

Technologies for Connecting and Using Databases and Server Applications on the World Wide Web

by

Adolfo G. Castellon Jr.

Submitted to the Department of Electrical Engineering and Computer Science on May 23,
1997, in partial fulfillment of the requirements for the degree of Bachelor of
Science in Computer Science

Abstract

This paper presents a study of current technologies used to build applications that make use of the World Wide Web. In particular, this paper discusses three different technologies (Java Beans, OLE/ActiveX and CORBA) born of very different heritage, that are evolving towards a common goal. The emphasis is on technologies that have been recently developed to connect databases to Web applications. Two applications created by the author are used to demonstrate specific types of emerging web technologies.

Thesis Supervisor: Dr. Amar Gupta

Title: Co-Director, Productivity from Information Technology (PROFIT) Initiative

Table of Contents

1	Introduction	3
1.1	Overview	3
1.2	Conventions: How to Read This Document.....	4
2	A Brief History of Time	5
2.1	Structured English Query Language (SEQUEL)	5
2.2	Powerpoint and the DLL problem.....	6
2.3	Object Management Group(OMG).....	6
2.4	The Set-Top Box and the OAK project.	7
3	The Tools of the Trade	9
3.1	The Standardized Interface Approach.....	9
3.1.1	Structured Query Language (SQL).....	9
3.1.2	Open DataBase Connectivity (ODBC)	10
3.1.3	Open Data Sources (ODS)	10
3.1.4	Common Gateway Interface (CGI)	11
3.2	Object Management Architecture (OMA).....	11
3.2.1	Common Object Request Broker Architecture (CORBA).....	13
3.2.2	OpenDoc and Fresco	14
3.3	Component Object Model (COM).....	14
3.3.1	Object Linking and Embedding (OLE).....	16
3.3.2	ActiveX	16
3.4	Java.....	17
3.4.1	JavaBeans.....	17
3.5	Extensions:.....	19

3.5.1	Java DataBase Connectivity (JDBC).....	19
3.5.2	InfoBus	19
3.5.3	Distributed COM (DCOM).....	19
3.5.4	OleDb/ActiveX Data Objects (ADO).....	20
3.5.5	General Overview.....	20
4	Example Applications	22
4.1	Active Server Pages: The Career Server (ADO, SQL)	22
4.1.1	Motivation.....	22
4.1.2	Typical Session at the Career Office	23
4.1.3	Typical Session at the Career Server.....	24
4.1.4	Functionality of the Career Server.....	25
4.1.5	The Implementation of the Career Server: Active Server Pages	25
4.1.6	ActiveX Data Objects and SQL in the Career Server	26
4.1.7	The Database (Microsoft Access 97)	27
4.1.8	Limitations and Issues	27
4.2	Merchant Server: Image Gallery (COM, SQL, ActiveX)	28
4.2.1	Motivation.....	29
4.2.2	Typical Purchasing Session at a Traditional Store	29
4.2.3	Typical Purchasing Session at the Image Gallery	29
4.2.4	Functionality of the Image Gallery	30
4.2.5	COM, SQL and ActiveX in Merchant Server.....	32
4.2.6	Limitations and Issues	32
5	Is It Worth It?.....	33
5.1	Making Money on the Internet is Hard	33
5.2	Where These Systems Are Useful	34
5.3	Is There Anything Beyond the Hype?.....	34
Appendix A	Background Material.....	36
A.1	The OMA Object Services	36
Appendix B	Lists and Raw Data	37
B.1	List of Figures	37
B.2	List of Tables	37
B.3	ASP Script: The Career Server.....	37
B.4	HTML Extended With Directives: Image Gallery	50
Bibliography	56

Section 1

Introduction

1.1 Overview

This paper focuses on three object technologies: Java Beans, OLE/ActiveX and CORBA. Born of very different heritage, these technologies are evolving towards common goals of network connectivity, World Wide Web applicability and emphasis on a distributed object model. The role of object technology in assisting the migration of traditional applications like databases towards integration with the World Wide Web is also discussed in this paper.

Section 2 provides a brief history of the origins of current standardization techniques and object models discussed later in the paper. It is important to note that all three evolving object models, Java Beans, CORBA and OLE/ActiveX, emerged from very different design requirements and were originally designed for applications very different from what they are now becoming popular for.

Section 3 contains descriptions of a number of technologies relevant to and including Java Beans, ActiveX and CORBA. Section 3.1 describes four standardization efforts that are both the basis for many of the popular object technologies as well as the means for creating multi-part, distributed applications using only static interfaces. This section is included to provide a common basis to compare Java Beans, ActiveX and CORBA.

Section 4 describes two example applications built by the author using some of the tools described in section 3. This was done to explore how useful and efficient the object models are by using existing OLE and ActiveX objects to create example applications like those that may eventually be created by serious developers on a larger scale.

Section 5 provides some statistics and commentary on the applications that developers are creating using these object technologies, the effectiveness of the object models themselves and the expected impact of these tools on the World Wide Web.

1.2 Conventions: How to Read This Document

Many of the names used in this paper are long and awkward to read. In addition, many of these topics are better known by their acronyms than by their true names. The acronyms for these names have been used wherever applicable. Further, the acronyms are placed next to their full names in the table of contents in order to provide a quick reference for the reader.

The reader may notice that the examples implemented are restricted to Microsoft products and object models. This is wholly due to the fact that these were the most available tools at the time that this paper was written. The implementation of these examples is not meant to be an endorsement for any technology over another. This is especially important given the fact that many of the technologies discussed in this paper are in their infancy. Many of these tools have neither been proven by many years of research nor by prolonged use in industry.

Section 2

A Brief History of Time

2.1 Structured English Query Language (SEQUEL)

The history of SQL begins with a paper by E. F. Codd, a member of the IBM Research Laboratory in San Jose, California. Published in 1970. The paper, "A Relational Model of Data for Large Shared Data Banks"[1], laid down a set of principles defining the relational model of database management. This paper encouraged significant experimentation on the relational database model and activities related to database design. Further research by members of the IBM San Jose Research Laboratory led to the development of a language that realizes the features of the relational model. This language was called the "Structured English Query Language" (SEQUEL). The language was revised for use with IBM's System R database system and renamed SQL for legal reasons. Other companies including Relational Software Inc. (later called Oracle Corporation), Data General Corporation, Relational Technology Inc. and Britton-Lee also created versions of SQL which were not necessarily compatible with IBM's original SQL implementation. Eventually, the American National Standards Institute (ANSI) and the International Standards Organization (ISO) defined a standard for SQL. Later, this standard was extended in SQL2. Another organization that was important in standardizing SQL was the SQL Access Group.

ODBC

One of the SQL Access Group's contributions to standardized interfaces was the Call Level Interface (CLI) specification. This specification and the specification for UNIX standard SQL (called X/OPEN) was part of the basis for Microsoft's Open DataBase Connectivity (ODBC) standard, which connects Windows-based databases through a common programming interface.

2.2 Powerpoint and The DLL Problem

When a team of researchers in Mountainview, California began building PowerPoint, a program for creating presentations, they probably had no idea that they would be helping to hatch the major innovation that OLE has now become. In creating PowerPoint, the team began building their own graphing software. However, Bill Gates forbade this idea as Microsoft had already created graphing software for use with the Excel spreadsheet software. As a result, software components were developed that allowed objects to be linked and embedded in other applications. For example, PowerPoint could hand off graphing information to an Excel component which does the actual rendering. This led Microsoft to develop the Component Object Model (COM) and begin the development of Object Linking and Embedding (OLE).

At the same time that OLE 1.0 was being developed, Microsoft was grappling with the problem of new versions of dynamic-link libraries (DLLs) causing general protection faults in old applications that relied on the original DLLs. The solution of the DLL evolution problem led to the upgrade of OLE 1.0 to OLE 2.0 in 1993.

After Netscape Communications Corp. began making news with its Web browser, Microsoft decided that it needed to get into Web technology. After adding a few new Web compatibility features and a great deal of hype, OLE 2.02 was renamed ActiveX and released to the public.

2.3 Object Management Group (OMG)

CORBA, however, has a far more structured history. In 1989, The Object Management Group was established as a consortium of eight companies: 3COM Corp., American Airlines, Canon Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems and Unysis Corporation (See [2]). The purpose of this consortium was to usher in a new era of software sales by providing standardized object software with which different developers could create marketable software components. Needless to say, this new era of software component markets is not quite here yet, but still on its way. The OMG now consists of over 570 members, making it the largest software development

consortium in the world. The OMG created the Object Management Architecture (OMA), from which CORBA was developed.

After the emergence of OLE's extension to COM, developers began trying to extend CORBA's basic model with something like OLE. These efforts include OpenDoc and Fresco.

2.4 The Set-Top Box and the Oak Project

Long before Javasoft, or even before Marc Andreessen made millions with Netscape Navigator, Sun Microsystems was attempting to create a new consumer electronics device called the set-top box. Similar to a cable box, the set-top box was designed to communicate with various house-hold electronics. However, unlike a cable box, the set-top box was supposed to be a fully digital instrument, allowing sophisticated control of one's TV, VCR and just about any other digital appliance with an on/off button. Of course, a new device such as the set-top box required a language to program it. After deciding that C++ was too bulky for the task, the members of the Sun team decided to come up with their own language, Oak.

Unfortunately for Sun, the set-top box was a dud. Sun tried to sell Oak as a language to program Personal Digital Assistants (PDA's). Unfortunately, Apple Computer's unmitigated failure with the Newton digital assistant, signalled the downfall of that market as well. In a last minute flash of ideas, James Gosling, leader of the OAK project, realized that Oak could be a great Web device. Sun gave the project a flashy new name and built a web browser around Gosling's idea. The results were Java and the HotJava Browser.

After Java's success on the Web, Sun decided that it needed a way to improve Java applets to combat the presence of ActiveX Components. The result is a new component model called Java Beans.

Section 3

The Tools of the Trade

3.1 The Standardized Interface Approach

Much effort has been spent in creating standards and standards organizations so that different applications can work together easily. Before we go into detail about particular standards, it is worthwhile to enumerate the reasons why a standard is useful.

- ¥ Reduced training costs: A standard way of doing things eliminates the need for developers to learn new interfaces.
- ¥ Application portability: Applications written under a particular standard can be used by other software vendors in different environments without costly reprogramming.
- ¥ Application longevity: Because standards are defined to be long lasting, applications written under standards can be assured at least a temporary respite from obsolescence caused by aging interfaces.
- ¥ Intersystem communication: Standards allow not only portability of different applications to other environments, but communication between applications in different environments.
- ¥ Customer choice: With standardization, a client or customer does not have to waste time learning and switching between different interfaces.

Unfortunately, standards can have significant disadvantages as well. These disadvantages are often enough for developers to decide that they are not worth the trouble.

- ¥ Standards can stifle creativity: As shown by the failure of HTML standards committees to keep up with the constant changes made by Microsoft and Netscape, standards are slow to react to pressures for new functionality. In turn, they waste developers' time by forcing developers to argue for new functionality into revisions of the standard.
- ¥ Standards can be inefficient: Having a standard interface may often mean giving up a more efficient way of doing things.

3.1.1 Structured Query Language (SQL)

As defined in *ÒA Guide to The SQL StandardÓ*[3], *ÒThe primary function of the SQL language is to support the definition, manipulation and control of data in relational databases.Ó* SQL provides a human

readable syntax and grammar with functions for basic data manipulation operations dealing with rows, columns and tables as well as some traditional programming statements such as conditional expressions. In addition, SQL provides support for cursors into rowsets, atomic transactions (for recovery purposes, an example of which is provided by the IBM System R database system), views (virtual tables consisting of parts of other tables and views), data integrity control and various security options. In addition, the SQL standard defines some general data types and provides certain advanced features.

3.1.2 Open DataBase Connectivity (ODBC)

Unlike SQL, ODBC is a programming interface built into existing languages like C/C++. ODBC is an example of a common interface architecture. The ODBC architecture consists of an application which performs calls to ODBC functions to submit SQL statements and retrieve results, a driver manager which loads ODBC drivers on behalf of an application and the driver itself which processes ODBC calls, submits SQL requests to a specific database and returns the results to an application. The driver modifies the applications request to conform to the syntax supported by its associated database management system.

ODBC is popular and efficient because any database that wishes to be used with any ODBC application need only implement an ODBC driver to extend its SQL capabilities. Unfortunately, ODBC is only supported on Windows platforms.

3.1.3 Open Data Sources(ODS)

Open Data Sources relies on a common gateway to provide communication between different servers. This is accomplished by a server with support for ODBC and DB-Library programming interfaces (DB-Library is like ODBC except that it is primarily designed for Microsoft SQL Server and is supported cross-platform).

3.1.4 Common Gateway Interface (CGI)

Unlike other programming interfaces, CGI does not define common functions for accessing data. CGI provides a means for communicating with applications by creating a standard way for servers to call programs written in an arbitrary language. In addition, CGI is only designed as an interface to Web servers.

Before the advent of Java applets or ActiveX Components, CGI was the only popular interface between Web servers (which otherwise were restricted to serving static content) and outside applications. Unfortunately, CGI is plagued by security problems and slow speeds.

3.2 Object Management Architecture (OMA)

The Object Management Architecture defines a general system of software components or Objects connected by a single bus called the Object Request Broker. See Figure 3.1 for the organization of the OMA.

Figure 3.1: The Object Management Architecture

The OMA defines four main sets of object interfaces as well as the Object Request Broker (See the next section). These interfaces are defined as follows:

- ¥ Object Services: Object Services are interfaces for services that are independent of application domain and are used by a large number of distributed object programs. Object Services include the naming service used to find particular objects in a network or the trading service used to identify particular objects by the properties of their interfaces. See Appendix A.1 for a full listing of Object Services.
- ¥ Common Facilities: Common Facilities are interfaces that are application independent, but fall within general areas for many end user applications. An example of a common facility object is the Distributed Document Component Facility (DDCF) based on the OpenDoc which provides facilities for linking different documents objects. An example of the DDCF in use is in linking a spreadsheet with a report document.
- ¥ Domain Interfaces: Domain Interfaces are oriented towards specific fields of applications like manufacturing, telecommunication, medicine and finance. A particular example of a Domain Interface object is for Product Data Management (PDM) for the manufacturing domain.
- ¥ Application Interfaces: Application Interfaces are application specific and therefore have no standards applied to them. Application Interfaces are created by specific developers who use CORBA. The OMG, however, has the responsibility of recognizing particular Application Interfaces that may become popular or useful enough to gain status as Domain Interfaces or Common Facilities.

Figure 3.2: Organization of Interfaces in the OMA

It is important to note that the different interface collections of the OMA reference each other in an organized manner. Common Facilities can serve Object Service interfaces to clients. Domain Interfaces can

serve Common Facilities or Object Services, etc. See Figure 3.2 for an overview of the relationship among the different types of interfaces.

3.2.1 Common Object Request Broker Architecture (CORBA)

The CORBA specification defines the architecture of a bus system for communication between different objects. This specification defines the architecture of both dynamic and static interfaces, adapters between different interfaces and an interface to the ORB itself.

Figure 3.3: The CORBA Architecture

All static object interfaces are defined in Interface Definition Language (OMG IDL). IDL is supported by a number of different languages including C++ and Smalltalk. Once an interface is defined, stubs are generated that allow the Client (the caller of the object) to interact with the Object Request Broker. In addition, an IDL skeleton is created for the Target Object which translates requests from the ORB to methods at the Target Object.

The Dynamic Invocation Interface allows Clients to request methods from Target Objects for whom interfaces were not known at compile-time. A dynamic skeleton allows the Target Object to receive method invocations from the ORB that were created dynamically.

The interface skeletons sit upon an Object Adapter (implementing various Object Services) which generally services a small range of object types. The Object Adapter provides such services as generation and interpretation of object references, method invocation, security of interactions, object and implementation activation and deactivation, mapping object references to implementations and registration of implementations.

3.2.2 OpenDoc and Fresco

OpenDoc and Fresco are both defined interfaces that are built upon the CORBA architecture. Both systems implement multimedia support for components from different vendors. For example, OpenDoc and Fresco provide interfaces for embedding various multimedia components (sounds, images and even web

browsers) into documents. The two interfaces differ in the level of depth of control of rendering (Fresco provides significant control for rendering screen objects) as well as support for a larger infrastructure (OpenDoc has a large infrastructure which makes it easy to integrate legacy applications, non-object oriented applications, etc.). For instance, OpenDoc provides an editing facility for all of its objects. This facility allows any OpenDoc object to be edited while being embedded within another document. This avoids the need for using multiple editors on a single document. One of the many things that OpenDoc and Fresco have in common is competition from Microsoft's OLE which is built upon COM. See Section 3.3.1 for more information on OLE.

3.3 Component Object Model (COM)

COM is like the OMA in many respects. Both promote reusable, distributed objects with both static and dynamic interfaces. Unlike CORBA, Microsoft's COM model is for a connection system that enables the connection between two objects and then drops out of the picture. COM was created with four ideals in mind. COM is supposed to provide reusable components, define a standard for binary interoperability and provide distributed capabilities. In addition, the COM specification states that COM is a "true system object model" in that it avoids inheritance problems, provides globally unique identifiers to avoid naming conflicts, provides a full life-cycle via reference counting, has a single programming model for cross-process or cross-process communication and provides security at the object level. Not surprisingly these specifications sound very much like the specifications for OMA and the CORBA Object Services.

The dynamic linking model that COM creates relies on the IUnknown interface which allows a client (calling object) to query for other interfaces within the object. As in CORBA, multiple interfaces may exist concurrently within an object. Also, like in CORBA, interfaces in COM are opaque in that they hide the implementation of the object from the user.

The COM specification provides three very important object services. These are Persistent Storage, Monikers (intelligent names) and Uniform Data Transfer.

Persistent Storage defines the file system within the file that describes the data contents of a COM object. Not only does this describe a uniform way of saving data in an object, it also provides a uniform way of saving the objects state as a whole. Persistent storage also defines incremental data access (Cursors) and transactions.

COM monikers (intelligent names) provide a means for assigning a unique identifier to a particular instantiation of an object. Identifiers are also created for particular operations such as queries. These monikers not only serve as a means for identifying an object, but also know how to relocate an object or operations without the help of a separate naming service.

Uniform Data Transfer describes the way that data are exchanged between client and object. This specification also defines interfaces for querying data formats, querying storage mediums and passing on event notifications.

3.3.1 Object Linking and Embedding (OLE)

Like OpenDoc, OLE is an extension of a lower level technology. OLE 2.02 extends COM to include the following capabilities:

- ¥ Drag and Drop: This interface allows data exchange by picking up the visual representation of an object with a mouse and dropping it in another window.
- ¥ Compound Documents: Like OpenDoc, Compound Documents provides the means for linking and embedding objects within documents as well for allowing in-document editing of embedded objects.
- ¥ Automation: Automation is very much like the dynamic invocation interface in CORBA. In COM terms, this is called late binding. As in CORBA, Automation is accomplished by defining type libraries from which clients can get information about dynamic interfaces.
- ¥ OLE Controls: This interface brings together Compound Documents and Automation under the same object. Originally, COM objects (from OLE 1.0) were restricted to writing either Compound Document objects (which have visual representation) or an invisible Oprogrammatic Automation Server. OLE Controls allows OLE objects to do both simultaneously.

3.3.2 ActiveX

When the Web became a popular source for markets, Microsoft decided that it needed a quick and easy way to make a strong presence on the Web. This was accomplished by adding a relatively small amount of Web functionality to OLE Controls to create ActiveX controls. Add a pinch of hype and you have a major

internet revolution. Unfortunately, Microsoft also added a little confusion by blurring the lines between ActiveX and OLE. When describing the number of existent ActiveX developers, Microsoft also includes developers that were originally creating OLE components without Web interaction. Thus, a confusion has developed as to where ActiveX begins and OLE ends. A table in Trevor Harmon's "Web Developer's Guide to Visual J++ and ActiveX" [13] describes ActiveX's relationship to OLE very well.

1992	1994	1996
OLE	+Internet	=ActiveX
OLE controls	+Internet	=ActiveX controls
OLE documents	+Internet	=ActiveX documents
OLE object model	+Internet	=ActiveX object model

Table 3.1: OLE is reborn as ActiveX

However, ActiveX does add two very important capabilities to the original OLE automation server. These are windows and events. Previously, ActiveX OLE Automation neither allowed windows nor the propagation of events. With these changes and some built in browser support, ActiveX was made.

3.4 Java

One might wonder why a language has entered the realm of object architectures. In fact, an important part of CORBA and COM is the independence of language and support by many different languages. Java's emergence into the component object realm is primarily due to three reasons. First, Java has a well developed object model that can be readily used in Web applications. For example, applets can be seen as the faint stirrings of a reusable software component which needs extra work to become a true component object. Second, Java is a particularly unique language in that it brings platform independence with it. Platform independence makes Java an incredibly important tool for distributed object computing. Lastly, Sun Microsystems wants to keep Java alive in the face of growing pressure from Microsoft's ActiveX.

3.4.1 JavaBeans

A JavaBean is Java's cross-platform equivalent to a CORBA or COM object. Javabeans provide five important object services. These service are introspection, customization, events, properties and persistence.

Introspection is the ability for a builder tool to analyze the internals of a bean in order to allow other objects to query the available methods, events and other interfaces that the bean has to offer. This provides dynamic linking between objects.

Customization is similar to editing services supplied by OpenDoc. However, Customization is oriented towards allowing the end user to customize the Bean rather than for providing editing functions within the Bean.

Event handling and propagation is similar to event propagation in CORBA and ActiveX objects.

Properties are also similar to those in CORBA and COM. Properties allow the existence of a Trading service similar to CORBA's.

Persistence follows a similar model to those of the CORBA and COM models.

JavaBeans also implement extended services like those of OLE. For instance, drag and drop data transfer is implemented as well as security for Bean data. In addition, JavaBeans also have internal Clipboard functions for cutting and pasting. This Clipboard is built upon Beans' support for Uniform Data Transfer which also exists in CORBA and COM.

The goals of Uniform Data Transfer are explicitly laid out in the Java Beans Specification as follow:

- ¥ to provide a simple conceptual framework for the encapsulation and interchange of data.
- ¥ to provide a general API that can support current and future data interchange models.
- ¥ to enable efficient, dynamic creation, registration and interchange of rich and diverse data types.
- ¥ to enable data interchange across process boundaries.
- ¥ to enable the interchange of data between Java-based and platform-native applications.

If it seems like a great deal of JavaBeans is similar to CORBA and COM, there is a good reason for it. JavaBean is the most recently devised component object model. The makers of JavaBeans borrowed extensively from existing technologies. For instance, JavaBeans use the exact IDL defined for CORBA objects in order to define static interfaces.

3.5 Extensions

3.5.1 JDBC

Java's developers have been attempting to maintain their foothold in the Web application world despite pressure from Microsoft. One of the results of these attempts is JDBC. The Java developers have created JDBC in order to promote easier access to database applications by Java applets and Beans. JDBC is essentially the same as Microsoft's ODBC except that it offers the added bonus of platform independence.

3.5.2 InfoBus

InfoBus is the product of collaboration between Javasoft and Lotus. Although COM and Javabeans have communication standards that offer faster speeds than achieved using a data bus, having the option of using a data bus between objects has the merits of greater organizational integrity and centralization of communication through a common server. With this in mind, the InfoBus was created as a platform independent bus that connects different JavaBeans. One of the potential bonuses of such a system is that support can be built in for allowing simplified intercommunication between CORBA, ActiveX and JavaBean objects.

3.5.3 Distributed COM (DCOM)

Distributed COM is Microsoft's attempt to extend the COM specification to include more efficient, transparent internet communication between COM objects. This extension mirrors support that already exists among JavaBeans and CORBA objects.

3.5.4 OLEDB/ActiveX Data Objects (ADO)

Figure 3.4: The Layered Structure of an Application that uses OLEDB

OLEDB provides extended support for creating OLE objects that access Microsoft databases. This extension falls under the Common Facilities extension that exist in CORBA. As the reader can probably expect, ActiveX Data Objects are the extension of OLEDB objects to the Web. In the terminology put forth by Trevor Harmon [13], $\text{OLEDB} + \text{Internet} = \text{ADO}$.

3.5.5 General Overview

It is important to note that developers of many of these object models have been very thorough in trying to provide support for connectivity between Beans, ActiveX and CORBA objects. This shows not only an attempt by them to gain support for a particular object model by providing support for other popular object models, but also that these different object models are evolving towards the singular goal of platform and process independent binary communication. Another item to note is that more recent developments in particular object models are very much a product of elements of existing object models. Examples of this can be seen in the great similarity between the object models as well as the development of extensions such as JDBC, InfoBus and OLEDB. This fact points toward the great amount of overlap between Beans, CORBA and COM. It seems that the

future of object component architectures will be filled with a great deal of growth due to competition as well as increased interconnection between the different object models.

Section 4

Example Applications

4.1 Active Server Pages: The Career Server (ADO, SQL)

The Career Server is an application written using Microsoft's Visual Interdev using Active Server Pages (ASP) as the basis for development.

Figure 4.1: The Structure of a Web Application that uses ASP

4.1.1 Motivation

The motivation for this endeavor is two fold. First, this application show ActiveX Data Objects and SQL in action. Second, the Career Server is an example of a useful application allowing convenient, paperless, large scale access to some of the most important services provided by MIT's Career Office.

Convenience and large scale access are provided by making the Career Server accessible from any Web capable machine, including all Athena¹ terminals.

Presently, the Career Office publishes thick collections of photocopied material for students to use in selecting companies to interview for. In addition, the Career Office

¹. Athena is an ongoing project to supply MIT students and faculty free and easy access to workstations and the internet for use in classwork and research. The percentage of users of Athena at MIT is extremely high, giving any Web based application developed for Athena a wide potential user base quickly.

maintains a large library of constantly changing material supplied by companies across the world. Moving even a small amount of the functionality that these collections provide into digital format will save a great deal of paper and space. In addition, the ultimate goal of the Career Server is to allow companies to update their own information on interviewee requirements remotely via Web access, distributing much of the effort of supporting the Career Office to the companies themselves. This is a feasible goal because the vast majority of companies that make use of the MIT Career Office are already connected to the Web.

4.1.2 Typical Session at the Career Office

A typical session at the Career Office begins by picking up a large packet of papers from a pile in the corner of the room between the hours of 9am and 5pm. After reading through the entire packet, a student proceeds to get in line in order to sign up for an interview. After a short wait, the student submits a paper copy of her/his resume and signs up for an available slot on a paper schedule. The student writes the selected interview time on a scrap of paper or in a notebook and leaves. The counselor at the Career Office then files the paper schedule away in a cabinet behind the desk until the interview day.

4.1.3 Typical Session using the Career Server

Figure 4.2: The Career Server Login Screen

A typical session using the Career Server starts at any time of day. A student logs on the Career Server with a password and is given a message of the day which may include deadlines for interview signups that are upcoming, etc. The student queries for companies that she/he is interested in and receives a list of pages that have interview schedules and company descriptions to choose from. The student works through the list and chooses available interview times to sign up for and then is sent a list of all of the interviews that she/he has signed up for.

4.1.4 Functionality of the Career Server

The Career Server presently provides services to both companies and students in querying for potential jobs, identifying good job applicants and setting up interview times.

Students can do the following:

- ¥ Maintain a general list of qualifications for the perusal of companies.
- ¥ Query a database of company information by various fields in order to find an appropriate company to apply to.
- ¥ Browse a list of human contacts for each company for the purpose of initiating a discussion with company representatives.
- ¥ Sign up for interviews with particular company representatives.
- ¥ Create a listing of all of the interviews that the student is signed up for.

In addition, companies that use the system can do the following:

- ¥ Maintain a profile and information to help prospective applicants to choose the appropriate company to apply to.
- ¥ Maintain a listing of company representatives that will be conducting on campus interviews, as well as an interview list for each contact.
- ¥ Query for particular students in order to identify prospective applicants that the company may wish to contact on their own.
- ¥ Create a listing of all the interviews that a particular contact is signed up for.

4.1.5 The Implementation of the Career Server: Active Server Pages

The Career Server is built using Active Server Pages, an extension to Microsoft's WinNT or Win95 Personal web servers. When an ASP is requested by a browser, the server executes any script on the page and sends the output back to the user's browser. This is very similar to the way CGI works except that ASP pages have some significant differences. Unlike CGI, ASP do not have to be contained within a particular directory to be accessed. In addition, security is done at the ASP file level instead of subjecting the system to security hole opened by having an entire directory available to the public for execution.

ASPs also provide extra functionality over a CGI script. The server creates internal representations of the various objects on the page which are made available to the script

ing language used in the ASP. Thus, generating HTML becomes largely automated. In addition, the ASP extensions to the web server provide COM including scripting support for ActiveX Data Objects.

Another part of the functionality of ASPs is the creation of Session and Application objects. Session objects are arbitrary variables that have a predetermined lifetime set by the ASP and are connected to a particular user. Session variables can be used to implement access control by creating a variable which is true only if the user has passed a particular login page. Subsequent pages can redirect the user to the login page if this variable is not true. Session variables are usually implemented as cookies (variables left on the users machine in a cookie file). Application variables are similar to Session variables except that they persist on the server for the entire length of the application. Session and Application variables are also important in that they are the only means of transferring information between ASPs as the pages themselves have no persistent state.

An important part of the reason ASPs are useful is the fact that they are run within the server itself. Because of this, ASP are much faster than CGI scripts because they avoid an extra layer of communication.

4.1.6 ActiveX Data Objects and SQL in the Career Server

An ASP can request that an ActiveX Data Object (ADO) be instantiated by the server for any data source accessible to the server machine. This includes databases that are not on the server machine. ADOs implement the ODBC interface, allowing access to virtually any database that runs on a Windows machine. Once an ADO is created, it can be used to maintain a connection to the database and provides methods for adding, deleting and updating records in the database. In addition, the ADO provides support for feeding hand written SQL into the database.

applicants they wish to interview and post a list of applicants that can interview as well as a list of alternates. Students on this list are then authorized to sign up with the company as if there was an open schedule. This service would require some additions to the current Career Server that are too time consuming. Lastly, many companies scan resumes into their own databases after receiving them. For this reason, digital resume submission and facilities to connect company databases to the Career Server are desirable. However, both of these issues are time consuming and have been left out of the Career Server. In addition, many other developers are working on these same problems. The Career Server was built with extensibility in mind and one could potentially see the current implementation of the Career Server combined with another system for the purposes of resume submission, etc.

4.2 Merchant Server: Image Gallery (COM, SQL, ActiveX)

The Image Gallery was designed using Microsoft's Merchant Server and the computational facilities of a local film company. The Merchant Server, in conjunction with Microsoft Wallet, makes use of COM, ActiveX and SQL in its implementation.

Figure 4.3: A Store Front Created with Merchant Server

4.2.1 Motivation

The motivation for the Image Gallery is to provide a system for selling reproductions of famous paintings. In particular, the Merchant Server implementation of the Image Gallery was created in order to test the ease-of-use, functionality and feasibility of the Merchant Server application in creating and maintaining the on-line Image Gallery store.

Some of the motivations for selling just about anything on-line include low start-up cost (now need to lease a store front to sell out of), low overhead (less employees needed to maintain a virtual store) and wide market base (the Web is everywhere). In particular, a painting is almost entirely a visual product. As almost all browsers are graphics capable, this provides a good mechanism for potential customers to view their collection previous to purchase.

4.2.2 Typical Purchasing Session at a traditional store

A customer in a traditional store must first find and physically go to that store in order to view and purchase the merchandise. In addition, this must usually be done during normal business hours, 9am to 5pm. Once at the store, the customer may view the actual painting they may purchase. If satisfied, the customer makes the purchase using credit card, cash or check.

At the end of the day, the manager of the store collects all of the cash and checks received during the day, counts them, files the transaction records, purchase orders and inventory reports. These actions may include entering information into a computerized inventory system. In addition, the register receipts for credit card transactions must be filed for subsequent comparison against reports from the issuing credit institutions.

4.2.3 Typical Purchasing Session at the Image Gallery

At any time of day or night, the customer at their home or office surfs to the Image Gallery web site. After browsing the site and viewing images of the reproductions, the customer checks reproductions she/he wants into a virtual shopping basket, navigates to a

check out screen and fills out a credit card form or a billing address¹ as well as a shipping address. Almost instantaneously, the transaction is forwarded via vPos (a system of direct electronic billing developed by Verifone) to the credit card issuing institution. The transaction information is passed on to archiving, accounting, inventory control and shipping programs at the server site.

At some time during the day, the manager (that maintains the Image Gallery server), runs a script or program that passes on transaction information to archiving, accounting, inventory control and shipping programs for a daily report.

4.2.4 Functionality of the Image Gallery

The Image Gallery provides a number of important services. These services are storefront creation and maintenance, server security, product line maintenance and advertising, transaction viewing and maintenance and customer security.

The creation and the maintenance of a store front are accomplished via Merchant Server directives. These directives are written directly into html pages and are processed and replaced by the Merchant Server by relevant information before the page is sent to the browser. For example, the Fetchrows directive retrieves a rowset from the Image Gallery database which can be listed on the html page using other directives. The link directive embeds other html pages which hold html describing the visual components of such a list. The arguments to the link directive provide the data obtained from Fetchrows that the list will show. These directives isolate the storefront developer from the need to understand the underlying details of the database connection and allow for easy modularity of pages and reuse of html components. The same link directive can be used to place the same header html at the top of every page in the store.

¹. Microsoft also promises that Merchant Server will eventually have support for digital cash. In addition, Microsoft or vPos Wallets may eliminate the need to re-enter credit information at a particular site.

The Merchant Server directives also supply a small amount of server security. Many directives reference values that are stored in the Windows registry on the server machine. This hides the internal directory structure of the server by hiding directory structure within directives.

Product line maintenance is also accomplished by directives. Directives allow SQL specified by the administrator or user to be run at the database. In addition, directives can also be used to create an interface for an administrator to update, add or delete records to the database. Advertisements can be made dynamic and users can customize their session with the help of directives that implement time services and database access for holding user preferences.

Transaction maintenance, viewing and report generation are also accomplished using directives that access the database and format administrator-only web pages.

Customer security is accomplished by use of the Wallet ActiveX component and a vPos COM object that commits the actual credit card transactions. Using Microsoft Wallet, a user can send her/his credit card information to the server in encrypted form. This prevents a common security risk in using credit cards. Often credit card numbers are sent to the server as arguments in the URL sent to the server. This allows snooping by outsiders to see the card numbers. In addition, because these URLs persist at the server, outsiders could potentially snoop large numbers of credit cards by sifting through the input queues of many servers. The vPos COM object that connects Merchant Server may also soon implement further security. At present, credit card numbers are available to the owner of the store at which a purchase is made. Credit card companies are creating Secure Electronic Transaction (SET) software to ensure that a credit card number never exists unencrypted at the server machine and is sent directly to the bank in encrypted form. This provides a level of customer security which is greater than that of a traditional store.

4.2.5 COM, SQL and ActiveX in Merchant Server

Three of the technologies discussed in this paper make up important parts of Merchant Server. SQL and the ODBC interface are used to access the Merchant Server database. As in the Career Server, this gives the Merchant Server user freedom in choosing an underlying database. For the Image Gallery, Microsoft's SQL database was used for the base application, while Microsoft's Access database was used for development.

The COM object model was used in order to create a connection to vPos software that makes connections to a particular bank for transaction support. By using a binary connection model with dynamic interfaces, Merchant Server is able to use independently developed vPos and bank software in order to make a secure connection. This may also allow a SET application make a secure connection to vPos objects using binary interfaces.

The Wallet component uses the ActiveX Component model to create a component that has a user interface for entering credit card and billing information, embeds this user interface within a web browser and maintains interfaces for a large number of other components such as the vPos transaction component.

4.2.6 Limitations and Issues

As of yet, Merchant Server does not provide support for digital cash. This makes anonymous transactions impossible. In addition, the server machine that runs the Merchant Server can only be administrated by the administrator of the Merchant Server or someone trusted to be the Merchant Server administrator. This is true because the administrator password for Merchant Server and for any database accessed by Merchant Server is kept in the clear in the Windows registry of the server machine.

Section 5

Is It Worth It?

5.1 Making Money on the Internet is Hard

Figure 5.1: Economist Magazine Survey Results

There has been a great deal of hype surrounding how various component object models give businesses an edge by providing internet and Web support for their applications. In truth, making any money on the Internet can be a trying experience. In 1996, only \$500 - \$600 million dollars in transactions were done on the Web¹. Communication giant, MCI's web Marketplace closed its virtual doors last year due to a lack of business. However, there is a great deal of useful Web material out there. A March 1996 survey of US and Canada Web users shows that 53% of Web users have used the Web to make purchases. However, only 15% of these completed the transaction on the Web itself.

¹. Survey information is taken from an article in Economist Magazine which makes reference to a number of independent studies in the United States and Canada [18].

Selling systems like the Career Server is also hard. Much of the investment a company makes can be downloaded by any user for their own purposes for free. In addition, web systems are easy enough to build that just about anyone can do it cheaper.

5.2 Where These Systems Are Useful

Where Web and Internet support has become very useful is for intra-company and inter-company transactions. In particular, on-line trading has a huge following among companies. For instance, Forrester reports \$111 billion worth of assets are managed on-line. On-line broker Charles Schwab estimates that one-sixth of its account holders have done on-line trading at least once during 1996 (See [18]). Manufacturing giants such as Ford and GE have web-based accounting and paperwork systems (popularly called intranets) that may save significant overhead costs.

A particularly successful example of company web sites that do make money is GE's Trading Process Network (TPN) which connects GE with suppliers across the Internet. The bonuses that TPN generate include better bidding processes for vendors and immediacy. GE reports \$1 billion worth of business with 1,400 suppliers over TPN. TPN's success single-handedly exceeds that of all consumer electronic commerce. These are the markets that ActiveX, JavaBeans and CORBA are being optimized for. Analysts believe that technologies like these are going to have a large impact on the Web and the businesses that use it.

5.3 Is There Anything Beyond the Hype?

With the large number of acronyms being thrown around in the media as well as the heavy competition between Microsoft and just about everyone else, there is a great deal of hype being generated. For instance, as discussed earlier in this paper, Microsoft added the snazzy new term "ActiveX" to its OLE objects after making relatively small additions.

However, one must not forget that the technology that ActiveX, CORBA and JavaBeans is based on comes from many years of research into distributed computing before object component architectures were considered useful. In addition, these technologies are more efficient, flexible and capable than their predecessors. As such, there is substance to component object computing indeed!

Appendix A

Background Material

A.1 The OMA Object Services

The following list includes the Object Services provided by the OMA. These interfaces are all specified in IDL. A few of these may not have their specifications formally adopted by the OMG.

- ¥ Naming: This is a directory service which enables objects to be referenced by their human readable `ÒnamesÓ`.
- ¥ Persistence: Objects can survive beyond the lifetime of their creator. This is usually accomplished by reference counting as in COM.
- ¥ Life Cycle: This provides a service for administering objects. Available methods include creating, destroying, copying and moving objects.
- ¥ Properties: This provides a means of giving object arbitrary properties which may be check by the Trading Service (See this list).
- ¥ Concurrency: This provides locking for the coordination of accessing shared resources.
- ¥ Security
- ¥ Collections
- ¥ Trader: This provides a service for identifying objects by arbitrary Properties (See this list). This is very much like the Naming Service.
- ¥ Externalisation: This allows streaming in and out an objects internal state.
- ¥ Events: This provides objects the ability to react to and exchange events. For example, an object can be activated when another object is destroyed or some window is opened.
- ¥ Transactions: This service provides a means to collect a series of statements into an atomic transaction. This is useful for crash recovery and control of concurrency.
- ¥ Query: This allows object selection according to attributes. For example, you can query for all objects which are `ÒredÓ`.
- ¥ Relationships: This allows arbitrary relationships to be defined between objects.
- ¥ Time
- ¥ Change Management
- ¥ Licensing

Appendix B

Lists and Raw Data

B.1 List of Figures

The Object Management Architecture	12
Organization of Interfaces in the OMA	13
The CORBA Architecture	14
The Layered Structure of an Application that uses OLEDB	21
The Structure of a Web Application that uses ASP	23
The Career Server Login Screen	25
A Store Front Created with Merchant Server	30
Economist Magazine Survey Results	35

B.2 List of Tables

OLE is reborn as ActiveX.....	17
-------------------------------	----

B.3 ASP Script: The Career Server, an example ASP page

```
<%@ LANGUAGE=ÓvbscriptÓ %>
<%
Ó-----
Ó Microsoft Visual InterDev - Data Form Wizard
Ó
Ó Form Page
Ó
Ó (c) 1997 Microsoft Corporation. All Rights Reserved.
Ó
Ó This file is an Active Server Page that contains the form view of a Data Form.
Ó It requires Microsoft Internet Information Server 3.0 and can be displayed
Ó using any browser that supports tables. You can edit this file to further
Ó customize the form view.
Ó
Ó Modes: The form mode can be controlled by passing the following
Ó name/value pairs using POST or GET:
Ó FormMode=Edit
Ó FormMode=Filter
Ó FormMode=New
Ó Tips: - If a field contains a URL to an image and has a name that
Ó begins with Óimg_Ó (case-insensitive), the image will be
Ó displayed using the IMG tag.
```

Ô - If a field contains a URL and has a name that begins with
Ô Öurl_Ö (case-insensitive), a jump will be displayed using the
Ô Anchor tag.
Ô-----

```

Dim strPagingMove
Dim strFormMode
Dim strDFName
strDFName = ÒrsCompanyCompaniesÓ
%>

```

```
<SCRIPT RUNAT=Server LANGUAGE=ÓVBScriptÓ>
```

```

Ó---- FieldAttributeEnum Values ----
Const adFldUpdatable = &H00000004
Const adFldUnknownUpdatable = &H00000008
Const adFldIsNullable = &H00000020

```

```

Ó---- DataTypeEnum Values ----
Const adUnsignedTinyInt = 17
Const adBoolean = 11
Const adLongVarChar = 201
Const adLongVarWChar = 203
Const adBinary = 128
Const adVarBinary = 204
Const adLongVarBinary = 205
Const adVarChar = 200
Const adWVarChar = 202
Const adBSTR = 8
Const adChar = 129
Const adWChar = 130
Ó---- Other Values ----
Const dfMaxSize = 100

```

```

Ó-----
Ó Purpose: Substitutes Empty for Null and trims leading/trailing spaces
Ó Inputs:  varTemp- the target value
Ó Returns:The processed value
Ó-----

```

```

Function ConvertNull(varTemp)
    If IsNull(varTemp) Then
        ConvertNull = ÒÒ
    Else
        ConvertNull = Trim(varTemp)
    End If
End Function

```

```

Ó-----
Ó Purpose: Embeds bracketing quotes around the string
Ó Inputs:  varTemp- the target value
Ó Returns:The processed value
Ó-----

```

```

Function QuotedString(varTemp)
    If IsNull(varTemp) Then
        QuotedString = Chr(34) & Chr(34)
    Else

```

```
        QuotedString = Chr(34) & CStr(varTemp) & Chr(34)
    End If
End Function
```

```
Ô-----
Ô Purpose: Tests string to see if it is a URL by looking for protocol
Ô Inputs:  varTemp- the target value
Ô Returns: True - if is URL, False if not
Ô-----
```

```
Function IsURL(varTemp)
    IsURL = True
    If UCase(Left(Trim(varTemp), 6)) = "HTTP:" Then Exit Function
    If UCase(Left(Trim(varTemp), 6)) = "FILE:" Then Exit Function
    If UCase(Left(Trim(varTemp), 8)) = "MAILTO:" Then Exit Function
    If UCase(Left(Trim(varTemp), 5)) = "FTP:" Then Exit Function
    If UCase(Left(Trim(varTemp), 8)) = "GOPHER:" Then Exit Function
    If UCase(Left(Trim(varTemp), 6)) = "NEWS:" Then Exit Function
    If UCase(Left(Trim(varTemp), 7)) = "HTTPS:" Then Exit Function
    If UCase(Left(Trim(varTemp), 8)) = "TELNET:" Then Exit Function
    If UCase(Left(Trim(varTemp), 6)) = "NNTP:" Then Exit Function
    IsURL = False
End Function
```

```
Ô-----
Ô Purpose: Tests whether the field in the recordset is updatable
Ô Assumes: That the recordset containing the field is open
Ô Inputs:  strFieldName- the name of the field in the recordset
Ô Returns: True if updatable, False if not
Ô-----
```

```
Function CanUpdateField(strFieldName)
    Dim intUpdatable
    intUpdatable = (adFldUpdatable Or adFldUnknownUpdatable)
    CanUpdateField = True
    If (rsCompanyCompanies(strFieldName).Attributes And intUpdatable) = False Then
        CanUpdateField = False
    End If
End Function
```

```
Ô-----
Ô Purpose: Handles the display of a field from a recordset depending
Ô         on its data type, attributes, and the current mode.
Ô Assumes: That the recordset containing the field is open
Ô         That strFormMode is initialized
Ô Inputs:  strFieldName - the name of the field in the recordset
Ô         strLabel     - the label to display
Ô         blnIdentity  - identity field flag
Ô         avarLookup   - array of lookup values
Ô-----
```

```
Sub ShowField(strFieldName, strLabel, blnIdentity, avarLookup)
    Dim blnFieldRequired
    Dim intMaxSize
```



```

        End Select
    End Select
    Response.Write <</FONT>></TD></TR>
    Exit Sub
End If

Ô Handle lookups using a select and options
If Not IsNull(avarLookup) Then
    Response.Write <<SELECT NAME= & QuotedString(strFieldName) & >>
    Ô Add blank entry if not required or in filter mode
    If Not blnFieldRequired Or strFormMode = <<Filter>> Then
        If (strFormMode = <<Filter>> Or strFormMode = <<New>>) Then
            Response.Write <<OPTION SELECTED>>
        Else
            Response.Write <<OPTION>>
        End If
    End If
End If

Ô Loop thru the rows in the array
For intRow = 0 to UBound(avarLookup, 2)
    Response.Write <<OPTION VALUE= & QuotedString(avarLookup(0, intRow))
    If strFormMode = <<Edit>> Then
        If ConvertNull(avarLookup(0, intRow)) = ConvertNull(strFieldValue) Then
            Response.Write << SELECTED>>
        End If
    End If
    Response.Write <>>
    Response.Write ConvertNull(avarLookup(1, intRow))
Next
Response.Write <</SELECT>>
If blnFieldRequired And strFormMode = <<New>> Then
    Response.Write << Required>>
End If
Response.Write <</FONT>></TD></TR>
Exit Sub
End If

Ô Evaluate data type and handle appropriately
Select Case rsCompanyCompanies(strFieldName).Type

    Case adBoolean, adUnsignedTinyInt        Ô Boolean
        If strFormMode = <<Filter>> Then
            strOption1State = <<>True>>
            strOption2State = <<>False>>
        Else
            Select Case strFieldValue
                Case <<True>>, <<1>>, <<-1>>
                    strOption1State = << CHECKED>>True>>
                    strOption2State = <<>False>>
                Case <<False>>, <<0>>
                    strOption1State = <<>True>>
                    strOption2State = << CHECKED>>False>>
                Case Else
                    strOption1State = <<>True>>
            End Select
        End If
    End Case
End Select

```

```

        strOption2State = <>False
    End Select
End If
Response.Write <<INPUT TYPE=Radio VALUE=1 NAME= & QuotedString(strFieldName) & strOption1State
Response.Write <<INPUT TYPE=Radio VALUE=0 NAME= & QuotedString(strFieldName) & strOption2State
If strFormMode = <<Filter>> Then
    Response.Write <<INPUT TYPE=Radio NAME= & QuotedString(strFieldName) & <<CHECKED>>Neither>>
End If

Case adBinary, adVarChar, adLongVarChar, adLongVarBinary, adBinary
    Response.Write <<[Binary]>>

Case adLongVarChar, adLongVarChar, adLongVarChar, adLongVarBinary, adBinary
    Response.Write <<TEXTAREA NAME= & QuotedString(strFieldName) & <<ROWS=3 COLS=80>>
    Response.Write Server.HtmlEncode(ConvertNull(strFieldValue))
    Response.Write <</TEXTAREA>>

Case Else
    Dim nType
    nType=rsCompanyCompanies(strFieldName).Type
    If (nType <> adVarChar) and (nType <> adWVarChar) and (nType <> adBSTR) and (nType <> adChar) and (nType
<> adWChar) Then
        intInputSize = (intInputSize-2)*3+2
        If strFormMode <> <<Filter>> Then intMaxSize = intInputSize - 2
    End If
    If blnIdentity Then
        Select Case strFormMode
            Case <<Edit>>
                Response.Write ConvertNull(strFieldValue)
                Response.Write <<INPUT TYPE=Hidden NAME= & QuotedString(strFieldName)
                Response.Write <<VALUE= & QuotedString(strFieldValue) & <>>
            Case <<New>>
                Response.Write <<[AutoNumber]>>
                Response.Write <<INPUT TYPE=Hidden NAME= & QuotedString(strFieldName)
                Response.Write <<VALUE= & QuotedString(strFieldValue) & <>>
            Case <<Filter>>
                Response.Write <<INPUT TYPE=Text NAME= & QuotedString(strFieldName) & <<SIZE= &
tInputSize
                Response.Write <<MAXLENGTH= & tMaxSize & <<VALUE= & QuotedString(strFieldValue) & <>>
        End Select
    Else
        If intInputSize > 80 Then intInputSize = 80
        Response.Write <<INPUT TYPE=Text NAME= & QuotedString(strFieldName)
        Response.Write <<SIZE= & intInputSize
        Response.Write <<MAXLENGTH= & intMaxSize
        Response.Write <<VALUE= & QuotedString(strFieldValue) & <>>
        << Check for special field types
        Select Case UCase(Left(rsCompanyCompanies(strFieldName).Name, 4))
            Case <<IMG_>>
                If strFieldValue <> <<>> Then
                    Response.Write <<BR><BR><IMG SRC= & QuotedString(strFieldValue) & <><BR>&nbsp;<BR>>>
                End If
            Case <<URL_>>
                If strFieldValue <> <<>> Then

```

Response.Write " "

```

        Response.Write "Go"
        Response.Write "</A>"
    End If
Case Else
    If IsURL(strFieldValue) Then
        Response.Write "&nbsp;&nbsp;<A HREF=" & QuotedString(strFieldValue) & ">"
        Response.Write "Go"
        Response.Write "</A>"
    End If
End Select
End If
End Select
If blnFieldRequired And strFormMode = "New" Then
    Response.Write " Required"
End If
Response.Write "</FONT></TD></TR>"
End Sub
</SCRIPT>

<%
strFormMode = "Edit" Initialize the default
If Not IsEmpty(Request("FormMode")) Then strFormMode = Request("FormMode")
If Not IsEmpty(Request("rsCompanyCompanies_PagingMove")) Then
    strPagingMove = Trim(Request("rsCompanyCompanies_PagingMove"))
End If
%>

<HTML>
<HEAD>
    <META NAME="GENERATOR" CONTENT="Microsoft Visual InterDev">
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
    <META NAME="Keywords" CONTENT="Microsoft Data Form, Companies Form">
    <TITLE>Companies Form</TITLE>
</HEAD>

<!-- Formatting Section -->

<BASEFONT FACE="Arial, Helvetica, sans-serif">
<LINK REL="STYLESHEET" HREF="./Stylesheets/Bluemood/Style2.css">
<BODY BACKGROUND="./Images/Bluemood/Background/Back2.jpg" BGCOLOR="White">

<!-- Lookups Section -->
<%
Dim avarCompanyCitReq
If IsEmpty(Application(strDFName & "_Lookup_CompanyCitReq")) Or strPagingMove = "Requery" Then
    Set DataConn = Server.CreateObject("ADODB.Connection")
    DataConn.ConnectionTimeout = Session("DataConn_ConnectionTimeout")
    DataConn.CommandTimeout = Session("DataConn_CommandTimeout")
    DataConn.Open Session("DataConn_ConnectionString"), Session("DataConn_RuntimeUserName"),
Session("DataConn_RuntimePassword")
    Set rsCompanyCitReq = DataConn.Execute("SELECT DISTINCT Citizenship Requirements, Citizenship
Requirements FROM CitReq")
    avarCompanyCitReq = Null
    On Error Resume Next

```


Response. Write ÒCurrent Filter: NoneÓ

```

Else
  If Session(ÒrsCompanyCompanies_FilterÓ) <> ÒÒ Then
    Response.Write ÒCurrent Filter: Ò & Session(ÒrsCompanyCompanies_FilterDisplayÓ)
  Else
    Response.Write ÒCurrent Filter: NoneÓ
  End If
End If
Case ÒFilterÓ
  Response.Write ÒStatus: Ready for filter criteriaÓ
Case ÒNewÓ
  Response.Write ÒStatus: Ready for new recordÓ
End Select
End If
%>
</FONT>
</TD>
</TR></TABLE>

```

<!------- Form Section ----->

```

<!--METADATA TYPE=ÓDesignerControlÓ startspan
  <OBJECT ID=ÓrsCompanyCompaniesÓ WIDTH=151 HEIGHT=24
    CLASSID=ÓCLSID:F602E721-A281-11CF-A5B7-0080C73AAC7EÓ>
    <PARAM NAME=ÓBarAlignmentÓ VALUE=Ó0Ó>
  <PARAM NAME=ÓPageSizeÓ VALUE=Ó1Ó>
    <PARAM Name=ÓRangeTypeÓ Value=Ó1Ó>
    <PARAM Name=ÓDataConnectionÓ Value=ÓDataConnÓ>
    <PARAM Name=ÓCommandTypeÓ Value=Ó0Ó>
    <PARAM Name=ÓCommandTextÓ Value=ÓSELECT ÓCompany IDÓ, ÓCompanyNameÓ, ÓCitizenship
RequirementsÓ, ÓResumes DueÓ, ÓURLÓ, ÓCover Letter RequiredÓ, ÓGrade Report RequiredÓ, ÓApplication RequiredÓ,
ÓNotesÓ FROM ÓCompaniesÓÓ>
    <PARAM Name=ÓCursorTypeÓ Value=Ó1Ó>
    <PARAM Name=ÓLockTypeÓ Value=Ó3Ó>
    <PARAM Name=ÓCacheRecordsetÓ Value=Ó1Ó>
  </OBJECT>
-->

<%
fHideNavBar = False
fHideNumber = False
fHideRequery = False
fHideRule = False
stQueryString = ÒÒ
fEmptyRecordset = False
fFirstPass = True
fNeedRecordset = False
fNoRecordset = False
tBarAlignment = ÒLeftÓ
tHeaderName = ÒrsCompanyCompaniesÓ
tPageSize = 1
tPagingMove = ÒÒ
tRangeType = ÒFormÓ
tRecordsProcessed = 0
tPrevAbsolutePage = 0

```

intCurPos = 0

```

intNewPos = 0
fSupportsBookmarks = True
fMoveAbsolute = False

If Not IsEmpty(Request(ÒrsCompanyCompanies_PagingMoveÓ)) Then
    tPagingMove = Trim(Request(ÒrsCompanyCompanies_PagingMoveÓ))
End If

If IsEmpty(Session(ÒrsCompanyCompanies_RecordsetÓ)) Then
    fNeedRecordset = True
Else
    If Session(ÒrsCompanyCompanies_RecordsetÓ) Is Nothing Then
        fNeedRecordset = True
    Else
        Set rsCompanyCompanies = Session(ÒrsCompanyCompanies_RecordsetÓ)
    End If
End If

If fNeedRecordset Then
    Set DataConn = Server.CreateObject(ÒADODB.ConnectionÓ)
    DataConn.ConnectionTimeout = Session(ÒDataConn_ConnectionTimeoutÓ)
    DataConn.CommandTimeout = Session(ÒDataConn_CommandTimeoutÓ)
    DataConn.Open Session(ÒDataConn_ConnectionStringÓ), Session(ÒDataConn_RuntimeUserNameÓ),
Session(ÒDataConn_RuntimePasswordÓ)
    Set cmdTemp = Server.CreateObject(ÒADODB.CommandÓ)
    Set rsCompanyCompanies = Server.CreateObject(ÒADODB.RecordsetÓ)
    cmdTemp.CommandText = "SELECT ÓCompany IDÓ, ÓCompanyNameÓ, ÓCitizenship RequirementsÓ, ÓResumes
DueÓ, ÓURLÓ, ÓCover Letter RequiredÓ, ÓGrade Report RequiredÓ, ÓApplication RequiredÓ, ÓNotesÓ FROM ÓCompaniesÓ"
    cmdTemp.CommandType = 1
    Set cmdTemp.ActiveConnection = DataConn
    rsCompanyCompanies.Open cmdTemp, , 1, 3
End If
On Error Resume Next
If rsCompanyCompanies.BOF And rsCompanyCompanies.EOF Then fEmptyRecordset = True
On Error Goto 0
If Err Then fEmptyRecordset = True
If fNeedRecordset Then
    Set Session(ÒrsCompanyCompanies_RecordsetÓ) = rsCompanyCompanies
End If
rsCompanyCompanies.PageSize = tPageSize
fSupportsBookmarks = rsCompanyCompanies.Supports(8192)

If Not IsEmpty(Session(ÒrsCompanyCompanies_FilterÓ)) And Not fEmptyRecordset Then
    rsCompanyCompanies.Filter = Session(ÒrsCompanyCompanies_FilterÓ)
    If rsCompanyCompanies.BOF And rsCompanyCompanies.EOF Then fEmptyRecordset = True
End If

If IsEmpty(Session(ÒrsCompanyCompanies_PageSizeÓ)) Then Session(ÒrsCompanyCompanies_PageSizeÓ) = tPageSize
If IsEmpty(Session(ÒrsCompanyCompanies_AbsolutePageÓ)) Then Session(ÒrsCompanyCompanies_AbsolutePageÓ) = 1

If Session(ÒrsCompanyCompanies_PageSizeÓ) <> tPageSize Then
    tCurRec = ((Session(ÒrsCompanyCompanies_AbsolutePageÓ) - 1) * Session(ÒrsCompanyCompanies_PageSizeÓ)) + 1
    tNewPage = Int(tCurRec / tPageSize)
    If tCurRec Mod tPageSize <> 0 Then

```

```

    tNewPage = tNewPage + 1
End If
If tNewPage = 0 Then tNewPage = 1
Session(ÒrsCompanyCompanies_PageSizeÓ) = tPageSize
Session(ÒrsCompanyCompanies_AbsolutePageÓ) = tNewPage
End If

If fEmptyRecordset Then
    fHideNavBar = True
    fHideRule = True
Else
    tPrevAbsolutePage = Session(ÒrsCompanyCompanies_AbsolutePageÓ)
    Select Case tPagingMove
        Case ÓÓ
            fMoveAbsolute = True
        Case ÓRequeryÓ
            rsCompanyCompanies.Requery
            fMoveAbsolute = True
        Case Ó<<Ó
            Session(ÒrsCompanyCompanies_AbsolutePageÓ) = 1
        Case Ó<Ó
            If Session(ÒrsCompanyCompanies_AbsolutePageÓ) > 1 Then
                Session(ÒrsCompanyCompanies_AbsolutePageÓ) = Session(ÒrsCompanyCompanies_AbsolutePageÓ) - 1
            End If
        Case Ó>Ó
            If Not rsCompanyCompanies.EOF Then
                Session(ÒrsCompanyCompanies_AbsolutePageÓ) = Session(ÒrsCompanyCompanies_AbsolutePageÓ) + 1
            End If
        Case Ó>>Ó
            If fSupportsBookmarks Then
                Session(ÒrsCompanyCompanies_AbsolutePageÓ) = rsCompanyCompanies.PageCount
            End If
    End Select
Do
    If fSupportsBookmarks Then
        rsCompanyCompanies.AbsolutePage = Session(ÒrsCompanyCompanies_AbsolutePageÓ)
    Else
        If fNeedRecordset Or fMoveAbsolute Or rsCompanyCompanies.EOF Then
            rsCompanyCompanies.MoveFirst
            rsCompanyCompanies.Move (Session(ÒrsCompanyCompanies_AbsolutePageÓ) - 1) * tPageSize
        Else
            intCurPos = ((tPrevAbsolutePage - 1) * tPageSize) + tPageSize
            intNewPos = ((Session(ÒrsCompanyCompanies_AbsolutePageÓ) - 1) * tPageSize) + 1
            rsCompanyCompanies.Move intNewPos - intCurPos
        End If
        If rsCompanyCompanies.BOF Then rsCompanyCompanies.MoveNext
    End If
    If Not rsCompanyCompanies.EOF Then Exit Do
    Session(ÒrsCompanyCompanies_AbsolutePageÓ) = Session(ÒrsCompanyCompanies_AbsolutePageÓ) - 1
Loop
End If

Do
    If fEmptyRecordset Then Exit Do

```

```

If tRecordsProcessed = tPageSize Then Exit Do
If Not fFirstPass Then
    rsCompanyCompanies.MoveNext
Else
    fFirstPass = False
End If
If rsCompanyCompanies.EOF Then Exit Do
tRecordsProcessed = tRecordsProcessed + 1
%>
<!--METADATA TYPE=ÓDesignerControlÓ ends-->

<%
If strFormMode = ÓEditÓ Then
    Response.Write Ó<P>Ó
    Response.Write Ó<TABLE WIDTH=100% CELSPACING=0 CELLPADDING=2 BORDER=0>Ó
    ShowField ÓCompany IDÓ, ÓCompany IDÓ, True, Null
    ShowField ÓCompanyNameÓ, ÓCompanyNameÓ, False, Null
    ShowField ÓCitizenship RequirementsÓ, ÓCitizenship RequirementsÓ, False, avarCompanyCitReq
    ShowField ÓResumes DueÓ, ÓResumes DueÓ, False, Null
    ShowField ÓURLÓ, ÓURLÓ, False, Null
    ShowField ÓCover Letter RequiredÓ, ÓCover Letter RequiredÓ, False, Null
    ShowField ÓGrade Report RequiredÓ, ÓGrade Report RequiredÓ, False, Null
    ShowField ÓApplication RequiredÓ, ÓApplication RequiredÓ, False, Null
    ShowField ÓNotesÓ, ÓNotesÓ, False, Null
    Response.Write Ó</TABLE>Ó
    Response.Write Ó</FORM></P>Ó
    stQueryString = Ó?FormMode=EditÓ
    fHideNavBar = False
    fHideRule = True
Else
    fHideNavBar = True
    fHideRule = True
End If
%>

<!--METADATA TYPE=ÓDesignerControlÓ startspan
<OBJECT ID=ÓDataRangeFtr1Ó WIDTH=151 HEIGHT=24
    CLASSID=ÓCLSID:F602E722-A281-11CF-A5B7-0080C73AAC7EÓ>
</OBJECT>
-->
<%
Loop
If tRangeType = ÓTableÓ Then Response.Write Ó</TABLE>Ó
If tPageSize > 0 Then
    If Not fHideRule Then Response.Write Ó<HR>Ó
    If Not fHideNavBar Then
        %>
        <TABLE WIDTH=100% >
        <TR>
            <TD WIDTH=100% >
                <P ALIGN=<%= tBarAlignment %> >
                <FORM <%= ÓACTION=ÓÓÓ & Request.ServerVariables(ÓPATH_INFOÓ) & stQueryString & ÓÓÓÓ %>
METHOD=ÓPOSTÓ>
                <INPUT TYPE=ÓSubmitÓ NAME=Ó<%= tHeaderName & Ó_PagingMoveÓ %>Ó VALUE=Ó &lt;&lt; Ó>

```

```

<INPUT TYPE=ÓSubmitÓ NAME=Ó<%= tHeaderName & Ò_PagingMoveÓ %>Ó VALUE=Ó &lt; Ò>
<INPUT TYPE=ÓSubmitÓ NAME=Ó<%= tHeaderName & Ò_PagingMoveÓ %>Ó VALUE=Ó &gt; Ò>
<% If fSupportsBookmarks Then %>
  <INPUT TYPE=ÓSubmitÓ NAME=Ó<%= tHeaderName & Ò_PagingMoveÓ %>Ó VALUE=Ó &gt;&gt; Ò>
<% End If %>
<% If Not fHideRequery Then %>
  <INPUT TYPE=ÓSubmitÓ NAME=Ó<%= tHeaderName & Ò_PagingMoveÓ %>Ó VALUE=Ó Requery Ò>
<% End If %>
</FORM>
</P>
</TD>
<TD VALIGN=MIDDLE ALIGN=RIGHT>
  <FONT SIZE=2>
  <%
  If Not fHideNumber Then
    If tPageSize > 1 Then
      Response.Write Ò<NOBR>Page: Ò & Session(tHeaderName & Ò_AbsolutePageÓ) & Ò</NOBR>Ó
    Else
      Response.Write Ò<NOBR>Record: Ò & Session(tHeaderName & Ò_AbsolutePageÓ) & Ò</NOBR>Ó
    End If
  End If
  %>
  </FONT>
</TD>
</TR>
</TABLE>
<%
End If
End If
%>
<!--METADATA TYPE=ÓDesignerControlÓ ends-->

<%
If strFormMode <> ÒEditÓ Then
  Response.Write Ò<P>Ó
  Response.Write Ò<TABLE WIDTH=100% CELSPACING=0 CELLPADDING=2 BORDER=0>Ó
  ShowField ÒCompany IDÓ, ÒCompany IDÓ, True, Null
  ShowField ÒCompanyNameÓ, ÒCompanyNameÓ, False, Null
  ShowField ÒCitizenship RequirementsÓ, ÒCitizenship RequirementsÓ, False, avarCompanyCitReq
  ShowField ÒResumes DueÓ, ÒResumes DueÓ, False, Null
  ShowField ÒURLÓ, ÒURLÓ, False, Null
  ShowField ÒCover Letter RequiredÓ, ÒCover Letter RequiredÓ, False, Null
  ShowField ÒGrade Report RequiredÓ, ÒGrade Report RequiredÓ, False, Null
  ShowField ÒApplication RequiredÓ, ÒApplication RequiredÓ, False, Null
  ShowField ÒNotesÓ, ÒNotesÓ, False, Null
  Response.Write Ò</TABLE>Ó
  Response.Write Ò</FORM><<P>Ó
End If
%>

<!------- Footer Section ----->

<%
Ó Display a message if there are no records to show

```

```

If strFormMode = "Edit" And fEmptyRecordset Then
    Response.Write "<p align=left>No Records Available</p>"
End If
' TEMP: This is here until we get a drop of the data range that has
' the CacheRecordset property
If fNeedRecordset Then
    Set Session("rsCompanyCompanies_Recordset") = rsCompanyCompanies
End If
%>

</BODY>
</HTML>

```

B.4 Example HTML Extended With Directives: Image Gallery

```

<!--
[# #####]
[#                                     #]
[# PRODUCT.EDIT.HTML                               #]
[# Microsoft Merchant Server v1.00                   #]
[#                                     #]
[# Copyright (c) 1996 Microsoft Corporation. All rights reserved. #]
[#                                     #]
[# #####]
-->

[nocache]

[# -- definitions: ]
[include "define.html"]

[# -- page vars: ]
[let pageTitle concat("Edit Product ", args.sku)]

[# -- header: ]
[include "admin.header.html"]

[# -- body: ]
[fetchrows product "product" args.sku]

<TABLE BORDER=0>
<TR>
<TD VALIGN=TOP>
    [form "product.delete.html" sku=product.sku]
        <INPUT TYPE="submit" VALUE="Delete Product...">
    [/form]

[xform admin.update table=var.tableProducts keys=sku goto="product.list.html" __sku=args.sku]

```

```
[include Òproduct.valid.htmlÓ]
```

```
<INPUT TYPE=ÓHIDDENÓ NAME=ÓskuÓ VALUE=Ó[value product.sku]Ó>
```

```
<INPUT TYPE=ÓsubmitÓ VALUE=ÓUpdate ProductÓ>
```

```
</TD>
```

```
<TD VALIGN=TOP>
```

```
<TABLE >
```

```
<TR>
```

```
  [# label: ]
```

```
  <TD VALIGN=TOP>
```

```
    ISBN:
```

```
  </TD>
```

```
  [# value: ]
```

```
  <TD VALIGN=TOP>
```

```
    <INPUT
```

```
      TYPE=ÓtextÓ
```

```
      SIZE=47
```

```
      NAME=ÓisbnÓ
```

```
      VALUE = Ò[value product.isbn]Ó>
```

```
    </TD>
```

```
</TR>
```

```
<TR>
```

```
  [# label: ]
```

```
  <TD VALIGN=TOP>
```

```
    Title:
```

```
  </TD>
```

```
  [# value: ]
```

```
  <TD VALIGN=TOP>
```

```
    <INPUT
```

```
      TYPE=ÓtextÓ
```

```
      SIZE=47
```

```
      NAME=ÓtitleÓ
```

```
      VALUE = Ò[value product.title]Ó>
```

```
    </TD>
```

```
</TR>
```

```
<TR>
```

```
  [# label: ]
```

```
  <TD VALIGN=TOP>
```

```
    Subtitle:
```

```
  </TD>
```

```
  [# value: ]
```

```
  <TD VALIGN=TOP>
```

```
    <TEXTAREA
```

```
      COLS=51
```

```
      ROWS=5
```

```
      NAME=ÓsubtitleÓ
```

```
      WRAP = ÒvirtualÓ>[value product.subtitle]</TEXTAREA>
```

```
    </TD>
```

```

</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Short Desc:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
    <TEXTAREA
      COLS=51
      ROWS=5
      NAME=Óshort_descriptionÓ
      WRAP = ÒvirtualÓ>[value product.short_description]</TEXTAREA>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Long Desc:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
    <TEXTAREA
      COLS=51
      ROWS=5
      NAME=Ólong_descriptionÓ
      WRAP = ÒvirtualÓ>[value product.long_description]</TEXTAREA>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Publisher:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
    <INPUT
      TYPE=ÓtextÓ
      SIZE=32
      NAME=ÓpublisherÓ
      VALUE = Ò[value product.publisher]Ó>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Date Pub:
  </TD>

```

```

[# value: ]
<TD VALIGN=TOP>
  <INPUT
    TYPE=ÓtextÓ
    SIZE=32
    NAME=Ódate_publishedÓ
    VALUE = Ò[date product.date_published]Ó>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Subject:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
  <SELECT NAME=Ósubject_idÓ>
    [fetchrows subject ÒsubjectsÓ]
    [eachrow subject]
      [option subject.subject_id product.subject_id] [value subject.subject_name]
    [/eachrow]
  </SELECT>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Author Id:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
  <SELECT NAME=Óauthor_idÓ>
    [fetchrows author ÒauthorsÓ]
    [eachrow author]
      [option author.author_id product.author_id] [value author.first_name] [value author.last_name]
    [/eachrow]
  </SELECT>
  </TD>
</TR>

<TR>
  [# label: ]
  <TD VALIGN=TOP>
    Featured:
  </TD>

  [# value: ]
  <TD VALIGN=TOP>
  <SELECT NAME=ÓfeaturedÓ>

```

```

        [option 1 product.featured] Yes
        [option 0 product.featured] No
    </SELECT>
</TD>
</TR>
<TR>
    [# label: ]
    <TD VALIGN=TOP>
        Rating:
    </TD>

    [# value: ]
    <TD VALIGN=TOP>
        <INPUT
            TYPE=ÓtextÓ
            SIZE=32
            NAME=ÓratingÓ
            VALUE = Ò[value product.rating]Ó>
    </TD>
</TR>
<TR>
    [# label: ]
    <TD VALIGN=TOP>
        Level:
    </TD>

    [# value: ]
    <TD VALIGN=TOP>
        <SELECT NAME=Ólevel_idÓ>
            [fetchrows level ÒlevelsÓ]
            [eachrow level]
                [option level.level_id product.level_id] [value level.level_name]
            [/eachrow]
        </SELECT>
    </TD>
</TR>
<TR>
    [# label: ]
    <TD VALIGN=TOP>
        Disk:
    </TD>

    [# value: ]
    <TD VALIGN=TOP>
        <INPUT
            TYPE=ÓtextÓ
            SIZE=32
            NAME=ÓbookdiskÓ
            VALUE = Ò[value product.bookdisk]Ó>
    </TD>
</TR>
<TR>

```

```

[# label: ]
<TD VALIGN=TOP>
  List Price:
</TD>

[# value: ]
<TD VALIGN=TOP>
  <INPUT
    TYPE=ÓtextÓ
    SIZE=32
    NAME=Ólist_priceÓ
    VALUE = Ò[money product.list_price]Ó>
</TD>
</TR>

</TABLE>
</TD>
<TD VALIGN=TOP>
  <IMG SRC=Ó[img]books/[value product.isbn].gifÓ></TD>
</TD>
</TR>
</TABLE>
[/xform]

[# -- footer: ]
[include Òadmin.footer.htmlÓ]

```


References

- [1] Communications of the ACM, Vol. 13, No. 6, June 1970
- [2] OMG Background Information:
<http://www.dbis.informatik.uni-frankfurt.de/WF/Hall1/A8/bacgrnd.html>
- [3] C.J. Date and Hugh Darwen: A Guide to the SQL Standard (Third Edition, Addison-Wesley, 1993)
- [4] Tim Brinson; Reviewer, Fresco & OpenDoc Comparison; Compound Presentation Facility, September 8, 1995
<http://www.omg.org/docs/1995/95-09-16.txt>
- [5] Douglas C. Schmidt, Distributed Object Computing with CORBA
<http://www.cs.wustl.edu/~schmidt/corba.html>
- [6] CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Communications Magazine, Vol. 14, No. 2, February, 1997.
- [7] OMG Home Page: <http://www.omg.org/>
- [8] OMG Publication: The Common Object Request Broker: Architecture and Specification; Revision 2.0 Copyright July 1995
- [9] OMG Publication, John Siegel Ph.D.: Common Object Services Specification; First Edition, Revision 1.0 Copyright 1994 OMG Doc# 94-1-1
- [10] Microsoft Corporation and Digital Equipment Corporation: The Component Object Model Specification; Draft Version 0.9, October 24, 1995 Copyright © 1992-95 Microsoft Corporation. Available at
<http://www.microsoft.com/oledev/olecom/title.htm>
- [11] To-yat Cheung, Joseph Fong, Brian Siu; Editors: Database Reengineering and Interoperability ISBN 0-306-45288-X, Copyright 1996 Plenum Press, NY
- [12] A. Abdellatif, J. Le Bihan, M. Limame: Oracle A user's guide ISBN 0-333-54215-0, Copyright 1990, Macmillan Education Ltd.
- [13] Trevor Harmon. Web Developers Guide to Visual J++ & ActiveX ISBN 1-57610-062-6, Copyright 1996, Coriolis Group Inc.
- [14] SIGMOD Record; Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data ISBN 0-89791-794-4, Copyright 1996, Association for Computing Machinery
- [15] Alexander Schill, Christian Mittasch, Otto Spaniol, Claudia Popien; Editors: Distributed Platforms ISBN 0-412-73280-7, Copyright 1996, Chapman & Hall
- [16] Lotus Corp.: The Kona InfoBus Technology Brief <http://kona.lotus.com/21c2.html>
- [17] Sun Microsystems, Javasoft: The JavaBeans 1.0 API specification Version 1.00-A Copyright 1996, Sun Microsystems, Available for download at
<http://splash.javasoft.com/beans/spec.html>
- [18] The Economist Magazine; In Search of a Perfect Market; A Survey of Electronic Commerce. May 10, 1997. Copyright 1997, The Economist Newspaper Ltd. Available at <http://www.economist.com/surveys/elcom/>