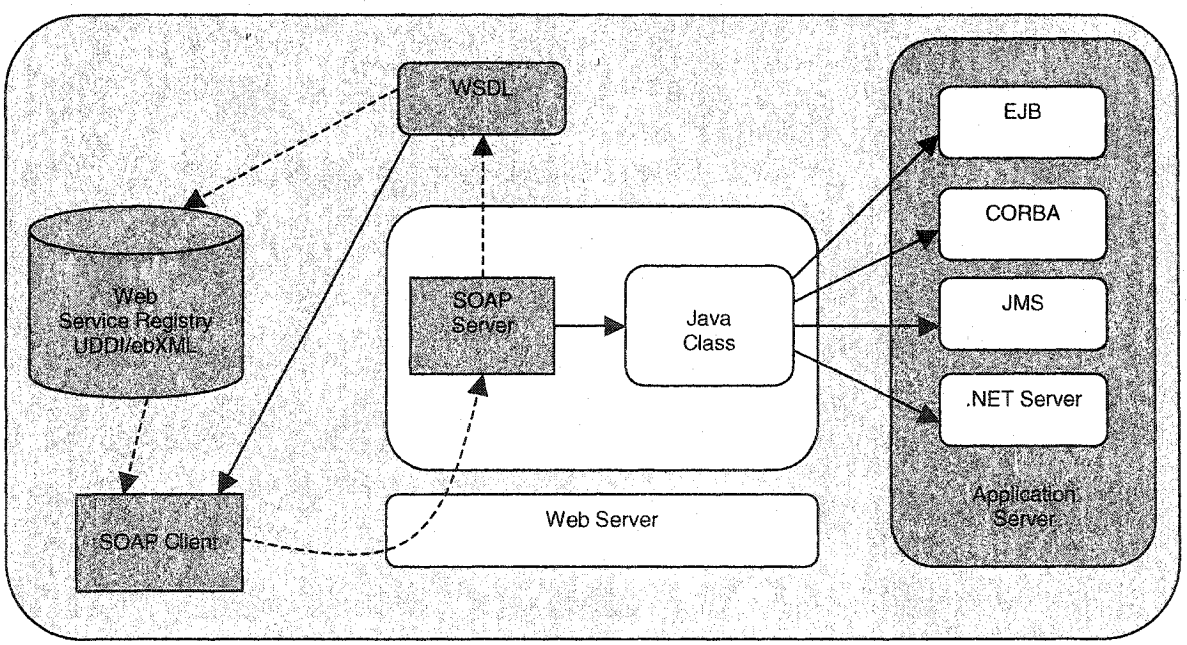


Figure 6-6: Web services and application servers

6.3.4.1 Java APIs for Web Services

J2EE includes new APIs for Web services. The J2EE standard defines Java APIs for enterprise services, such as security, messaging, transaction management, and directory lookup. The J2EE APIs are provided in application server vendor products from BEA Systems, HP, IBM, IONA, Oracle, Sun Microsystems, and others. Several efforts are under way to extend J2EE APIs for use with Web services. These APIs facilitate parsing and manipulating XML from within J2EE servers and provide access from J2EE servers to external Web services technologies, such as UDDI.

The Java Community Process (JCP) initiatives, led by Sun Microsystems, that are relevant to Web services comprise:

- *Java API for XML Parsing (JAXP)* - Provides users with pluggable APIs for XML parsing and transformation.
- *Java API for XML Messaging (JAXM)* - This specification defines Java APIs for exchanging business documents—XML documents and other arbitrary data—among trading partners.
- *Java APIs for XML-based RPC (JAX RPC)* - This specification defines Java APIs for SOAP and potentially other XML-based RPC-oriented protocols.
- *Java APIs for XML Registries (JAXR)* - This specification defines Java APIs for access to UDDI and ebXML registries for storing and retrieving information about a business's Web services and other interfaces.
- *The Java Web Services Developer Pack* - Provides a complete set of technologies for tools vendors to incorporate support for Web services in their products.
- *SourceForge ebXML Registry/Repository Open Source Project* - Sun Microsystems donated an internal implementation of the ebXML registry and repository to the open source community at SourceForge. A growing developer community from many companies and countries is working on this project (http://sourceforge.net/projects/ebxmlrr/).

6.3.5 The Differences in Approaches to Web Services

The .NET environment treats Web services as an integral part of the major pieces of .NET. .NET hosts Web services as client and server technologies equally. With the .NET approach, development and deployment of a Web service occur in a single stage. Once a program is written in any one of the .NET languages, it can be easily created and deployed as a Web service. Every program is a Web service to .NET, or at least potentially so.

The J2EE community, which relies upon a large vendor community consensus-process to define and incorporate new technologies, views Web services more as an evolution of application servers rather than a technology requiring any fundamental architectural change. For J2EE, adding Web services support means adding more APIs to an already long list of APIs that integrate applications servers with other, external technologies.

J2EE hosts EJBs within a server container, which can't host Web services. Therefore, a Web service that connects to an EJB, always requires a separate Java class "proxy" or wrapper. (This isn't necessary within the .NET environment.) The additional development step complicates design and development activities. For .NET, deciding to expose a server object as a Web service is one of several options supported natively within the VS.NET framework. For J2EE application servers, deciding to expose a server object as a Web service involves an additional, completely separate development and deployment step.

Within J2EE application servers, vendors are moving toward adopting the Java API for XML-Based RPC (JAX-RPC), a standard API for sending and receiving SOAP messages. The goal is to standardize the APIs developers use, as well as the formats and protocols for communication transports, configuration parameters, and data interchange formats. Application servers that conform to J2EE reassure customers that their investments in application development are

protected by, at least, the potential to port to another application server product, and assures them that their application server products work well with external standards.

In practice, customers don't often port applications from one application server to another after they've chosen one and deployed upon it successfully. However, customers tend to insist on J2EE conformance.

The .NET environment provides a single, generic mechanism for supporting Web services in multiple languages. The common language runtime (CLR) hosts objects or components developed in multiple languages using the same infrastructure, which is basically the .NET Framework. So far, VS.NET for C#, Visual Basic, and J# offer native support for Web services, and support for C++ Web services is provided through the Active Template Library (ATL).
ASP.NET, the recommended approach for creating Web services, and ADO.NET, the recommended approach for accessing data resources, also create objects that are hosted by the CLR, providing a consistent, seamless architecture.

NET provides a binary and XML version of its communication protocol. For J2EE, the binary protocol is RMI or RMI/IIOP, and the XML protocol isn't supported currently. J2EE requires developer to create a protocol bridge or gateway class to accept a Web services invocation over HTTP and call an EJB using RMI or RMI/IIOP.

The key advantage, perhaps, of using the .NET approach to Web Services is that it has been designed for that purpose, whereas J2EE is being retrofitted by the addition of further APIs. However, as of the maturity of the platform, J2EE has proven to be a robust, scalable
and a mature platform over the last six years. Addition of support for Web Services is just another feature for this platform. On other side, Although .NET inherits a lot of features from Windows DNA architecture, it is still relatively new and has to prove itself to be able to offer an enterprise-wide framework.

6.4 Building the Sample Application of Application Servers and Web Services

Our previous course registration applications were built on the COM/CORBA on distributed environments, which can only be accessed from inside the LAN network of the university. This application can be built as a trivial and not so real example of Web Services. In the sample application, student can use the university's registration portal to register courses and view their status and other related information. The front-end of the portal is built using Microsoft technology (ASP.NET, Internet Information Services (IIS) Web Server, VB Script, etc.). One of the features provided within the portal application is course registration. Using this feature, the students can retrieve real-time course registration information for any available course (assume the registration is based on first come first service). When a student requests a registration of a course, the request is sent from the browser to the Web Server.

As may potentially happen within any real enterprise application, it is assumed that the course registration is provided to multiple clients as a Web Service by a middleware application within the university, with the online registration portal being just one of those clients. Another client, as shown in the figure, is a VB application. The typical users of the VB application are the people of registrar office.

The information about Web Services offered by this middleware application is obtained from the private internal UDDI registry and invoked over the intranet. The implementation of the business methods exposed by the Web Service is provided by EJBs contained in another application server.

This is a typical example of .NET-J2EE application server integration using Web Services. The binding information for frequently used Web Services, such as those for course registrations, can

be cached by the client application, to avoid the resource intensive and time consuming dynamic binding. In this example, Web Services loosely integrates Microsoft technology-based application with the J2EE-based middleware application that interfaces with the Mainframe to receive the course information.

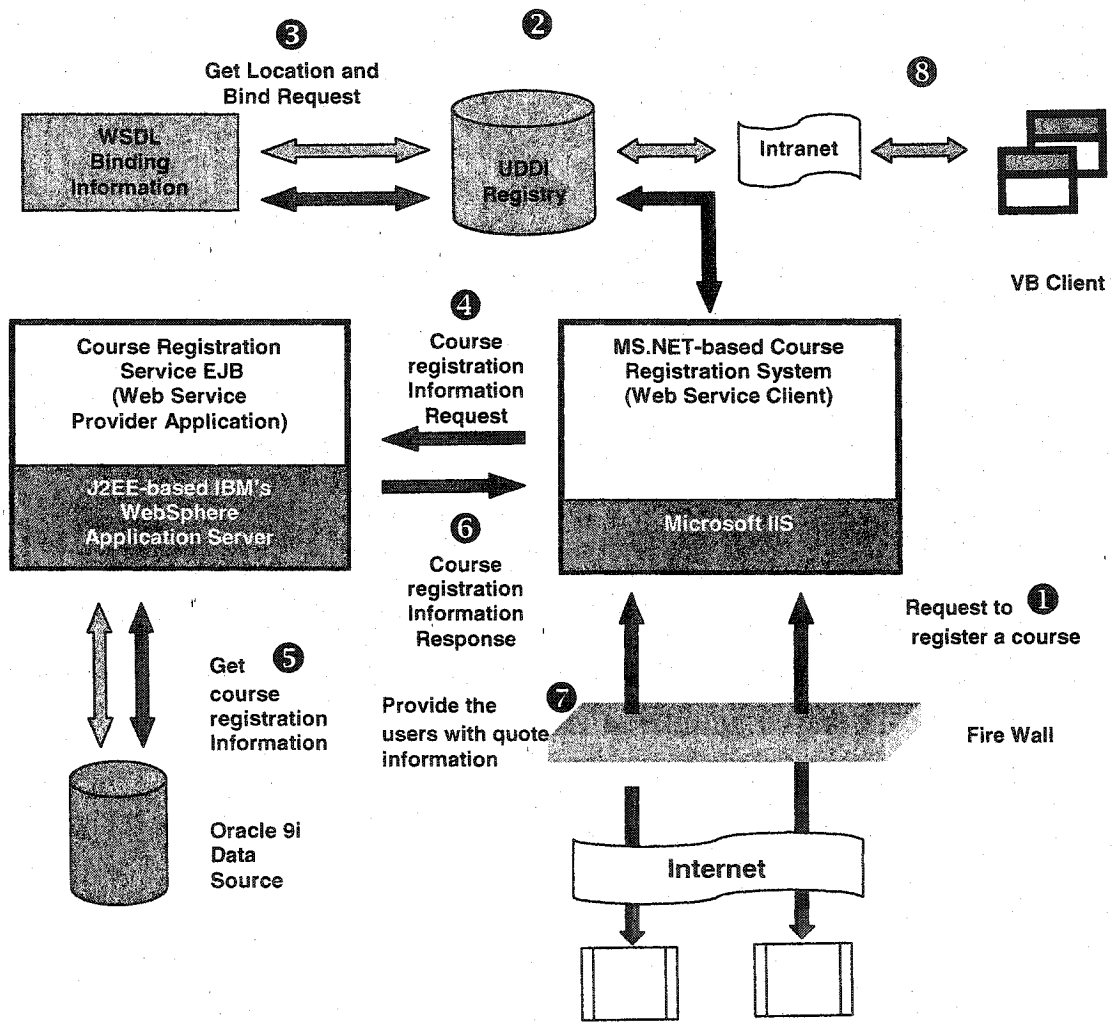


Figure 6-7: The Sample Web Services Application

The typical use case scenario and Web Service workflow is as follows:

1. The user requests to register a course on an ASP.NET/VBScript/HTML front-end that is passed over to the course registration portal running within Microsoft IIS.

2. The .NET-based portal application gets information about Web Services made available by the J2EE-based middleware application by looking for them in the private UDDI registry.
3. The location of and WSDL binding information for Web Services is sent to the portal application as a SOAP-based message.
4. The portal application invokes the Web Service published by the middleware application, passing course registration request as part of a SOAP-based message.
5. The actual implementation of the Web Service is provided by EJBs running within a J2EE-based application server. The EJBs use the JDBC API to get information from the data source, which in this case is Oracle 9i Database.
6. The EJBs send the Web Services response to the portal application as a SOAP-based message.
7. The response is formatted in HTML and sent back to the browser-based client application.
8. Another VB custom application within the university intranet invokes the same Web Service, thereby being another client of the course registration Web Service. The communication happens based on SOAP.

6.5 Conclusion

Both J2EE and .NET have their strengths and weaknesses. For organizations that must have platform neutrality, J2EE is the only alternative.

Great strengths of .NET lies in multiple languages support with a common set of class libraries; unified programming model with the Services Framework; high productivity development environment; Web application performance; runs on commodity servers that can be scaled out; separation of application logic and presentation exists; greater integration with XML and Web Services protocols; and tight integration with the dominant client platform (Windows).

.NET has its weaknesses. The OO nature of CLR makes development initially more complicated for VB/ASP developers. .NET is a single-vendor solution. In the area of data access, .NET does not support an automatic mapping of database rows to objects as with container-managed persistence (CMP), although there is widespread disagreement within the industry as to whether entity beans are desirable because they do not promote transactional consistency and are questionable in terms of performance.

J2EE systems are generally regarded as being more secure. This comes partly through Microsoft's focus on ease of use over security. J2EE is also architected for enterprise systems and so is currently more scalable.

The key weaknesses of J2EE are it's a single-language solution; smaller base of developers. EJBs are complex and not widely used. Thus, JSPs are overused, resulting in spaghetti code. Varying levels of J2EE conformance across vendors. The need to use vendor-specific features leads to vendor lock-in. Performance varies due to vendor-specific implementations. Software cost is generally higher, developer costs are higher, and deployment time is increased.

7 J2EE Vs .NET – Decision Making in Choosing for New Enterprise Software Development and Deployment

7.1 Background

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

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

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

Before examining the differences between J2EE and .NET, it is worthwhile mentioning that there are some key similarities between J2EE and the .NET Platform.

- Both J2EE and .NET architecture model brings the OO (Object Oriented) approach to mainstream enterprise computing, with powerful OO frameworks (class libraries) for services such as enterprise components management, object persistence, transactions,

Web services, asynchronous communication, loosely coupled event services, messaging and more.

- UML-based tools are typically used by both platforms because UML's object orientation is ideal for the J2EE and .NET implementation model.
- The use of a virtual machine (VM) architecture is common to J2EE and .NET. Application development tools produce intermediate-level code instead of platform-specific binary code, which means the VM interprets that code in real time or performs Just-In-Time (JIT) compilation.
- Both models are of inherently object-oriented natures that leads to a potentially steep learning curve for developers who are accustomed to the more traditional program-flow approach.
- J2EE and .NET share a broad common foundation that implements the multi-tiered approach. The high level conceptual architecture models for both platforms looks similar as shown in the following figures.

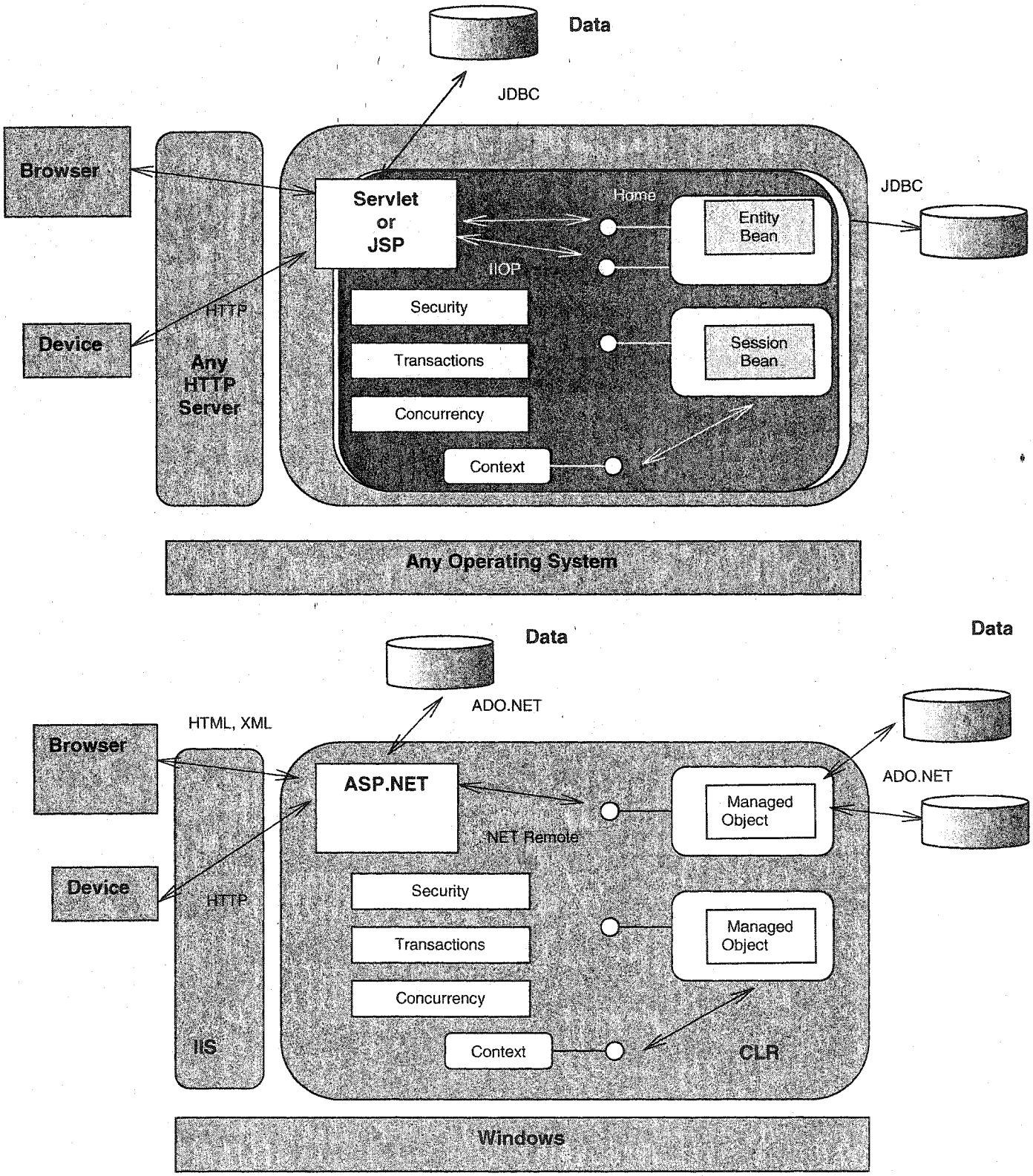


Figure 7-1: High level architectural similarities in .NET and J2EE

7.2 Scorecard

In this thesis, a scorecard is used as a tool to record of the analysis results when comparing the two platforms in each area. The rating is from 1 to 4, ranging from acceptable to excellent. For a given organization, this scorecard can be used a tool to evaluation the suitability of each platform. A weight for each area has to be defined according to the organization's real situation and priorities.

7.3 Vendor Neutrality and Platform Independency

	J2EE	.NET
Vendor Neutrality	••••	•

The.NET platform is not vendor neutral at this time; it is tied to the Microsoft. However, Microsoft has implemented an open-source policy for selected academic institutions. There is an on going progress to port .NET to non-Microsoft platform. An example is the Mono Project [18]. Mono was initiated by Ximian in July 2001, aims to give developers a set of open source tools for building .NET applications that can run on Windows or any Mono-supported platform, including Linux and UNIX. Incorporating key .NET compliant components, including a C# compiler, a Common Language Runtime just-in-time compiler and a precise garbage collection system, the Mono Project extends the functionality of .NET to the open source developer community. Commercial offerings from companies such as Tipic and OpenLink Software demonstrate the progress of Mono in enabling on Linux, a self-hosting C# compiler, CLR and web applications utilizing Microsoft .NET components, including ASP.NET and ADO.NET.

Java is the platform independent language with *Java Virtual Machine* (*JVM*) provided by *Sun Microsystems.* Java codes in *J2EE* are compiled to Java byte codes as in *J2SE*. The Java byte codes can run on any platform such as *Unix* (*Linux*) or *Windows* environment, in which the

platform has *Java Virtual Machine* (*JVM*) installed. Almost all platforms have their *JVMs* to make Java byte codes executable on them.

Currently, there are over thirty-seven vendors who license J2EE and at least seventeen who supply J2EE compatible application servers. The licensees include IBM, IONA Technologies, Apache Software Foundation, BEA Systems, Borland Corp.

The vendor neutrality topic in itself is hotly debated. One camp claims that the ability to choose and replace vendors is very important because an organization can become tied to a single supplier. The other camp claims that vendor neutrality is a myth. They claim that the J2EE model has not matured enough, and the specification is too incomplete to allow you to move a component from one application to another. The J2EE specification leaves many specifics up to the implementer; a wide variety of J2EE configurations and implementations are possible because the implementer has to fill-in the gaps. Thus, a J2EE component in IBM's WebSphere is not necessarily the same component in BEA's WebLogic. So, when a company uses a component that is tied to a specific vendor, the company is no longer vendor neutral. Thus, the reality today may be that there is no such thing as vendor neutrality.

The both camps have good points, and the decision maker has to decide how important vendor neutrality is to his organization. The organization needs to devise a plan to mitigate the risk.

The portability aspect of J2EE is attractive. This large number of vendors gives you the ability to find a supplier that will provide the best component to meet your needs at a competitive price. It also gives you leverage at the negotiating table because moving to a different vendor is always an option.

Cross-vendor portability for J2EE is definitely not the imaginary " Write Once, Run Anywhere (WORA) " scenario. This is due in part to its organic growth model, where vendors add their own

features and other extensions ahead of the J2EE specification process. J2EE vendor portability can be achieved by focusing on the parts of J2EE that are fully covered in the J2EE specification, such as Servlets, but doing so is currently very constraining and even impossible [22]. For example, EJB deployment dimensions aren't completely addressed in the J2EE specification [29].

7.4 Platform Maturity

	J2EE	.NET
Platform Maturity	●●●●	●●

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

The first Microsoft .NET equivalent "Microsoft DNA and MTS" came out in 1996, while the first J2EE specification came out in 1998 and first beta product in 1999. Microsoft .NET was the evolution of the Microsoft DNA, e.g. MTS to COM+ becomes an important piece of the .NET Enterprise Services. Microsoft .NET have delivered high volume and high reliable web sites such as Reuters, NASDAQ, BMO Financial Group, DELL and many others [16].

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

Table 3 shows the history of the evolution of the two platforms.

	1995	1996	1997	1999	2000	2002
Microsoft	COM	MTS ASP ADO	Windows DNA		.NET C# .NET	VS.NET
CORBA J2EE	Java	CORBA 2.0	JDK 1.1 EJB Java Servlet	JSP J2EE		CORBA 3.0

Table 3: Timeline of Microsoft and Java technologies

7.5 Scalability and Performance

	J2EE	.NET
Performance	●●●	●●●●

When considering enterprise architecture, the two most obvious places where system performance can be improved are data access and program speed. Both J2EE and .NET provide the ability to maintain business process state in memory and long-term data caching to reduce database traffic, and thus increase the performance of data access. As far as actual program speed is concerned, Java applications are often as fast as their native equivalents. .NET has also incorporated highly optimized, just-in-time compilers that produce code that often runs faster than native equivalents. .NET and Java both will cache the results of the JIT for quick access of programs that are run more often.

Scalability refers to the ability to add more workload and achieve maximum throughput. Its measurement is a function of the software platform and not the underlying hardware. Performance refers to the speed of a single unit of work under different workloads. Its measurement is a function of the software platform and not the underlying hardware.

All existing J2EE benchmarks published by J2EE vendors are analytical results which real systems may or may not be able to actually achieve. Because of the J2EE portability, it happens to run on different sets of hardware that should not be a factor in measuring platform scalability.

Sun released an enterprise application (basic data-driven Web application, distributed transaction benchmark, and the XML Web service benchmark) called "PetStore" to be the blueprint of the J2EE platform, Microsoft then released another version of the same application implemented on the .NET platform which showed an extremely higher performance/scalability compared to the J2EE version. The Middleware Company announced that it was not a fair comparison. Middleware re-wrote a second revised and fully optimized J2EE™ application and re-ran the comparison tests. The results of this study were varied across not only .NET vs. J2EE but also between the two J2EE application servers. In general the performance of the fastest J2EE application server was 66% of that of .NET. Here are some results [34] highlights:

WEB APPLICATION TEST RESULTS

The charts below show the results of testing the web application test scripts with increasing user loads for the five codebases against both databases, and the maximum throughput they achieved. Figures 7-2 and 7-3 show the results for the applications running using Oracle 9i. Figures 7-4 and 7-5 show the results for the applications running using Microsoft SQL Server.

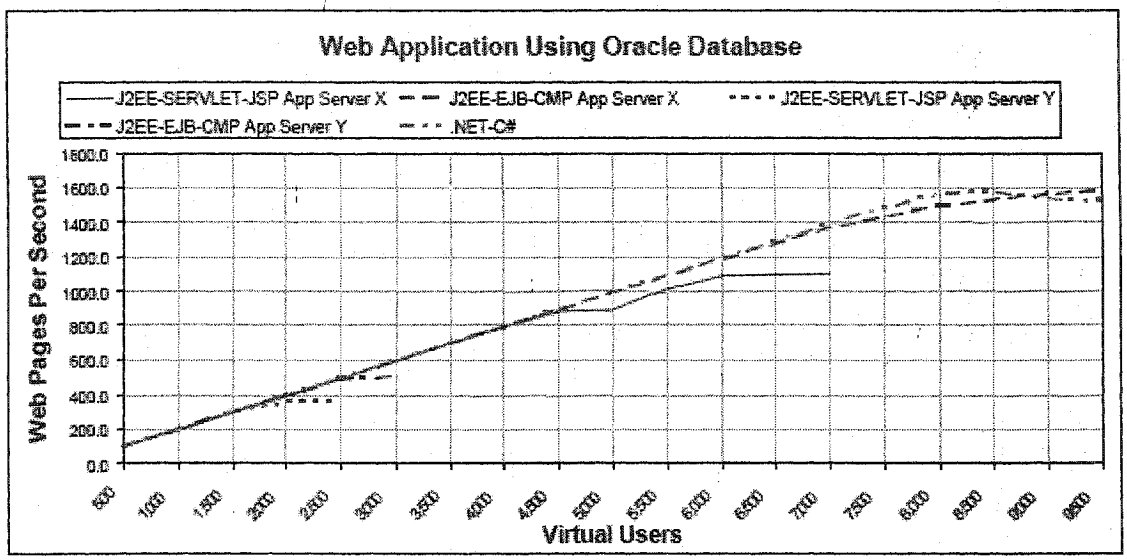


Figure 7-2: Throughput, web pages per second, increase as user load increases running the web application codebases using Oracle 9i database[34].

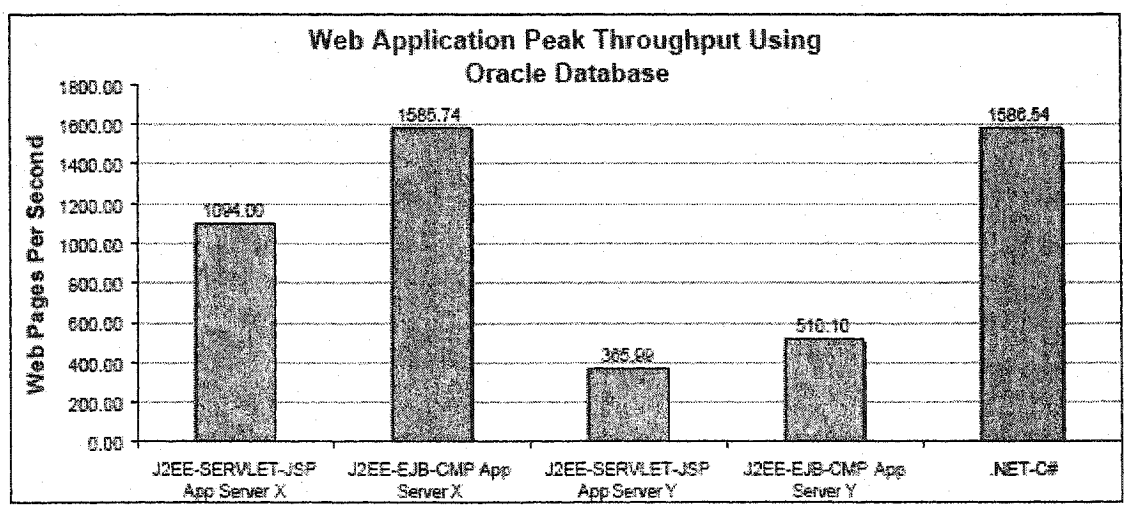


Figure 7-3: The maximum throughput achieved during the web application tests using Oracle 9i database[34].

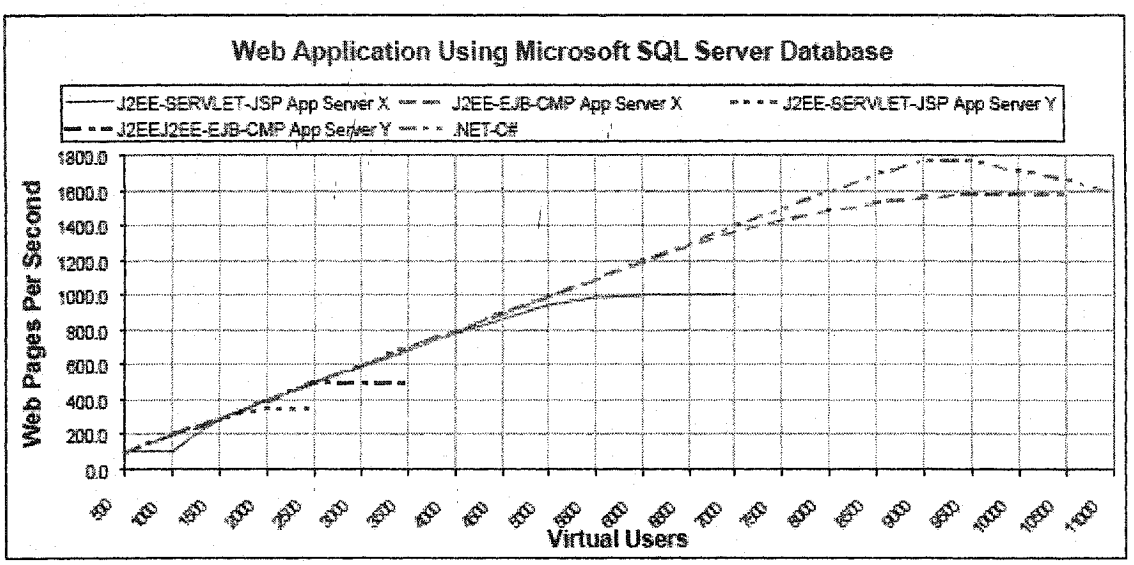


Figure 7-4: Throughput, web pages per second, increase as user load increases running the web application codebases using Microsoft SQL Server 2000 database[34].

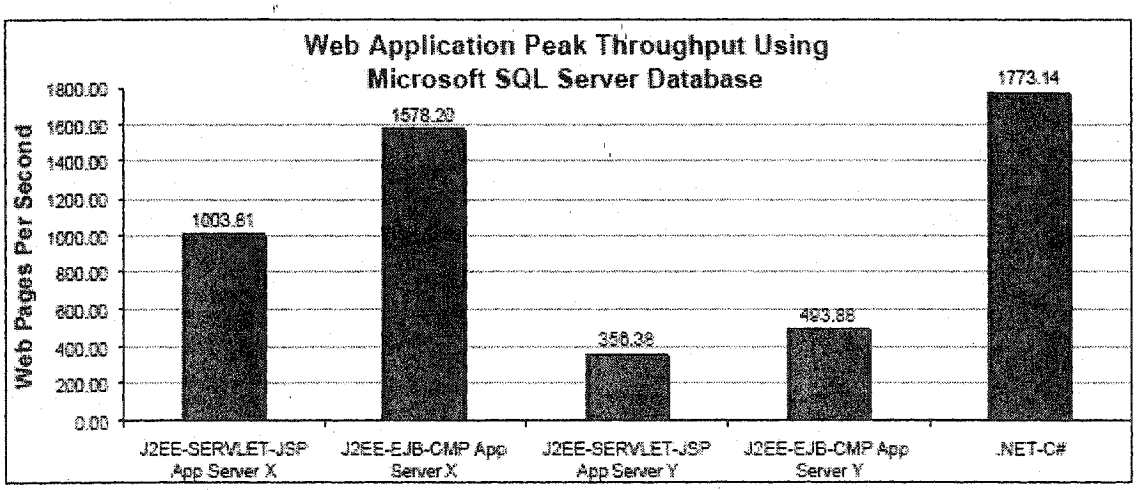


Figure 7-5: The maximum throughput achieved during the web application tests using Microsoft SQL Server 2000 database[34].

WEB SERVICES TEST RESULTS

The charts below, Figures 7-6 and 7-7, show the results of testing the web service test scripts with increasing user loads for the five codebases against their selected database, and the maximum throughput they achieved.

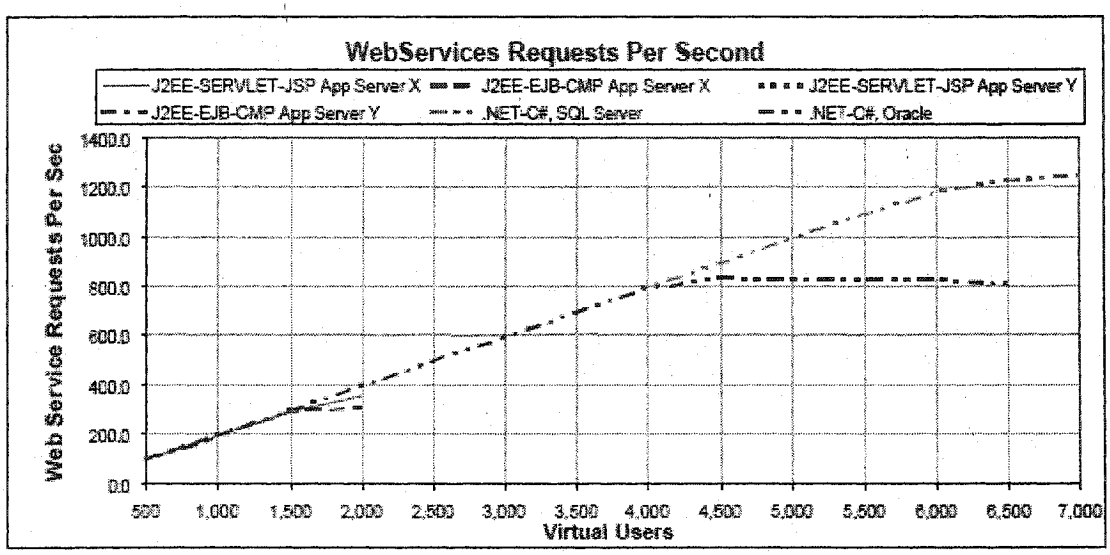


Figure 7-6, The throughput, web service requests per second, increase as user load increase for the various configurations[34].

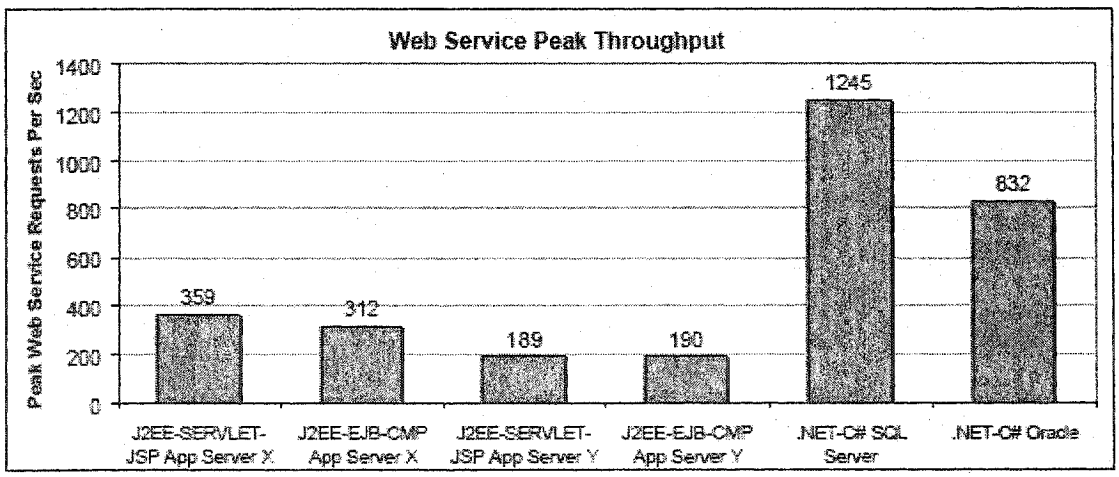


Figure 7-7, The maximum throughput achieved by each configuration during the web service tests[34].

7.6 Development

7.6.1 Programming language

	J2EE	.NET
Multiple Language	●	●●●●

J2EE only supports Java. Although both IBM's WebSphere and BEA's WebLogic support other languages; neither does it through their J2EE technology. There are only two official ways in the J2EE platform to access other languages, one through the Java Native Interface and the other through CORBA interoperability which Sun recommends. Building any new code in CORBA architecture is similar to building in COM in windows environment.

Microsoft .NET platform supports dozens of well-known programming languages and open to third parties to add more .NET enabled programming languages. Microsoft has it's own implementation of Java – J#. [20]. Multiple language support by .NET is clear advantage over J2EE.

7.6.2 Development Community

	J2EE	.NET
Development Community	●●●●	●●●●

Both .NET and J2EE have sizable and quality development community. By 2005, 80 present of all new IT application-development projects will be based on J2EE or Microsoft's proprietary .NET, according to Gartner, a technology market research firm based in Stamford, Connecticut.

.NET has a large potential development community, especially since Microsoft enfranchised COBOL and Perl developers with .NET. The quality of the existing Microsoft community is good.

The size of the existing Microsoft development community is huge, assuming all of those developers choose to get retrained in the .NET platform.

One sore spot for Microsoft is that .NET is completely object-oriented. This will result in a large number of developers needing to be retrained from being traditional Visual Basic programmers to object-oriented software engineers. This change will be difficult for many, but the final result will be a tremendous boon to the industry. Microsoft also hopes to attract new audiences through the .NET support for alternative languages like Perl and Python.

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

Unlike Microsoft, Sun does not really offer any support for training initiatives, other than the training that is offered by 3rd party ISVs. It is uncertain that the size of the J2EE community will increase because there is no prior precedent to compare it to.

In general Java programmers are paid about 25% higher salaries than equivalent VB or Cobol programmers. This will increase the development cost for J2EE solutions over .NET. Also, retraining of existing VB or Cobol programmers on Java constitutes very high cost, the other option will be outsourcing.

7.6.3 Tools Support

	J2EE	.NET
Tool support	••	••••

Microsoft has a long history of developing highly successful tools for developers. With .NET it has redeveloped its tools offering into Visual Studio .NET, designed to fully exploit the architectural changes in the Microsoft platform. Visual Studio .NET provides the developer with a dynamic interface for developing traditional client, server, and web based applications, as well as web services. Both web and desktop user interfaces are developed using drag and drop and the familiar Visual Basic "code behind", irrespective of language. Visual Studio .NET offers a single development environment for all of its supported languages – VB, C++, C#, Perl, Python, etc. Microsoft's strength in this area has been giving developers a seamlessly integrated development environment.

The explosion of developers in the Java market has created intense competition in the J2EE tools market. Unlike the Microsoft market where Visual Studio .NET dominates, the J2EE market has a wide variety of development tools available from IBM, BEA, Sun and others, as well as a great number available from the Open Source community. This competition has continued to push the improvement of J2EE tools.

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

7.7 Framework Support and Productivity

	J2EE	.NET
Productivity	••	•••

Framework support decreases development time and increases productivity by providing sets of libraries. The.NET platform includes an eCommerce framework called Commerce Server [13], this eCommerce platform helps developers to build solutions on top of a well defined and tested

eCommerce framework. The use of such a framework can dramatically reduce development costs. There is no equivalent vendor-neutral framework in the J2EE arena.

The.NET Frame includes a huge system libraries (called Namespaces) combined with the Visual Studio .NET dramatically reduce development time and thus cost. J2EE's best development tool IBM's WebSphereis out of competition. The Middleware Company benchmark tests mentioned above shows this superiority, table 4 lists lines of code needed to develop same enterprise application using these development tools [12].

Lines of code / Element	.NET PetStore 2.0	J2EE PetStore 2.0
User Interface	1,002	5,576
Middle Tier	795	6,187
Data Tier	197	197
Configuration	102	2,053
Total lines of code	2,096	14,004

Table 4: Line of Code comparison

7.8 Client Device Independence

	J2EE	.NET
Client Device Independence	•	•••

J2EE Java Applets are packaged code that run in browser and automatically installed on client (if enabled by client), this is analogous to the Microsoft ActiveX packages. Both have little usage because of client security restrictions that might be in place.

J2EE Java Servlets and Java Server Pages are analogous to the old Microsoft Active Server Pages. Both makes the presentation tier the programmer's responsibility to determine the

ultimate destination browser (or other thin client system), what the capabilities of that thin client system are, and how to generate HTML to best take advantage of that thin client system.

This Java approach has problems:

- It requires a lot of code on the presentation tier, since every possible thin client system requires a different code path.
- It is very difficult to test the code with every possible thin client system.
- It is very difficult to add new thin clients to an existing application, since to do so involves searching through, and modifying a tremendous amount of presentation tier logic.

The .NET approach is to write device independent code that interacts with visual controls. It is the control, not the programmer, that is responsible for determining what HTML to deliver, based on the capabilities of the client device.

7.9 Security

	J2EE	.NET
Security	●●●	●

J2EE and .Net both provide quite comprehensive security services, though with a different focus. Both platforms follow standards such as X.509 and Kerberos. A user or a group may secure code blocks in each platform. .NET has encryption built in, as does J2EE.

However, on taking a closer look one must consider that .NET is still relatively young compared with J2EE, which is widely used as an enterprise platform. Another point of interest is that Sun has a history of supporting security. Recall that Java started out with security as part of its original conception understanding that users would be downloading programs from an insecure Internet to run on their machines. Microsoft has only recently begun their focus on "Trustworthy Computing". According to the OS Vulnerability report performed by a London-based technology

risk management company Mi2g Ltd, Windows OS were the most vulnerable to attack and damage from hackers, worms and viruses in 2002.

Upon further analysis of security, one must consider the primary programming language of both platforms: Java and C#. Both of these languages were created with security in mind, and both have similar ways of enforcing security, including garbage collection and namespaces.

The most significant difference between J2EE and .NET security is that .NET enforces security through an "evidence-based" system that determines whether code is safe based on where it comes from. Java relies on a sandbox security model using runtime security checks.

Another difference to note from a security point of view reaches into the heart of Sun's business model and Microsoft's business model. Sun has always encouraged open source systems, and the Java Virtual Machine specification is publicly available. Microsoft, on the other hand, keeps the components of the Common Language Runtime locked down.

Both platforms use similar concepts for handling user and code access to resources, with permissions being key to both, and the concept of roles being used to associate permissions with principals in both environments. While J2EE uses the concept of Organizational Roles to delineate responsibilities at various stages of the development and deployment process, .NET does not define the hierarchy as clearly.

While .NET provides a solid security model through managed code in the CLR, the ability to run unmanaged code confers the ability to bypass CLR security through direct calls to the underlying OS APIs. In the Java world, signed, trusted code has unrestricted access to system resources. Java calling out to native (C/C++) code through JNI confers the ability to bypass the JRE's security as surely as running unmanaged code in the CLR can be used to bypass .NET's security.

Java has also had its share of security problems in the past, especially in the area of certain JREs and malicious applets; problems have also been reported in the class loading process, a fundamental part of JVM security.

One of the crucial challenges for both Microsoft and J2EE vendors when developing their respective platforms is securely handling code obtained from multiple sources, outside of the local machine. The code verification functions of the JVM are quite mature at this stage and mistakes made in the past have been learned from. The CLR model is similar, but the implementation is relatively untested. It will be interesting to see how the .NET environment stands the test of time from a security perspective, once .Net deployments become more widespread.

7.10 Legacy Application Integration

	J2EE	.NET
Legacy Integration	••••	•••

J2EE legacy integration support: [17]:

- 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.
- 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 others.

.NET legacy integration support:

- 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.
- BizTalk Server 2000 can be used to integrate with systems based on B2B protocols, such as Electronic Data Interchange (EDI). Also provides integration connectors to SAP, Siebel, Onyx, and J.D. Edwards, and more.

7.11 Portability

	J2EE	.NET
Portability	●●●●	●

J2EE server applications support operating system portability. .NET server applications only run on Windows. The path to achieve J2EE portability is to stick with a given J2EE vendor, this is due to differences in vendor's implementation extensions of J2EE specification. The portability is beneficial to the Independent Software Vendors (ISVs). The software vendors cannot impose the operating system constraint to their clients and thus need to provide the solutions which can be easily ported to many platforms. Simply saying, the software vendor will not be able to sale their product written in .NET to a client who is only using UNIX.

7.12 High Server Availability

	J2EE	.NET
High Server Availability	●●●	●

The J2EE platform is deployable onto reliable operating environments that support high availability. While some J2EE deployments will be on Windows, customers also have the choice

of deploying J2EE solutions on more robust operating system environments, such as Sun's Solaris Operating Environment or IBM OS/390. The most robust operating system environments can reach levels of 99.999% availability, or as little as 5 minutes of downtime per year. This level of availability is ideal for businesses building mission-critical commerce systems.

Deployments based upon Microsoft technology have historically been unable to reach this level of availability. .NET is tied to the Windows operating system, which has been criticized in the past for unreliability.

7.13 System Cost

	J2EE	.NET
Lower System Cost	••	•••

Microsoft created a .NET Pet Store application using Visual Studio and further extended with a Web Service. They also extended the Java Pet Store application with the same Web Service using IBM Websphere 4.0 The result shows that the cost of the licensing requirement to deploy J2EE is extremely higher than deploying .NET solutions.

Metric	Visual Studio .NET	IBM WebSphere 4.0
Estimated software purchase costs to deploy Web Services on a single server with full authenticated per-user access	$5,998 1 x 8-CPU Application Servers to Host Web Services running Windows 2000 Advanced Server with .NET Framework @ $3999.00 per server + 1 Internet Connector License @ $1,999 per server.	$64,000 1 x 8-CPU Application Servers to Host Web Services running IBM WebSphere 4.0 Advanced Single Server Edition (on Solaris, W2k Or AIX) @ $8,000 per CPU
Estimated software purchase costs to deploy Web Services to a 4-server Cluster	$23,992 4 x 8-CPU Application Servers to Host Web Services running Windows 2000 Advanced Server with .NET Framework @ $3999.00 per server + 4 Internet Connector Licenses @ $1,999 per server.	$384,000 4 x 8-CPU Application Servers to Host Web Services running IBM WebSphere 4.0 Advanced Edition (on Solaris, W2k or AIX) @ $12,000 per CPU

Table 5: Comparison of development cost for a sample Web Service

However according to a recent IBM white paper [27], which reflects IBM's overall marketing plan to aggressively combat Microsoft, IBM Middleware Platform provides an affordable entry point to e-Business on demand, and it is competitive on a short-term basis as well as long term. IBM provides express solutions, which are designed, developed and priced exclusively for mid-market businesses and include hardware, software, services and financing offerings. Unlike similar Microsoft middleware offerings, IBM's Express offerings include one year of maintenance and support while providing the customer with the greatest flexibility to leverage existing IT resources. With Microsoft, customers need to pay money each time they have a question. In addition, customers must purchase a separate software assurance package to be entitled to upgrades of the product. All of this is included in IBM's competitive price.

IBM's Express products portfolio are also priced very competitively versus similar Microsoft middleware offerings as illustrated in the below pricing scenarios.

Inventory Management Provide support for 50 employees to inventory management system plus Internet access for customers to track order status.	**WebSphere Application Server – Express** **$2,000** • Unlimited users • Maintenance & upgrade protection • 1 set of developer tools • Technical Support	**Microsoft Windows 2003 Server** **$5,845** • Upgrade access to 50 additional users (10 included with Windows 2000 Server) • Maintenance & upgrade protection • 1 set of developer tools • No technical support
Employee HR Portal Provide access to Human Resources portal supporting 200 registered users.	**WebSphere Portal – Express** **$17,000** • 200 users • Maintenance & upgrade protection • Technical Support	**Microsoft Sharepoint Portal Server** **$22,749** • 200 users • Maintenance & upgrade protection • No technical support
Business Integration Provide distributor ability to execute exchange business information with small number of partners.	**WebSphere Business Integration Connect – Express** **$3,750** • Gateway & support for 5 partner connections • Browser based tooling for definition and management • Maintenance & upgrade protection • Technical Support	**Microsoft BizTalk Server 2002 Standard Edition** **$14,997** • Gateway & Support for 10 partner connections • SQL Server Standard Edition • Maintenance & upgrade protection • No technical support
Basic Commerce Site Deliver a basic B2B or B2C e-commerce site, one-processor license plus pre-production staging environment.	**WebSphere Commerce– Express** **$20,000** • Production & staging server • Database (DB2 Express) • Developer license • Maintenance & upgrade protection • Technical Support	**Microsoft Commerce Server Standard** **$26,795** • Production & staging server • Database (SQL) • Developer license • Maintenance & upgrade protection • No technical support

Table 6: Cost comparison between IBM's Express products portfolio and Microsoft's.NET suite products

7.14 Web Services

	J2EE	.NET
Web Services	••	•••

Microsoft initiated industry focus on Web services. Microsoft's strategic .NET initiative is highly Web-centric. J2EE views Web services more as an evolution of application servers, rather than a revolution. J2EE includes new APIs for Web services. For .NET, deciding to expose a server object as a Web service is one of several options supported natively within the VS.NET framework. For J2EE application servers, deciding to expose a server object as a Web service involves an additional, completely separate development and deployment step.

7.15 Risk (Stability)

	J2EE	.NET
Stability	••••	••

The Java Platform is an industry standard, supported by the all the largest software vendors. .NET is not a standard but purely a proprietary framework. Microsoft architecture is good. The implementation is where the problem lies. Microsoft, quick to market the new technologies but stability does not enter the offering until the fourth or fifth release of the product. For instance, Microsoft MFC was released five times, and the instance pooling was documented in MTS but only fully supported COM+.

7.16 The overall ratings

Tables 7 shows the overall results of the analysis.

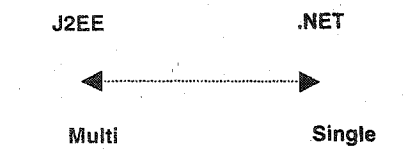
Criteria	J2EE	.NET
Vendor Neutrality	●●●●	●
Platform Maturity	●●●●	●●●
Performance	●●●	●●●●
Multiple Language	●	●●●●
Development Community	●●●●	●●●●
Tool support	●●	●●●●
Framework Support/Productivity	●●	●●●
Client Device Independence	●	●●●
Security	●●●	●
Legacy Integration	●●●●	●●●●
Portability	●●●●	●
Lower System Cost	●●	●●●
Web Services	●●	●●●
Stability	●●●●	●●
High Server Availability	●●●	●
Overall	**72%**	**68%**

Tables 7: Scorecard of the comparisons

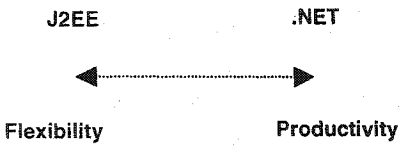
7.17 Assess the Suitability by Questions

To take the business decision on J2EE or .NET, the company needs to consider the following questions.

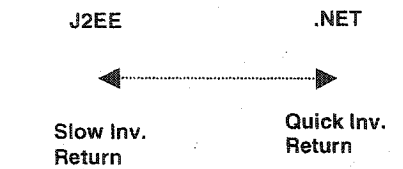
- ❑ How important is multi-vendors support and cross-platform deployment?



- ❑ How important is productivity vs. flexibility?



- ❑ What are the established skill sets of your enterprise's developers?
- ❑ With which software and system vendors does your enterprise have ongoing and long-term relationships?
- ❑ Can you afford to recoup ROI benefits over time, or do you demand a quick return on your investment?



- ❑ How large is your enterprise, and your development organization?
- ❑ Are you willing to through out your previous software investments?

The answers to these questions can be made based on our previous analysis and the ratings in scorecard. For a given organization, the business requirements can be converted to list of concrete questions. The choice will of course ultimately be dependent on the prevalent conditions in the organization. The considerations above should give a number of pointers as to how each of these conditions should be weighted in making decision.

7.18 How J2EE and .NET will Evolve

Trends of shifting to J2EE and .NET Programming model

By 2005, 80 present of all new IT application-development projects will be based on J2EE or .NET, according to Gartner, a technology market research firm based in Stamford, Connecticut.

Mainstream industry community support

The industry community support for J2EE will continuously to be strong, whereas .NET will high likely fall short of building community support among third-party vendors.

Market Orientation

J2EE dominates high-value enterprise deployment today and is expected to retain the lead in these segments. Microsoft .NET is working to gain presence and will do so.

Impact on Microsoft Developers for .NET Migration

Migration to .NET for Microsoft developer is a question of "When" not "If". There are some strategy considerations for migration plan including platform Immaturity, developer skill migration, and learning curve of developers.

Impact on Java Developers for J2EE Migration

Vendor product strategies are shifting to "one-stop shopping" strategies of these buyers, and this will subsequently increase the degree of vendor-specific technology. The company should carefully chose an enterprise Java technology provider. The Java market is moving into a consolidation phase, during which a small number of vendors will dominate the market.

7.19 Conclusion and Recommendations

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

Small-to-midsize businesses should focus on one platform. Microsoft will offer the most-attractive technology to these enterprises. Microsoft based solutions are generally more suitable for smaller companies, which need simpler lighter systems with less need for scalability. This is both in terms of the type of hardware supported, and in the level of flexibility available to modify low level aspects of the platform for specific needs.

Large enterprises must support both platforms and should focus on integration technologies. Java platform will generally be more attractive. Application integration strategies must be developed to manage interoperability between the technologies.

For mission-critical application, J2EE is definite a better choose.

Emerging Web services models have begun to dominate new e-business development initiatives for both platforms. Whether choose .NET, J2EE, or even choose to wait a while, there are a few things a company can begin to do today that will help the organization move in the right direction. First, the company should begin to train and equip developers to use XML. Second, then should begin moving the internal systems to using XML as a communication mechanism at a minimum, and for data storage when appropriate. Finally, the company should begin to insist that its vendors and business partners make the transition to web services.

References

[1] [COM 95] Microsoft Corporation. *The Component Object Model Specification*, Version 0.9, October 24, 1995. http://www.microsoft.com/Com/resources/comdocs.asp (1995).

[2] [DCOM 97] Microsoft Corporation. *Distributed Component Object Model Protocol-* DCOM/1.0, http://www.microsoft.com/Com/resources/comdocs.asp , draft, November 1996.

[3] The Common Object Request Broker: Architecture and Specification. OMG (http://www.omg.org/), Revision 2.6 December 2001

[4] Sun's Java page (http://java.sun.com/)

[5] BEA's CORBA page. http://e-docs.bea.com/wle/interm/corba.htm

[6] IONA's CORBA product page, ORBIX (http://www.iona.com/products/orbix.htm)

[7] *Borland unveils C++ application development strategy for 2002* (http://community.borland.com/article/0,1410,28296,00.html) , David Intersimone, January 30, 2002

[8] *A Guide to Reviewing and Evaluating Microsoft Transaction Server* (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmts/html/msdn_mtsrevgd.asp), Microsoft Corporation

[9] Microsoft's .Net page: http://www.microsoft.com/net/

[10] Sun's J2EE page (http://java.sun.com/j2ee/)

[11] *Java™ 2 Platform Enterprise Edition Specification, v1.4 –Final Release* (http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf), November 24, 2003

[12] A Microsoft-sponsored site on .Net (http://www.gotdotnet.com/)

[13] Microsoft eCommerce site http://www.microsoft.com/commerce

[14] *Java 2 Platform Enterprise Edition Specification*, http://java.sun.com/j2ee/

[15] Sun's J2EE Tutorial, http://java.sun.com/j2ee/learning/tutorial/index.html, April 24, 2002

[16] Microsoft .NET case studies http://www.microsoft.com/net/casestudies/

[17] Sun Partner's and Solutions (http://solutions.sun.com/)

[18] Ximian™ and the Mono Project (http://developer.ximian.com/projects/mono/)

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

[20] Microsoft's implementation of Java -.NET J#" (http://www.microsoft.com/jsharp)

[21] Authorized Licensees of the J2EE Platform http://java.sun.com/j2ee/licensees.html

[22] *Java 2 Enterprise Edition (J2EE) versus The .NET Platform - Two Visions for eBusiness,* Roger Sessions ObjectWatch Inc.(http://www.Objectwatch.com), March 28, 2001.

[23] *Rumble in the jungle: J2EE versus .Net, Part 1 & 2 ,* Java World, (http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevsnet.html), Humphrey Sheil and Michael Monteiro , June 21, 2002

[24] *The Great Debate: .Net Vs. J2EE,* Java World, (http://www.javaworld.com/javaworld/jw-03-2002/jw-0308-j2eenet.html), Jonathan Lurie and R. Jason Belanger, March 8, 2002

[25] *J2EE vs. Microsoft.NET - A comparison of building XML-based web services* (http://www.theserverside.com/articles/article.tss?l=J2EE-vs-DOTNET), Chad Vawter and Ed Roman, June 2001

[26] *Developer's Guide to Building XML-based Web Services with the Java 2 Platform, Enterprise Edition (J2EE)* (http://www.middleware-company.com/), James Kao, June 2001, Prepared for Sun Microsystems, Inc.

[27] IBM J2EE Middleware Platform White paper - *IBM J2EE Middleware Platform vs. the Microsoft .NET Platform* (http://www-1.ibm.com/linux/files/middleware_final7.pdf?ca=vgr-isv02NetvsJ2EE), January 2004

[28] *To EJB, or Not to EJB?* JavaWorld(http://www.javaworld.com/javaworld/jw-12-2001/jw-1207-yesnoejb.html), Read Humphrey Sheil December 2001.

[29] *Building a Large, Integrated, Multi-EJB Server System,* Paul Harmon.

[30] *The Evolution of Web Service*, Graham Glass, June 2002

[31] *Building and Integrating XML Web Services with Visual Studio .NET vs. IBM Websphere 4.0 (* http://www.gotdotnet.com/team/compare/webservicecompare.aspx)

[32] *An Overview of J2EE with IBM WebSphere* by ADDISON-WESLEY, Jim Amsden, Daniel Berg et al, May 13, 2004

[33] *The Great Debate: .Net Vs. J2EE*, JavaWorld, (http://www.javaworld.com/javaworld/jw-03-2002/jw-0308-j2eenet.html), Jonathan Lurie and R. Jason Belanger, March 2002,

[34] *J2EE and .NET (RELOADED) Yet Another Performance Case Study* (http://www.middleware-company.com/casestudy/), June 2003

[35] "Advanced CORBA Programming with C++", Published by Addison-Wesley, Michi Henning, Steve Vinoski. July 1999

[36] ActiveX Controls Home page (http://www.microsoft.com/com/tech/activex.asp), Microsoft Corporation

[37] *What is Middleware?* (http://java.about.com/b/a/099316.htm), Kevin Taylor, July 2004

[38] Rotor Project (http://www.research.microsoft.com/programs/europe/rotor/), Microsoft Corporation.

[39] JOVE (http://www.instantiations.com/jove/product/productdetails030314.htm), Instantiations, Inc.

[40] IBM WebSphere MQ (http://www-306.ibm.com/software/integration/wmq/), IBM

[41] Microsoft Host Integration Server (http://www.microsoft.com/hiserver/), Microsoft Corporation.

[42] JDBC Overview (http://java.sun.com/products/jdbc/overview.html), Sun Microsystems.

[43] JDBC Data Access APIs (http://servlet.java.sun.com/products/jdbc/drivers), Sun Microsystems.

[44] Microsoft Developer Network (MSDN), Microsoft Corporation.

[45] Java API for XML Processing (http://java.sun.com/xml/jaxp/), Sun Microsystems.

[46] XML Parser for Java (http://www.alphaworks.ibm.com/tech/xml4j), IBM

[47] Microsoft annual report, 2001 (http://www.microsoft.com), Microsoft Corporation.

[48] Professional Java Server Programming, Subrahmanyam Allamaraju et al., Wrox, 2002