DESIGN AND IMPLEMENTATION OF A WEB DATABASE MANAGEMENT SYSTEM

By

YIJING ZHANG

Bachelor of Science

Zhejiang University of Technology

Hangzhou, China

1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
In partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1998

DESIGN AND IMPLEMETATION OF A WEB DATABASE MANAGEMENT SYSTEM

Thesis Approved:

H. Lu
Thesis Adviser

[illegible]

J. Chandler

Wayne B. Powell
Dean of the Graduate College

ACKNOWLEDGEMENTS

I sincerely thank my major advisor, Dr. Huizhu Lu for her intelligent supervision, constructive guidance, warm encouragement and valuable time she has given to me toward the completion of my thesis. My sincere appreciation extends to Dr. J.P. Chandler and Dr. G. E. Hedrick for serving on my committee, whose guidance, encouragement, assistance and friendships are also invaluable.

Moreover, I wish to express my sincere gratitude to Dr.Yue Zhang, Mr. Yuwen Lin, and Mr. Hongchi Su for their suggestions and helps.

I would like to give my special thanks to my husband, Dr. Jiaqi Cai, for his precious suggestions, encouragement, patience, love and understanding throughout this whole process. My respectful thanks go to my parents Mr. Junhong Zhang, Mrs. Binfeng Ma, and my brother Mr. Yiqun Zhang, for their support and encouragement.

Finally, I would like to thank all the faculty of the Department of Computer Science for supporting during these two years's study.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. LITERATURE REVIEW 5

Active Server Page 5

Open Database Connectivity 8

Structured Query Language 12

Component Object Model 14

III. REQUIREMENT SPECIFICATION OF THE SYSTEM 19

Requirement Analysis of the System 19

User Requirement Specification 19

Functional Requirement Specification 22

Formal Requirement Specification 25

Search Operations 27

Modify and Update Product Information 30

Modify and Update Company Information 34

IV. IMPLEMENTATION AND RESULTS 38

Email COM 38

Background 38

Active Template Library 38

Socket 39

Chapter Page

Implementation 42

Navigation Record Set 48

Customized Message Box 52

Some Outputs of this Database Management System 54

V. SUMMARY, CONCLUSIONS AND SUGGESTED FUTURE WORK 83

BIBLIOGRAPHY 85

APPENDIXES 87

APPENDIX A – NOTATIONS OF Z SPECIFICATION LANGUAGE USED IN THIS THESIS 88

APPENDIX B -- ABRVIATION USED IN THIS THESIS 89

APPENDIX C -- GLOSSARY OF TERMS IN THIS THESIS 91

LIST OF FIGURES

Figure Page

1. Four Components of ODBC Architecture 9
2. Basic Flow Control for ODBC Applications 11
3. A COM Object's Services Accessed via Its Interface 14
4. Component Object Model Vtable 15
5. Three Kind of Servers 17
6. E-R Diagram of Electronic Parts Database Management System 20
7. Users and Their Operations 23
8. Protocols and Networks in the TCP/IP Model Initially 39
9. Socket Communications for Connection-oriented Protocols 41
10. Code for atlmail.h 42
11. Structure for COM's Binary String 44
12. Code for atlmail.cpp 45
13. An ASP Program Calls E-mail COM 48
14. Code for Navigating the Record Set 48
15. Code for Customized Error Message Box 52
16. Code for Customized Confirmation Box 53
17. Code in Confirmbox.html 53
18. Three Types of Users 57
19. The Example of User Search Page 58

Figure Page

20. The Output of Searching for Products Made by King/Allied Signal Company.... 59

21. View Product 60

22. Company's Home Page 61

23. Searching Product by Avionics Catalog 62

24. Search Result for Avionics Products 63

25. Selection of Product: TMA-330D, TN-200D, and TX-760D 64

26. Order Forms 65

27. Completed Order Form 66

28. Message Box of Order Status 67

29. Order Mail 1 68

30. Order Mail 2 69

31. Order Mail 3 70

32. Log-on Page for Company Administrator 71

33. First Record of Product in This Company 72

34. Enable Previous Button 73

35. Disable Next Button 74

36. Insertion with Invalid Data 75

37. Error Message 76

38. Insertion with Valid Data 77

39. A Success Insertion 78

Figure Page

40. A Confirmation Box In Action 79

41. Current Record Success Removed 80

42. Log-on Page of Server Administrator 81

43. Company Information Page 82

LIST OF TABLES

Table Page

I. The Socket Primitives for TCP 40

II. Cursor Types, Meanings, and Values 51

CHAPTER I

Introduction

Today, the World Wide Web is gaining increasing popularity. People utilize the Web for information inquiry, entertainment, and shopping. More and more businesses realize the importance of the Internet, and actively seek opportunities through the Internet. The increasing presence of businesses on the World Wide Web reflects the intensive effect of marketing on the Web. Facing a huge volume of product information, potential buyers who want to make comparisons among different brands and/or manufacturers encounter the difficulty of information selection. This thesis intends to provide a bridge between buyers and manufacturing companies, providing the buyers with the information about different products and companies on the Internet from one site, saving the potential buyer substantial effort in their searching. With such a bridge the companies will be able to maintain and update their product information in the database remotely; the customers can search and compare commodities they are interested in, and place orders directly from this site. An order placed can be sent to the manufacturer directly.

This study consists of two phases: the design of an abstract model using requirement specification and the implementation of the system.

In developing high quality software, one of the crucial stages is to write a software requirement specification document. The requirement specification focuses carefully upon what the software is supposed to do. We must make sure that requirement specification statements completely and systematically describe what the software needs to do. A good software requirement specification should have the following features: first, it should be unambiguous; second, it should be accurate; finally it should be complete. It is common that misunderstandings between the client and the developer cause compromises which conceal some important requirements and may lead to dissatisfaction by the client. A formal specification provides a way to achieve no compromise (Ford, 1993). "Formal methods originated from the work of Dijkstra and Hoare on program verification, and Scott, Strachey and others on program semantics"(Goldsack and Kent, 1996). Many mathematical languages were developed from program specifications. One of the most established languages is Z. Created by J.R. Abrial, the Z language has been expanded and industrially applied by many others in the 1980s. Z is a "model-based" formal specification language, that is, it uses logic and set theory to build abstract models of required systems using set sequences and functions (Goldsack and Kent, 1996). Since the Z language is based on the mathematical discipline of first-order-logic and set theory, the Z specification has many advantages. It leads to concise, unambiguous and exact specifications through which the reasoning process easily can be seen. First-order logic deals with predicates, also known as propositions, which are either true or false. Z contains true and false (two primitive predicates), which are the always true and the always false propositions respectively (Diller, 1990).

This study uses Microsoft SQL Server 6.5 to create a database system and use ASP (Active Server Page), JavaScript and VBScript to provide an Internet interface. Microsoft SQL Server 6.5 uses SQL commands to create and to maintain databases. SQL is a language that provides a tool that can create and operate a relational database; that is, sets of related information stored in tables. Using SQL, relational database information can be created, retrieved, updated and transferred. Because of its elegance and independence from machine specifics, as well as the support from the industry leaders in relational database technology, SQL has become the standard language in relational database. The SQL standard is defined by the American National Standards Institute (ANSI) and is accepted by the International Standard Organization (ISO) (Gruber, 1990).

While SQL is employed to create the database management system, JavaScript and VBScript are employed to provide the Web interface. However, the functionality of JavaScript and VBScript is limited. The Component Object Model, better known as COM, is used to build a custom component providing extra functions. COM is a method for developing software components that are small binary executables providing services for applications, operating systems, and other components. Developing custom COM components is like developing dynamic-object-oriented API (Application Programming Interface). COM components are connected to form applications or systems of components. As indicated by Rogerson "components can be unplugged and replaced at run-time without re-linking or recompiling the application." (Rogerson, 1997). In this thesis, Active Template Library is used to build a DLL (Dynamic Link Library) as the component.

The objectives of the thesis are to create a multi-function relational database management system, Electronic Parts Database Management, for both manufacturers and customers; to provide a user-friendly web-interface for this database system. Customers are able to access all information about different products and companies from one site. The companies can remotely maintain and update product information of their own company. The customers are allowed to place orders directly from the web site.

This thesis consists of five chapters. The current chapter (Chapter I) is introduction, providing an overview of formal requirement specification, Z specification language, Microsoft Active Server Page, Structured Query Language and Component Object Model.

Chapter II is the literature review for this thesis. It includes ASP, ODBC (Open Database Connectivity), SQL, and COM.

Chapter III is the System requirement specification. It presents the requirement specification statements, and it analyzes the statements for the Electronic Parts Database Management System.

Chapter IV is the implementation of the Electronic Parts Database Management System. It states how the E-mail COM is created, how the customized message box is implemented, and how the record set navigation is achieved through the Web interface.

Chapter V is the summary, conclusions and suggested future work of this thesis.

CHAPTER II

Literature Review

As mentioned in the introduction, this thesis provides a tool by which a user can search product information through an Web interface. Active Server Page (ASP) provides a way that dynamically presents information to a Web page corresponding to the users request.

2.1 Active Server Page

Active Server Page (ASP) is a technology that is used in the creation of dynamic web content by supporting several server-side programming languages to generate an HTML file. HTML, the HyperText Markup Language, is a standard language to provide a common presentation method to globally shared information on World Wide Web (Fedorchek and Rensin, 1997).

Being a presentation description language rather then a programming language, HTML doses not determine or control a program's behavior. To address this problem, Common Gateway Interface (CGI) is introduced as a solution. CGI presents dynamically generated information on the World Wide Web. It allows the computer to generate Web pages according to the user's request. It lets the Internet offer interactive advanced user-driven applications. CGI opens up an entire class of modern applications for the Web (Gundaraim, 1996). It is a set of standards that describe how a web server program should communicate with an external application. Pages can be created by an external program which returns dynamic HTML. But CGI applications suffer from some problems. The major problem of CGI is that it can lead to slow performance, because whenever a user loads a Web page that calls the CGI application, web server machine creates a new process. When sites grow bigger and CGI applications grow more complex, the problem would become more severe.

An alternative to CGI in the presentation of dynamic web page is Active server Page (ASP). The majority of ASP functionality is controlled by an asp.dll file, which is an OLE (Object Linking and Embedding) component. According to Fedorchek and Rensin (Fedorchek and Rensin, 1997) "The first time a call is made to a page that is to be handled by ASP, the web server creates an instance of the ASP OLE component in memory and passes it on to the request. From then on, that same component handles all user requests. This means that every call to ASP after the first will be extremely fast." ASP runs in the same memory space with the web server application since it is an in-process component, and it does not require the operating system to start a new process for each user.

ASP usually performs its work on server-side, which provides the security. ASP provides some built-in objects and components that make programming much easier. The built-in objects are as follows(Fedorchek and Rensin, 1997):

• Application:	used to share information with other users who are accessing the same ASP application
• Request:	used to retrieve information submitted by a client
• Response:	used to send output data to client browser
• Server:	used to access some internal properties of the web server

• Session:	used to track information specific to a given user's session

ASP not only provides the objects, but also provides a way enabling developers to add in their own objects. The following are five components that are provided with ASP (Fedorchek and Rensin, 1997):

• The Ad Rotator Component:	automatically rotates advertisements displayed on a page
• The Browser Capabilities Component:	determines the capabilities of the client browser
• The Database Access Component:	interacts with any ODBC-compliant database.
• The Content Linking Component:	creates a table of contents for web pages that the information of the site can be organized like pages in a book.
• The File Access Component:	creates and reads text files from the local disk.

Relational databases information can be created, retrieved, altered and transferred by using ASP built-in component, the Database Access Component, through Open Database Connectivity technology.

2.2 Open Database Connectivity

The Open Database Connectivity (ODBC) interface defines a group of functions. An application can use the interface to access a Database Management System (DBMS) by Structured Query Language (SQL).

ODBC defines a method to connect to a data source. This open connectivity is accomplished by a common method of accessing the database on which the application and database must agree. This agreement is implemented as a standard that defines a set of API (Application Programming Interface) function calls and a SQL syntax set. The open connectivity on the database side is provided by drivers which are contained in Dynamically linked libraries (DLL). These drivers transform the ODBC API functions into function calls that are supported by the particular data source being used. These drivers also transform the ODBC SQL syntax into syntax that is acceptable to the data source. Therefore different data sources can be accessed by just loading their corresponding drivers.

The ODBC architecture consists of four main components: the application, a driver manager, the driver, and the data source (Whiting, Morgan, and Perkins, 1996), as shown in Figure 1.

- Application: A program calls ODBC functions to interact with data sources.
- Drive manager: A Dynamically Linked Library (DLL) that loads drivers and provides a single entry point to ODBC functions for different drivers.
- Driver: A DLL, passes SQL statements from application to data

source, and passes results back to the application.

- Data source: includes the data , the associated database, Management System (DBMS), the platform , and the network used to access the platform.

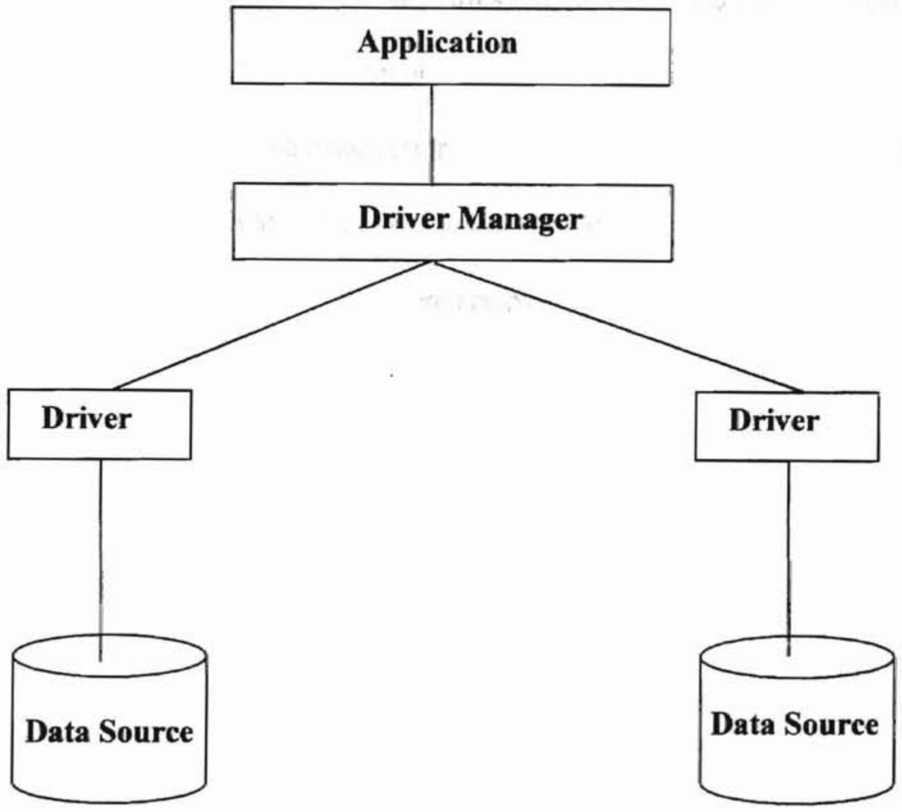


Figure 1.Four Components of ODBC Architecture

(Whiting, Morgan, and Perkins, 1996)

The ODBC interface functions have seven groups of functions, which contain the functions defined by the X/Open and SQL Access Group (SAG) CALL Level Interface (CLI) draft specifications. The SAG CLI groups the interface functions as follows (Signore, Creamer, and Stegman, 1995):

- Allocate and deallocate
    - Environment handle: Identifies the memory storage for global information
    - Connection handle: Identifies the memory location for information about a particular connection
    - Statement handle: Identifies the memory location for information about a SQL statement
- Connection: Establish connection to a server and upon exiting an application, the SAG CLI also provides the closing connection to the server
- Executing SQL statements: There are two ways to execute SQL statements: prepared and direct. If the developers plan to have the application submit the SQL statement multiple times, with possibly changes to the parameter values then, a prepared execution method is used. If application does not require information about a result set prior to completing the SQL request and the developers plan to have the application submit the SQL request only once, then the direct execution method is used.
- Receiving results: Retrieving data from an SQL statement's result set and retrieving information about a result set.
- Transaction control: Allows the developers to commit or roll back a transaction.

- Error handling and miscellaneous: Returns error information and allows attempting the cancellation of a SQL statement.

The elementary flow control for ODBC application is showed as follow (Figure 2)

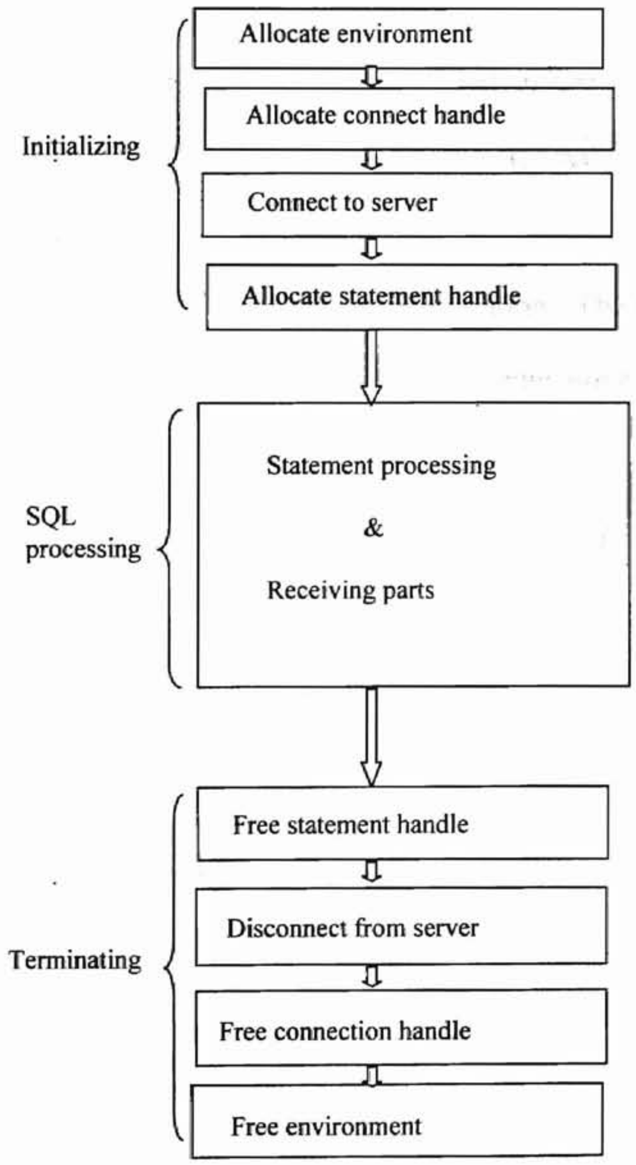


Figure 2. Basic Flow Control for ODBC Applications

(Signore, Creamer, and Stegman, 1995)

Structure Query Language is employed to create and manipulate the data in the database system.

2.2 Structured Query Language (SQL)

A relational database is related information stored in two-dimensional tables. Tables are interrelated so that sophisticated and powerful operations can be performed. The power of the relational database lies in the relationship that the developers can construct between the pieces of information. The relational model now is being used as the primary data model for numerous applications outside the domain of traditional data processing. The first database systems used either the network model or the hierarchical model which are tied more closely to the underlying implementation of the database than is the relational model. There is a substantial theory for a relational database. The theory assist in the design of relational databases and in the efficient processing of user requests for information from database. (Silberschatz, Korth, and Sudarshan, 1997).

In 1970, E.F. Codd introduced the relational model (Codd, 1970). A theoretical basis for database languages was founded. The model consists of simple concepts for recording data and operators for manipulating the information in a database. Codd later published an article that brought forth ideas for the improvement of the initial model (Codd, 1979). Database languages were developed based on the concept and ideas of that relational model. Therefore, these languages are called relational database languages, SQL was one of them.

SQL originated with a project at IBM. The IBM project, known as System R, was to develop an experimental relational database management system, System R. One of the objectives of System R was to demonstrate that the relational model could be implemented in a system and could satisfy the demands of a modern database management system. The Sequel language was chosen as its database language. In the System R project, the language was renamed SQL. The System R project was completed in 1979. IBM publicized the development of system R a great deal during and after the project advocating relational database management systems. In the mean time, other manufacturers began to build relational database management systems. Some of them implemented SQL with their own adaptive adjustment, while others adopted on SQL immediately. As the result, there were a number of competing SQL products on the market, ANSI leading to set up the standard SQL then call for conformation of the standard (Van and Rick, 1988).

The structure of SQL is that it has two groups of SQL commands. One is called Data Definition Language (or DDL also called Schema Definition Language in ANSI) which is used to define relation schemas, delete relations, create indices, and modify relation schemas. The other is called Date Manipulation Language (DML), which is used to insert information into, delete information from, and modify information in the tables at any given time. Data Control Language (DCL) is used to determine whether a user is authorized to perform a particular operation in the database. DCLis considered part of DDL in ANSI (Derlans, 1988).

While SQL is employed to define and manipulating data, JavaScript and VBScript are used to present the data to a user. Unfortunately, the functionality of a scripting language usually is limited. As mentioned in chapter one, ASP allows developers to build their own components. Therefore, component object model (COM) technology is adopted to provide additional functions.

2.3 Component Object Model (COM)

The advantage of using COMs is their abilities to plug into and unplug from an application dynamically. Components link dynamically and hide the details of how they were implemented. A program or a component that uses another component is called a client. A client and a component connected to each other through an interface as shown in Figure 3. Each interface is a contract between the object and its clients.

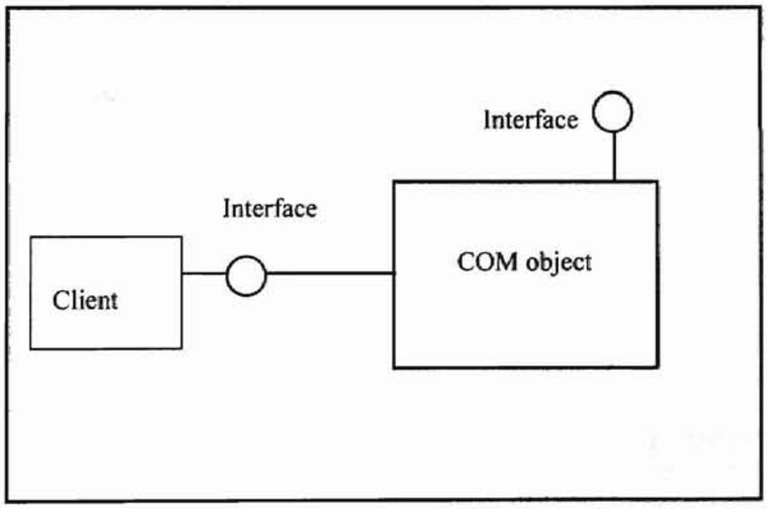


Figure 3. A COM object's Services Accessed via Its Interface

All COM interfaces are required to inherit from IUnKnown, which a standard COM interface. Following is the definition of IUnknown.

```
Class IUknown

{

        virtual HRESULT QueryInterface(REFIID riid, void** ppv) =0;

        virtual ULONG AddRef()=0;

        virtual ULONG Release()=0;

};
```

Each COM interface has three methods: QueryInterface, AddRef, and Release, which are inherited from IUnKnown as the first three functions in its virtual function table (Vtable) (Figure 4). A client can discover whether a component supports a particular interface by passing the interface identifier structure (IID) to QueryInterface. If the COM supports that particular interface, QueryInterface returns a pointer to the interface; otherwise, QueryInterface returns an error code, and the client can ask for another interface.

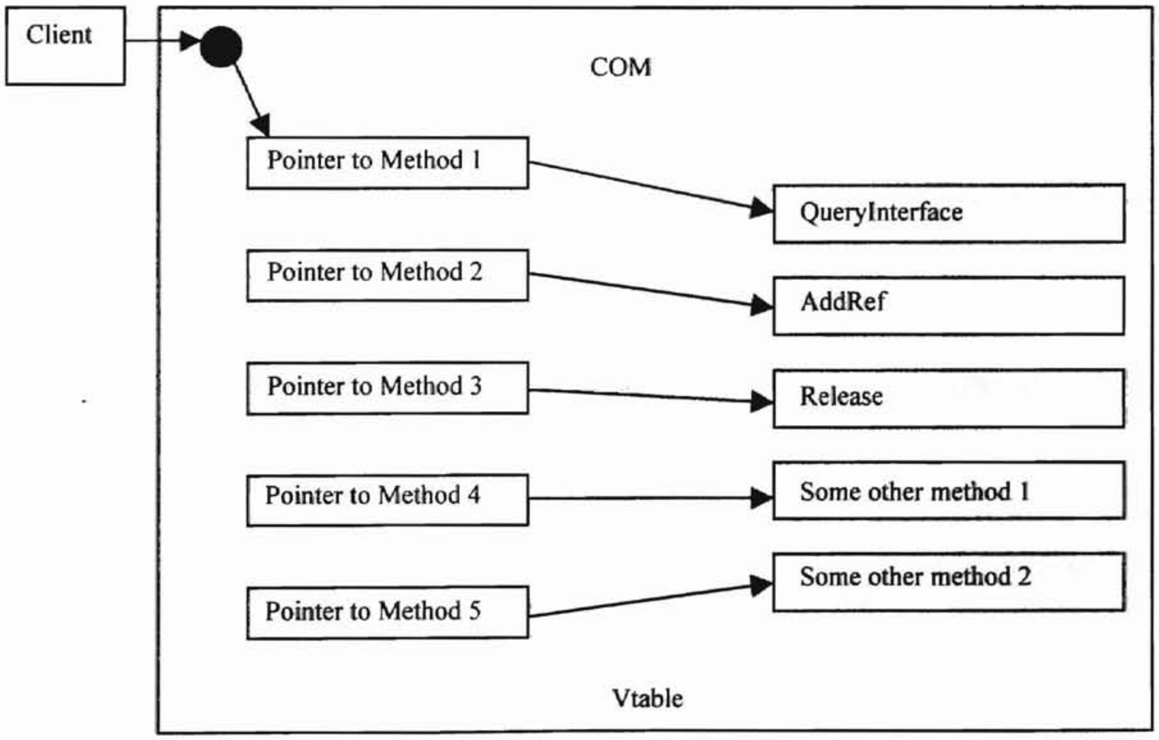


Figure 4. Component Object Model Vtable

Client brings COM into action by calling it. But stopping COM execution should not depend on the client. If there are more than two clients using the COM simultaneously, one client finishes using the COM and stops COM, the other client would mysteriously lose the COM. Therefore COM must have the ability to know when it should stop its own execution. Reference counting is a simple and fast method to enable components to stop their own executions. A running COM component maintains a reference count. When a client gets an interface from a component, the reference count is incremented. This is handled by the AddRef method. When the client finishes the interface, the reference count is decremented, handled by the Release method. Release method also check the reference count if it reaches zero, and if it dose so, Release deletes the component from the memory (unloads COM itself).

Every COM component is implemented inside a server. There are three primary kinds of servers (Chappell, 1996), as shown in Figure 5:

- In-process Servers: implemented in a dynamic link library and execute in the same process as the client
- Local Servers: implemented in a separate process running on the same machine as the client
- Remote Servers: implemented in a DLL or in a separate process that runs on a different machine than the client does, Distributed COM (DCOM) chooses this option

Among the above types of servers, in-process server is chosen because its performance resulted from running in the same memory space as client process provided by Dynamic Link Library (DDL).

Dynamic Link Libraries generally are not executable directly, and they usually do not receive messages. They are separate files containing functions that can be called by programs and other DLLs to perform certain jobs. A DLL is brought into action only when another module calls one of the functions in the library (Petzold, 1996).

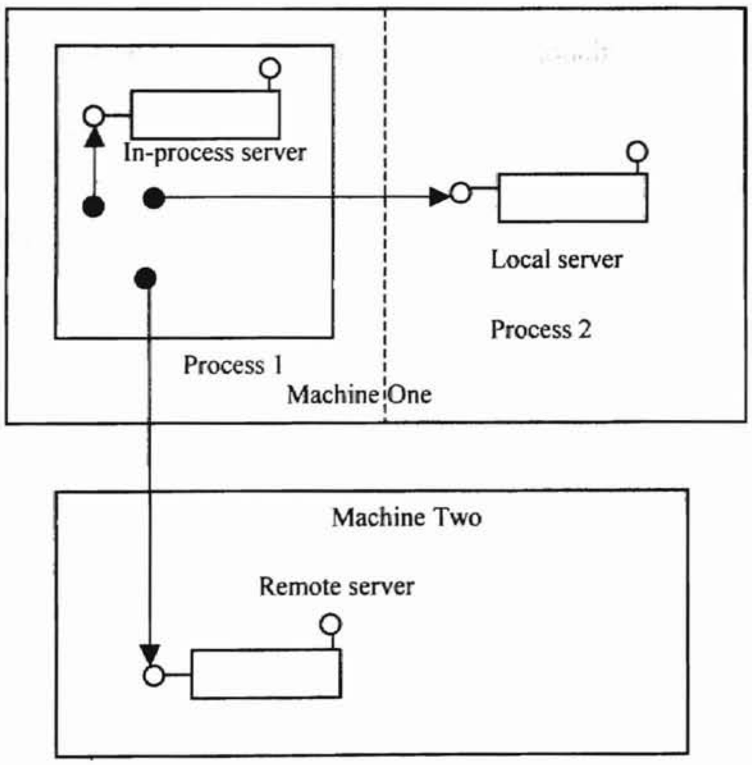


Figure 5. Three Kind of Servers

Template is a technique to create a model for generic functions and classes. A generic function or class is a set of functions or members that can applied to various types of data. The particular data type is specified when it is instantiated. COM has some basic requirements, such as, all interfaces that must inherit from IUnknown. All interfaces must have three functions, which are almost the same to all the COMs. The Active Template Library (ATL) uses template technology to provide basic COM needs. ATL facilitates the creation of small, COM-based components. ATL provides the following features (Armstrong, 1998):

- AppWizard: creates the initial ATL project
- Object Wizard: produces code for basic COM components
- Built-in support for elementary COM functionality such as IUnKnown, Class factories and self-registration
- Support for Microsoft's Interface Definition Language (IDL). This provides marshaling (transferring function arguments and return value across process and machine boundaries) support for custom Vtable interface as well as component self-description through a type library
- Support for IDispatch (Automation) and dual interfaces
- Support for developing efficient ActiveX controls

CHAPTER III

REQUIREMENT SPECIFICATION OF THE SYSTEM

Requirements specification includes both user requirements specification and functional requirements specification. Specification of user requirements is statements that characterize the data needs of the prospective database users. Specification of functional requirements is statements that describe operations or transactions users expect to perform on the data. The following sections present the detailed overview of the Electronic Parts Database Management System. The requirement specification is stated with Z specification language.

3.1 Requirement Analysis for the System

3.1.1 User requirements specification

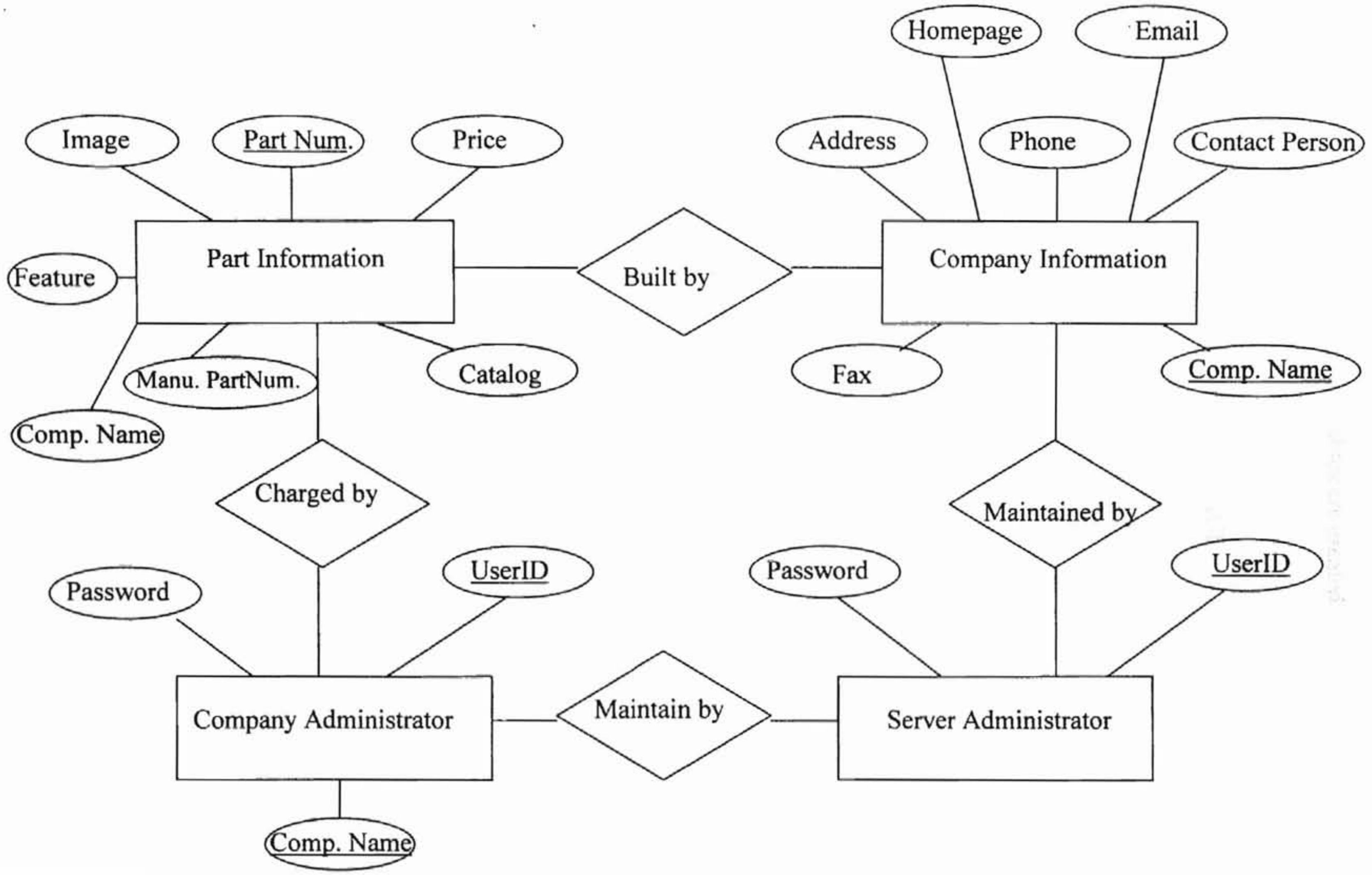
A company is located in a particular, street, city, and is identified by its unique company name. The company can be contacted through a contact person, or the company's email, phone, and fax. The company also provides additional information on its homepage.

A product is made by some particular companies; it is identified by a particular part number. A product has additional information such as features, image, price and catalog.

A company administrator is protected by his or her user id, password, and association with the particular company whose database is his or her charge.

The server administrator also is protected by user id and password. The E-R diagram for the electronic parts database management system is shown in Figure 6.

Figure 6 E-R Diagram of Electronic Parts Database Management System



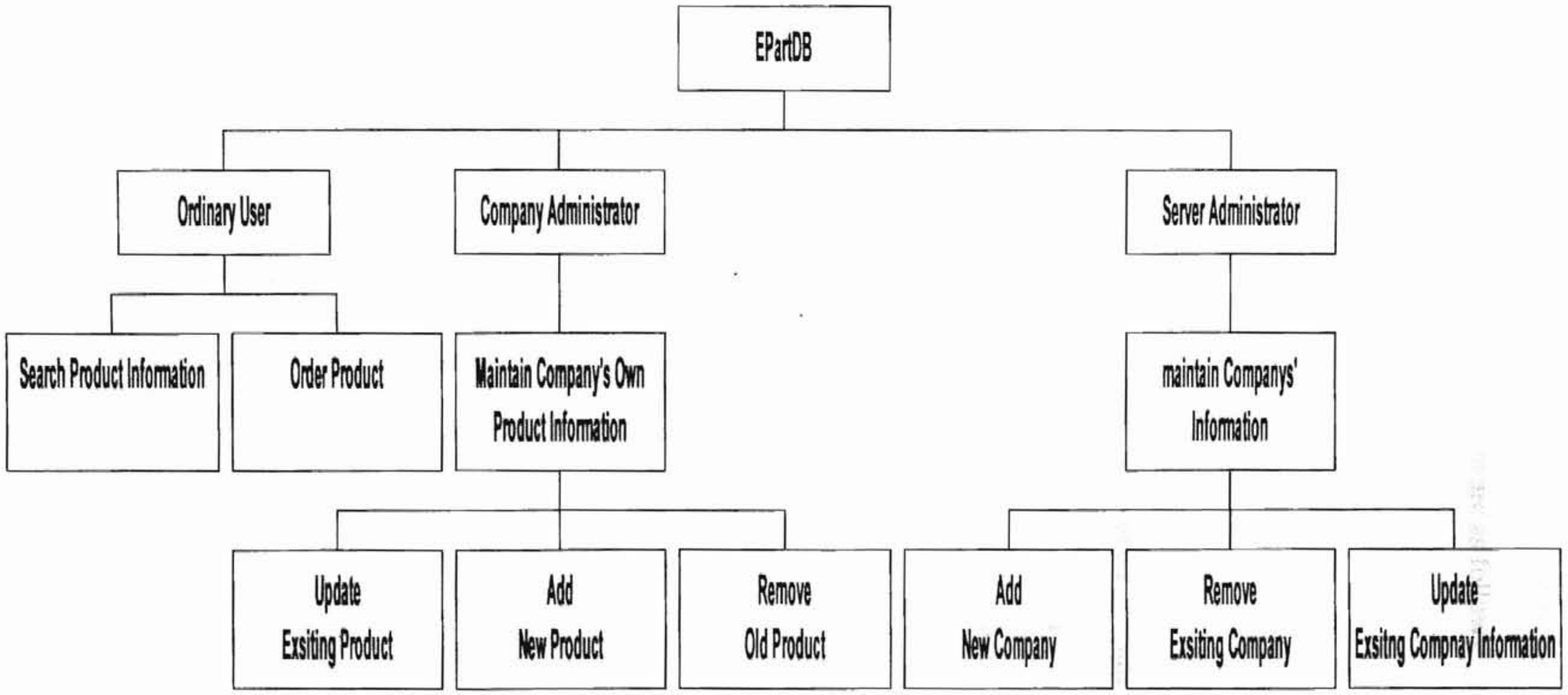
Base on the E-R diagram following tables are created.

```
CREATE TABLE ADTABLE
(SADMOINISTARTOR VARCHAR (50)   NOT NULL,
PASSWORD       VARCHAR (50)  NOT NULL,
PRIMARY KEY (SADMINISTRATOR, PASSWORD))

CREATE TABLE CADIMINISTORTABLE
( COMPANYNAME  VARCHAR (50)       NOT NULL,
CADMINISTRATOR          VARCHAR (50),
PASSWORD  VARCHAR (50)
FOREIGN KEY (COMPANYNAME))

CTREATE TABLE COMPANYTABLE
( COMPANYNAME  VARCHAR (50)       NOT NULL,
WHOLENAME      VARCHAR (100)      NOT NULL,
STREET         VARCHAR (50),
CITY           VARCHAR (30),
STATE          VARCHAR (20),
ZIP            VARCHAR (20),
CONTECTPERSON  VARCHAR (30),
HOMEPAGE       VARCHAR (50),
EMAIL          VARCHAR (50),
PHONE          VARCHAR (30),
FAX            VARCHAR (30),
PRIMARY KEY (COMPANYNAME))

CREATE TABLE PARTTABLE
( COMPANYNAME          VARCHAR (50)      NOT NULL,
MANUPARTNUMBER         VARCHAR (100)     NOT NULL,
CATALOG                VARCHAR (100)     NOT NULL,
PARTNUMBER             VARCHAR (100)     NOT NULL,
PFEATURE               VARCHAR (255),
IMAGEFILE              VARCHAR (255),
PRICE                  NUMERIC (18,2),
PRIMARY KEY (PARTNUMBER, COMPANYNAME))
```

3.1.2 functional requirements specification

Users in three different levels access this electronic parts database system. The first level is for ordinary users who can search all product information of the system, but are not authorized to make modifications. The second level is company administrators who are responsible for maintaining or updating their company product information. The highest level is a server administrator who can maintain and update any company information. When a new company joins, the server administrator must record all company information, the company administrators' user ID and password. Figure 7 shows the operations that can be performed on this database management system.

Figure 7 Users and Their operations



The major transactions in this database system are as follows:

- Search Product Information:
  1. Search product information by company name
  2. Search product information by catalog
  3. Search product information by part number
  4. Search product information by company name and part number
  5. Search product information by company name and catalog
  6. Search product information by catalog and product number
  7. Search product information by company name, catalog and part name
- Maintain and Update Product Information:
  1. Add new product information
  2. Modify existing product information
  3. Remove existing product information
- Maintain and Update Company Information
  1. Add new company information
  2. Modify existing company information
  3. Remove existing company information

Ordinary users only can perform search product operations. Maintaining and updating company product information is restricted to company administrators only. However, server administrators can place themselves as company administrators to modify product information, but maintaining and updating company information only can be performed by server administrators. When a new company applies to join in the database system, company information, its company administrator user ID and password must be recorded. Company administrators only can maintain and update their own product database

3.2 Formal requirement specifications

the Z specification language is adopted for writing requirements specification. In this requirements specification, the following data types are defined:

company = [[companyname, person, password, address,

contactperson, phone, fax, homepage, e-mailaddress]]

product = [[partnumber, price, image, feature, catalog, companyname]]

The data type, *company*, represents any company information that may or may not be in the database. The data type, *product*, represent any product information that may or may not be in the database. The same thing applies to the *partnumber*, *catalog* and *companyname*. The date type *Company* $\in$ *company* represents company information that is currently in the database. The data type *Product* $\in$ *product*, which represents product information, is currently in the database. So do data types *PartNumber* $\in$ *partnumber*, *Catalog* $\in$ *catalog* and *CompanyName* $\in$ *companyname.*

-------- EpartDBState-----------

| CAdministrator, SAdministrator: F person

| CompanyName: F companyname

| Catalog: F catalog

| PartNumber: F partnumber

| Product: F product

| Company: F company

| build: CompanyName $\longrightarrow$ Product

| about: Catalog $\longrightarrow$ Product

| identify: PartNumber $\longrightarrow$ Product

| ensure: CAdministrator $\longrightarrow$ Password

| inchargeof: CAdministrator $\longrightarrow$ CompanyName

| secure: SAdministrator $\longrightarrow$ Password

| aboutcompany: CompanyName $\longrightarrow$ Company

| CAdministrator $\cap$ SAdministrator = { }

Δ EPartDBState $\triangleq$ EPartDBState $\wedge$ EPartDBState'

Ξ EPartDBState $\triangleq$ Δ EPartDBState |

about' = about $\wedge$

build' = build $\wedge$

identify' = identify $\wedge$

ensure' = ensure $\wedge$

inchargeof' = inchargeof $\wedge$

CAdministrator' = CAdministrator $\wedge$

SAdministrator' = SAdministrator $\wedge$

Product' = Product $\wedge$

Catalog' = Catalog $\wedge$

CompanyName' = CompanyName

In the initial state every variable is the empty set:

InitialEPartDBState'=EPartDBState' |

about' = { } $\wedge$

build' = { } $\wedge$

identify' = { } $\wedge$

ensure' = { } $\wedge$

$inchargeof' = \{\ \} \wedge$

$CAdministrator' = \{\ \} \wedge$

$SAdministrator' = \{\ \} \wedge$

$Product' = \{\ \} \wedge$

$Catalog' = \{\ \} \wedge$

$CompanyName' = \{\ \}$

The electronic database system can be interrogated in various ways. The following session describes an operation that outputs all product information by a particular company name, a transaction that lists all the product information about a particular catalog and a transaction that lists all the product information of a particular part number.

3.2.1. Search Operations

3.2.1.1 Search by Company Name

```
-------- SearchByCompanyName-----------
|        Ξ EpartDBState
|        c?: CompanyName
|        out!: F Product
|--------------------------------------------------
|        out! = build (|c?|)
--------------------------------------------------
```

Here c? is an input, and $c? \in$ *CompanyName*, out! is an output. Since *build* is a function that maps from *CompanyName* to *Product*, *build*(|c?|) will give all the information of the products that are made by that company.

3.2.1.2 Search by Catalog

```
-------- SearchByCatalog-----------
|        Ξ EpartDBState
|        a?: Catalog
|        out!: F Product
---------------------------------------------------
|        out! = about (|a?|)
---------------------------------------------------
```

3.2.1.3 Search by Part Number

```
------- SearchByPartNumber----------
|        Ξ EpartDBState
|        p?: PartNumber
|        out!: F Product
--------------------------------------------------
|        out! = identify (|p?|)
---------------------------------------------------
```

This session presents an operation that outputs all the product information by specifying company name and catalog; an operation that outputs all the product information by specifying company name and part number; and an operation that outputs all the product information by specifying part number and catalog.

3.2.1.4 Search by Company and catalog

```
------- SearchByCompanyAndCatalog----------
|        Ξ EpartDBState
|        a?: Catalog
|        c?: CompanyName
|        out!: F Product
--------------------------------------------------
|        out! = build (|c?|) ∩ about (|a?|)
---------------------------------------------------
```

Here a? and c? are inputs. And a?∈ *Catalog*, c? ∈*CompanyName*, out! is an output. The function *about* maps from *Catalog* to *Product*, so *about*(|a?|) will give us all the product information that belongs to that catalog. Since *build*(|c?|) gives all the products that are made by that company, *build* (|c?|) ∩ *about* (|a?|) will give products that are made by that particular company and belong to that specified catalog. It should be noticed that out! may be an empty set.

3.2.1.5 Search by Company and Part Number

```
-------- SearchByCompanyAndPartNumber-----------
|        Ξ EpartDBState
|        p?: PartNumber
|        c?: CompanyName
|        out!: F Product
-------------------------------------------------
|        out! = build (|c?|) ∩ Identify (|p?|)
-------------------------------------------------
```

3.2.1.6 Search by Catalog and Part Number

```
-------- SearchByCatalogAndPartNumber-----------
|        Ξ EpartDBState
|        p?: PartNumber
|        a?: Catalog
|        out!: F Product
-------------------------------------------------
|        out! = about (|a?|) ∩ Identify (|p?|)
--------------------------------------------------
```

The following operation will output all product information by choosing particular company name, part number and catalog.

3.2.1.7 Search by Company, Catalog and Part Number

```
-------- SearchByCompanyCatalogAndPartNumber-----------
|        Ξ EpartDBState
|        c?: Company
|        p?: PartNumber
|        a?: Catalog
|        out!: F Product
-------------------------------------------------
|        out! = about (|a?|) ∩ Identify (|p?|) ∩ build(|c?|)
-----------------------------------------------------------------------
```

Here a?, c? and p? are inputs, $c? \in$ *CompanyName*, $a? \in$ *Catalog*, $p? \in$ *PartNumber*. The function *identify* maps from *PartNumber* to *Product*. Because *identify* (|p?|) gives product information that is identified by the *PartNumber*, *build* (|c?|)

gives a list of products that are made by the company, and *about*(|a?|) gives a list of products that belongs to that specified catalog, therefore, outputs out! is the product that is made by that particular company, belonged to that specified catalog and can be identified by that specified part number. Again out! may be an empty set.

3.2.2 Modify and Update Product Information

3.2.2.1. Add a New Product

```
-------- AddNewProduct---------------
|        Δ EPartDBState
|        p?: product
|        n?: person
|        c?: catalog
|        a?: partnumber
|--------------------------------------------
|        a? ∉ dom identify
|        p? ∉ ran identify
|        n? ∈ dom inchargeof
|        Product'=Product ∪ {p?}
|        Catalog'=Catalog ∪ {c?}
|        build' = build ∪ {inchargeof (|n?|) ⟶ p?}
|        identify' = identify ∪ {a? ⟶ p?}
|        about' = about ∪ {c? ⟶ p?}
-----------------------------------------------------------------
```

```
--------- NotNewProduct----------------
|        Ξ EPartDBState
|        a?: partnumber
|        p?: product
|        rep!: Report
|------------------------------------------
|        a? ∈ dom identify ∨
|        p? ∈ ran identify
|        rep! = 'Not a new product!'
----------------------------------------------
```

```
----------AuthorizedCAdministrator---------------
```

```
|       n?: person
|       c?: companyname
|       p?: password
|-------------------------------
|       n? ∈ dom inchargeof
|       p? ∈ ran ensure
|       c? ∈ ran inchargeof
|       c? = inchargeof(|n?|)
|       p? = ensure(|n?|)
----------------------------------------------------------


------------UnauthorizedCAdministrator------------
|       Ξ EPartDBState
|       n?: name
|       p?:password
|       c?:companyname
|       rep!:Report
-----------------------------------------------------
|       n? ∉ dom inchargeof ∨
|       p? ∉ ran ensure ∨
|       c? ∉ ran inchargeof
|       p? ≠  ensure(|n?|) ∨
|       c? ≠  inchargeof(|n?|)
|       rep! = 'Not Authorized as company administrator!'
----------------------------------------------------------------------
```

DoAddNewProduct $\triangleq$ AuthorizedCAdministrator $\wedge$ AddNewProduct $\wedge$ Success
$\vee$
UnauthorizedCAdministrator
$\vee$
NotNewProduct

Here is an operation that adds a new product to the database. p? $\notin$ ran *identify* specifies that this product information does not exist in this database, since the *identify* is the function that maps from existing part number in this database to product information. a? $\notin$ dom *identify* means that this new part number is not currently being used for identifying some other products. In this case it is considered as this 'new' product not a really new. Here n? $\in$ dom *inchargeof* guarantees that function *inchargeof*(|n?|) gives the company name. And function *build* is add a new mapping *inchargeof*(|n?|) $\longrightarrow$ p? guarantees that the company administrator only modify his or her own company's product information.

Product' = Product $\cup$ {p?} updates new Product set, so does Catalog set. The function *identify* adds a new mapping from part number to new product information. So does the function *about*.

When a new product is added, three scenarios can happen. First, if the requester is an authorized company administrator, and the product is new, then he/she can add the new product successfully. Second, if the requester is an authorized company administrator, because the product is not new (either product information not new or product part number being used), then insertion cannot be done. Third if the requester is unauthorized, then the insertion cannot be done.

3.2.2.2 Remove a product from database

```
----------RemoveProduct-----------------------
|        Δ EpartDBState
|        p?: product
|        c?: catalog
|        n?: person
|        a?: partnumber
|---------------------------------------------
|        a? ∈ dom identify
|        p? ∈ ran identify
|        n? ∈ dom inchargeof
|        build' = build \ { inchageof(|n?|) ——→p?}
|        identify' = identify \ { a? ——→p? }
|        about' = about \ { c? ———→ p? }
|        Product' = Product \ {p?}
|        Catalog' = Catalog \ {c?}
-----------------------------------------------------------
```

```
---------NotExistProduct------------------
|        Ξ EPartDBState
|        a?: partnumber
|        p?: product
|        rep!: Report
```

```
|----------------------------------------
|       a? ∉ dom identify ∨
|       p? ∉ ran identify
|       rep! = 'Not exist such product!'
---------------------------------------------------
```

DoRemoveProduct ≙ AuthorizedCAdministrator ∧ RemoveProduct ∧ Success
∨
UnauthorizedCAdministrator
∨
NotExistProduct

3.2.2.3 Update a product information

```
--------UpdateProduct--------------
|        Δ EpartDBState
|       p?: product
|       c?: catalog
|       n?: person
|       a?: partnumber
-----------------------------------------
|       a? ∈ dom identify
|       n? ∈ dom inchargeof
|       p? ∈ ran identify
|       build' =(build ⩥ identify(|a?|) ) ∪ { inchargeof(|n?|) ⟶ p? }
|       about' =(about ⩥ idendify(|a?|) ) ∪ { c? ⟶ p? }
|       identify' = (identify ⩥ identify(|a?|) ) ∪ { a? ⟶ p? }
|       Product' = Product
|       Catalog' = Catalog
---------------------------------------------------------------------------------
```

DoUpdateProduct ≙ AythorizedCAdministrator ∧ UpdateProduct ∧ Success
∨
UnauthorizedCAdministrator
∨
NotExistProduct

a? ∈ dom *identify* specifies that this part number does exist in the database. n? ∈ dom *inchargeof* guarantees that *inchargeof*(|n?|) will give the company name. p? ∈ ran *identify* means that product information is included in the database. (build ⩥ *identify*(|a?|)) ∪ { *inchageof*(|n?|) ⟶ p? } means *build* filters the mapping which is associated with product information *identify*(|a?|) , and adds a new mapping from company name *inchargeof*(|n?|) to the current product information. So *about* and *identify* functions do the same thing.

3.2.3 Modify and Update Company Information

3.2.3.1 Add New company

```
---------AddNewCompany-------------------------------
|        Δ EPartDBState
|        c?: Company
|        a?: CompanyName
|        b?: person
|        p?: password
|-----------------------------------------------------------
|        a? ∉ dom Company
|        inchargeof' = inchargeof ∪ {b? ⟶ a?}
|        ensure' = ensure ∪ {b? ⟶ p?}
|        aboutcompany' = aboutcompany ∪ {a? ⟶ c?}
-----------------------------------------------------------------------------

--------AuthorizedSAdministrator------------
|        n?: person
|        p?: password
|----------------------------------------
|        n? ∈ dom secure
|        p? = secure(|n?|)
--------------------------------------------------

---------UnauthorizedSAdministrator-----------
|        Ξ EPartDBState
|        n?: person
|        p?:password
|        rep!: Report
|------------------------------------
|        n?∉ dom secure ∨
|        p? ≠ secure(|n?|)
|        rep! = 'Not Authorized as company administrator!'
---------------------------------------------------------------------

------   NotNewCompany-----------------------------------
|        Ξ EPartDBState
|        a?: CompanyName
|       rep!: Report
|-----------------------------------------------------
|       a? ∈ dom aboutcompany
|       rep! = 'Not a New company!'
------------------------------------------------------
```

```
DoAddNewCompany ≙ AuthorizedSAdministrator ∧ AddNewCompany ∧ Success
                  ∨
                  UnauthorizedSAdministrator
                  ∨
                  NotNewCompany
```

3.2.3.2 Remove a company information

```
------  RemoveCompany-----------------------------
|       Δ EPartDBState
|       a?: CompanyName
|       c?: company
|-----------------------------------------------------
|       a? ∈ dom aboutcompany
|       c? = aboutcompany(|a?|)
|       aboutcompany' = aboutcompany ⩥ {a?}
|       ensure' = ensure ⩥ {a?}
|       inchargeof' = inchargeof ⩥ {a?}
|       identify'^-1 = identify^-1 ⩥ build(|a?|)
|       about'^-1 = about^-1 ⩥ build(|a?|)
|       Product' = Product' \ build(|a?|)
|       build'^-1 = build^-1 ⩥ build(|a?|)
|       Company' = Company \ {c?}
|       Catalog' = dom about
------------------------------------------------------
```

```
------  NotExsitCompany-----------------------------------
|       Ξ EPartDBState
|       a?: CompanyName
|       c?: company
|       rep!: Report
|-----------------------------------------------------
|       a? ∉ dom aboutcompany ∨
|       c? ≠ aboutcompany(|a?|)
|       rep! = 'The Company Not Exsit !'
------------------------------------------------------
```

```
DoRemoveCompany ≙ AuthorizedSAdministrator ∧ RemoveCompany ∧ Success
                  ∨
```

UnauthorizedSAdministrator
$\vee$
NotExistCompany

. a? $\in$ dom *aboutcompany* denotes that this company name exists in the database. c? = *aboutcompany*(|a?|) makes sure that the company name and company information truly are associated with each other in the database. *aboutcompany*' = *aboutcompany* $\ntriangleright$ {a?} filters this association from this function. When the company information is removed from the database, all the products that are related to this company should be removed as well. *identify*'$^{-1}$ = *identify*$^{-1}$ $\triangleright$ *build*(|a?|) filters information which is associated with this company, since the *identify*$^{-1}$ maps product information to part number. So does the other two functions of *about*$^{-1}$ and *build*$^{-1}$. Therefore, functions of *build*, *about*, and i*dentify* are updated.

When the company information is removed, the following three cases may occur. First, the requester might not be authorized; second, requester might be authorized but the company does not exist in the database; finally, if the requester is an authorized server administrator, and if the company exists in the database, then he/she can remove company information successfully.

3.2.3.3 Update a company information

```
------  UpdateCompany---------------------------
|       Δ EPartDBState
|       a?: companyname
|       c?: company
|       b?: Name
|       p?: password
|---------------------------------------------------------
|       a? ∈ Dom aboutcompany
|       aboutcompany' = aboutcompany ⊕ {a? ⟶ c?}
|       inchargeof' = inchargeof ⊕ {b? ⟶ a?}
|       ensure' = ensure ⊕ {b? ⟶ p?}
|       Company' = ran aboutcompany
|       Product' = Product
|       Catalog' = Catalog
------------------------------------------------------------------
```

```
DoUpdateCompany  ≙ AuthorizedSAdministrator ∧ UpdateCompany ∧ Success
                   ∨
                   UnauthorizedSAdministrator
                   ∨
                   NotExistCompany
```

CHAPTER IV

IMPLEMENTATION AND RESULTS

4.1 E-mail COM

As mentioned in the objectives section, when a user selects products, then clicks the "order" button, the order should be sent to manufacturers. To send the order to the manufacturers automatically, an E-mail COM should be created. This E-mail COM is created using the Active Template Library (ATL) and Winsock.

4.1.1 Background

4.1.1.1 Active Template Library

ATL is a framework that easily creates small, lightweight COM objects. It gives software developers the flexibility to implement their components without any dependencies on secondary DLLs, including the standard C run-time DLL. It makes components as small and fast as possible. The ATL AppWizard provides the basic housing support that the COM component needs. All COM objects must support the IUnknown interface and expose its specific functionality. It must provide a class factory, which facilitates the creation of COM objects, and it should support self-registration, which is the ability to add the COM registry entries for each of its components automatically. Since the ATL includes its functionality as part of the implementation, there is no need to link to any external DLLs. It encapsulates a component's support housing for in-process component in its CcomModule class, taking care of self-registration by exporting two standard COM functions: DllRegisterServer and

DllUnregisterServer. ATL supports IUnknown, so the developer need not write any code for QueryInterface, AddRef, Release methods. ATL supports for class Factories, so the developer need not write any code for CreateInstance, LockServer these methods. ATL provides build-in support for each of these requirements. Because of this, COM developers can concentrate on the unique functions that they want the COM to provide.

4.1.1.2 Socket

TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that allows a byte stream to be delivered without error on any other machines in the internet. UDP (User Datagram Protocol), is an unreliable, connectionless protocol. The protocols and networks is shown in Figure 8 (Tanenbaum, 1996).

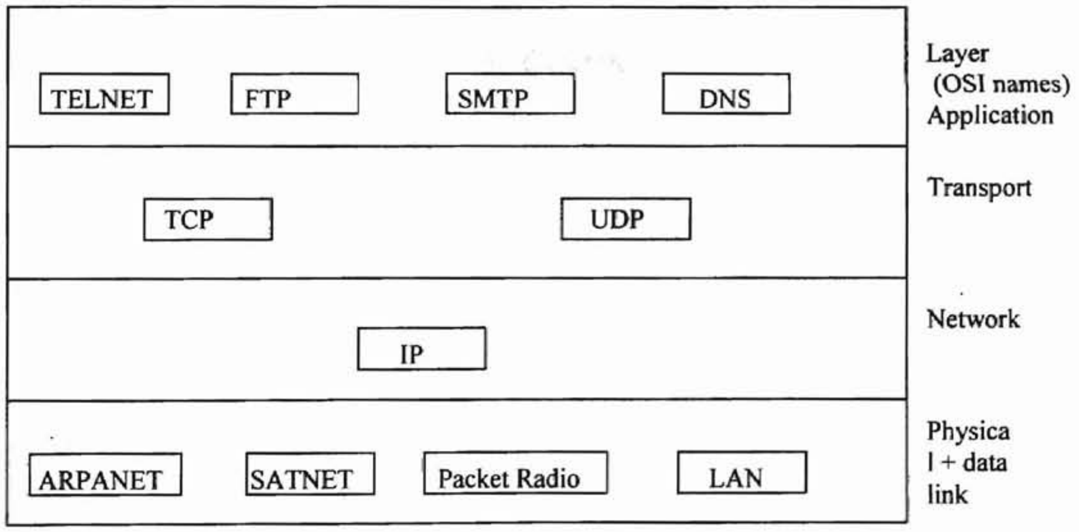


Figure 8. Protocols and Networks in the TCP/IP Model Initially (Tanenbaum, 1996).

The primitives and meanings are list as follow (Table 1):

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections, give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Table 1 The Socket Primitives for TCP (Tanenbaum,1996)

There are some implementation differences between Winsock and the UNIX version of Berkeley sockets. One of the differences is that socket descriptors and file descriptors cannot be interchanged in. Another is that when developers use Winsock functions, they must first call the WSAStartup function; and they should call WSACleanup for proper termination.

Some structures used in this COM are as follows:

```
struct sockaddr_in
{
        short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];
.
}
struct hostent
{
        char FAR * h_name;
        char FAR * FAR * h_aliases;
        short h_addrtype;
        short h_length;
        char FAR * FAR * h_addr_list;
};
```

The overview of setting up and using TCP connection:

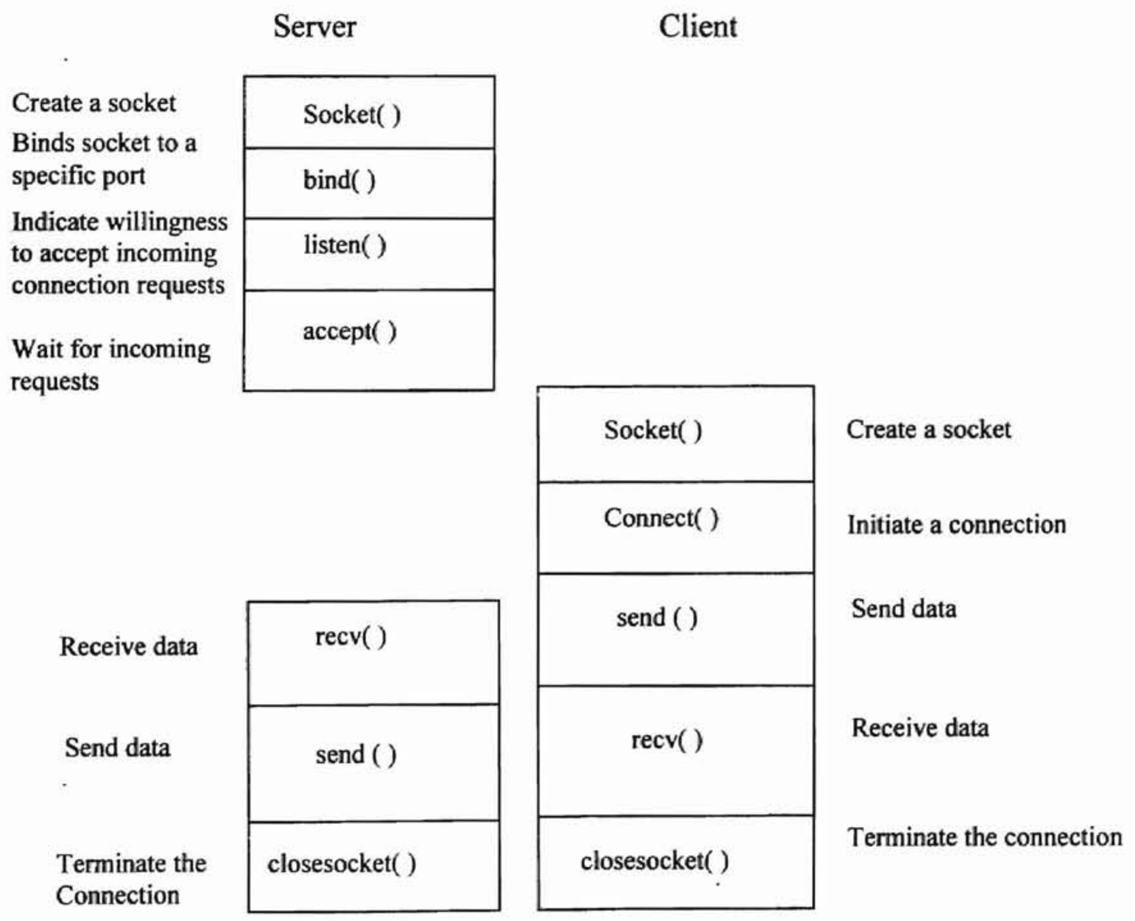


Figure 9. Socket Communications for Connection-oriented Protocols (Toth, 1997)

From the figure 9 we know that the E-mail COM needs to do is create a socket and connect to mail server, then send data and receive data, after that terminate the connection. Following section is the detail of the implementation of the E-mail COM.

4.1.2 Implementation

following code are developed to implement E-mail COM. The file atlmail.h declare the object and its unique function sendmail. The file atlmail.cpp implement the sendmail mathod.

```
 // atlmail.h : Declaration of the Catlmail
 #ifndef __ATLMAIL_H_
 #define __ATLMAIL_H_

 #include "resource.h"       // main symbols
 #include <winsock2.h>

/////////////////////////////////////////////////////////////////////////////
// Catlmail
     class email
     {
     public:
     int init;
     int mysocket;
     struct sockaddr_in a;
     struct hostent *h;
     WSADATA wsadata;
     email();
     ~email();
     int Logon(BSTR server);
     int sendmsg(BSTR sender, BSTR address, BSTR message);
     };
```

Figure 10. Code for atlmail.h (to be continued)

```
class ATL_NO_VTABLE Catlmail :
   public CComObjectRootEx<CComSingleThreadModel>,
   public CComCoClass<Catlmail, &CLSID_atlmail>,
   public IDispatchImpl<Iatlmail, &IID_Iatlmail,&LIBID_WATL1SOCKLib>
{
 public:

      Catlmail()
      {
      }


DECLARE_REGISTRY_RESOURCEID(IDR_ATLMAIL)

BEGIN_COM_MAP(Catlmail)
     COM_INTERFACE_ENTRY(Iatlmail)
     COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()



// Iatlmail
public:
     STDMETHOD(sendmail)((/*[in]*/BSTR server,/*[in]*/BSTR
     sender,/*[in]*/BSTR address,/*[in]*/BSTR message);
};

#endif //__ATLMAIL_H_
```

Figure 10. Code for atlmail.h

In the OBJBASE.H, three COM C++ macros are defined as shown below:

```
#define STDMETHODCALLTYPE _stdcall

#define STDMETHOD(method) virtual HRESULT STDMETHODCALLTYPE
method

#define STDMETHODIMP HRESULT STDMETHODCALLTYPE
```

The STDMETHOD macro is used in the declaration of the interface methods within the implementing class. The STDMETHODIMP macros are used when the developers actually implement the interface function.

The 'in' and 'out' keywords specify the direction of the parameter. By providing this information, the developers provide COM with information that will help to make the parameter marshaling process more efficient. The 'retval' keyword specifies that the parameter should be treated as the return value for the method.

COM uses a special string data type called a binary string or basic string or BSTR. It is declared as OLECHAR *, which indicates that it is a Unicode string. The structure of BSTR is as follow:

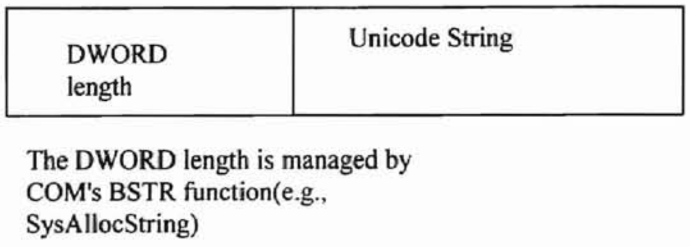

DWORD length
Unicode String
The DWORD length is managed by COM's BSTR function(e.g., SysAllocString)

Figure 11. Structure for COM's Binary String (BSTR)(Armstrong,1998)

BSTRs are represented as OLECHAR pointers. To convert a BSTR to ANSI, developers can use:

USES_CONVERSION;

OLE2T(someBSTRdata);

This conversion is being used in the atlmail.cpp(figure 7.b).

```
// atlmail.cpp : Implementation of Catlmail
#include "stdafx.h"
#include "watl1sock.h"
#include "atlmail.h"

////////////////////////////////////////////////////////////////////////
// Catlmail
email::email()
{
        init=0;
        mysocket=0;
}
email::~email()
{
        if (mysocket!=0)//has socket
                closesocket(mysocket);//shut down the connection
        if (init!=0)
          WSACleanup(); //clean up
        init=mysocket=0;//reset
}

int email::Logon(BSTR server)
{
        char *mserver=new char[50];
        USES_CONVERSION;
        strcpy(mserver,OLE2T(server));
        //change BSTR to ANSI and     copy to mserver
        if (WSAStartup(0x101, &wsadata)) //can not initiated
        {
                delete mserver;
                return 0;
        }
        mysocket=socket(AF_INET, SOCK_STREAM,
                        IPPROTO_TCP);//create a socket
        init=1; //initialization success
```

Figure 12. Code for atlmail.cpp (to be continued)

```
  if (mysocket==0)//can not create a socket

  {
          delete mserver;
          return 0;
  }
  h=gethostbyname(mserver);//resolve hostname
  if (h==NULL) //can not resolve hostname
  {
          delete mserver;
          return 0;
  }
  delete mserver;
  return 1;
}

int email::sendmsg(BSTR sender, BSTR address, BSTR message)
{
  char *t=new char [2000];
  char *temp=new char [2000];
  char *msender=new char[50];
  char *maddress=new char[100];
  char *mmessage=new char[1500];
  USES_CONVERSION;
  strcpy(msender,OLE2T(sender));
  strcpy(maddress,OLE2T(address));
  strcpy(mmessage,OLE2T(message));
  a.sin_family = AF_INET;
  a.sin_port=htons(25); //SMTP port number is 25
  memcpy(&(a.sin_addr.s_addr), h->h_addr, sizeof(int));
  if (connect(mysocket, (struct sockaddr *)&a, sizeof(a)))
  {
          return 0;
  }
  int i=recv(mysoceket, temp, 2000,0);
  strcpy(t,"MAIL FROM:");
  msender[strlen(msender)]='\n';
  strcat(t,msender);
  send(mysocket, t,strlen(t),0);//send data :"MAIL FROM:" + msender +'\n'
  i=recv(mysocket,temp,2000,0);
  strcpy(t,"RCPT TO:");
  strcat(t,maddress);
  t[strlen(t)]='\n';
  send(mysocket,t, strlen(t),0);//send data:"RCPT TO:"+maddress+'\n'
  i=recv(mysocket,temp,2000,0);
  strcpy(t,"DATA\n");
  send(mysocket,t,strlen(t),0);//send data: "DATA\n"
```

Figure 12. Code for atlmail.cpp (to be continued)

```
        i=recv(mysocket, temp,2000,0);
        strcpy(t,mmessage);
        strcat(t,"\n\n\n.\n");
        send(mysocket,t,strlen(t),0);//send data: mmessage+"\n\n\n.\n"
        i=recv(mysocket,temp,2000,0);
        strcpy(t,"QUIT\n");//send data: "QUIT\n"
        send(mysocket,t,strlen(t),0);
        delete t;
        delete temp;
        delete msender;
        delete maddress;
        delete mmessage;
        return 1;
}

STDMETHODIMP Catlmail::sendmail(BSTR server, BSTR sender,
                                          BSTR address, BSTR message)
{
        email Email;
        int i=Email.Logon(server);//logon the mail server
        if (i==1)//if success
           i=Email.sendmsg(sender,address,message);//send message
        return S_OK;
}
```

Figure 12. Code for atlmail.cpp

Sendmsg method formats an SMTP-compliant message and sends it. The basic SMTP session looks like this:

MAIL FROM: <sender>

RCPT TO: <recipient address>

DATA

<message>

.

QUIT

Calling E-mail COM is very simple as shown below (Figure 13):

```
if (isobject(session("e_mail")))then
        set e_mail=session("e_mail")
else
        set e_mail=Server.CreateObject("atlmail.atlmail.1")
        set session("e_mail")=e_mail
end if
emailserver="a.cs.okstate.edu"
e_mail.sendmail emailserver, sender, recipientaddress ,message
```

Figure 13. An ASP Program Calls E-mail COM

The E-mail COM uses ATL. To add a new ATL object a Simple Object is chosen instead of an Active Server Component; therefore, this component can be used not only in ASP but also in other application programs such as Visual Basic. And it is portable with any windows operating system.

4.2 Navigation Record Set

In order for users to navigate the database backward and forward through the web interface, the following code is developed:

```
function recordform(str,b)
dim s,rs,getnext,getprevious

if (isobject(session("rs")) and b )then
          set rs=session("rs")
          total=request.form("total")
 else
          total=0
          currec=0
          SQL="select catalog,manupartnumber,price,imagefile,pfeature, "
         SQL=SQL & "keynumber from parttable where companyname='"
         SQL=SQL & cname & "'"
         set rs=conn.execute(SQL)
```

Figure 14. Code for Navigating the Record Set (to be continued)

```
          do while not rs.eof
           total=total+1
           rs.MoveNext
loop
rs.close
rs.open , ,2
set session("rs")=rs
end if

select case str
 case "Get First"
 rs.MoveFirst
  currec=0
 case "Get Next"
 currec=currec+1
  rs.MoveNext

 case "Get Previous"
   currec=currec-1
    rs.MovePrevious
case "Get Last"
   currec=total-1
   rs.MoveLast
case "Get Current"
end select

 if (cint(total)-1>currec) then
          getnext=true
else
           getnext=false
end if
if (currec>0) then
           getprevious=true
else
           getprevious=false
end if

s="<table align=right cellspacing=0><tr><td><input type=image border=0
          src=first.gif name=First></td><td>"
 if getprevious then
            s=s & "<input type=image border=0 src=prev.gif
                          name=Prev></td><td>"
 else
            s=s & "<image src=prev1.gif></td><td>"
 end if
  s=s & "<input type=text size=3  name=currec value='" & currec &"'></td>
        <td>"
if getnext then
  s=s & "<input type=image border=0 src=next.gif name=Next></td><td>"
```

Figure 14. Code for Navigating of Record Set (to be continued)

```
  else
    s=s & "<image src=next1.gif></td><td>"
   end if
   s=s & "<input type=image border=0 src=last.gif
  name=Last></td></tr></table><br><br><br>"
  s=s&" <table align=center><tr><td width=100><center><b><font
  color=#0000ff>Catalog<td width=200><center><b><font color=#0000ff
          >Manufacture Part Number</td>"
   s=s&"<td width=100><center><b><font color=#0000ff>Price</td></tr
          >"
   s=s&"<tr><td><input type=text name=Clog size=20
          value='"&rs(0)&"'></td><td>"
   s=s&"<input type=text name=mpartnumber size=30
          value='"&rs(1)&"'></td><td>"
    s=s&"<input type=text name=price size=20 "
  if (csng(rs(2))=int(csng(rs(2)))) then
      s=s & "value=$" &rs(2)&".00></td></tr>"
  else
      s=s & "value=$" &rs(2) & "></td></tr>"
   end if
   s=s&"<tr><td width=100><center><b><font color=#0000ff>Image File
          Name"
    s=s &"<td width=200><center><b><font color=#0000ff>Product
          Features</td>"
     s=s&"<td width=100><center><b><font color=#0000ff>Key
          Number</td></tr>"
    s=s&"<td><input type=text name=ifile size=20 value='"&rs(3)&"'></td>
          <td>"
    s=s & "<input type=text name=pfeature size=30 value='"
    s=s & rs(4) & "'></td><td>"
    s=s & "<input type=text name=knumber size=20
          value='"&rs(5)&"'></td></tr></table>"
   s=s & "<input type=hidden name=cname value='" & cname & "'>"
  s=s & "<input type=hidden name=rec0 value='" & rs(0) & "'>"
  s=s & "<input type=hidden name=rec1 value='" & rs(1) & "'>"
  s=s & "<input type=hidden name=rec2 value='" & rs(2) & "'>"
   s=s & "<input type=hidden name=rec3 value='" & rs(3) & "'>"
  s=s & "<input type=hidden name=rec4 value='" & rs(4) & "'>"
  s=s & "<input type=hidden name=rec5 value='" & rs(5) & "'>"
  s=s & "<input type=hidden name=total value='" & total & "'>"
  recordform=s
end function
```

Figure 14. Code for Navigating of Record Set

There are four different cursor types when opening a Record set object: Dynamic cursor, Keyset cursor, static cursor and Forward-only cursor (By default ADO opens a forward-only cursor). Followings are CursorTypeEnum values:

Constant	Value	Description
adOpenForwardOnly	0	Forward-only cursor Identical to a static cursor except that it only allows scrolling forward through records.
adOpenKeyset	1	Keyset cursor. Like a dynamic cursor, except that you can't see records that other user add, although records that other users delete are inaccessible from your recordset. Data changes by other users are still visible.
adOpenDynamic	2	Dynamic cursor. Addition, changes, and deletions by other users are visible, and all types of movement through the recordsst are allowed, except for bookmarks if the provider doesn't support them.
adOpenStatic	3	Static cursor, Astatic copy of a set of records that you can use to find data or generate reports. Additions, changes, or deletions made by other users are not visible.

Table 2 Cursor Types, Meanings, and Values

The cursor cannot be changed once a recordset is opened. Instead, the recordset must be closed and reopened using a new cursor type. After this is done, any new operations supported by the cursor are immediately available.

4.3 Customized Message Box

VBScript has a function called msgbox, and it supports many styles. Unfortunately, Netscape does not support the VBScript language. JavaScript has only one function called alert that can be used as message box although Netscape and Internet Explorer support JavaScript. Therefore, a unique error message and confirmation box needs to be developed using JavaScript. JavaScript has a window object that has the 'open' method. The following code is developed for trapping this method and sending error message to users:

```
function errorbox(str)
%>
<script>
str1="ERROR"
str2="<%=str%>"
aPopUp=window.open('','messagebox','toobar=yes,location=no,directories
=no,status=no,scrollbars=yes,resizable=no,copyhistory=no,width=250,
height=150,ScreenX=200,ScreenY=200')
 ndoc=aPopUp.document
  astr='<html><head><br><title>' + str1 + '</title>'
 astr +='</head><form>' +'<body'+' background="backgrnd1.gif">'
  astr +='<table><tr><td><image border=0 src="stop.gif"><td>' +str2+
'<tr></table><br>'
  astr +='<center><input type=button name=closebtn value="OK"
onclick="closebox()">'
  astr +='<script>'
  astr +='function closebox()' +'{' + 'self.close()'+ ' }'+'</'
+'script>'+'</body></form></html>'
  ndoc.write(astr)
  ndoc.close()
 self.messagebox=aPopUp
</script><% end function%>
```

Figure 15. Code for Customized Error Message Box

It is necessary for the confirmed box not only to send a confirmed message to users but also to get a feedback from the users. Hence the author uses a hidden field that allows the user to send information back to the server. The following code serves this purpose (here the hidden field name is called "cbutton")

```
s="<script language=javascript>"
  s=s & "function confirmbox() {"
  s=s &
"window.open('confirmbox.htm','confirm','height=150,width=200,ScreenX=
200,ScreenY=200');"
  s=s & "} </script>"
  s=s & "<br>"
s= s & "<table align=center><tr><td><input type=submit name=mode1
value=ADD></td><td>"
  s=s&"<input type=submit name=mode1 value=UPDATE></td><td>"
  s=s&"<input type=button name=mode1 value=REMOVE
OnClick='confirmbox()'></td><td>"
  s=s&"<input type=submit name=mode1
value=LOGOFF></td></tr></table>"
```

Figure 16. Code for Customized Confirmation Box

In the confirmbox.htm:

```
<html><head><title>CONFIRM</title>
<script language=javascript>
function yesbox () {
opener.document.forms[0].cbutton.value="yes";
opener.document.forms[0].submit();
self.close();
}
function nobox () {
opener.document.forms[0].cbutton.value="no";
opener.document.forms[0].submit();
self.close();
}
</script></head>
<body background="backgrnd1.gif"><form>
<table><tr><td><image border=0 src="question.gif"><td>
Are you sure you want to remove current record?<tr></table><br>
<center><input type=button  value="yes" onclick="yesbox()">
<input type=button  value="No" onclick="nobox()">
</form></body></html>
```

Figure 17. Code in Confirmbox.html

4 . 4 Some Outputs of the Database Management System

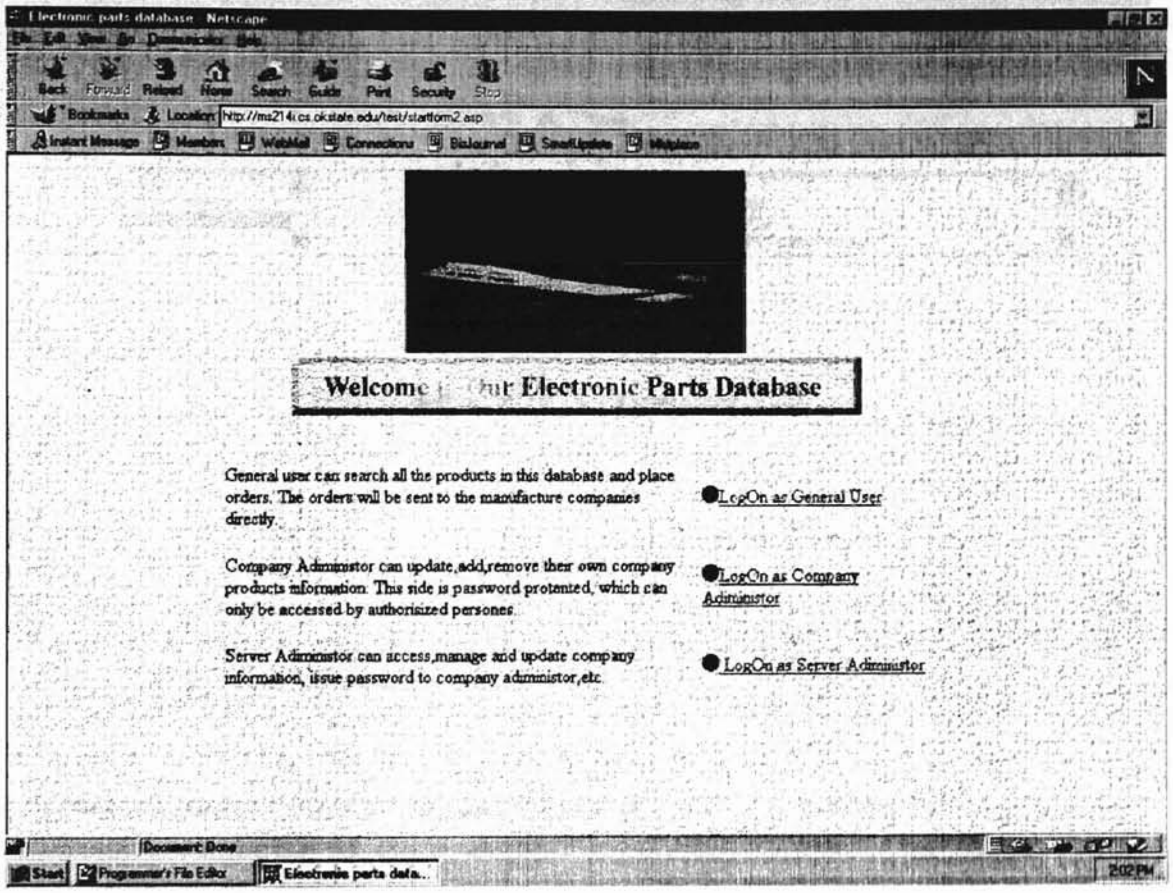
As stated in the requirements specification, there are three groups of users. The first group of users is ordinary users, who can search all the products in this database, select products, and place an order for the products. The second group of users is company administrators, who can maintain their own company's product information remotely. The third group of users is server administrators, who will maintain company information and company administrator information.(see Figure 18). The ordinary users can search for products by adding some specifications, such as, company name, catalog, part number. Figure 19 is the result when a user selects company King/Allied Signal as the search specification. A user can view the product by clicking "view" (see Figure 21). The user can continue to search for or order products, but must select products and add them to his/her shopping cart before clicking the 'order' button. Otherwise, an alert message pops up. The user can also jump to a company's homepage to view more information about the company by clicking the company name (see Figure 22). A user can also search for products by catalog (see Figure 23). All company names are hyperlinked to their home pages. After selecting products, the user can click the 'order' button.(see Figure 25), then, an order form pops up that shows shopping cart items, with default quantity 1. The user can change the quantity to 0 or more than 1. On the order form, name, street, city, state and zip code fields must be completed; otherwise, the user will be given an alert message, and the order is not sent to the server. After the order information sent back to server, zip code and phone number (if user filled in) are checked. If either of them is not valid, the order message is not sent to the manufacturer, and an error message appears. If all the information is valid the server will check whether company's email information is in the database. If it is, the order message is sent to the manufacturer; otherwise, it gives a message to show which orders were sent successfully and which orders failed due to lack of email information. If a user orders three products then the system sends three order messages instead of one, since the user may order from three different companies.(see Figures 28, 29, 30, 31).

Company administrators can update company product information remotely by passing a security check (see Figure 32). After a company administrator logs on, he/she can navigate the company product information. Figure 24 is the first record of the product information. The "Previous" button is dark, which means it is disabled. After clicking the "Next" button, the "Previous" button is enabled (see Figure 34). Company administrator can move forward by clicking "Next" button, or move backward by clicking the "Previous" button. Administrator can also jump to the last record by clicking the "Last" button(see Figure 35) or back to the first record by clicking the "First" button.

Company administrators can insert, delete or update records (see Figures 36,37, 38,39). After clicking the "Add" button, company administrators can fill in the information about "catalog", "manufacture part number", "price", "part number"(which are required) and "image file name", "product feature" (which are optional). If the required fields are not completed, the information is not be sent to the server, and an alert message pops up. When the information is at the server, it checks whether "price" is numeric and "part number" is unique. If any of the checks fail, then insertion aborts and the user is given an error message. If the insertion is successful, the last record is shown and the number of the record is increased by one.

A company administrator can delete records as well. When the administrator clicks the "Remove" button, a confirmation message box will pop up. The action of the deletion depends on the information sent back by the confirmation box (see Figure 40). If the administrator clicks "Yes" in the confirmation box, the server checks the data in this record. If any data changes, the deletion aborts and an error message shows up. If it is a successful deletion, the last record is shown, and the record number is decreased by one.

A server administrator can log in and do insertion, deletion and modification to the company information table and company administrator table (see Figure 42). The server administrator can navigate the records too, just like the company administrator does (see Figure 43).


Electronic parts database - Netscape
http://ms214i.cs.okstate.edu/test/startform2.asp
Document: Done

Welcome to Our Electronic Parts Database

General user can search all the products in this database and place orders. The orders will be sent to the manufacture companies directly.

Company Administor can update,add,remove their own company products information. This side is password protented, which can only be accessed by authorisized persones.

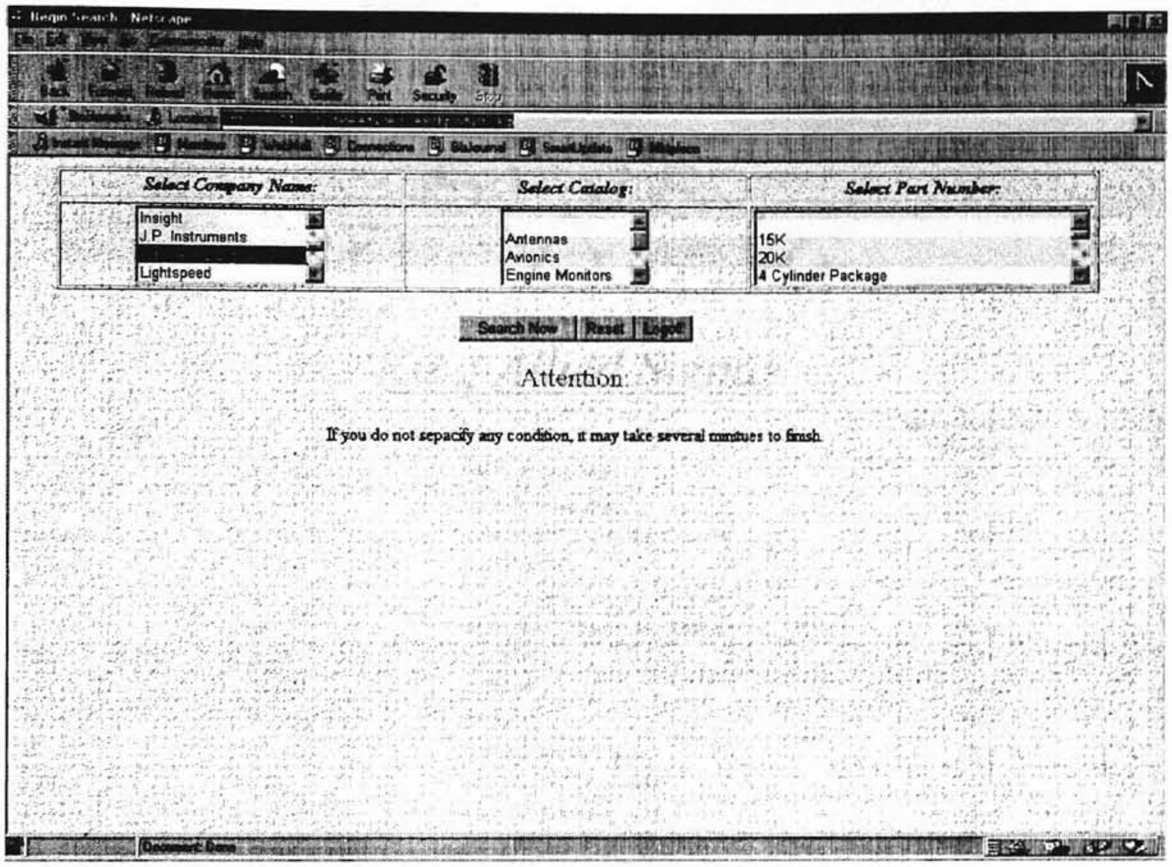
Server Administor can access,manage and update company information, issue password to company administor,etc.

● LogOn as General User

● LogOn as Company Administor

● LogOn as Server Administor

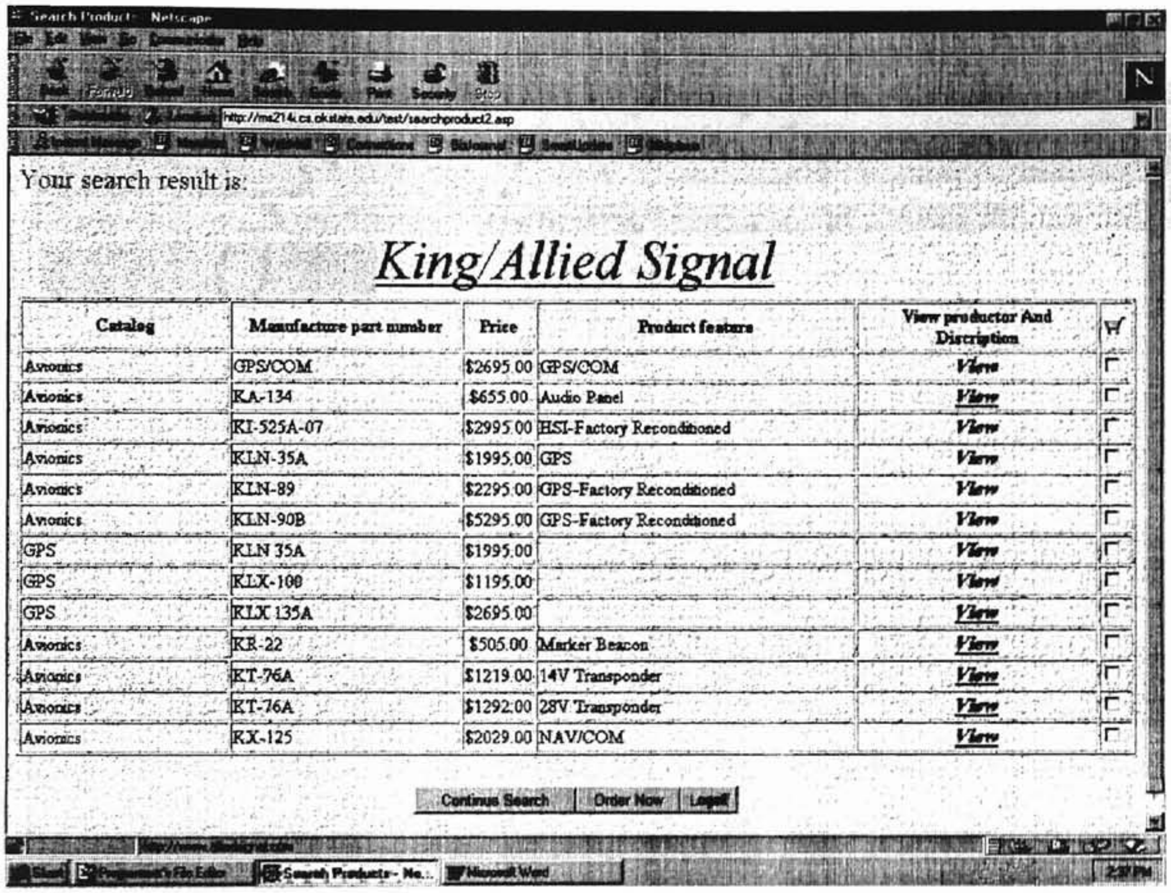
Figure 18. Three Types of Users



Attention:

If you do not sepacify any condition, it may take several mintues to finish.

Figure 19. The Example of User Search Page


Search Products - Netscape
http://ms214i.cs.okstate.edu/test/searchproduct2.asp

Your search result is:

King/Allied Signal

Catalog	Manufacture part number	Price	Product feature	View productor And Discription	🛒
Avionics	GPS/COM	$2695.00	GPS/COM	View	☐
Avionics	KA-134	$655.00	Audio Panel	View	☐
Avionics	KI-525A-07	$2995.00	HSI-Factory Reconditioned	View	☐
Avionics	KLN-35A	$1995.00	GPS	View	☐
Avionics	KLN-89	$2295.00	GPS-Factory Reconditioned	View	☐
Avionics	KLN-90B	$5295.00	GPS-Factory Reconditioned	View	☐
GPS	KLN 35A	$1995.00		View	☐
GPS	KLX-100	$1195.00		View	☐
GPS	KLX 135A	$2695.00		View	☐
Avionics	KR-22	$505.00	Marker Beacon	View	☐
Avionics	KT-76A	$1219.00	14V Transponder	View	☐
Avionics	KT-76A	$1292.00	28V Transponder	View	☐
Avionics	RX-125	$2029.00	NAV/COM	View	☐

Figure 20. The Output of Searching Products Made by King/Allied Signal Company

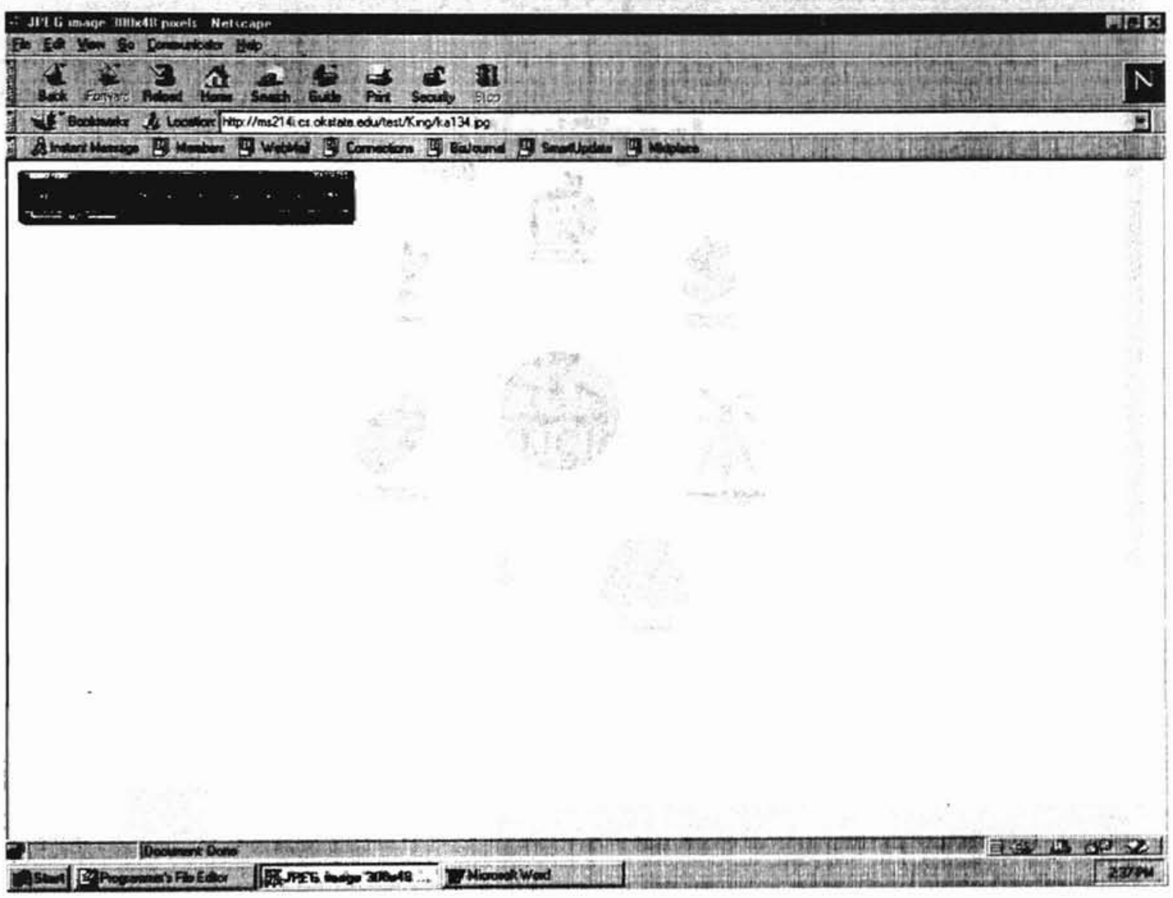

JPEG image 300x48 pixels - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://ms214i.cs.okstate.edu/test/King/ka134.jpg
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Document: Done
Start Programmer's File Editor JPEG Image 300x48 ... Microsoft Word
2:37 PM

Figure 21. View Product

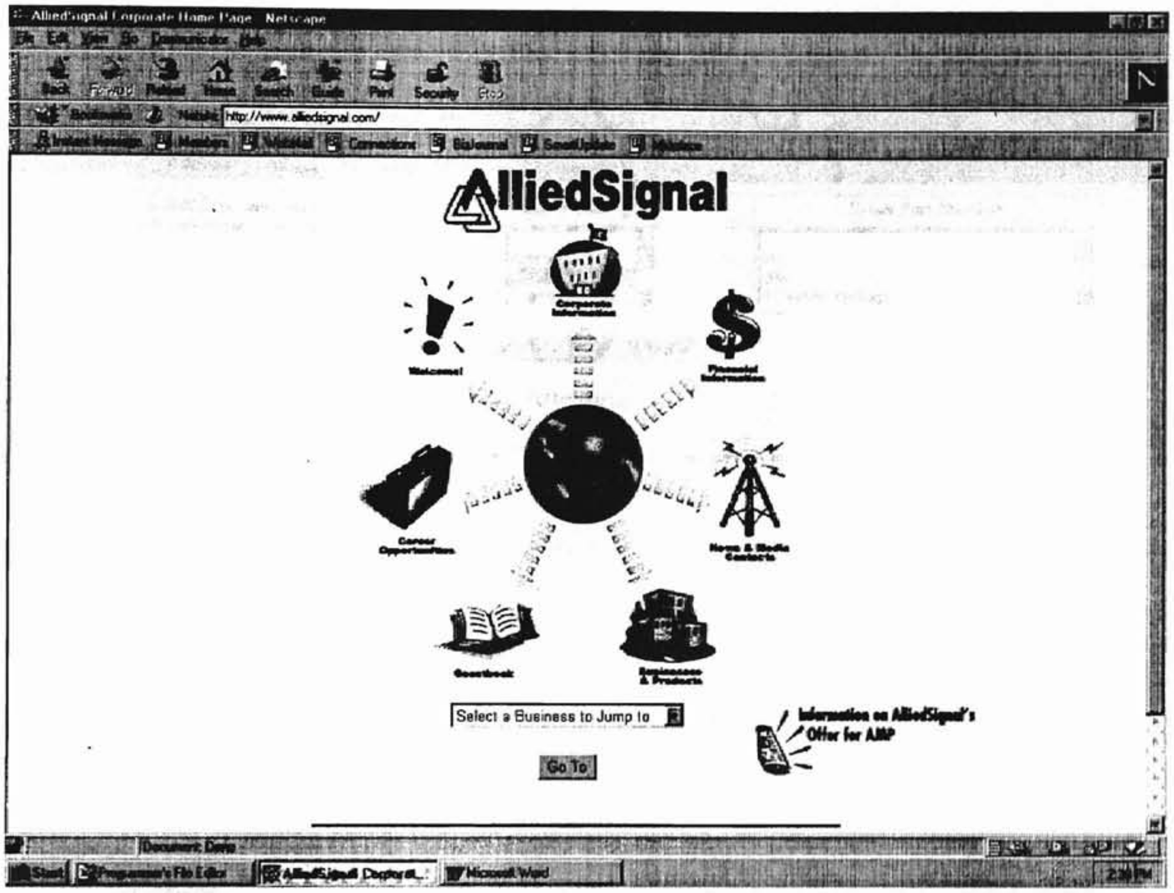
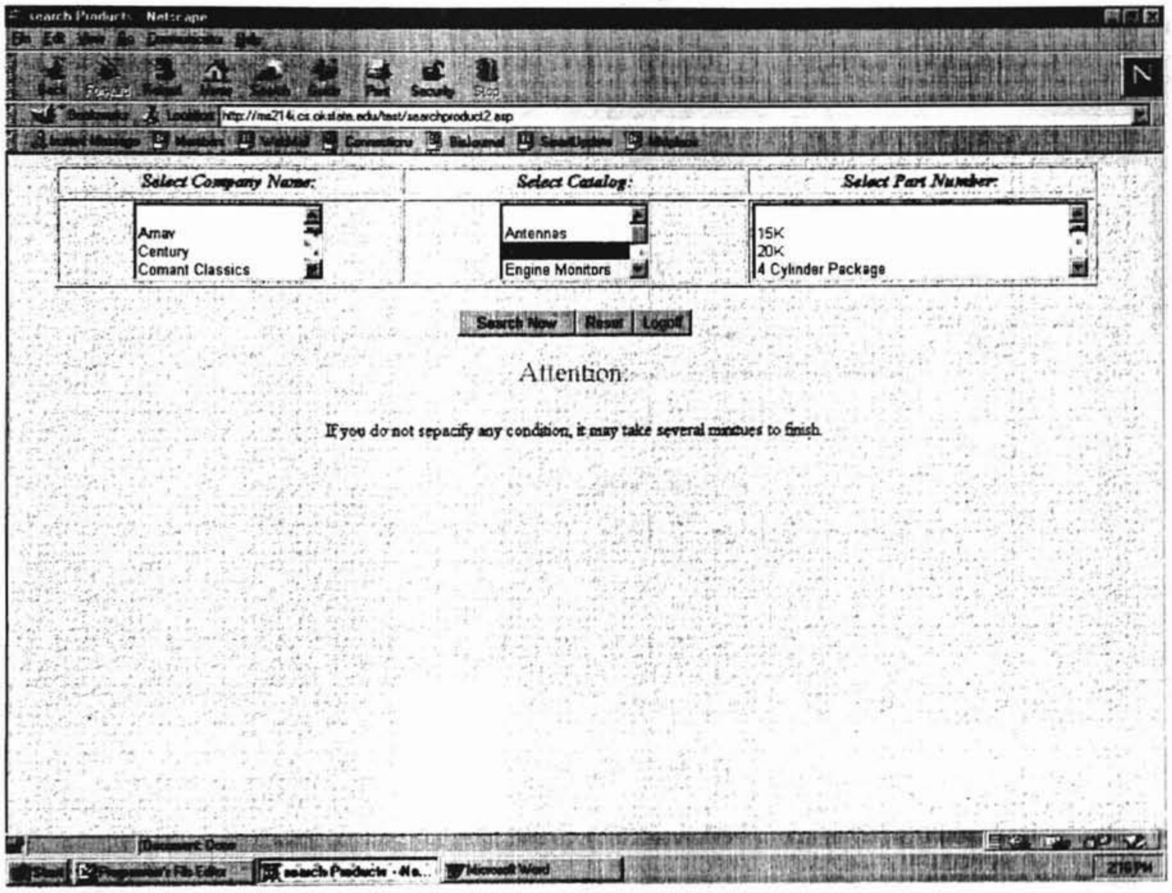


Figure 22. Company's Home Page


Select Company Name:

Select Catalog:

Select Part Number:

Arnav
Century
Comant Classics

Antennas

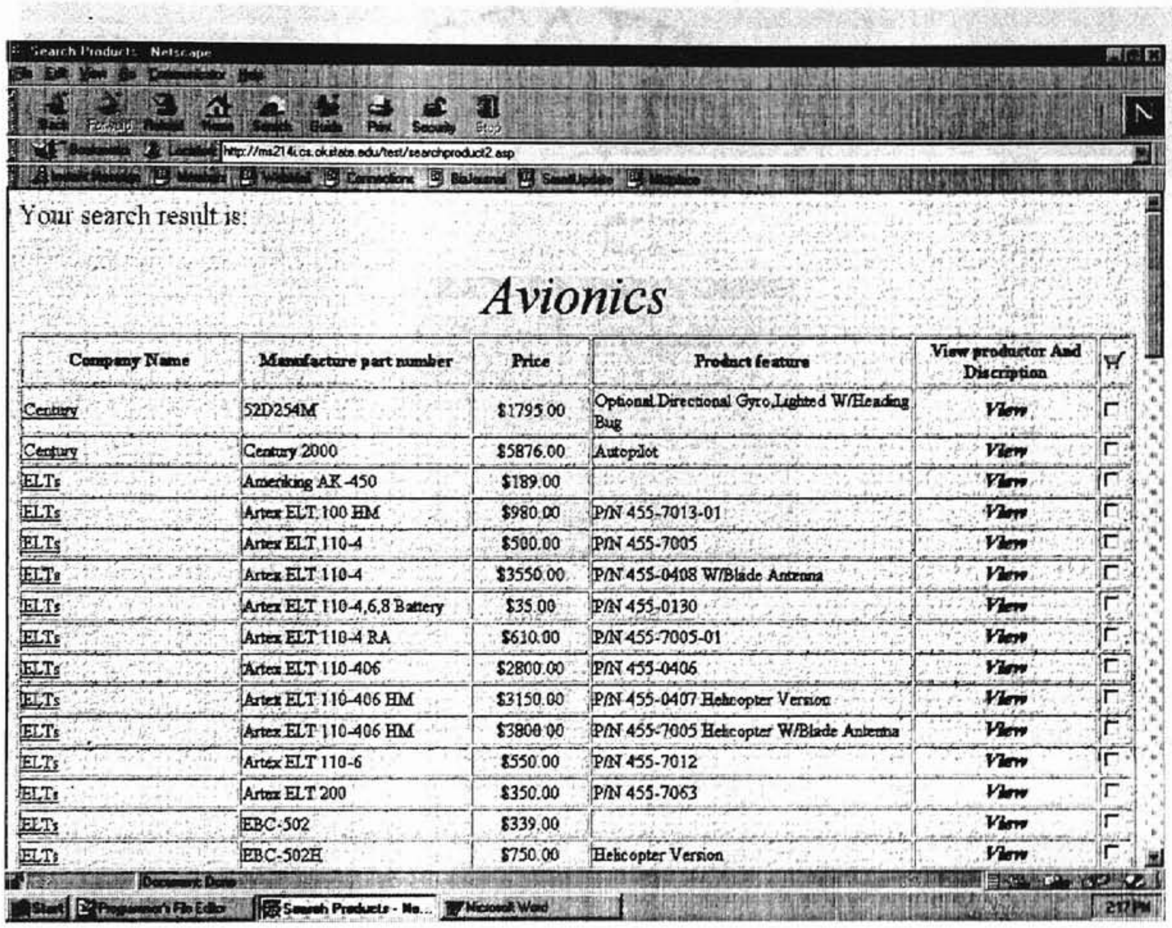
Engine Monitors

15K
20K
4 Cylinder Package

Attention:

If you do not sepacify any condition, it may take several minutes to finish.

Figure 23. Searching Product by Avionics Catalog



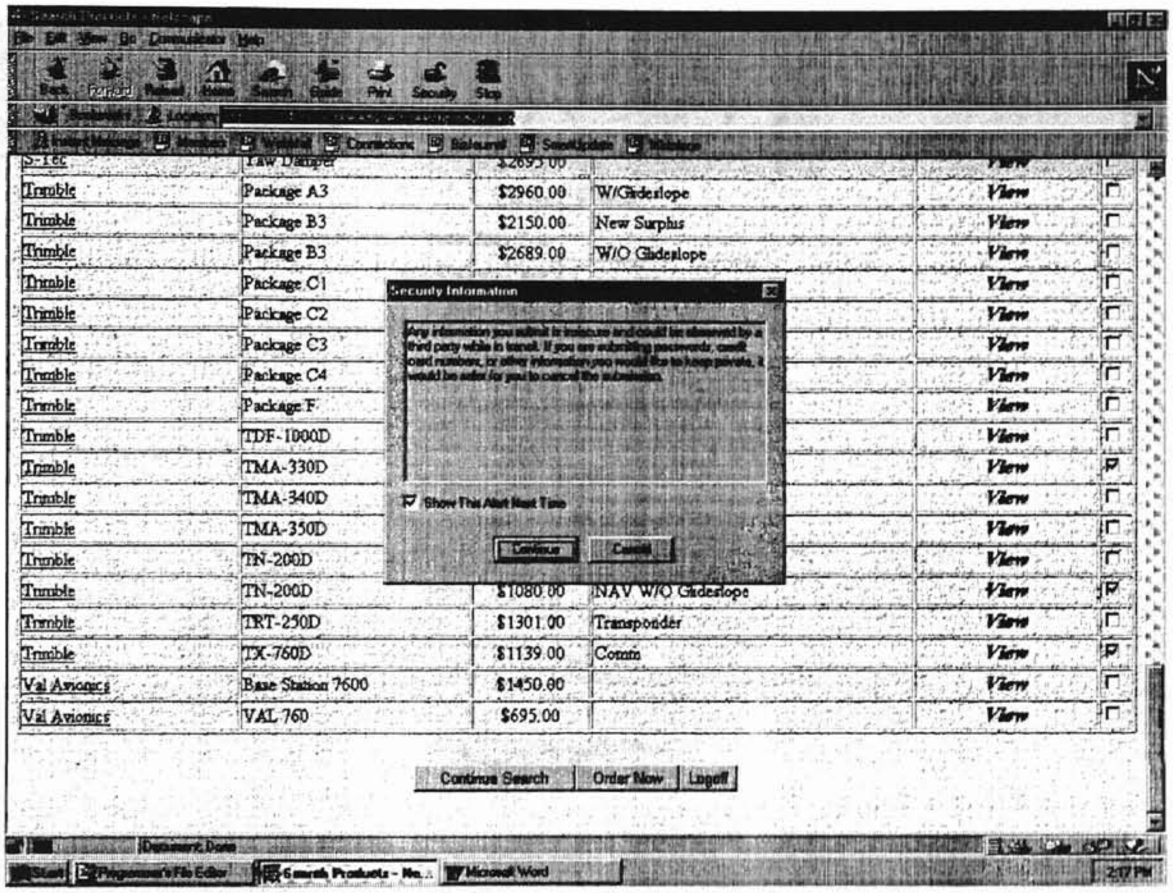
http://ms214i.cs.okstate.edu/test/searchproduct2.asp

Your search result is:

Avionics

Company Name	Manufacture part number	Price	Product feature	View productor And Discription	
Century	52D254M	$1795.00	Optional Directional Gyro,Lighted W/Heading Bug	View	☐
Century	Century 2000	$5876.00	Autopilot	View	☐
ELTs	Ameriking AK-450	$189.00		View	☐
ELTs	Artex ELT 100 HM	$980.00	P/N 455-7013-01	View	☐
ELTs	Artex ELT 110-4	$500.00	P/N 455-7005	View	☐
ELTs	Artex ELT 110-4	$3550.00	P/N 455-0408 W/Blade Antenna	View	☐
ELTs	Artex ELT 110-4,6,8 Battery	$35.00	P/N 455-0130	View	☐
ELTs	Artex ELT 110-4 RA	$610.00	P/N 455-7005-01	View	☐
ELTs	Artex ELT 110-406	$2800.00	P/N 455-0406	View	☐
ELTs	Artex ELT 110-406 HM	$3150.00	P/N 455-0407 Helicopter Version	View	☐
ELTs	Artex ELT 110-406 HM	$3800.00	P/N 455-7005 Helicopter W/Blade Antenna	View	☐
ELTs	Artex ELT 110-6	$550.00	P/N 455-7012	View	☐
ELTs	Artex ELT 200	$350.00	P/N 455-7063	View	☐
ELTs	EBC-502	$339.00		View	☐
ELTs	EBC-502H	$750.00	Helicopter Version	View	☐

Figure 24. Search Result for Avionics Products



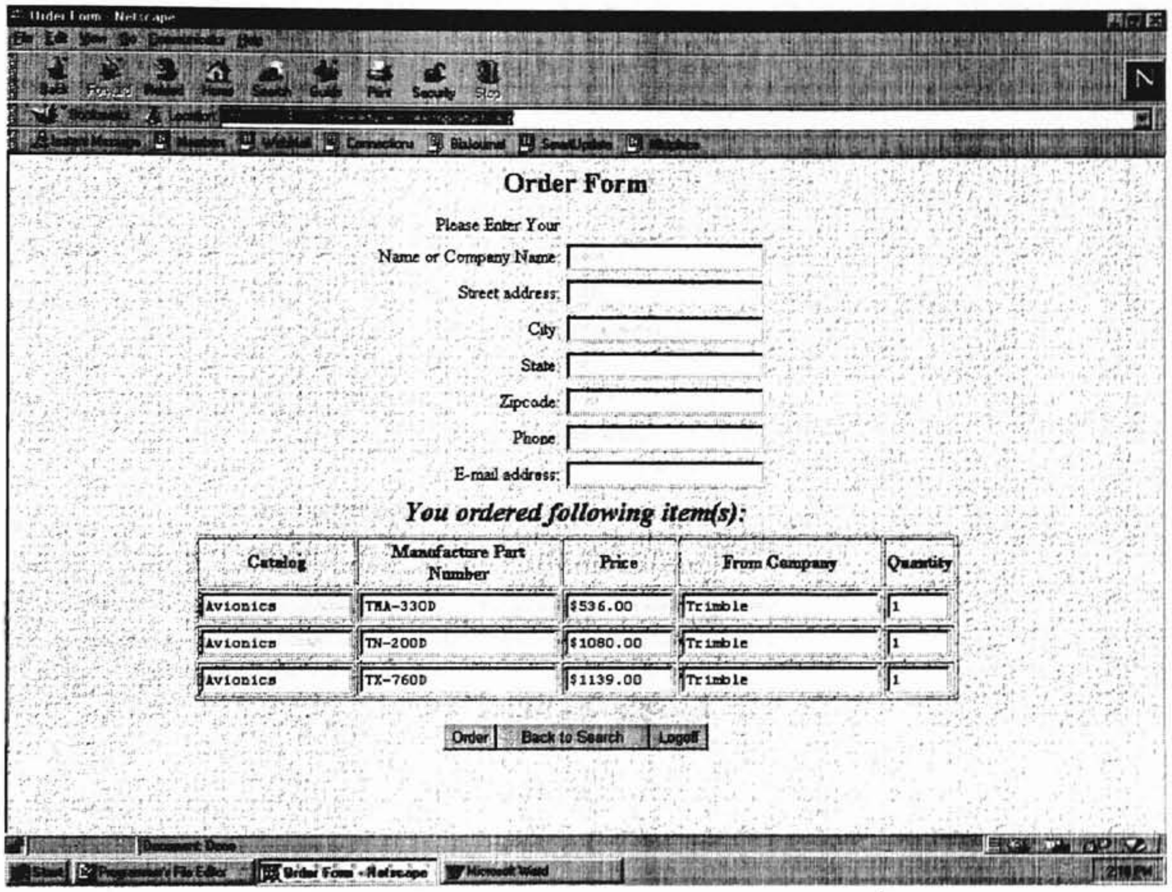
S-Tec	Yaw Damper	$2695.00		View	☐
Trimble	Package A3	$2960.00	W/Glideslope	View	☐
Trimble	Package B3	$2150.00	New Surplus	View	☐
Trimble	Package B3	$2689.00	W/O Glideslope	View	☐
Trimble	Package C1			View	☐
Trimble	Package C2			View	☐
Trimble	Package C3			View	☐
Trimble	Package C4			View	☐
Trimble	Package F			View	☐
Trimble	TDF-1000D			View	☐
Trimble	TMA-330D			View	☑
Trimble	TMA-340D			View	☐
Trimble	TMA-350D			View	☐
Trimble	TN-200D			View	☐
Trimble	TN-200D	$1080.00	NAV W/O Glideslope	View	☑
Trimble	TRT-250D	$1301.00	Transponder	View	☐
Trimble	TX-760D	$1139.00	Comm	View	☑
Val Avionics	Base Station 7600	$1450.00		View	☐
Val Avionics	VAL 760	$695.00		View	☐

Security Information

Any information you submit is insecure and could be observed by a third party while in transit. If you are submitting passwords, credit card numbers, or other information you would like to keep private, it would be safer for you to cancel the submission.

☑ Show This Alert Next Time

Figure 25. Selection of Product: TMA-330D, TN-200D, and TX-760D



Order Form

Please Enter Your

Name or Company Name:

Street address:

City:

State:

Zipcode:

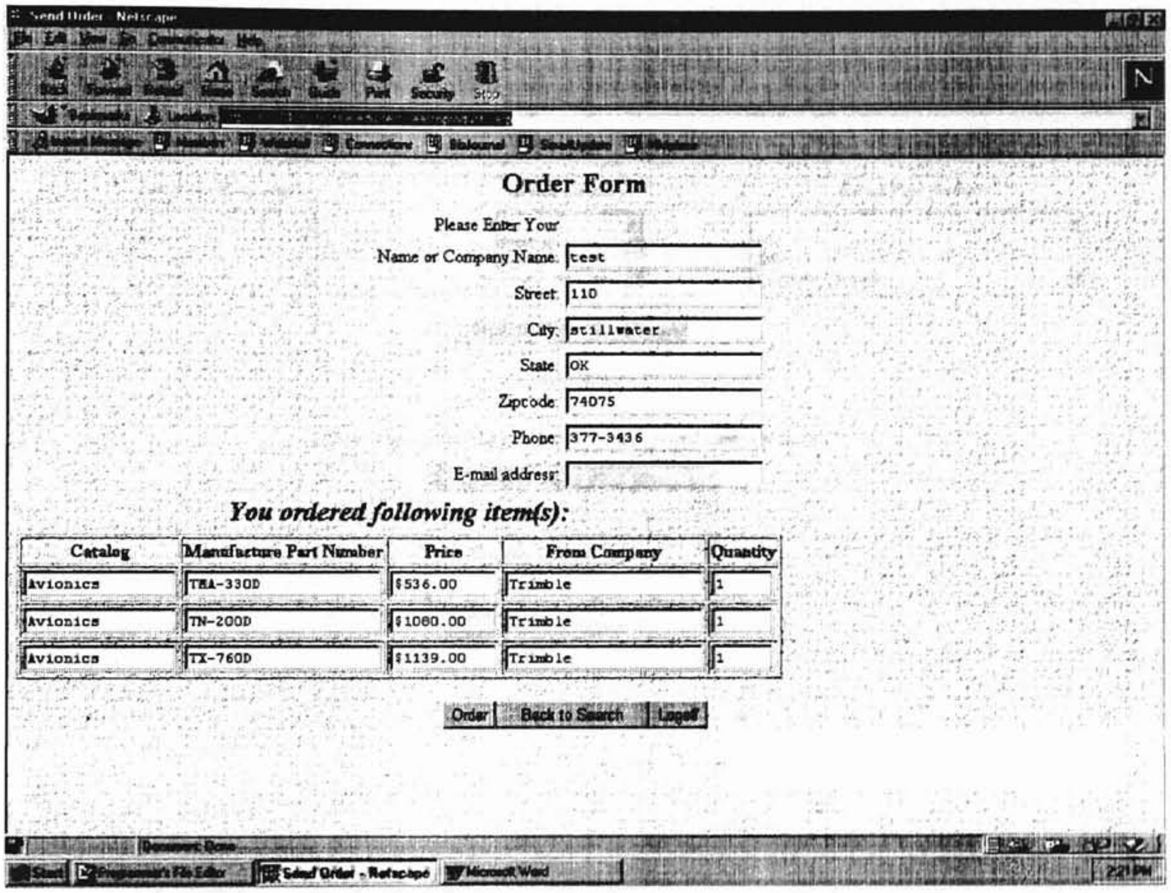
Phone:

E-mail address:

You ordered following item(s):

Catalog	Manufacture Part Number	Price	From Company	Quantity
Avionics	TMA-330D	$536.00	Trimble	1
Avionics	TN-200D	$1080.00	Trimble	1
Avionics	TX-760D	$1139.00	Trimble	1

Figure 26. Order Forms



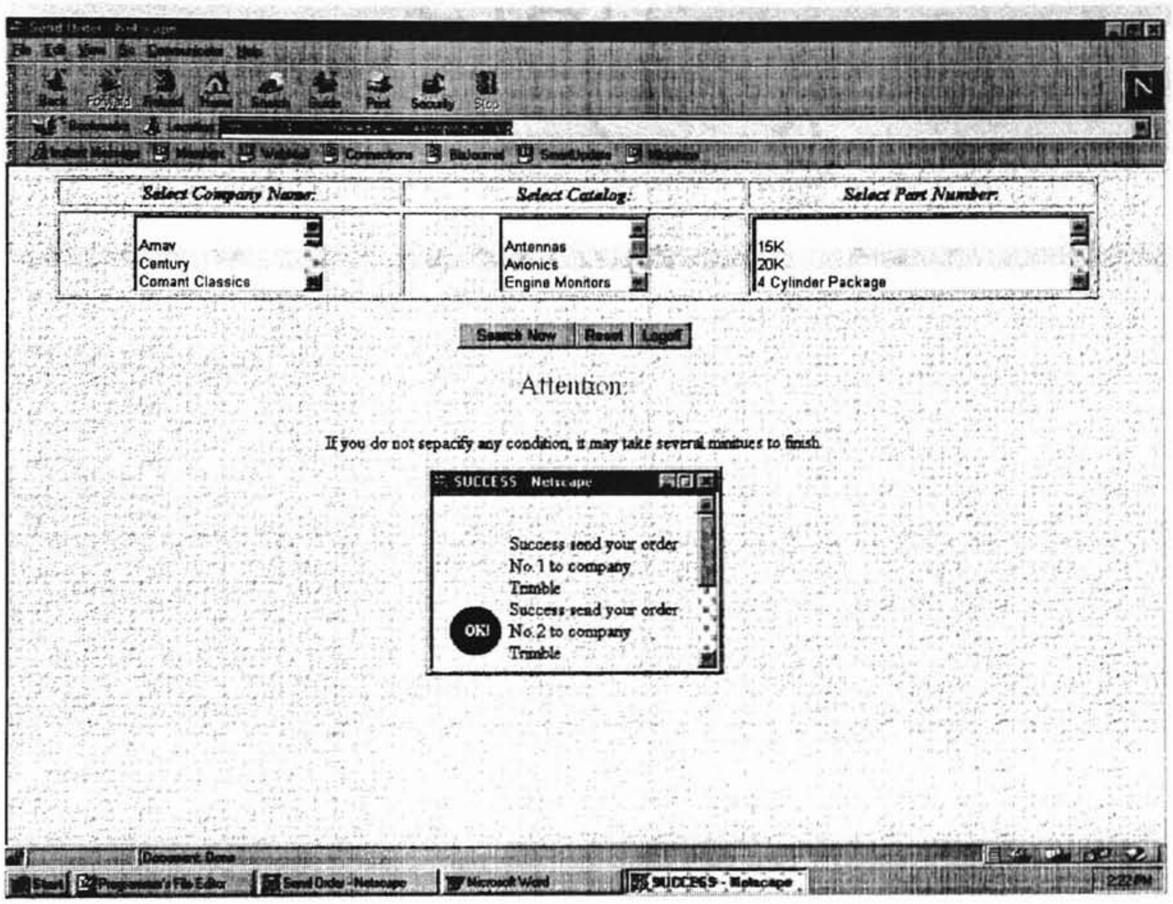
Please Enter Your

Name or Company Name:	test
Street:	110
City:	stillwater
State:	OK
Zipcode:	74075
Phone:	377-3436
E-mail address:	

You ordered following item(s):

Catalog	Manufacture Part Number	Price	From Company	Quantity
Avionics	TSA-330D	$536.00	Trimble	1
Avionics	TN-200D	$1080.00	Trimble	1
Avionics	TX-760D	$1139.00	Trimble	1

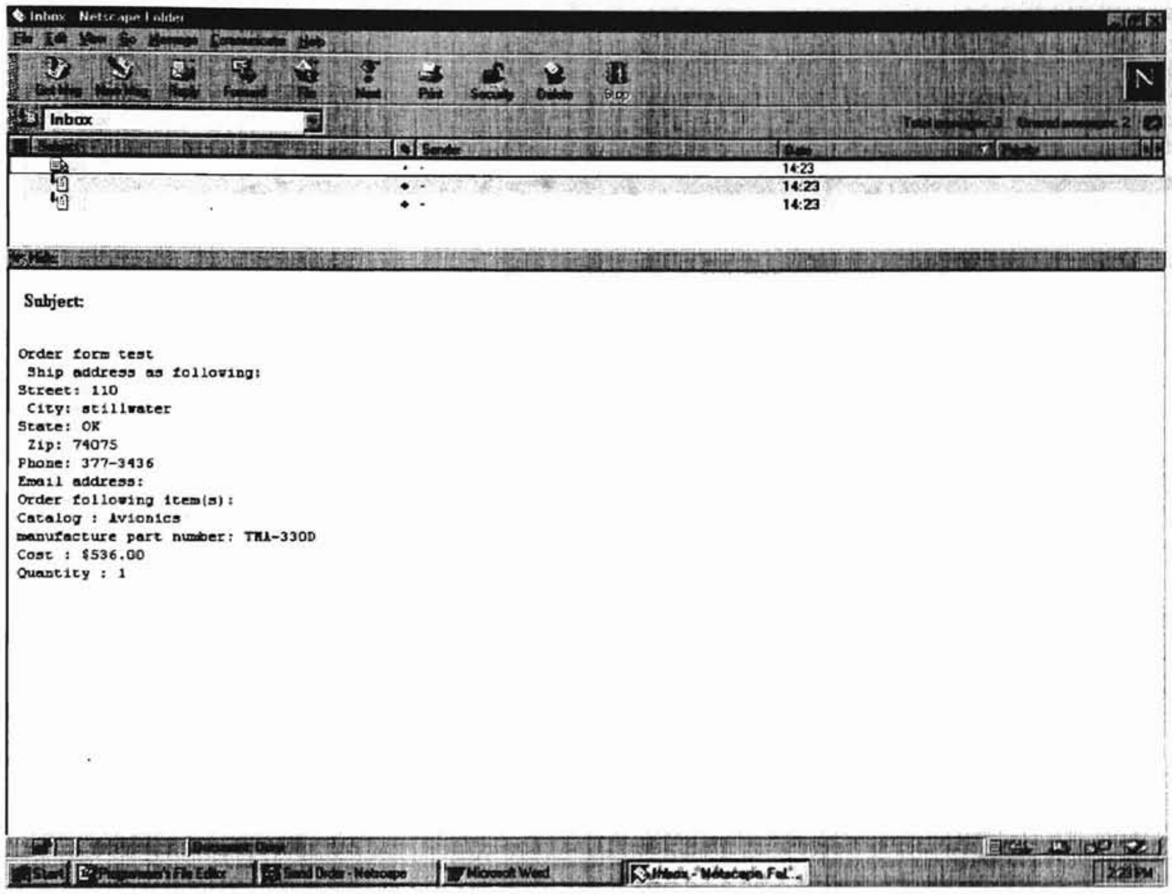
Figure 27. Completed Order Form



If you do not sepacify any condition, it may take several minitues to finish.

Success send your order No.1 to company Trimble
Success send your order No.2 to company Trimble

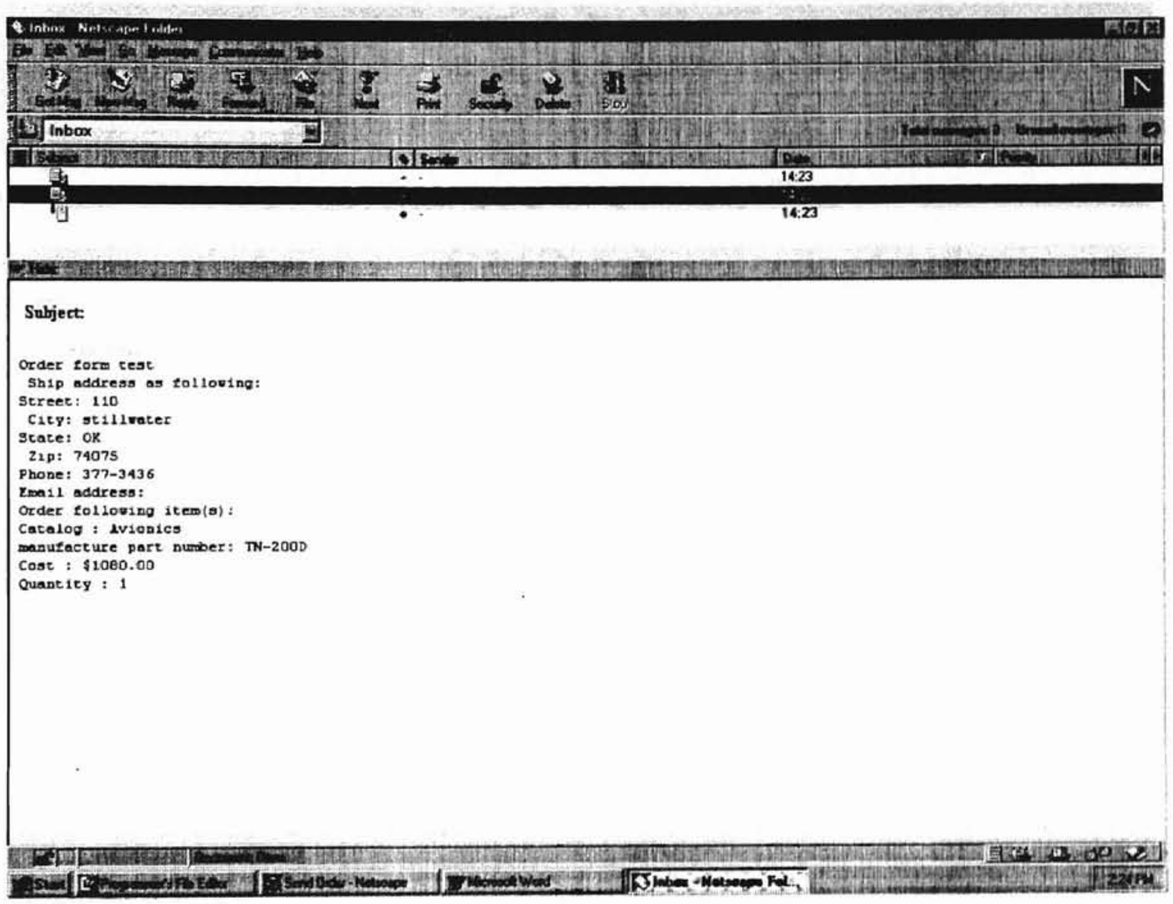
Figure 28. Message Box of Order Status



Subject:

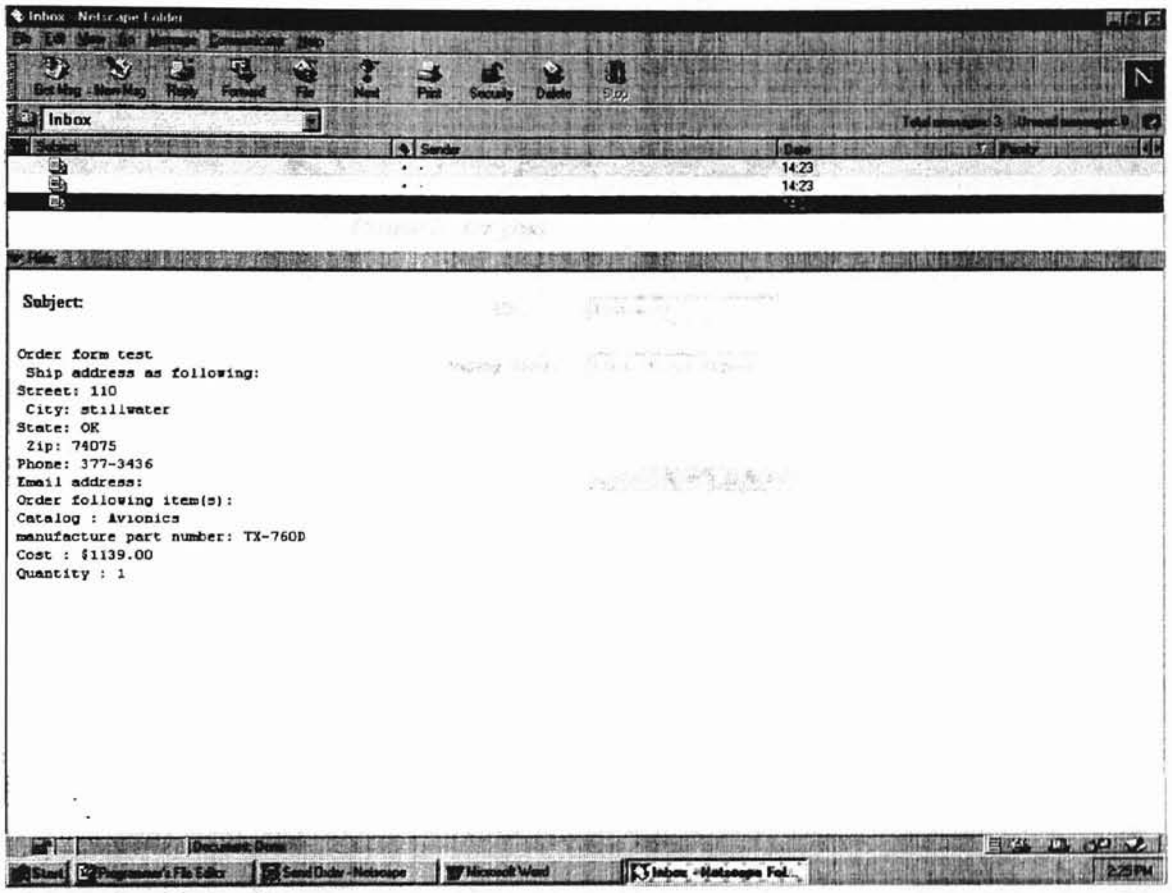
```
Order form test
 Ship address as following:
Street: 110
 City: stillwater
State: OK
 Zip: 74075
Phone: 377-3436
Email address:
Order following item(s):
Catalog : Avionics
manufacture part number: TMA-330D
Cost : $536.00
Quantity : 1
```

Figure 29. Order Mail 1



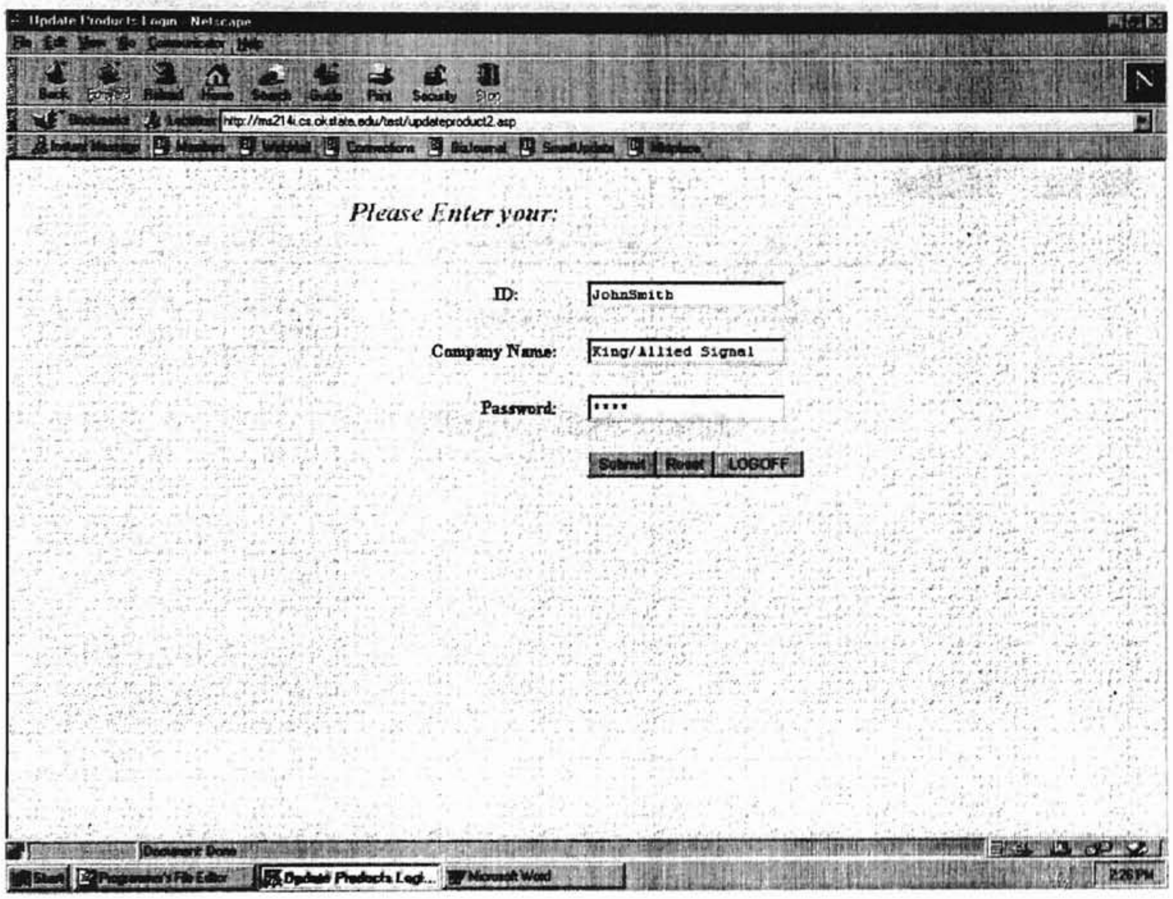
```
Order form test
 Ship address as following:
Street: 110
 City: stillwater
State: OK
 Zip: 74075
Phone: 377-3436
Email address:
Order following item(s):
Catalog : Avionics
manufacture part number: TN-200D
Cost : $1080.00
Quantity : 1
```

Figure 30. Order Mail 2



```
Order form test
 Ship address as following:
Street: 110
 City: stillwater
State: OK
 Zip: 74075
Phone: 377-3436
Email address:
Order following item(s):
Catalog : Avionics
manufacture part number: TX-760D
Cost : $1139.00
Quantity : 1
```

Figure 31. Order Mail 3


Update Products Login - Netscape
http://ms214i.cs.okstate.edu/test/updateproduct2.asp
Submit
Reset
LOGOFF
Document: Done

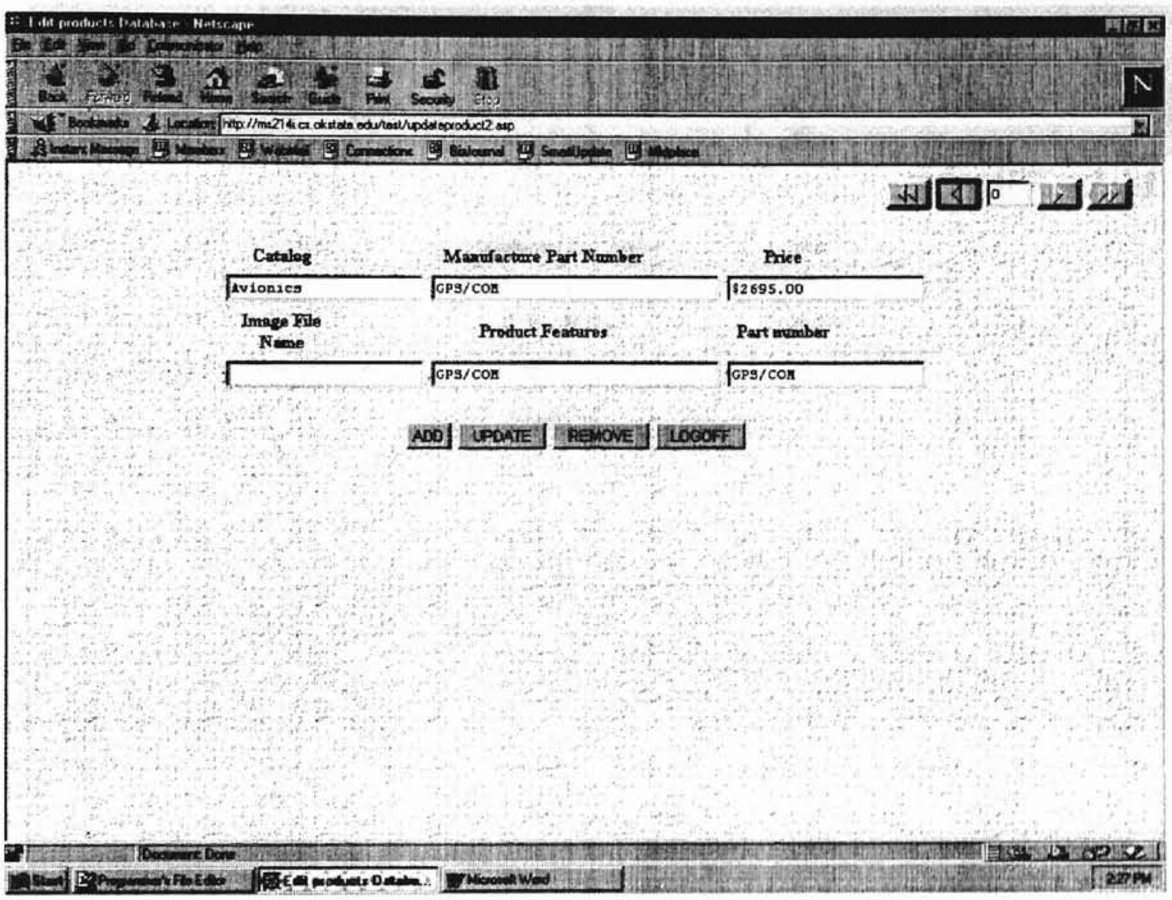
Please Enter your:

ID: JohnSmith

Company Name: King/Allied Signal

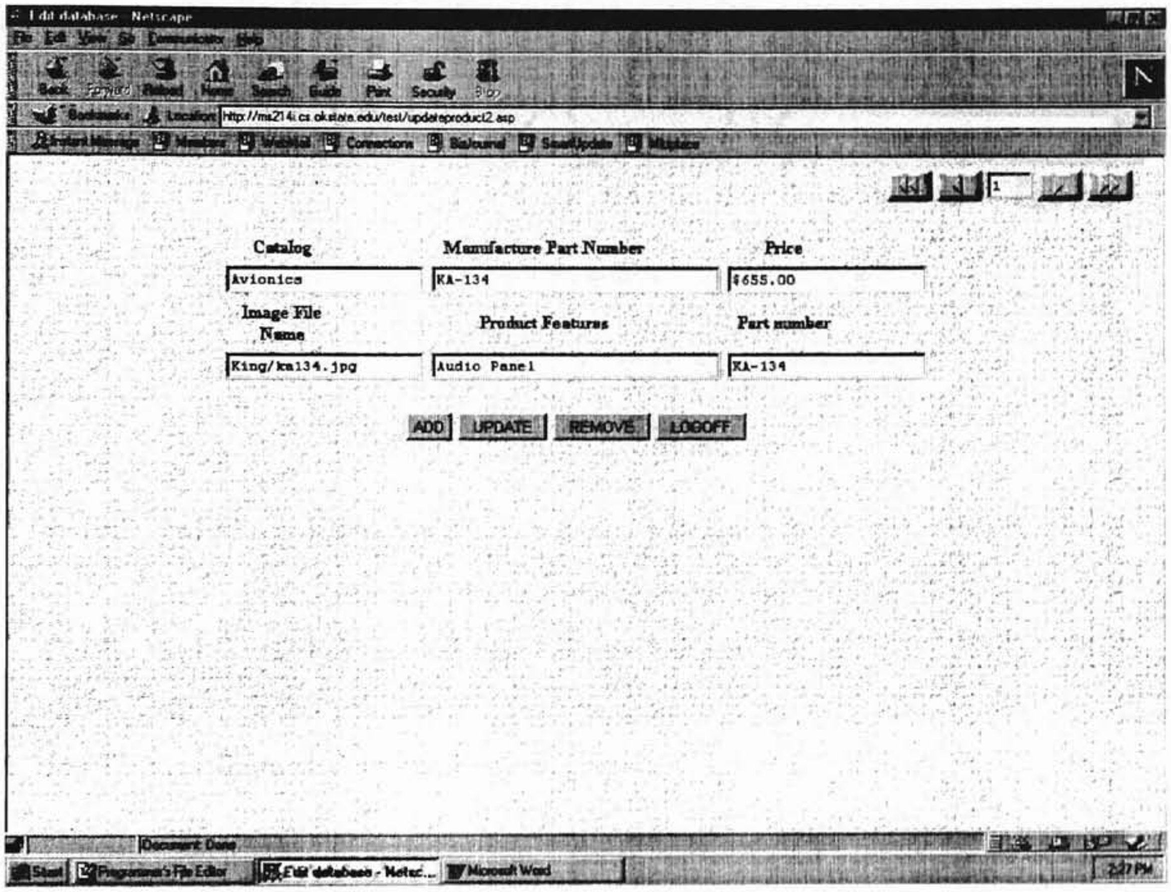
Password: ****

Figure 32. Log-on Page for Company Administrator


Edit products Database - Netscape
http://ms214i.cx.okstate.edu/test/updateproduct2.asp
0
ADD
UPDATE
REMOVE
LOGOFF
Document: Done
2:27 PM

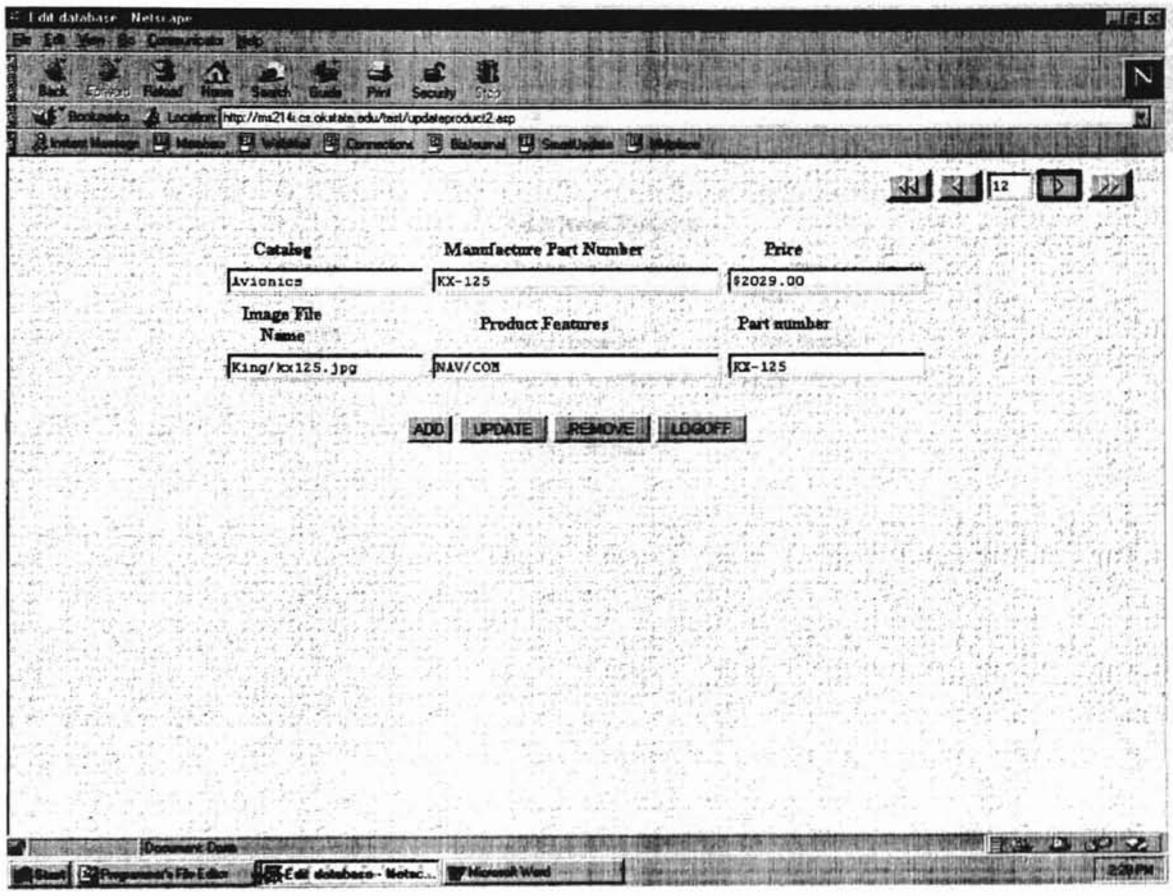
Catalog	Manufacture Part Number	Price
Avionics	GPS/COM	$2695.00
Image File Name	**Product Features**	**Part number**
	GPS/COM	GPS/COM

Figure 33. First Record of Product in This Company



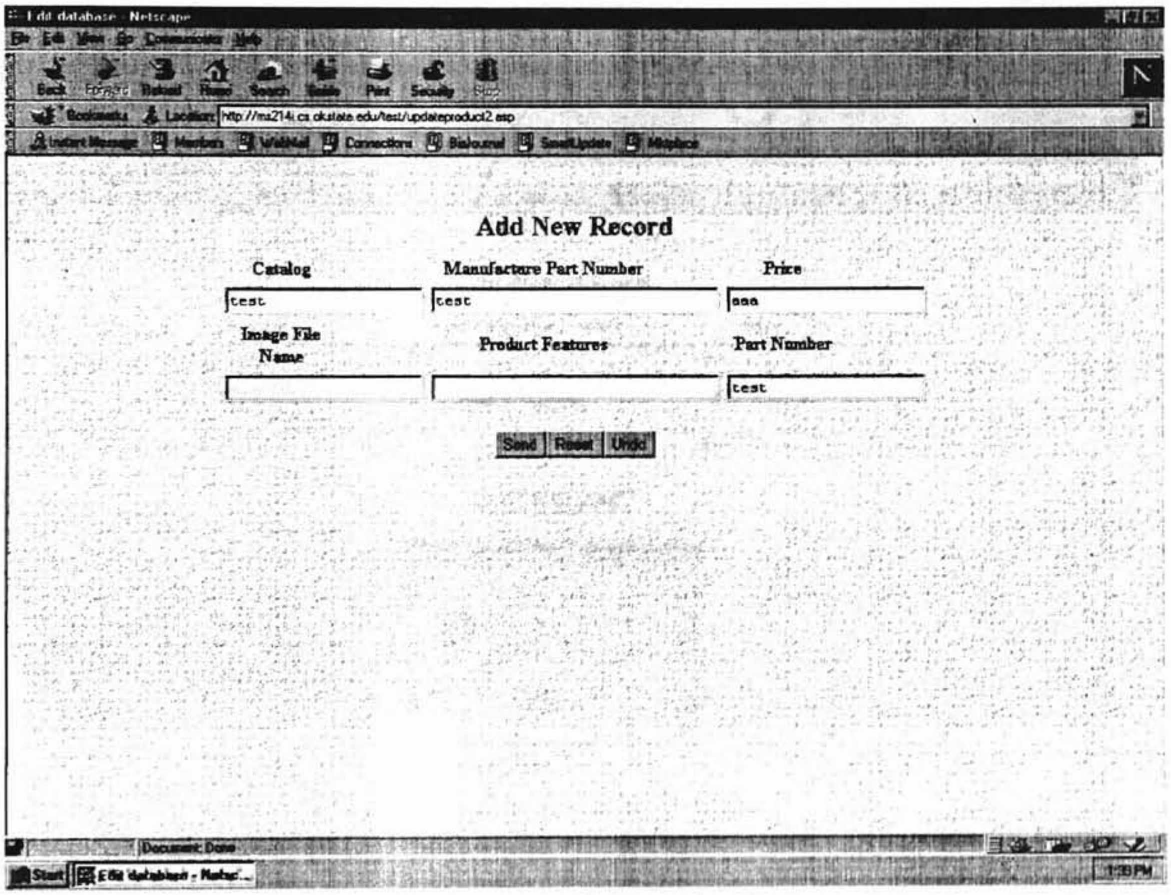
Catalog	Manufacture Part Number	Price
Avionics	KA-134	$655.00
Image File Name	**Product Features**	**Part number**
King/ka134.jpg	Audio Panel	KA-134

Figure 34. Enable Previous Button



Catalog	Manufacture Part Number	Price
Avionics	KX-125	$2029.00
Image File Name	**Product Features**	**Part number**
King/kx125.jpg	NAV/COM	KX-125

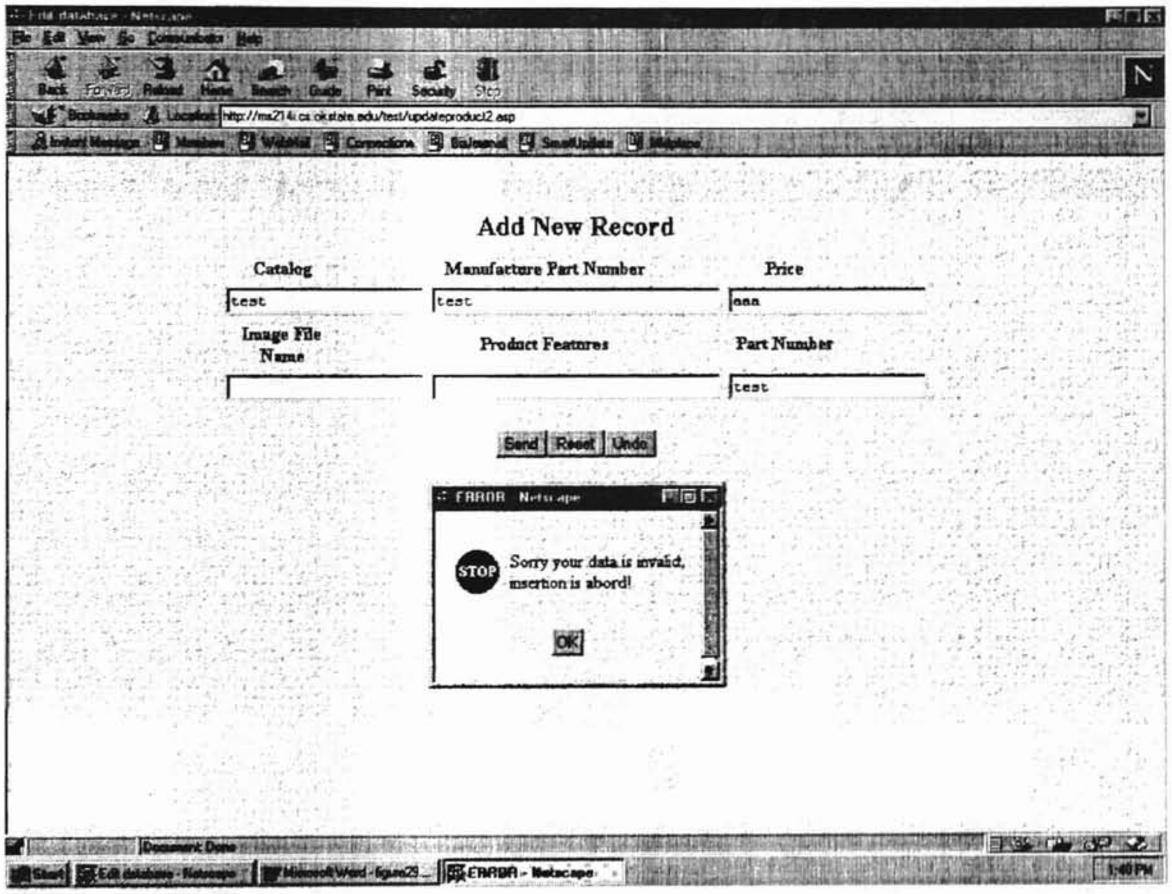
Figure 35. Disable Next Button


Edit database - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://ms214i.cs.okstate.edu/test/updateproduct2.asp
Document: Done
Start
1:38 PM

Add New Record

Catalog	Manufacture Part Number	Price
test	test	aaa
Image File Name	**Product Features**	**Part Number**
		test

Figure 36. Insertion with Invalid Data

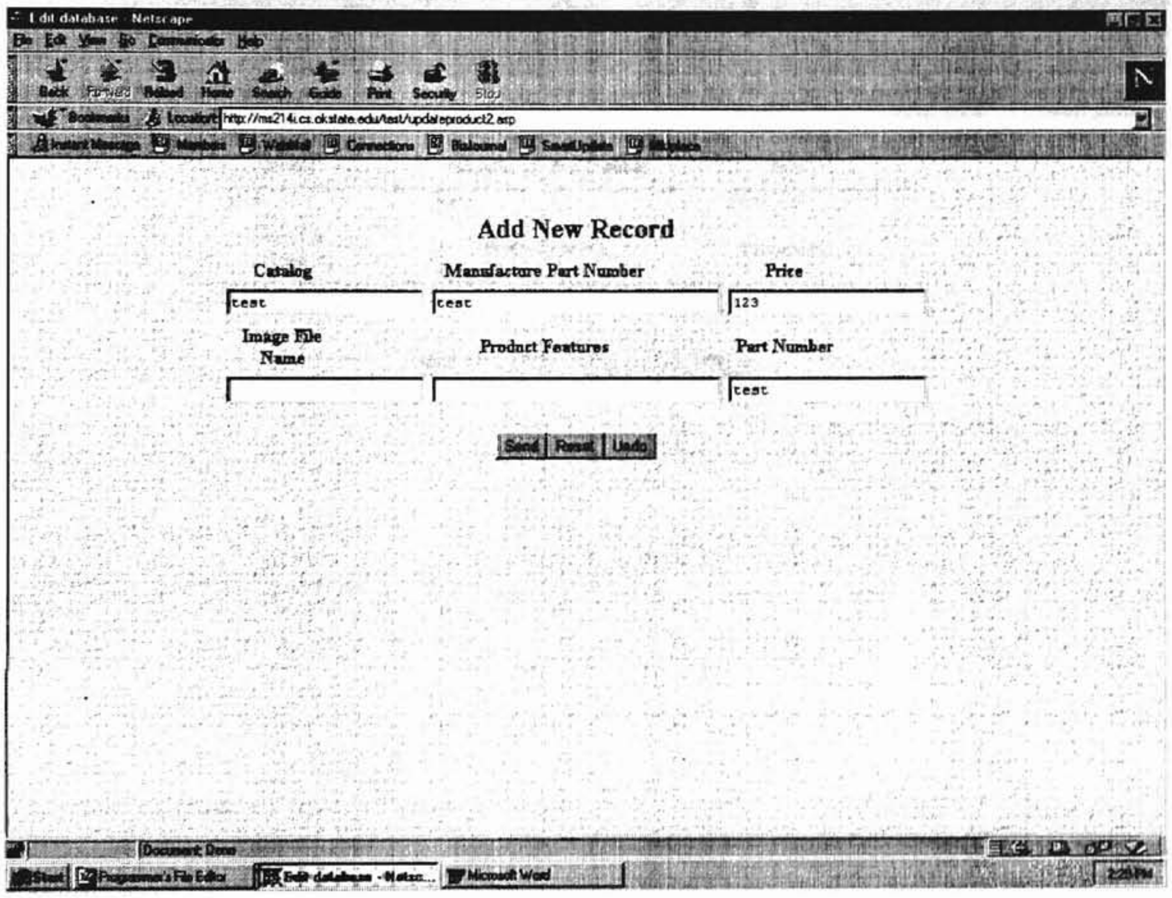

File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://ms214i.cs.okstate.edu/test/updateproduct2.asp
Send Reset Undo
ERROR - Netscape
STOP
OK
Document: Done
Start

Add New Record

Catalog	Manufacture Part Number	Price
test	test	aaa
Image File Name	**Product Features**	**Part Number**
		test

Sorry your data is invalid, insertion is abord!

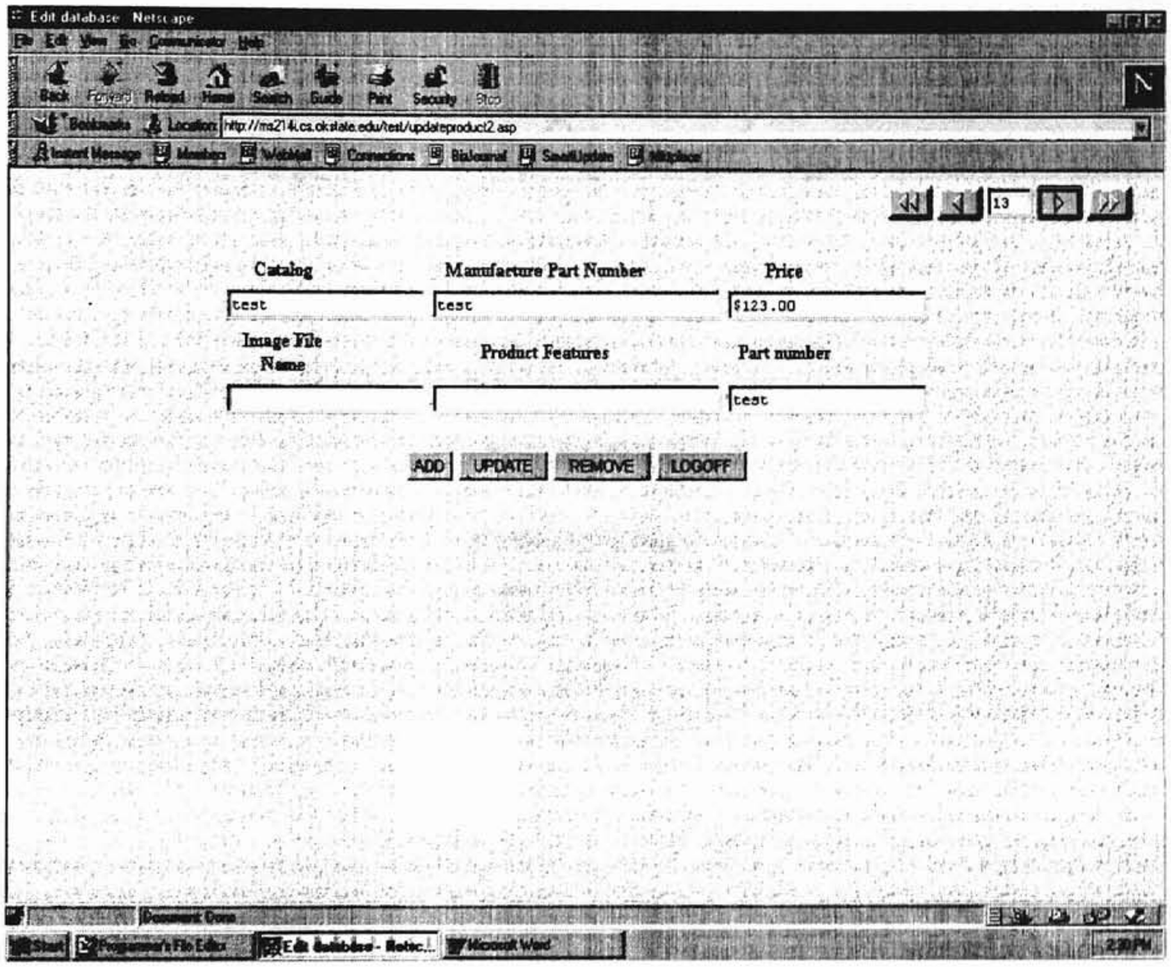
Figure 37. Error Message



Add New Record

Catalog	Manufacture Part Number	Price
test	test	123
Image File Name	Product Features	Part Number
		test

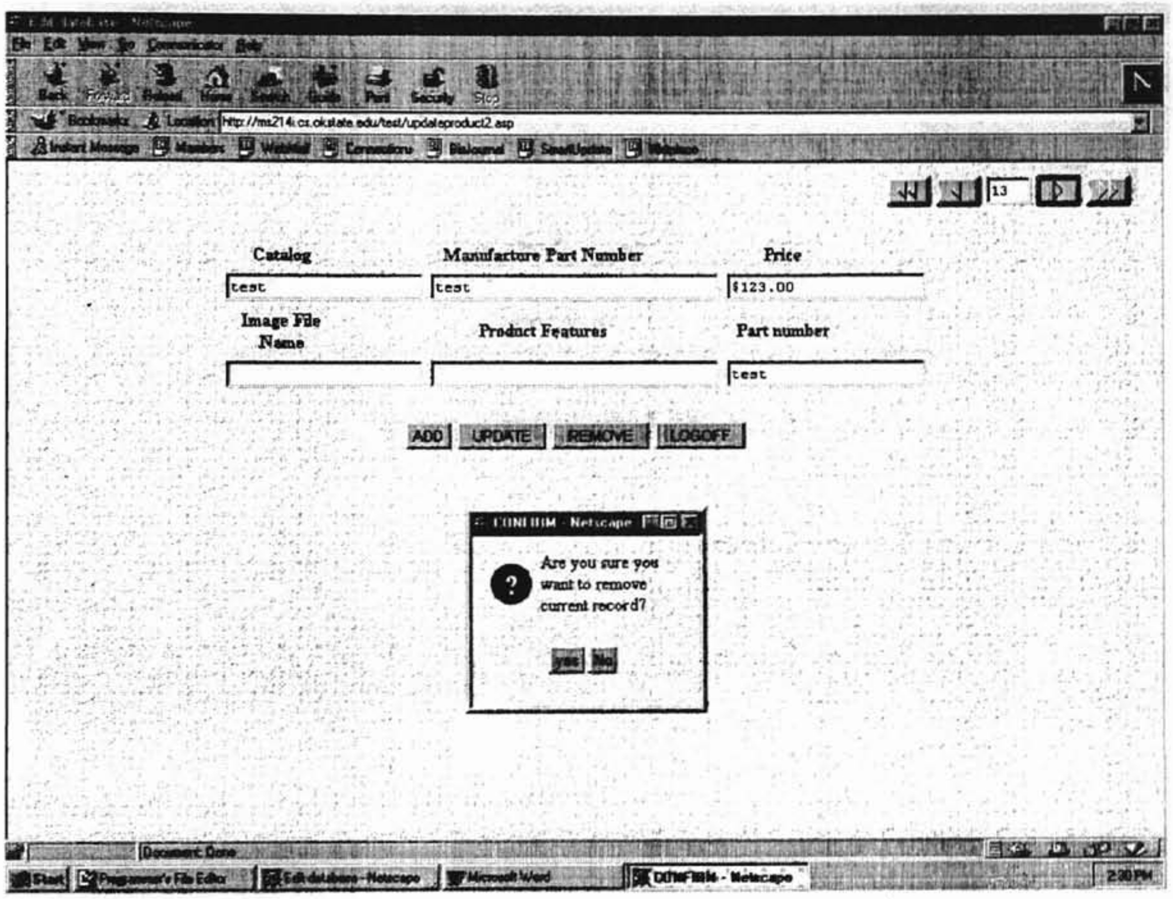
Figure 38. Insertion with Valid Data



Catalog	Manufacture Part Number	Price
test	test	$123.00

Image File Name	Product Features	Part number
		test

Figure 39. A Success Insertion.



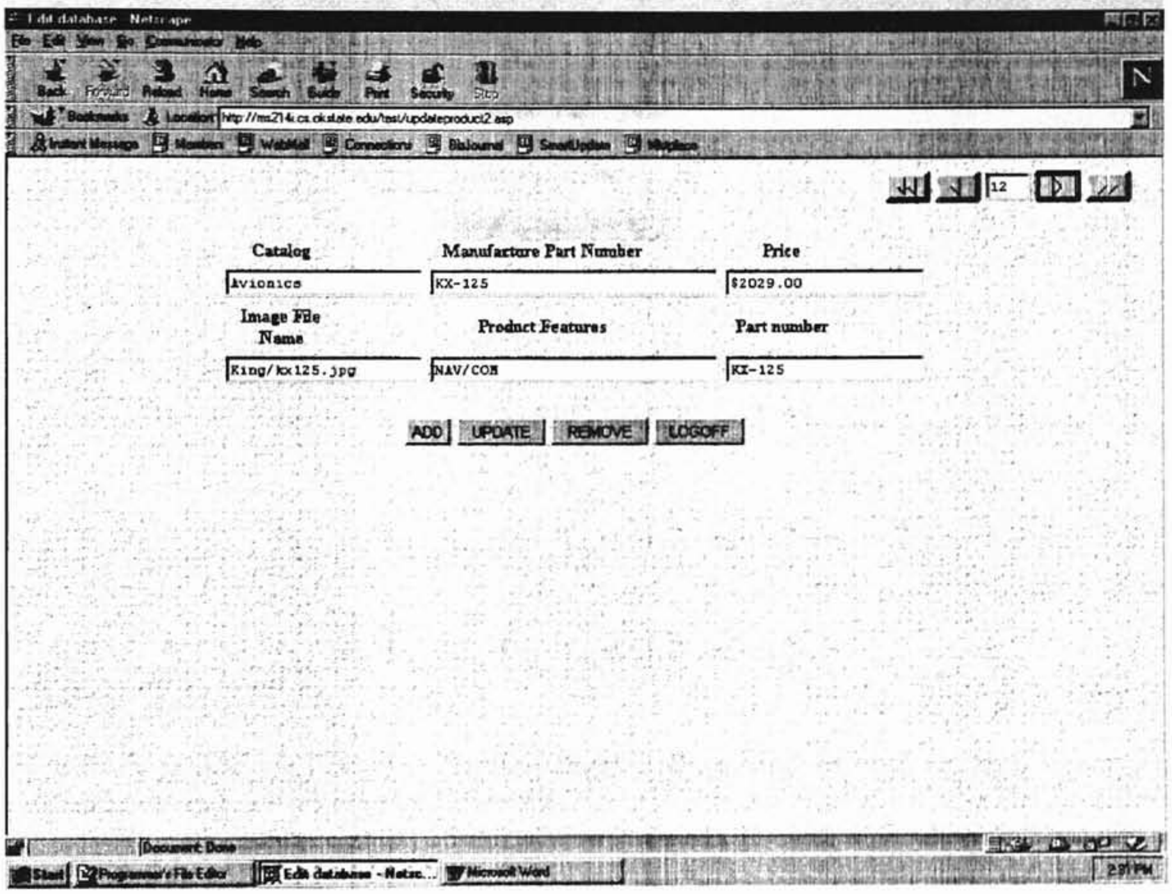
http://ms214.cs.okstate.edu/test/updateproduct2.asp

Catalog	Manufacture Part Number	Price
test	test	$123.00
Image File Name	Product Features	Part number
		test

CONFIRM - Netscape

Are you sure you want to remove current record?

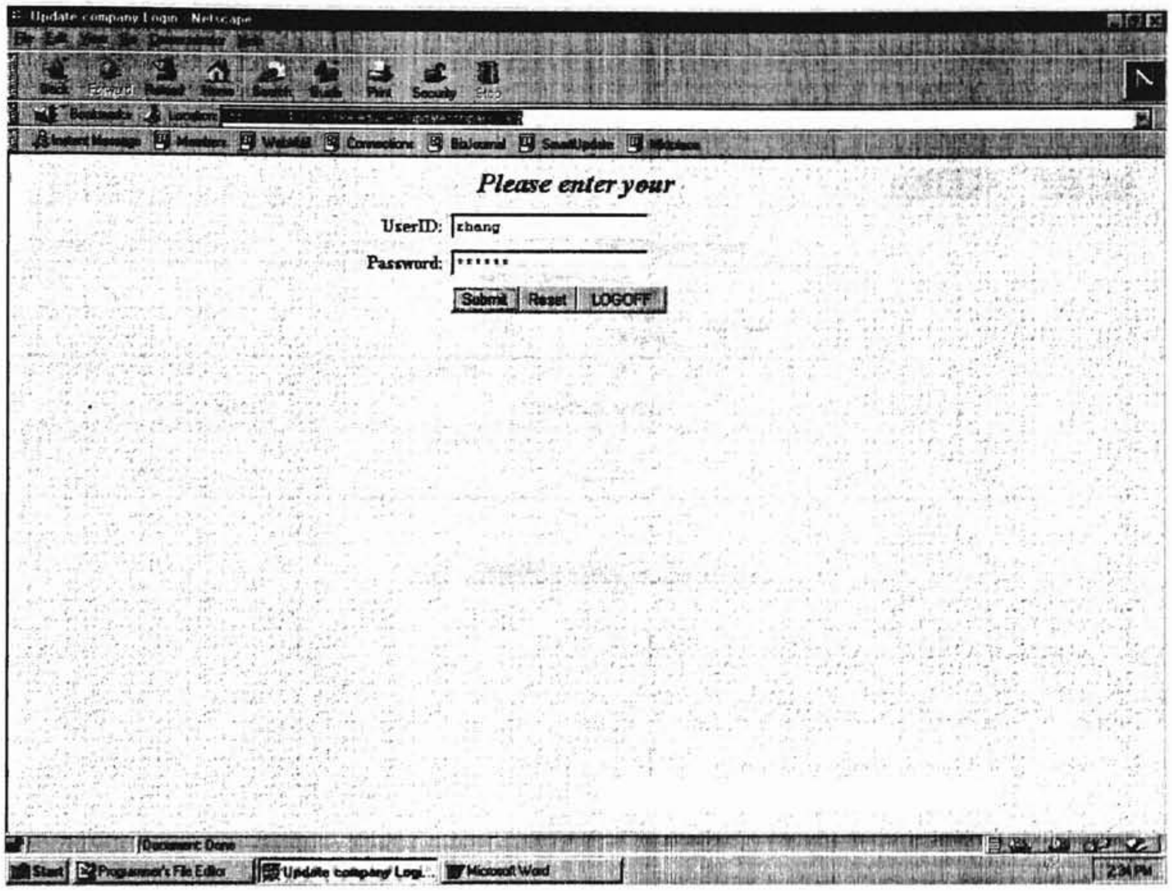
Figure 40. A Confirmation Box in Action



http://ms214.cs.okstate.edu/test/updateproduct2.asp

Catalog	Manufacture Part Number	Price
Avionics	KX-125	$2029.00
Image File Name	Product Features	Part number
King/kx125.jpg	NAV/COM	KX-125

Figure 41. Current Record Success Removed.

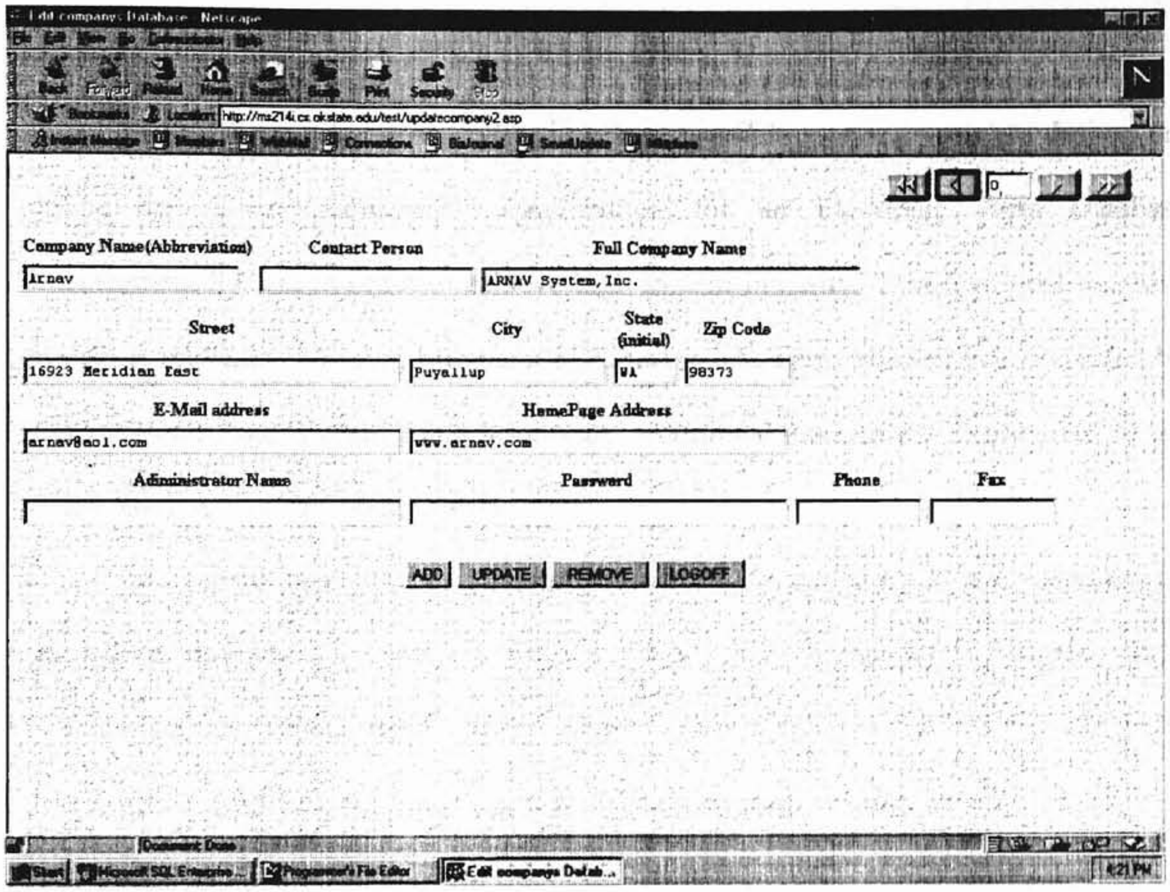


Please enter your

UserID: zhang

Password: ******

Figure 42. Log-on Page of Server Administrator.


Edit companys Database - Netscape
Back
Forward
Reload
Home
Search
Print
Security
Stop
Location: http://ms214i.cs.okstate.edu/test/updatecompany2.asp
ADD
UPDATE
REMOVE
LOGOFF
Document: Done

Company Name(Abbreviation)	Contact Person	Full Company Name
Arnav		ARNAV System, Inc.

Street	City	State (initial)	Zip Code
16923 Meridian East	Puyallup	WA	98373

E-Mail address	HomePage Address
arnav@aol.com	www.arnav.com

Administrator Name	Password	Phone	Fax

Figure 43. Company Information Page

CHAPTER V

SUMMARY, CONCLUSIONS AND SUGGESTED FUTURE WORK

In this thesis a formal specification language, Z language, is used to design and define the system requirement specification for an Electronic Parts Database Management System in order to achieve high system integrity. This relational database system is implemented using Microsoft SQL server 6.5. The web-interface is created by Active Server Page, VBScript and JavaScript. Additional functionality is provided by the COM using Active Template Library.

The Active Server Page is used to generate HTML dynamically, corresponding to the user's responses. It contains built-in objects and components to make ASP development tasks much easier. Although the default language with ASP is VBScript, JavaScript is used with the intention of reducing data traffic on the Internet. JavaScript can trap events such as a mouse click, a mouse move etc. In this thesis, JavaScript is used to preprocess the data to reduce data traffic and to improve system performance. JavaScript also is applied to build both the customized error message boxes and the confirmation boxes.

The Active Template Library (ATL) is employed because it allows the COM developers to concentrate on the unique functions that they want their COMs to provide; thus, helping to improve both the efficiency and effectiveness of the COM developers.

Special efforts are made in the design of a customized error message and confirmation box, and in the design of the E-mail COM. The design of the customized error and confirmation message box makes the message more eye-catching, so that the user can respond in a more timely manner in addition to a more pleasant graphical user interface.

The design of the E-mail COM enables its user to send an order form to the manufacturer directly rather then going through a server administrator, avoiding the human interference of the server administrator; thereby reducing and eliminating possible errors that may be involved, and providing a 24-hour availability of the service to the customers without the physical presence of the server administrator.

The system developed provides a user-friendly interface. It is also very easy to maintain. Since e-mail COM can be reused, it can be plugged in to some other applications (not only for ASP applications) to provide the same functions for the client. In its current version the e-mail COM supports only the sending mail function. Additional mail functions should be added in the future, such as, reading, replying, forwarding, and deleting mails. By adding this, e-mail COM not only can support current customers, but also can provide new service for new customers in the future. Currently, the system allows that one company only can have one company administrator. Sometimes, it is convenient that one company has more than one company administrator. Concurrency control should be added in the future in order to allow each company to have multiple company administrators.

REFERENCES

Armstrong, T. (1998) Active Template Library: A Developer's Guide. Foster City, CA: IDG Books Worldwide, Inc.

Chappell, D. (1996) ActiveX OLE A Guide for Developers & Managers. Redmond, Washington: Microsoft Press.

Codd, E.F. (1970) A Relational Model for Large Shared Data Banks. In Communications of ACM, Volume 13, Number 6, (June 1970), pages 377-387.

Codd, E.F. (1979) Extending the Database Relational Model to Compute More Meaning. ACM Transaction on Database Systems, Volume 4, Number 4 (December 1997), pages 397-434.

Diller, Antoni (1990) Z: an Introduction to Formal Methods. New York: John Wiley & Sons, Inc.

Fedorchek, A.M., Rensin, D.K. (1997) ASP Active Server Pages. Foster City, CA: IDG Books Worldwide, Inc.

Ford, N.J., Ford, J.M. (1993) Introducing Formal Methods: a Less Mathematical Approach. New York: Ellis Horwood Limited.

Goldsack, S.J., Kent, S.J.H. (1996) Formal Methods and Object Technology. London: Springer-Verlag London Limited.

Gruber, Martin (1990) Understanding SQL. San Francisco: SYBEX Inc.

Gundavaram, S. (1996) CGI Programming on the World Wide Web. Sebastopol, CA: O'Reilly & Associates, Inc.

Petzold, C. (1996) Programming Windows 95 The Definitive Developer's Guide to the Window 95 API. Redmond, Washington: Microsoft Press.

Rogerson, Dale (1997) Inside COM: Microsoft's Component Object Model. Redmond: Microsoft Press.

Siberschatz, A., Korth, H.F., Sudarshan, S. (1997) Database System Concepts. New York: McGraw-Hill, Inc.

Signore, R., Creamer, J., Stegman, M.O. (1995) The ODBC Solution Open Database Connectivity in Distributed Environments. New York: McGraw-Hill, Inc.

Tanenbaum, S., A., (1996) Computer Networks. Third edition, New Jersey: Prentice Hall PTR

Toth, V., (1997) Visual C++ 5. Second edition, Indianapolis, IN: SAMS Publishing

Van der Lans, Rick, F. (1988) Introduction to SQL, translated by Andrea, Gary Workingham, England (1988): Addison-Wesley Publishing Company, Inc.

Whiting, B., Morgan, B., Perkins, J. (1996) Teach Yourself ODBC Programming in 21 Days. Indianapolis, IN: Sams Publishing.

APPENDIXES

APPENDIX A

NOTATIONS OF Z SPECIFICATION LANGUAGE USED IN THIS THESIS

$\in$	Set membership
$\rightarrow$	Function mapping
$\cap$	Set intersection
{ }	Empty set
Δ	Combining the before and after specifications of state
Ξ	Used in the specification of operation that does not change the state of the database
?a	Input variable a
!a	Output variable a
F(|U|)	set of all those thing that can be reached from U
dom	Domain
ran	Range
$\rhd$	Range anti-restriction
$\oplus$	Overwriting Operator

APPENDIX B

ABBREVIATION USED IN THIS THESIS

ANSI	American National Standards Institute
API	Application Programming Interface
ASP	Active Server Page
ATL	Active Template Library
CGI	Common Gateway Interface
CLI	Call Level Interface
COM	Component Object Model
DBMS	DataBase Management System
DCL	Data Control Language
DDL	Data Definition Language
DLL	Dynamic Link Library
DML	Data Manipulation Language
HTML	HyperText Markup Language
IDL	Interface Definition Language
ISO	International Standard Organization
ODBC	Open DataBase Connectivity
OLE	Object Linking and Embedding
SAG	SQL Access Group

SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

APPENDIX C

GLOSSARY OF TERMS IN THIS THESIS

System R	An IBM project Name
System R	The relational database management system experimented in IBM project System R
IUnknown	Standard interface for COM, it contains three methods: QueryInterface, AddRef, and Release
Vtable	C++ virtual function table
IDispatch	A mechanism by which an object can provide access to its methods via dynamic invocation
ActiveX	Build on the component Object Model and a document-focused technology

4

VITA

Yijing Zhang

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND IMPLEMENTATION OF A WEB DATABASE SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Huzhou, China on February 5, 1969, the daughter of Junhong Zhang and Binfeng Ma.

Education: Graduated from Economics Department, Zhejiang University of Technology in July, 1990; received Bachelor of Science degree in Industry Accounting. Completed the requirements of the Master of Science at Oklahoma State University in December, 1998.

Professional Experience: Employed by Huzhou Vocational School, Huzhou, China, as a Senior Instructor, 1990 to 1994; employed by Oklahoma State University, Department of Computer Science as a Research Associate, Oklahoma State university, Department of Computer Science, 1997 to present.