Andrew Dzhigo. Building Online Applications: Sun Java Servlets and Microsoft Active Server Pages. A Master's paper for M.S. in IS degree. April, 2001. 67 pages. Advisor: Paul M. Jones

This paper describes two approaches of building database driven online applications: Microsoft Active Server Pages (ASP) and Sun Java Servlets. It provides programs for both implementations and demonstrates the advantages and disadvantages of each one. It concludes that ASP is better for use for small scale systems and Servlets for more complex ones. Source codes of the programs are provided in the text.

Headings:

Servlets

Active Server Pages

Web sites -- Design

Web servers

Application software -- Development

BUILDING ONLINE APPLICATIONS: SUN JAVA SERVLETS AND MICROSOFT ACTIVE SERVER PAGES

By
Andrew Dzhigo

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

April, 2001

Approved by:

Advisor

Table of Contents

INTRODUCTION

If we take a brief look at the history of the World Wide Web (WWW) we could spot a strong tendency of moving from static Hypertext Markup Language (HTML) pages to a more dynamic and interactive environment which allows system-user interactions, database searching, application level service and much more. All of the described functionality involves making some significant changes on the server side and sometimes on the client side. Typically on the server side those changes would allow some programs to interact with the server and to process client requests.

Probably, the first attempts to implement this functionality were made by introducing Server Side Includes (SSI) and Common Gateway Interface (CGI) technologies. CGI programming has become very popular since it allows running programs written in many languages if the appropriate compiler is installed on the system. The downside of the CGI is that each time a new client interacts with a server CGI-program runs as a separate process thus eating up system resources.

In the recent years many new WWW-programming technologies have emerged. These technologies provide additional functionality for both clients and servers and can be divided into two categories:

1. Scripting languages:
    - *client-side*: JavaScript, VBScript
    - *server-side*: Active Server Pages, Java Server Pages, ColdFusion, CGI programs (compiled on runtime)
2. Programs:
    - *client-side*: Java Applets, ActiveX components, plug-ins
    - *server-side*: Java Servlets, JavaBeans, ISAPI programs, precompiled CGI programs

The above examples of technologies show the variety of possible solutions that exist now to create interactive WWW systems and applications. Those shown are just the major ones. There are many more technologies available.

In this paper I would like to evaluate two of these technologies: Java Servlets and Active Server Pages. Both of these technologies are widely used in today’s WWW systems and have their strengths and weaknesses. I’m proposing building two identical online database driven systems using these technologies and looking at advantages and disadvantages of both implementations.

LITERATURE REVIEW

As with all new technologies literature on Servlets and ASP is mostly limited to architecture reviews ("white papers"), tutorials, use case studies, and textbooks. There is little literature on technology comparisons or evaluations. Some of the sources that will be covered in this section are available only online. Others were taken from specialized journals and magazines.

Probably the best description of Java Servlet technology is given in the Servlet "white paper" [18]. This online document provides very detailed information on the Servlet technology. It also addresses the possible uses of Servlets such as:

- processing HTML forms
- supporting collaborative applications
- sharing work among different clients
- forwarding requests to other services

The large part of the article covers Servlet's Application Program Interface (API). The document also compares Servlet technology with CGI and Fast-CGI approaches and clearly explains the benefits of using Servlets:

- Servlets "run in parallel within the same process as the server" and "provide compelling performance advantages" over CGI which creates new child process to handle each request and even Fast-CGI which creates single child process to handle multiple requests
- "100% Pure Java programs don't care what operating system they use, you have the power to choose whatever system vendor best addresses your requirements in any given application"

ASP technology overview is also available online. Microsoft Developer Network (MSDN) Online Web Workshop hosts a number of articles covering ASP. "ASP Technology Feature Overview" [28] by Ron Wodaski gives the basic outline of ASP. It emphasizes one of the big advantages of ASP as a technology based on the use of scripting languages – fast application development. The article addresses the performance and security issues of ASP as well as guidelines for developing web applications using ASP. The following key features are provided by ASP for web applications support:

- application object – allows setting and getting application properties and using its methods
- isolated processes – allows running each application in its own process on the server
- start/end events – allows starting and ending applications as needed

Large part of this article covers ASP object model and use of the Cookies technology in ASP.

Probably, the most valuable articles in terms of this paper would be those that compare and evaluate different web applications technologies. There is a fair number of works that contain reviews of these technologies, however, almost none of them make comparisons or evaluations. And there is almost no works available on the comparison of different conceptual approaches such as scripting language systems (ASP) vs. compiled program modules (Servlets).

"Database-Driven Pages" [12] gives an overview of two technologies: ASP and ColdFusion. This report contains a good introduction to ASP, covers its syntax and describes ASP tools that allow database access via WWW. This paper also discusses alternative platforms for ASP besides Microsoft's Internet Information Server on Windows NT. Apache server on UNIX is given as an example of such an alternative. The last part of the article describes several Interactive Development Environments (IDEs) available to work with ASP technology: NetObjects' ScriptBuilder, Microsoft's Visual InterDev and FrontPage.

In "Integrating Webpages with Databases" [11] David Cox mentions an important issue with modern e-commerce and web sites - they are not only about user interface but also about using and interacting with databases. I strongly agree with his view and would say that this is true not only for e-commerce but also for any other sites that provide rich information content. Further in his article Mr. Cox reviews three technologies: ColdFusion, Microsoft's Internet Database Connector (IDC) and ASP. Describing these technologies the author states that all of them are proprietary and thus not portable. This is not exactly true, ASP applications can be transferred to several platforms as was noted in the "Database-Driven pages" article discussed above.

As I stated in the beginning of the literature review most articles on these technologies are limited to manuals and tutorials. Good examples of such works are

"Programming for Active Server Pages: Creating Dynamic Content using ASP" [8] by Leon Chalnick and "Visual InterDev and ASP" [21] by John Lam. Both articles contain excellent tutorials on building ASP applications. Leon Chalnik gives a throughout description of ASP server object model and ActiveX Data Object (ADO) provider to ODBC databases. John Lam focuses more on working with ASP using Visual InterDev IDE.

So far I've been mostly discussing works related to ASP technology. Unfortunately, there is a lack of articles covering both Servlets and ASP technology. The situation with works on Java Servlets technology is very similar to the situation with ASP - many reviews and tutorials are available, but aside from that there are almost no works comparing Servlets with other technologies.

Java Servlets technology appeared in 1997 as an alternative to CGI based applications and scripting languages systems such as ASP. And, as with almost all new technologies, it required a new environment to work in. Java Web Server 1.1 developed by Sun Microsystems was the first web server supporting Servlets. A very good article by Greg Alwang "Java Web Server" [1] is one of a few that compares some features of ASP with Servlets. The article explains and evaluates the difference between ASP scripting and Servlets and concludes that Servlet technology "delivers immense flexibility to developers creating dynamic Web applications" by using actual Java language for programming. The article also speaks about Java Web Server 1.1 drawbacks, that is not very fast performance and some server administration problems. However, it comments on the server's high extensibility and considers it as "an ideal platform for Java developers looking to distribute customized Web applications".

Another article that talks about actual use of Servlets and gives an evaluation of this technology is "Java Servlets" [27] by Jon Udell. He emphasizes the multithreaded nature of Java language and states that Servlets allow developers to create "lightweight Web services" which are extremely responsive. The author does not see the Java applets on the client side as much of an improvement over using standard HTML page combined with some JavaScript functions. At this point Jon Udell does not take into account the ability of Java Servlets on the server side and Java Applets on the client side to maintain a persistent connection, which is not available in typical browser-server interactions. I believe that this feature does represent a significant step towards large distributed Web applications. I will discuss in my paper the significant role that persistent connection could play in online system.

DATABASE OVERVIEW

Before I begin talking about Servlet and ASP implementation approaches I must describe the underlying database structure used in the system.

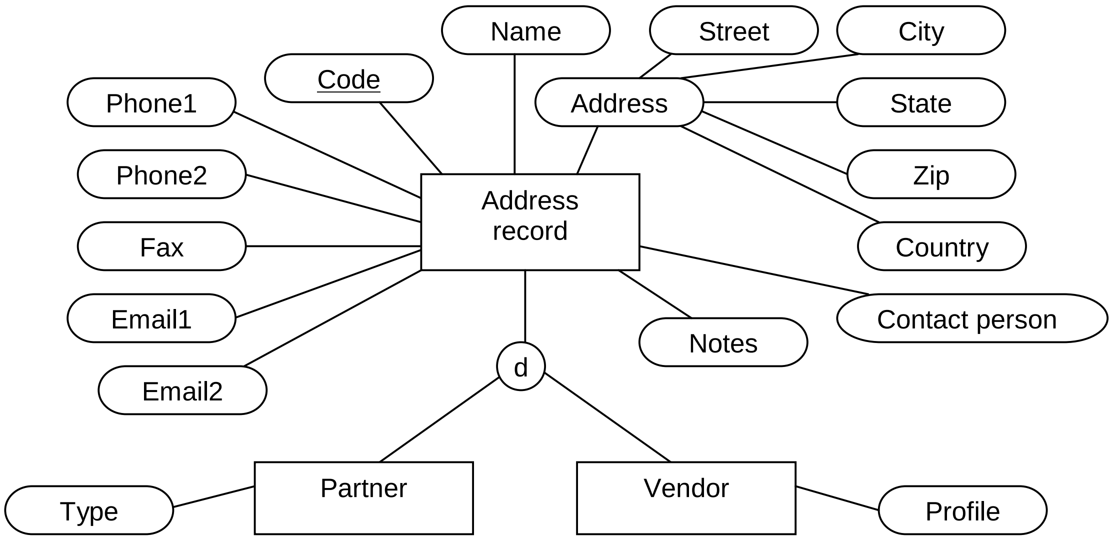
This database is a part of the integrated application developed for Slavic and East European Resources department of Davis Library in the University of North Carolina at Chapel Hill. The database keeps track of departmental foreign vendors and exchange partners address information. Developed in Microsoft Access 97 the database provides the following capabilities for end users:

- adding a new partner/vendor
- modifying/deleting information about existing partners/vendors
- browsing/searching existing partners/vendors

The functions described above are implemented in Access using forms and Visual Basic programming language modules.

Lately it has become clear that this information could be of use for other library departments, especially, for the Acquisitions department. Since all the information in the database will still be entered and modified only by the SEER department the only part that other departments will have to gain access to is the browsing/searching interface. In this paper I will describe and evaluate two possible ways of providing such capability using WWW application model.

The relational schema of this database. is not very complex.



The corresponding database schema based on the above shown relational schema puts *Partner* and *Vendor* subclasses attributes into separate tables. Also I made a decision to put *Notes* attribute into a separate table since it is a big field (255 characters) which will not always be used.

Database schema:

PartnersVendors (<u>pcode</u>, type, name, street, state, zip, city, country, phone1, phone2, fax, email1, email2, cperson)

PartnerTypes (FK: <u>pcode</u>, ptype)

VendorProfiles (FK: <u>pcode</u>, profile)

PartnerVendorNotes (FK: <u>pcode</u>, notes)

SERVLETS TECHNOLOGY OVERVIEW

Java Servlets are server side programs written in Java which allow developers to extend web server capabilities. In order to execute Servlets a server must be Java enabled and must support Servlet Application Program Interface (API). Servlets are very similar to Applets but on the server side. One of the major differences between them is that Servlets do not have to support and implement user interfaces since they run only on servers.

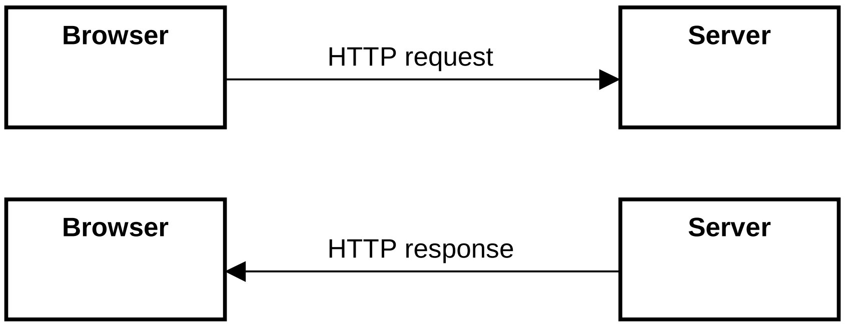
Servlets are protocol independent and thus can support a variety of clients from FTP and Telnet programs to audio and video streaming applications. In this paper I am going to focus on a Servlet subclass – HTTP Servlet. This Servlet is specifically written to work with HTTP connections. It supports all standard HTTP methods, including the most widely used – *GET* and *POST*.

The beauty of Servlet technology is that being written in Java it is platform independent. Developers can use in Servlets all their existent Java classes and libraries without changing them. Java has very powerful tools to work with databases. JDBC (Java Database Connectivity) API allows Java programs to work with any SQL compliant database. The support of CORBA (Common Object Request Broker Architecture) gives Java classes the ability to communicate with other programs regardless of the language they were written in. The above features make Java Servlets a perfect architecture to use for middle tiers of distributed applications.

Together with Servlets in my project I will be using two very powerful Java technologies: JDBC and RMI (Remote Method Invocation). As I said earlier JDBC allows Java programs to communicate with different databases. RMI gives developers an ability of calling server methods remotely and therefore eliminates the need for writing their own data transferring routines. All of these will be described in more detail later in the "Servlets Implementation" part of this paper.

SERVLETS IMPLEMENTATION

In order to get a better understanding of the online applications built on Servlet technology I would like first to take a look at a typical online communication process model: web browser client – web server interaction. The simplified scheme of this interaction is given below.

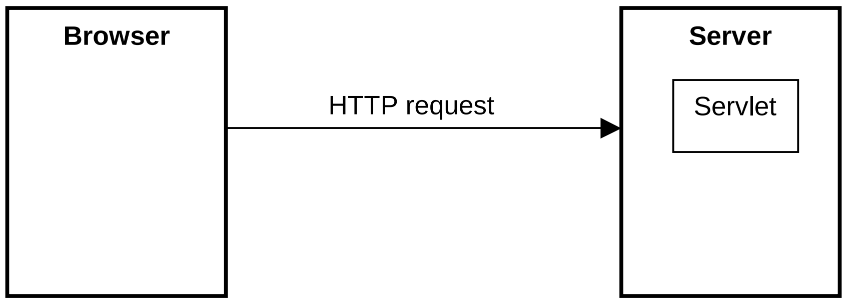


Browser sends server a request for a specific object (HTML page, picture, etc.). Server reads the request and sends browser a response together with the requested object if it was found or a not found error. In an interaction like this all the communication starts with a browser request and ends with a server response. The notion of connection between client and server in this example is a two-step process which exists only until a server sends a response. If browser wants to get another object from server it has to send a new request which starts a new connection. Server never knows if a client has finished communicating or not. And in simple activities such as providing access to static HTML pages it makes perfect sense. Server does not have a need to know about the state of a communication between it and client.

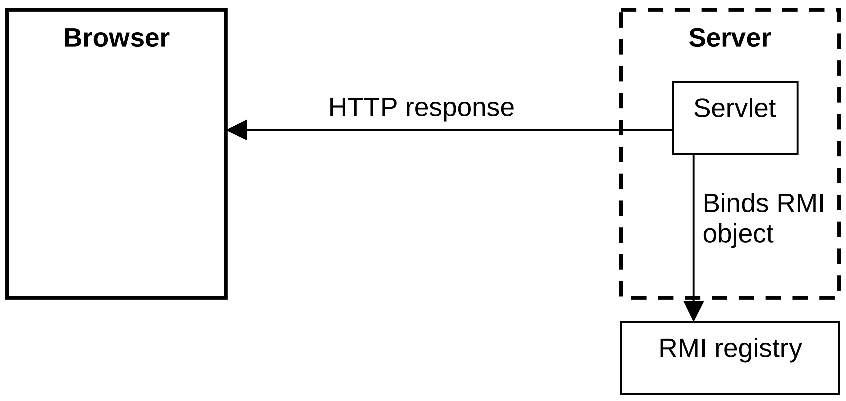
However, if our server does more than just send static HTML pages, for example, providing database access, the state of communication, or connection, between server and client becomes very important. The very simple example of this is an online search engine which displays only a certain number of found results per page. Whenever the user clicks on *Next* link the server displays the following page with results in order. If this server used our simple two-step model it would be impossible to do. In this case the server must know which state is the client is in. Thus it is crucial to maintain connection state information. There are several ways to do this and in this chapter I am going to focus on the Java Servlet – Java Applet approach.

The following scheme describes the steps in the process of establishing connection between Servlet and Applet.

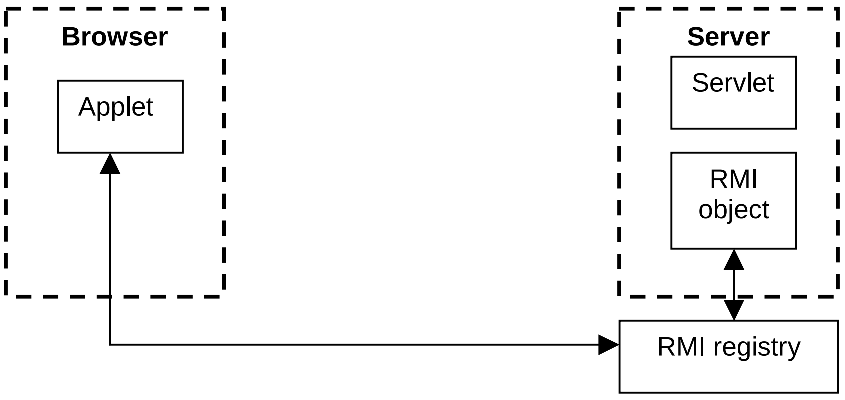
1. Browser sends an HTTP request to server. Server starts Servlet.



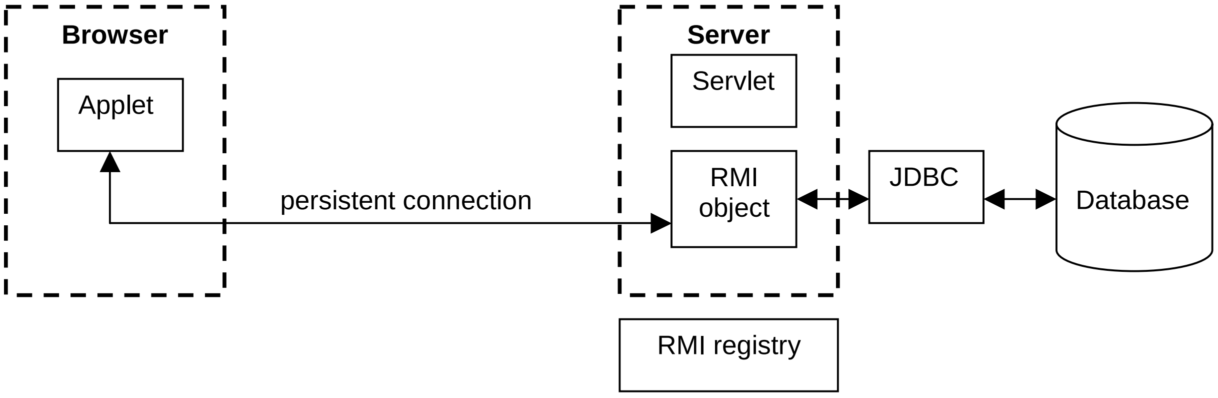
2. Servlet binds an RMI object into RMI registry and sends an HTML page with Applet tag to client.



3. Browser loads Applet, Applet connects to RMI registry and looks up the RMI object.



4. Applet invokes methods of the RMI object, the RMI object communicates with database.



As we can see at the fourth step there is persistent connection established between Applet on the client side and RMI object on the server side. Each client invokes its own instance of the RMI object so at all times there are connection states available for each of the instances of the RMI object.

Another benefit of this implementation is that connection between Applet and RMI object is not an HTTP but pure Java type. So it is very easy to transmit any type of data.

As it is clear from the above schema this implementation consists of two components - Java Applet and Servlet. In my example, I used Sun's Java Web Server 1.1.3 on the server side and Sun's Java 2 Runtime Environment Plug-in 1.3.0_02 on the client side.

The Servlet component has 5 parts:

1. *PartnerVendorServer* class
2. *PartnerVendorConnectorInt* interface class
3. *PartnerVendorConnectorImp* interface implementation class
4. *PartnerVendorRecordObject* class
5. *Java2AppletTagCodeMaker* class

All source codes for these classes are given in "Appendix A.1".

PartnerVendorServer class is a subclass of Servlet. Its primary function is to bind RMI object class *PartnerVendorConnectorImp* in RMI registry and return an HTML page with an Applet tag code to a browser. *PartnerVendorServer* uses

Java2AppletTagCodeMaker to generate this Applet tag code. Since RMI functionality and Swing interface packages used to build Applet user interface are fully supported only in Java 2 virtual machines it was necessary to use the appropriate plug-in supplied by Sun for the version of browsers that do not have Java 2 (all versions of Internet Explorer, all versions of Netscape Navigator before 6.0). *Java2AppletTagCodeMaker* determines which version of a browser a client uses and builds an applet tag accordingly.

PartnerVendorConnectorImp is an RMI object which implements methods to communicate with Access database using standard Java JDBC-ODBC bridge. Having a separate interface class, in this case *PartnerVendorConnectorInt*, which is implemented by *PartnerVendorConnectorImp* is a requirement for all RMI objects.

PartnerVendorRecordObject is a serializable class. It is used to return database records from *PartnerVendorConnectorImp* methods.

Client side Applet consists of four parts:

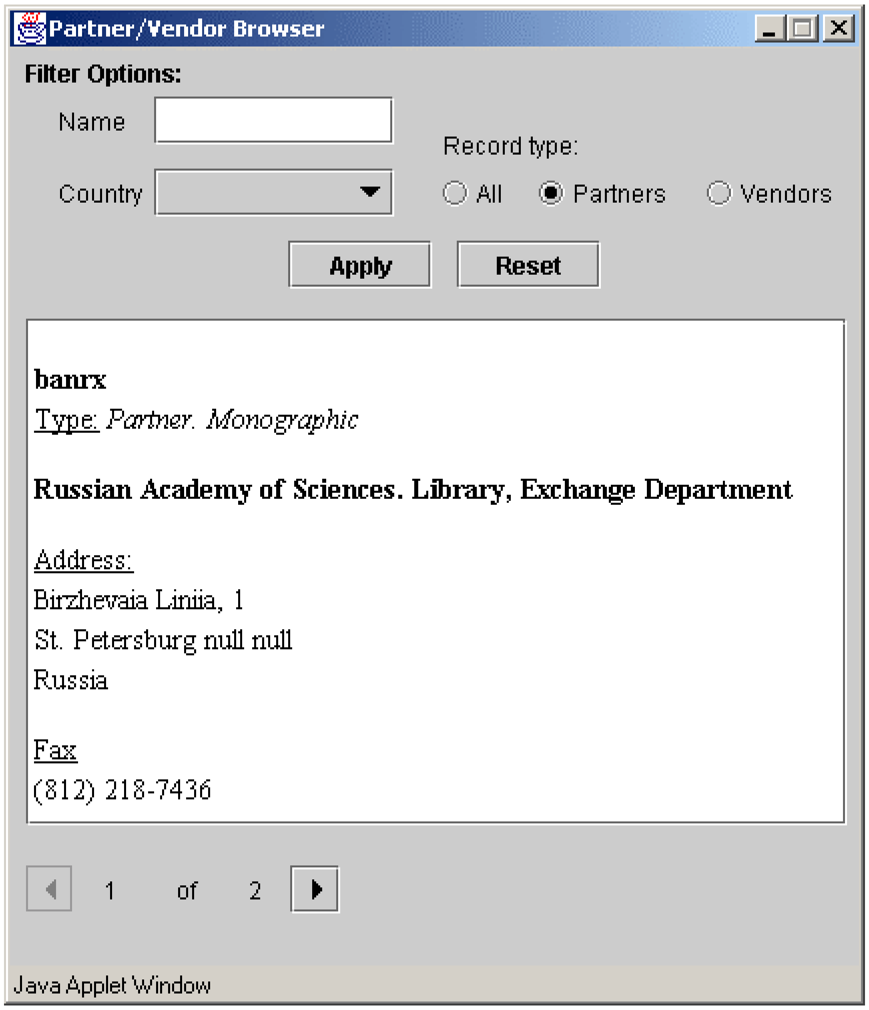
1. *PartnerVendorBrowser* class
2. *ConnectionHandler* class
3. *PartnerVendorConnectorInt* class
4. *PartnerVendorRecordObject* class

All source codes for these classes are given in "Appendix A.2".

PartnerVendorBrowser is an Applet user interface class. It handles all user interactions and displays database controls and records. In order to communicate with server this class uses *ConnectionHandler* which runs as a separate thread.

ConnectionHandler does *PartnerVendorConnectorImp* RMI object lookup and returns its interface *PartnerVendorConnectorInt*. After that *ConnectionHandler* calls *PartnerVendorConnectorImp* methods specified by *PartnerVendorConnectorInt* interface. *PartnerVendorRecordObject* is used to extract data returned by *PartnerVendorConnectorImp* methods.

There is a screenshot of *PartnerVendorBrowser* user interface on the next page.



Record type: All [✓ Partners] Vendors

banrx
Type: *Partner. Monographic*

Russian Academy of Sciences. Library, Exchange Department

Address:
Birzhevaia Liniia, 1
St. Petersburg null null
Russia

Fax
(812) 218-7436

PartnerVendorBrowser user interface screenshot

ASP TECHNOLOGY OVERVIEW

Active Server Pages (ASP) developed by Microsoft is another technology which allows developers to build online database driven systems. It is different from Servlets in a way that ASP is a scripting language approach. Instead of using compiled programs ASP adds scripting language commands right into HTML pages stored on a server. Whenever such pages are accessed server parses them first and runs scripting language commands and only after that sends processed pages to clients.

Two scripting languages are supported by ASP – Visual Basic Script and JScript (Microsoft's analog of JavaScript). This technology allows adding components written in other languages using Microsoft ActiveX architecture. Those components are compiled programs created in compliance with ActiveX standard and registered on an ASP server.

Standard ASP included with Microsoft's Internet Information Server (IIS) already comes with some built-in ActiveX components. Two of them are especially critical for creating online database driven applications: Request object and ADODB object. Request object allows server to get information about client's system and to read data sent by client. ADODB object is a database connection program.

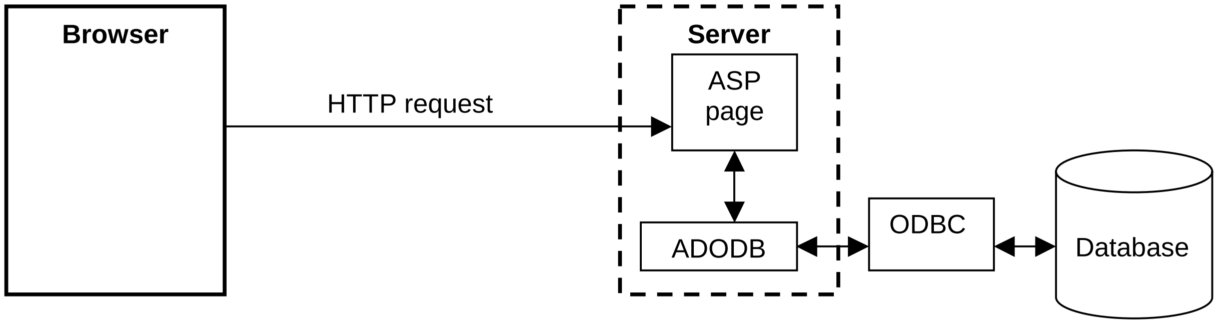
ASP IMPLEMENTATION

ASP communication model is very similar to the typical browser – server communication model described in the "Servlet Implementation" chapter. Since there is no persistent connection between client and server in this model the method of maintaining the connection state is different from Servlet implementation. The two most popular ways of maintaining connection state in ASP application are:

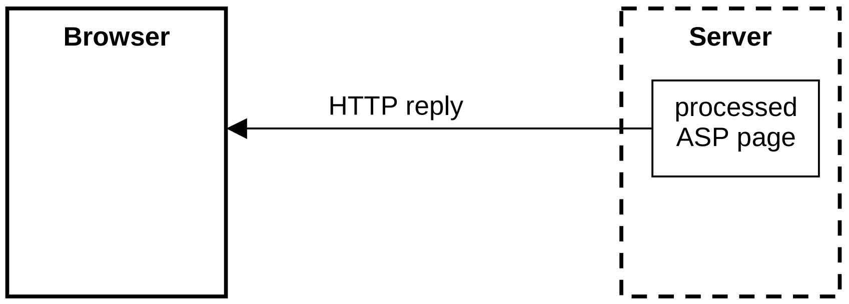
1. Cookies. ASP has a Session object which assigns a unique identification to every client which is stored as a Cookie on a client side. Developer can add variables describing connection to Session object. Server will automatically map those variables with client identifier. This Cookie exists only during browser runtime, it is not saved to client's computer.
2. HTML form fields. Server can set connection state to a hidden HTML form field tag which will be sent to client as a part of HTML page. This approach is used in my example.

Below is the application model of my ASP implementation.

1. Browser sends request to server. Server loads corresponding ASP page and processes it. While processing ASP page server reads client's form fields and determines the connection state (if no fields are available server sets default connection state values). After that server communicates with database and formats the results.



2. Server sends processed ASP page with connection state data to client.



The ASP implementation has a significantly simpler application model than Servlet implementation and consists of just three components represented as ASP pages:

1. *default.asp* – HTML frameset structure of user interface
2. *topmenu.asp* – top frame of user interface. This page sets search filter
3. *main.asp* – main frame of user interface. This page performs database connection and displays the results

I used Internet Information Server 5.0 to process my ASP pages. I used Visual Basic script as ASP scripting language. Client side can be any fourth generation or above browser.

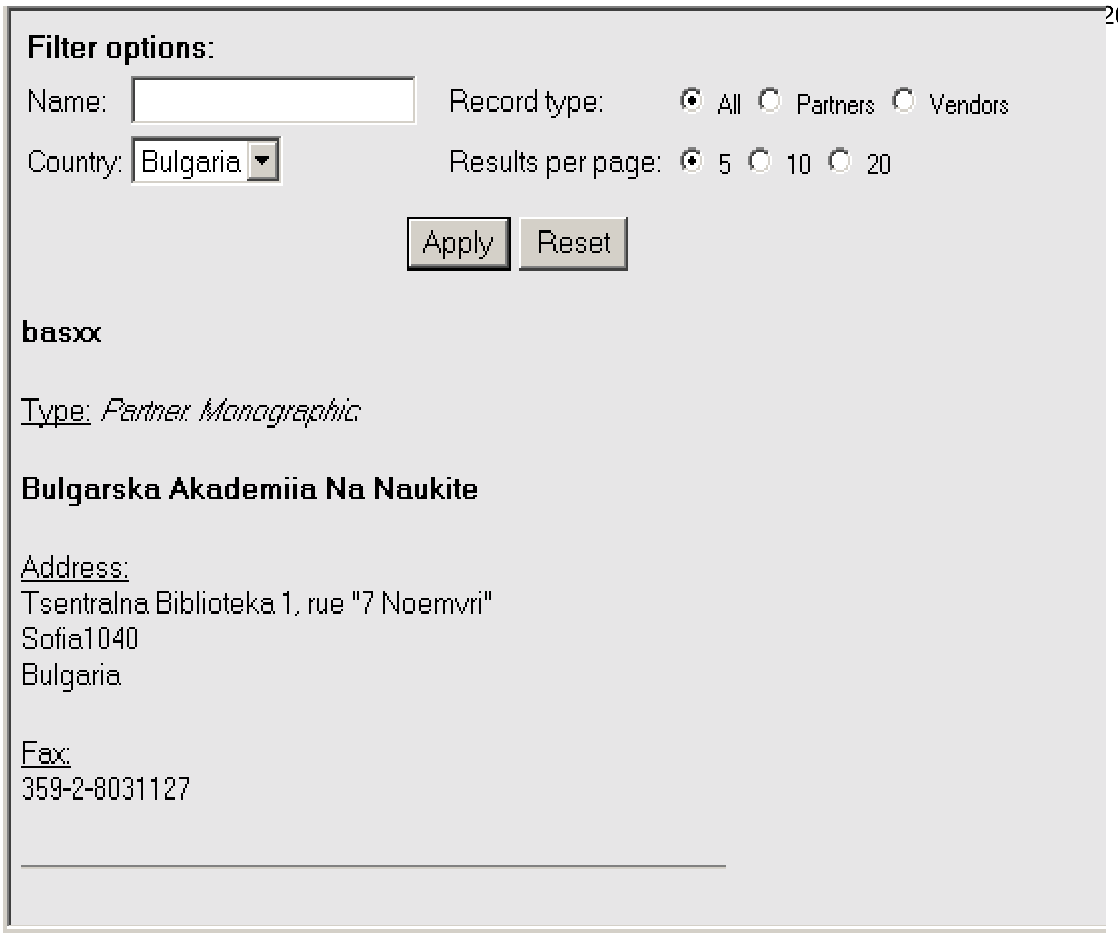
All source codes for ASP pages are given in "Appendix B".

default.asp – just defines the frame layout of user interface, no ASP script here.

topmenu.asp – uses ADODB object to connect to the database and sets search fields of application.

main.asp – uses Request object to get connection state parameters, sets connection state parameters, communicates with database and formats the results.

There is a screenshot of ASP user interface on the next page.


Filter options:

Name: Record type: [✓ All] Partners Vendors

Country: Bulgaria Results per page: [✓ 5] 10 20

basxx

Type: *Partner: Monographic*

Bulgarska Akademiia Na Naukite

Address:
Tsentralna Biblioteka 1, rue "7 Noemvri"
Sofia1040
Bulgaria

Fax:
359-2-8031127

ASP user interface

COMPARISON OF ASP AND SERVLET TECHNOLOGY

If we go back and take a look at communication model diagrams for both Servlet and ASP implementations one thing can be spotted right away: The ASP diagram is far less complex than the Servlet diagram. Developing and programming ASP application takes significantly less time than the Servlet application. The ASP learning curve is also smaller. It took me about four days to make a working ASP implementation. But with Servlet it required one week just to understand the principles behind the architecture, and then another week and a half to develop the application.

On the other hand, the Servlet approach is very object oriented. This means that the components initially developed for a project can later be used in other applications with very few or no modifications. Thus Servlet technology speeds and simplifies the development process. Since ASP uses scripting language it embeds the code right into the HTML pages which makes programming very task specific. This creates problems when someone wants to use the existing code for new applications. The code might have to be modified and this takes more time and work. In some cases with ASP it is easier to start from scratch than to modify the existing program.

From a client’s point of view, ASP implementation is "lighter". It requires only a web browser program to work. All the processing is done on the server. In comparison to ASP the Servlet approach does some processing on the client. In my implementation I used Java Applet as a browsing and search tool on the client side. This Applet calls Servlet methods via RMI; it then formats and displays the received data. This approach makes the client do some processing on his side though the most processing is done on the server. Although it is not as "lightweight" as the ASP implementation, the Servlet approach gives developers a persistent connection between client and server for free since it is a part of the architecture. Developers also do not need to worry about maintaining the connection status since Applet has an instance of Servlet object provided by RMI. The ASP approach binds developers to program connection status monitoring routines. In this case there is no persistent connection available and the client has to supply the server with its connection state information each time communication occurs.

Another useful feature of the Servlet approach is the ability of Applet to implement new functionality which otherwise would not be available in a client’s browsing program. Of course, with ASP approach developers can use a big variety of features supported by modern browsers to incorporate dynamic elements into HTML pages. Unfortunately, those features are non-standard and differ from one software manufacturer to another. Thus an HTML page that uses the additional features of Internet Explorer 5.x would not work properly in Netscape Navigator 4.x and vice versa. On the other hand Applet, being a pure Java program, runs in all of these browsers.

Table of advantages and disadvantages of the two approaches

	Servlet	**ASP**
Learning curve	*slow*	*fast*
Code reusability	*high*	*low*
Maintaining connection state	*integral part of architecture*	*has to be programmed*
Amount of processing performed on a client	*moderate*	*small*
Ability to extend standard browser functions	*high; developer can implement new functionality*	*moderate; developer is limited to functions supported by current version of browser*

In conclusion, I would recommend use of the ASP implementation in cases of small to medium size applications. For example, to provide an access to Partners/Vendors database described in this paper, ASP implementation would be the better choice. For more complex and larger applications like a library acquisition system, the Servlet approach would provide better scalability and systems support and management.

REFERENCE

1. Alwang, Greg. Java Web Server. PC Magazine (May 5, 1998) v17 n9 p187-188
2. Betz, Mark. Active data objects & ASP -- Executing scripts on a server. Dr. Dobb's Journal (May 1, 1998) v23 n5 p88-91, 111+
3. Breeding, Marshall. Perking up library applications. Java making a presence in library automation. Information Today v. 17 no11 (Dec. 2000) p. 52-3 Journal Code: Inf. Today
4. Carr, David F. As Java matures, developers focus on applications rather than hype. Internet World (June 21, 1999) v5 n23 p1, 45
5. Carr, David F. House blends -- Applets, Servlets, JavaScript - why developers mix and match. Internet World (February 1, 2000) v6 n3 p54
6. Carr, David F. The 'servlet' is emerging as the next frontier for Java. Internet World (February 2, 1998) v4 n4 p25, 27
7. Carr, David F. Too early to gauge Microsoft app server's success. Internet World (July 27, 1998) v4 n25 p28, 31
8. Chalnick, Leon. Programming for Active Server pages -- Creative dynamic content using ASP. Web Techniques (October 1, 1997) v2 n10 p55-67
9. Clip, Paul. Servlets: CGI the Java way -- These server-oriented Java classes offer system independence and better performance over CGI programs. BYTE (May 1, 1998) v23 n5 p55-56
10. Comparing Java Server Pages and Microsoft Active Server Pages Technologies. http://java.sun.com/products/jsp/jsp-asp.html
11. Cox, David. Integrating Web pages with databases -- Preprocessing HTML. Dr. Dobb's Journal (September 1, 2000) v25 n9 p94-98
12. Database-driven pages. Digital Publishing Technologies (January 1, 1999) v4 n1 p8-12
13. David Finkel, Craig E. Wills, Brian Brennan, Chris Brennan. Distriblets: Java-based distributed computing on the Web. Internet Research: Electronic Networking Applications and Policy 9, no. 1 (1999): 35
14. Dyck, Timothy. Four scripting languages speed development -- ColdFusion is the most productive choice; JSP technology offers best growth path for enterprises. eWeek (October 30, 2000) v17 n44 p28-35
15. Fiedler, David Clark, Scott. Executing files from the Web; moving to Active Server Pages. Internet World (May 4, 1998) v4 n17 p23-24
16. Hippenhammer, Craighton T., Wilhelm, Bryan. Interlibrary loan form Java programming and Direct Request. at Olivet Nazarene University. Journal of Interlibrary Loan, Document Delivery & Information Supply v. 9 no4 (1999) p. 5-13
17. Jacsó, Péter. Publishing textual databases on the Web. Java applets and CGI/DLL programs which are common gateway interfaces. Information Today v. 15 no11 (Dec. 1998) p. 33+ Journal Code: Inf Today

18. Java Servlet Technology. White Paper. http://java.sun.com/products/Servlet/whitepaper.html

19. Kochmer, Casey. JSP VS ASP Part 2: The Future According to ASP+. JSPBuzz Vol I: Issue 7 http://www.jspinsider.com/jspbuzz/oct2000/buzz_10_17_2000.html - topic

20. Lam, John. Database connectivity and the Internet -- Active Server Pages and Remote Data Service. PC Magazine (December 16, 1997) v16 n22 p249-257

21. Lam, John. Visual InterDev and ASP -- Building a dynamic Web site with Visual InterDev and Active Server Pages. PC Magazine (December 16, 1997) v16 n22 p265-272

22. Malarvannan, Mani. A multithreaded server in Java -- Building upon frameworks. Web Techniques (October 1, 1998) v3 n10 p46-51

23. Mischo, William H Schlembach, Mary C. Web-based access to locally developed databases. Library Computing (March 1, 1999) v18 n1 p51-58

24. Scott, Ralph Lee..Java and the Web. North Carolina Libraries v. 56 no1 (Spring '98) p. 39

25. Simone, Luisa. ASP easy as 1-2-3.PC Magazine (April 4, 2000) v19 n7 p50

26. Tal, Guy. Provide dynamic Web content with Java servlets -- An e-business application should be interactive, dynamic, and fast. Server-side Java is the key. e-Business Advisor (October 1, 1998) v16 n10 p48-51

27. Udell, Jon. Java servlets -- Servlets, the Java equivalent of CGI applications, can deliver on many of Java's promises while dodging some of its worst limitations. BYTE (June 1, 1997) v22 n6 p115-118

28. Wodaski, Ron. ASP Technology Feature Overview. 1998. http://msdn.microsoft.com/workshop/server/asp/aspfeat.asp

29. Zoltan, Erik. PHP, Perl, Java servlets - Which one's right for you? April 2001. http://www-106.ibm.com/developerworks/web/library/wa-sssl.html?t=gr,p=PHP-Perl-JSP

APPENDIX A.1

PartnerVendorServer.class

```
/**
 * HttpServlet subclass
 * Provides client browser with applet tag code
 * Sets RMI string parameter and binds RMI object
 * on first initialization
 */
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.rmi.*;

public class PartnerVendorServer extends HttpServlet
{
     private String host, rmiString, message;
     private int port;
     private static boolean firstInit = true;

     /**
      * Calls Post method which handles all of the operations
      */
     public void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
     {
          doPost(req, resp);
     }

     /**
      * Binds RMI object
      * Returns HTML page to client
      */
     public void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
     {
          /**
           * On first initialization:
           */
          if (firstInit)
          {
               firstInit = false;
               setHostPort(req);
               /**
                * Sets RMI string
                */
               rmiString = "//" + host +
":4444/PartnerVendorRMIObj";
```

```
                PartnerVendorConnectorImp PartnerVendorRMIObj =
new PartnerVendorConnectorImp();
                try
                {
                     /**
                      * Binds RMI object
                      * Displays message on error
                      */
                     Naming.rebind(rmiString,
PartnerVendorRMIObj);
                     message = "PartnerVendorConnectorImp bound
in RMI registry: port 4444";
                }
                catch(Exception except)
                {
                     message = "Error binding
PartnerVendorConnectorImp: " + except;
                }
           }

           /**
            * Sets output stream
            * Creates appropriate applet tag code
            */

           ServletOutputStream out;
           Java2AppletTagCodeMaker codeMaker;

           resp.setContentType("text/html");
           out = resp.getOutputStream();

           codeMaker = new Java2AppletTagCodeMaker(req,
"PartnerVendorBrowser.class", "/mpservlet/");
           codeMaker.addParameter("rmiString", rmiString);

           out.println("<html>");
           out.println("<head>");
           out.println("<title>Partner/Vendor Browser</title>");
           out.println("</head>");
           out.println("<body bgcolor=\"#ffffff\">");
           out.println("<p>" + message + "</p>");
           out.println(codeMaker.getTagCode());
           out.println("</body>");
           out.println("</html>");
           out.close();
      }

      /**
       * Sets current serve host and port
       */
      private void setHostPort(HttpServletRequest req)
      {
            host = req.getServerName();
            port = req.getServerPort();
        }
    }
```

Java2AppletTagCodeMaker.class

```
/**
 * Creates appropriate Java 2 plug-in tag
 * code based on User-Agent value
 */
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class Java2AppletTagCodeMaker
{
     private String host, codeName, codePath, userAgent;
     private int port;
     private Hashtable parameters = new Hashtable();

     /**
      * Class constructor
      */
     public Java2AppletTagCodeMaker(HttpServletRequest req,
String name, String path)
     {
          host = req.getServerName();
          port = req.getServerPort();
          userAgent = req.getHeader("User-Agent");
          codeName = name;
          codePath = path;
     }

     /**
      * Method to add parameters to applet tag code
      */
     public void addParameter(String name, String value)
     {
          parameters.put(name,value);
     }

     /**
      * Returns tag code
      */
     public String getTagCode()
     {
          String tagCode, tmp;
          String codeBase = "http://" + host + ":" +
Integer.toString(port) + codePath;
          Enumeration enum = parameters.keys();

          /**
           * Tag code for Internet Explorer
           */
          if (userAgent.indexOf("MSIE") > -1)
          {
                tagCode = "<object classid=\"clsid:8AD9C840-044E-
11D1-B3E9-00805F499D93\" width=0 height=0
codebase=\"http://java.sun.com/products/plugin/1.3/jinstall-13-
win32.cab#Version=1,3,0,0\">\n";
                tagCode += "<param name=\"code\" value=\"" +
codeName + "\">\n";
                tagCode += "<param name=\"codebase\" value=\"" +
codeBase + "\">\n";
                tagCode += "<param name=\"type\"
value=\"application/x-java-applet;version=1.3\">\n";
                tagCode += "<param name=\"scriptable\"
value=\"false\">\n";
                while(enum.hasMoreElements())
                {
                     tmp = (String) enum.nextElement();
                     tagCode += "<param name=\"" + tmp + "\"
value=\"" + (String) parameters.get(tmp) + "\">\n";
                }
                tagCode += "</object>";
           }
           /**
            * Tag code for Netscape 4.x
            */
           else if ((userAgent.indexOf("Mozilla") > -1) &&
(userAgent.indexOf("Netscape6") == -1))
           {
                tagCode = "<embed type=\"application/x-java-
applet;version=1.3\" code=\"" + codeName + "\" codebase=\"" +
codeBase + "\" width=0 height=0";
                while(enum.hasMoreElements())
                {
                     tmp = (String) enum.nextElement();
                     tagCode += " " + tmp + "=\"" + (String)
parameters.get(tmp) + "\"";
                }
                tagCode += " scriptable=\"false\"
pluginspage=\"http://java.sun.com/products/plugin/1.3/plugin-
install.html\">\n";
                tagCode += "</embed>";
           }
           /**
            * Tag code for Netscape 6
            */
           else
           {
                tagCode = "<applet code=\"" + codeName + "\"
codebase=\"" + codeBase + "\" width=0 height=0>\n";
                while(enum.hasMoreElements())
                {
                     tmp = (String) enum.nextElement();
                     tagCode += "<param name=\"" + tmp + "\"
value=\""+ (String) parameters.get(tmp) + "\">\n";
```

```
                        }
                        tagCode += "</applet>";
                }
                return tagCode;
        }
}
```

PartnerVendorConnectorInt.class

```
/**
 * Interface class for PartnerVendorConnectorImp
 * RMI object
 */
import java.rmi.*;
import java.sql.*;

public interface PartnerVendorConnectorInt extends Remote
{
     void startConnection() throws RemoteException, SQLException,
ClassNotFoundException, Exception;
     boolean hasNextRecord() throws RemoteException;
     boolean hasPreviousRecord() throws RemoteException;
     PartnerVendorRecordObject nextRecord() throws
RemoteException, SQLException;
     PartnerVendorRecordObject previousRecord() throws
RemoteException, SQLException;
     int getCurrentPosition() throws  RemoteException;
     int getTotalRecords() throws  RemoteException;
     boolean search(String sqlString) throws  RemoteException,
SQLException;
     String[] getCountryNames() throws RemoteException,
SQLException;
}
```

PartnerVendorConnectorImp.class

```
/**
 * RMI object implementation class
 * Uses JdbcOdbcDriver to connect to Access database
 */
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.io.*;
import java.sql.*;
import java.util.*;

public class PartnerVendorConnectorImp extends
UnicastRemoteObject implements PartnerVendorConnectorInt
{
     /**
      * JDBS connection object
      */
     private Connection dbConnection;

     /**
      * Vector to store record sets
      */
     private Vector RecordSet = new Vector(30);

     private int totalRecords = 0, currentPosition = 0;
     private static final int GET_NEXT = 0, GET_PREVIOUS = 1;

     /**
      * Class constructor
      */
     public PartnerVendorConnectorImp() throws RemoteException
     {
          super();
     }

     /**
      * Staring new connection
      */
     public void startConnection() throws RemoteException,
SQLException, ClassNotFoundException, Exception
     {
          Statement dbStatement;
          ResultSet dbResultSet;


     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
          dbConnection =
DriverManager.getConnection("jdbc:odbc:PartnersVendors");
          dbStatement = dbConnection.createStatement();
          dbResultSet = dbStatement.executeQuery("SELECT * FROM
PartnersVendors ORDER BY PCODE;");
```

```
			populateRecordSet(dbResultSet);
			dbResultSet.close();
			dbStatement.close();
			setTotalRecords();
		}

		/**
		 * Checks if next record exists
		 */
		public boolean hasNextRecord() throws RemoteException
		{
			if (currentPosition < RecordSet.size())    return
	true;
			else return false;
		}

		/**
		 * Checks if previous record exists
		 */
		public boolean hasPreviousRecord() throws RemoteException
		{
			if (currentPosition > 1) return true;
			else return false;
		}

		/**
		 * Returns current record index
		 */
		public int getCurrentPosition() throws RemoteException
		{
			return currentPosition;
		}

		/**
		 * Returns total number of records
		 */
		public int getTotalRecords() throws RemoteException
		{
			return totalRecords;
		}

		/**
		 * Gets next record
		 */
		public PartnerVendorRecordObject nextRecord() throws
	RemoteException, SQLException
		{
			return getRecord(GET_NEXT);
		}

		/**
		 * Gets previous record
```

```
       */
      public PartnerVendorRecordObject previousRecord() throws
RemoteException, SQLException
      {
            return getRecord(GET_PREVIOUS);
      }

      /**
       * Searches based on the SQL string
       * Populates record set Vector with records from database
       * Returns true if results were found,
       * false otherwise
       */
      public boolean search(String sqlString) throws
RemoteException, SQLException
      {
            Statement dbStatement;
            ResultSet dbResultSet;

            dbStatement = dbConnection.createStatement();
            dbResultSet = dbStatement.executeQuery(sqlString);
            populateRecordSet(dbResultSet);
            dbResultSet.close();
            dbStatement.close();
            currentPosition = 0;
            setTotalRecords();

            if (totalRecords > 0) return true;
            else return false;
      }

      /**
       * Returns String array of unique contry names
       * in database
       */
      public String[] getCountryNames() throws RemoteException,
SQLException
      {
            Statement dbStatementTemp;
            ResultSet dbResultSetTemp;
            Vector resultVector = new Vector();
            String[] result;

            dbStatementTemp = dbConnection.createStatement();
            dbResultSetTemp = dbStatementTemp.executeQuery("SELECT
DISTINCT COUNTRY FROM PartnersVendors ORDER BY COUNTRY;");

            while(dbResultSetTemp.next())
            {

      resultVector.addElement(dbResultSetTemp.getString("COUNTRY")
);
```

```
          }

          dbResultSetTemp.close();
          dbStatementTemp.close();

          result = new String[resultVector.size()];
          resultVector.copyInto(result);

          return result;
     }

     /**
      * Sets total number of records in record set
      */
     private void setTotalRecords()
     {
          totalRecords = RecordSet.size();
     }

     /**
      * Returns record from record set based on actionCode
      */
     private PartnerVendorRecordObject getRecord(int actionCode)
     {
          if (actionCode == GET_NEXT) currentPosition++;
          else currentPosition--;

          return (PartnerVendorRecordObject)
RecordSet.elementAt(currentPosition-1);
     }

     /**
      * Populates record set Vector with records from database
      *
      */
     private void populateRecordSet(ResultSet records) throws
SQLException
     {
          Statement dbStatementTemp;
          ResultSet dbResultSetTemp;
          String pcode;
          PartnerVendorRecordObject record;

          if (RecordSet.size() > 0)
RecordSet.removeAllElements();

          while(records.next())
          {
               record = new PartnerVendorRecordObject();
               pcode = records.getString("PCODE");
               record.setPcode(pcode);
               record.setType(records.getInt("TYPE"));
```

```
                /**
                 * Checks record type: Partner 0 or Vendor 1
                 */
                if (record.getType() == 0)
                {

                     /**
                      * Gets partner type
                      */
                     dbStatementTemp =
dbConnection.createStatement();
                     dbResultSetTemp =
dbStatementTemp.executeQuery("SELECT PTYPE FROM PartnerTypes
WHERE PCODE = ’" + pcode + "’;");
                     dbResultSetTemp.next();

     record.setPtype(dbResultSetTemp.getInt("PTYPE"));
                     dbResultSetTemp.close();
                     dbStatementTemp.close();
                }
                else
                {
                     /**
                      * Gets vendor profile if there is one
                      */
                     dbStatementTemp =
dbConnection.createStatement();
                     dbResultSetTemp =
dbStatementTemp.executeQuery("SELECT PROFILE FROM VendorProfiles
WHERE PCODE = ’" + pcode + "’;");
                     if (dbResultSetTemp.next())
record.setProfile(dbResultSetTemp.getString("PROFILE"));
                     dbResultSetTemp.close();
                     dbStatementTemp.close();
                }

                record.setName(records.getString("NAME"));
                record.setStreet(records.getString("STREET"));
                record.setCity(records.getString("CITY"));
                record.setState(records.getString("STATE"));
                record.setCountry(records.getString("COUNTRY"));
                record.setPhone1(records.getString("PHONE1"));
                record.setPhone2(records.getString("PHONE2"));
                record.setFax(records.getString("FAX"));
                record.setEmail1(records.getString("EMAIL1"));
                record.setEmail2(records.getString("EMAIL2"));
                record.setCperson(records.getString("CPERSON"));

                /**
                 * Gets Notes if there are ones
                 */
                dbStatementTemp = dbConnection.createStatement();
                dbResultSetTemp =
dbStatementTemp.executeQuery("SELECT NOTES FROM
PartnerVendorNotes WHERE PCODE = ’" + pcode + "’;");
                if (dbResultSetTemp.next())
record.setNotes(dbResultSetTemp.getString("NOTES"));
                dbResultSetTemp.close();
                dbStatementTemp.close();

                RecordSet.addElement(record);
           }
      }
}
```

PartnerVendorRecordObject.class

```
/**
 * Serializable class to store and
 * to transmit records from database
 */

import java.io.*;

public class PartnerVendorRecordObject extends Object implements
Serializable
{
     private String pcode;
     private int type;
     private int ptype;
     private String name;
     private String street;
     private String city;
     private String state;
     private String zip;
     private String country;
     private String phone1;
     private String phone2;
     private String fax;
     private String email1;
     private String email2;
     private String cperson;
     private String notes;
     private String profile;

     public PartnerVendorRecordObject()
     {
          pcode = null;
          type = -1;
          ptype = -1;
          name = null;
          street = null;
          city = null;
          state = null;
          zip = null;
          country = null;
          phone1 = null;
          phone2 = null;
          fax = null;
          email1 = null;
          email2 = null;
          cperson = null;
          notes = null;
          profile = null;
     }

     public void setPcode(String s)
```

```
{
     pcode = s;
}

public String getPcode()
{
     return pcode;
}

public void setType(int i)
{
     type = i;
}

public int getType()
{
     return type;
}

public void setPtype(int i)
{
     ptype = i;
}

public int getPtype()
{
     return ptype;
}

public void setName(String s)
{
     name = s;
}

public String getName()
{
     return name;
}

public void setStreet(String s)
{
     street = s;
}

public String getStreet()
{
     return street;
}

public void setCity(String s)
{
     city = s;
}

public String getCity()
{
     return city;
}

public void setState(String s)
{
     state = s;
}

public String getState()
{
     return state;
}

public void setZip(String s)
{
     zip = s;
}

public String getZip()
{
     return zip;
}

public void setCountry(String s)
{
     country = s;
}

public String getCountry()
{
     return country;
}

public void setPhone1(String s)
{
     phone1 = s;
}

public String getPhone1()
{
     return phone1;
}

public void setPhone2(String s)
{
     phone2 = s;
}
```

```
public String getPhone2()
{
     return phone2;
}

public void setFax(String s)
{
     fax = s;
}

public String getFax()
{
     return fax;
}

public void setEmail1(String s)
{
     email1 = s;
}

public String getEmail1()
{
     return email1;
}

public void setEmail2(String s)
{
     email2 = s;
}

public String getEmail2()
{
     return email2;
}

public void setCperson(String s)
{
     cperson = s;
}

public String getCperson()
{
     return cperson;
}

public void setProfile(String s)
{
     profile = s;
}

public String getProfile()
{
```

```
            return profile;
        }

        public void setNotes(String s)
        {
            notes = s;
        }

        public String getNotes()
        {
            return notes;
        }
}
```

APPENDIX A.2

PartnerVendorBrowser.class

```
/**
 * Applet GUI class. Uses ConnectionHandler.class
 * implemented as a separate Thread to communicate
 * with server.
 * Gets RMI parameter from web page.
 */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.*;

public class PartnerVendorBrowser extends JApplet implements
ActionListener, WindowListener
{
     /**
      * GUI components
      */
     private JFrame mainFrame;
     private JLabel labelFilter, labelName, labelCountry,
labelRecordType, labelCurrentPosition, labelOf, labelTotal;
     private JTextField textFieldName;
     private JComboBox comboBoxCountry;
     private JRadioButton radioButtonAll, radioButtonPartners,
radioButtonVendors;
     private JButton buttonApply, buttonReset, buttonPrevious,
buttonNext;
     private JEditorPane browsePane;
     private ImageIcon iconPrevious, iconNext;

     /**
      * Connection object
      */
     private ConnectionHandler handler;

     /**
      * Gets RMI parameter from web page
      * Starts connection with server
      */
     public void init()
     {
          createDialog();
          handler = new ConnectionHandler(this,
getParameter("rmiString"));
          handler.setActionCode(handler.ACTION_START);
          handler.run();
}

/**
 * Method to set text in browsePain object
 * browsePain is used to display records from database
 */
public void setBrowsePane(String s)
{
     browsePane.setText(s);
}

/**
 * Method to enable/disable buttonPrevious
 */
public void setButtonPrevious(boolean b)
{
     buttonPrevious.setEnabled(b);
}

/**
 * Method to enable/disable buttonNext
 */
public void setButtonNext(boolean b)
{
     buttonNext.setEnabled(b);
}

/**
 * Returns true if radioButtonAll is selected
 */
public boolean isAllSelected()
{
     return radioButtonAll.isSelected();
}

/**
 * Selects/deselects radioButtonAll
 */
public void setAll(boolean b)
{
     radioButtonAll.setSelected(b);
}

/**
 * Returns true if radioButtonPartners is selected
 */
public boolean isPartnersSelected()
{
     return radioButtonPartners.isSelected();
}

/**
```

```
 * Selects/deselects radioButtonPartners
 */
public void setPartners(boolean b)
{
     radioButtonPartners.setSelected(b);
}

/**
 * Returns true if radioButtonVendors is selected
 */
public boolean isVendorsSelected()
{
     return radioButtonVendors.isSelected();
}

/**
 * Selectes/deselects radioButtonVendors
 */
public void setVendors(boolean b)
{
     radioButtonVendors.setSelected(b);
}

/**
 * Returns value of textFieldName
 */
public String getNameField()
{
     return textFieldName.getText();
}

/**
 * Sets value of textFieldName
 */
public void setNameField(String s)
{
     textFieldName.setText(s);
}

/**
 * Returns selected value of comboBoxCountry
 */
public String getCountryName()
{
     return (String) comboBoxCountry.getSelectedItem();
}

/**
 * Sets value of comboBoxCountry based on index
 */
public void selectCountryName(int i)
{
        comboBoxCountry.setSelectedIndex(i);
    }

    /**
     * Sets comboBoxCountry values from String array
     */
    public void setCountryNames(String[] names)
    {
        int i;

        comboBoxCountry.addItem("");
        for(i=0;i<names.length;i++)
        {
            comboBoxCountry.addItem(names[i]);
        }
    }

    /**
     * Sets labelCurrentPosition value
     */
    public void setCurrentPosition(String s)
    {
        labelCurrentPosition.setText(s);
    }

    /**
     * Sets labelTotal value
     */
    public void setTotal(String s)
    {
        labelTotal.setText(s);
    }

    /**
     * Creates user interface dialog
     */
    private void createDialog()
    {
        mainFrame = new JFrame();

        Container mainContainer = mainFrame.getContentPane();

        String codebase = getParameter("codebase");

        JPanel panelNorth = new JPanel();
        panelNorth.setLayout(null);
        panelNorth.setBounds(0,0,420,130);

        mainFrame.setSize(430,500);
        mainFrame.setTitle("Partner/Vendor Browser");
        mainFrame.setResizable(false);
        mainFrame.setLocation(150,150);
        mainFrame.addWindowListener(this);

        labelFilter = new JLabel("Filter Options:");
        labelFilter.setBounds(8,0,84,24);
        labelFilter.setFont(new Font("Dialog", Font.BOLD,
12));
        labelFilter.setForeground(Color.black);
        panelNorth.add(labelFilter);

        labelName = new JLabel("Name");
        labelName.setBounds(24,24,48,24);
        labelName.setFont(new Font("Dialog", Font.PLAIN, 12));
        labelName.setForeground(Color.black);
        panelNorth.add(labelName);

        textFieldName = new JTextField();
        textFieldName.setBounds(72,24,120,24);
        panelNorth.add(textFieldName);

        labelCountry = new JLabel("Country");
        labelCountry.setBounds(24,60,48,24);
        labelCountry.setFont(new Font("Dialog", Font.PLAIN,
12));
        labelCountry.setForeground(Color.black);
        panelNorth.add(labelCountry);

        comboBoxCountry = new JComboBox();
        comboBoxCountry.setBounds(72,60,120,24);
        panelNorth.add(comboBoxCountry);

        labelRecordType = new JLabel("Record type:");
        labelRecordType.setBounds(216,36,72,24);
        labelRecordType.setFont(new Font("Dialog", Font.PLAIN,
12));
        labelRecordType.setForeground(Color.black);
        panelNorth.add(labelRecordType);

        radioButtonAll = new JRadioButton("All");
        radioButtonAll.setFont(new Font("Dialog", Font.PLAIN,
12));
        radioButtonAll.setBounds(216,60,36,24);
        radioButtonAll.setSelected(true);
        radioButtonAll.addActionListener(this);
        panelNorth.add(radioButtonAll);

        radioButtonPartners = new JRadioButton("Partners");
        radioButtonPartners.setFont(new Font("Dialog",
Font.PLAIN, 12));
        radioButtonPartners.setBounds(264,60,72,24);
        radioButtonPartners.addActionListener(this);
        panelNorth.add(radioButtonPartners);
```

```
          radioButtonVendors = new JRadioButton("Vendors");

          radioButtonVendors.setFont(new Font("Dialog",
Font.PLAIN, 12));
          radioButtonVendors.setBounds(348,60,72,24);
          radioButtonVendors.addActionListener(this);
          panelNorth.add(radioButtonVendors);

          buttonApply = new JButton("Apply");
          buttonApply.setBounds(139,96,72,24);
          buttonApply.addActionListener(this);
          panelNorth.add(buttonApply);

          buttonReset = new JButton("Reset");
          buttonReset.addActionListener(this);
          buttonReset.setBounds(223,96,72,24);
          panelNorth.add(buttonReset);

          browsePane = new JEditorPane();
          browsePane.setContentType("text/html");
          browsePane.setEditable(false);

          JScrollPane centerScrollPane = new JScrollPane();
          centerScrollPane.setBounds(8,135,410,253);
          centerScrollPane.getViewport().add(browsePane);

          JPanel panelSouth = new JPanel();
          panelSouth.setLayout(null);
          panelSouth.setBounds(8,408,410,39);

          try
          {
               iconPrevious = new ImageIcon(new
URL(codebase+"previous.gif"));
          }
          catch(Exception except)
          {
          }

          buttonPrevious = new JButton(iconPrevious);
          buttonPrevious.setBounds(0,0,24,24);
          buttonPrevious.addActionListener(this);
          panelSouth.add(buttonPrevious);

          labelCurrentPosition = new JLabel();

     labelCurrentPosition.setHorizontalAlignment(SwingConstants.C
ENTER);
          labelCurrentPosition.setForeground(Color.black);
          labelCurrentPosition.setFont(new Font("Dialog",
Font.PLAIN, 12));
          labelCurrentPosition.setBounds(24,0,36,24);
          panelSouth.add(labelCurrentPosition);

          labelOf = new JLabel("of");
          labelOf.setHorizontalAlignment(SwingConstants.CENTER);
          labelOf.setForeground(Color.black);
          labelOf.setFont(new Font("Dialog", Font.PLAIN, 12));
          labelOf.setBounds(72,0,16,24);
          panelSouth.add(labelOf);

          labelTotal = new JLabel();

     labelTotal.setHorizontalAlignment(SwingConstants.CENTER);
          labelTotal.setForeground(Color.black);
          labelTotal.setFont(new Font("Dialog", Font.PLAIN,
12));
          labelTotal.setBounds(96,0,36,24);
          panelSouth.add(labelTotal);

          try
          {
               iconNext = new ImageIcon(new
URL(codebase+"next.gif"));
          }
          catch(Exception except)
          {
          }

          buttonNext = new JButton(iconNext);
          buttonNext.setBounds(132,0,25,24);
          buttonNext.addActionListener(this);
          panelSouth.add(buttonNext);

          mainContainer.setLayout(null);
          mainContainer.add(panelNorth);
          mainContainer.add(centerScrollPane);
          mainContainer.add(panelSouth);
          mainFrame.setVisible(true);
     }

     /**
      * Checks user action and performs operations
      * based on it
      */
     public void actionPerformed(ActionEvent e)
     {
          /**
           * Sets radioButtonAll behavior
           */
          if (e.getSource() == radioButtonAll)
          {
               radioButtonAll.setSelected(true);
               radioButtonPartners.setSelected(false);
```

```
                radioButtonVendors.setSelected(false);
           }
           /**
            * Sets radioButtonPartners behavior
            */
           if (e.getSource() == radioButtonPartners)
           {
                radioButtonPartners.setSelected(true);
                radioButtonAll.setSelected(false);
                radioButtonVendors.setSelected(false);
           }
           /**
            * Sets radioButtonVendors behavior
            */
           if (e.getSource() == radioButtonVendors)
           {
                radioButtonVendors.setSelected(true);
                radioButtonAll.setSelected(false);
                radioButtonPartners.setSelected(false);
           }
           /**
            * Runs search filter
            */
           if (e.getSource() == buttonApply)
           {
                handler.setActionCode(handler.ACTION_SEARCH);
                handler.run();
           }
           /**
            * Resets search filter
            */
           if (e.getSource() == buttonReset)
           {
                handler.setActionCode(handler.ACTION_RESET);
                handler.run();
           }
           /**
            * Displays next record
            */
           if (e.getSource() == buttonNext)
           {
                handler.setActionCode(handler.ACTION_GETNEXT);
                handler.run();
           }
           /**
            * Displays previous record
            */
           if (e.getSource() == buttonPrevious)
           {

      handler.setActionCode(handler.ACTION_GETPREVIOUS);
                handler.run();
```

```
		}

	}

	public void windowActivated(WindowEvent e)
	{
	}

	public void windowClosed(WindowEvent e)
	{
	}

	/**
	 * Close dialog window and exits
	 */
	public void windowClosing(WindowEvent e)
	{
		mainFrame.dispose();
		System.exit(0);
	}

	public void windowDeactivated(WindowEvent e)
	{
	}

	public void windowDeiconified(WindowEvent e)
	{
	}

	public void windowIconified(WindowEvent e)
	{
	}

	public void windowOpened(WindowEvent e)
	{
	}
}
```

ConnectionHandler.class

```
/**
 * RMI connection handling class. Implemented as a separate
 * Thread
 */
import java.rmi.*;
import java.awt.*;
import javax.swing.*;
import java.sql.*;

public class ConnectionHandler extends Thread
{
     private PartnerVendorBrowser source;

     /**
      * Instance of RMI object
      */
     private PartnerVendorConnectorInt dbConnector;

     private int actionCode = 0;

     /**
      * Action codes
      */
     public static final int ACTION_START = 0;
     public static final int ACTION_GETNEXT = 1;
     public static final int ACTION_GETPREVIOUS = 2;
     public static final int ACTION_SEARCH = 3;
     public static final int ACTION_RESET = 4;

     /**
      * Class constructor
      * Connects to RMI registry, binds rmiString to interface
      */
     public ConnectionHandler(PartnerVendorBrowser obj, String
rmiString)
     {
          super();
          source = obj;

          try
          {
               dbConnector = (PartnerVendorConnectorInt)
Naming.lookup(rmiString);
          }
          catch(Exception except)
          {
               new
JOptionPane().showMessageDialog(source,"Error:
"+except,"Error",(new JOptionPane().ERROR_MESSAGE));
          }
	}

	/**
	 * Sets action code
	 */
	public void setActionCode(int i)
	{
		actionCode = i;
	}

	/**
	 * Thread run method. Performs actions based
	 * on the value of actionCode
	 */
	public void run()
	{
		switch(actionCode)
		{
			/**
			 * Start new connection
			 */
			case 0:
				try
				{
					dbConnector.startConnection();
					if (dbConnector.hasNextRecord())
					{
						getNextRecord();

	source.setCountryNames(dbConnector.getCountryNames());
					}
					else new
JOptionPane().showMessageDialog(source,"Database is
empty","Error",(new JOptionPane().INFORMATION_MESSAGE));
				}
				catch(Exception except)
				{
					new
JOptionPane().showMessageDialog(source,"Error:
"+except,"Error",(new JOptionPane().ERROR_MESSAGE));
				}
				break;
			/**
			 * Get next record
			 */
			case 1:
				getNextRecord();
				break;
			/**
			 * Get previous record
			 */
			case 2:
```

```
                    getPreviousRecord();
                    break;
               /**
                * Search, do not reset
                */
               case 3:
                    search(false);
                    break;
               /**
                * Reset
                */
               case 4:
                    search(true);
                    break;
          }
     }

     /**
      * Gets next record
      * First checks if there is one available
      * Displays message on error
      */
     private void getNextRecord()
     {
          try
          {
               setDialogFromRecord(dbConnector.nextRecord());
               setNavButtons();
          }
          catch(Exception except)
          {
               new
JOptionPane().showMessageDialog(source,"Error:
"+except,"Error",(new JOptionPane().ERROR_MESSAGE));
          }
     }

     /**
      * Gets previous record
      * First checks if there is one available
      * Displays message on error
      */
     private void getPreviousRecord()
     {
          try
          {

     setDialogFromRecord(dbConnector.previousRecord());
               setNavButtons();
          }
          catch(Exception except)
          {
                new
JOptionPane().showMessageDialog(source,"Error:
"+except,"Error",(new JOptionPane().ERROR_MESSAGE));
            }
      }

      /**
       * Gets record values from PartnerVendorRecordObject,
formats them
       * and sets browsePane in PartnerVendorBrowser.class
       */
      private void setDialogFromRecord(PartnerVendorRecordObject
record)
      {
            String tmp;

            tmp = "<p><b>" + record.getPcode() + "</b><br>\n";
            tmp += "<u>Type:</u> <i>";
            switch(record.getType())
            {
                  case 0:
                        tmp += "Partner. ";
                        switch(record.getPtype())
                        {
                              case 0:
                                    tmp += "Monographic";
                                    break;
                              case 1:
                                    tmp += "Serial";
                                    break;
                              case 2:
                                    tmp += "Serial and Monographic";
                                    break;
                        }
                        break;
                  case 1:
                        tmp += "Vendor";
                        break;
            }
            tmp += "</i></p>\n";
            tmp += "<p><b>" + record.getName() + "</b></p>\n";
            tmp += "<p><u>Address:</u><br>\n";
            tmp += record.getStreet() + "<br>\n";
            tmp += record.getCity() + " " + record.getState() + "
" + record.getZip() + "<br>\n";
            tmp += record.getCountry() + "</p>\n";

            if (record.getPhone1() != null || record.getPhone2()
!= null)
            {
                  tmp += "<p><u>Phone:</u><br>\n";
                  if (record.getPhone1() != null)
                {
                     tmp += record.getPhone1();
                     if (record.getPhone2() != null) tmp +=
"<br>\n";
                }
                if (record.getPhone2() != null) tmp +=
record.getPhone2();
                tmp += "</p>\n";
           }

           if (record.getFax() != null)
           {
                tmp += "<p><u>Fax</u><br>\n";
                tmp += record.getFax() + "</p>\n";
           }

           if (record.getEmail1() != null || record.getEmail2()
!= null)
           {
                tmp += "<p><u>Email:</u><br>\n";
                if (record.getEmail1() != null)
                {
                     tmp += record.getEmail1();
                     if (record.getEmail2() != null) tmp +=
"<br>\n";
                }
                if (record.getEmail2() != null) tmp +=
record.getPhone2();
                tmp += "</p>\n";
           }

           if (record.getCperson() != null)
           {
                tmp += "<p><u>Contact person</u><br>\n";
                tmp += record.getCperson() + "</p>\n";
           }

           if (record.getProfile() != null)
           {
                tmp += "<p><u>Profile</u><br>\n";
                tmp += record.getProfile() + "</p>\n";
           }

           if (record.getNotes() != null)
           {
                tmp += "<p><u>Notes</u><br>\n";
                tmp += record.getNotes() + "</p>\n";
           }

           source.setBrowsePane(tmp);
      }
```

```
     /**
      * Enables/disables previous and next buttons and
      * sets values of current position and total number of
records
      * in PartnerVendorBrowser.class
      */
     private void setNavButtons() throws Exception
     {

     source.setButtonPrevious(dbConnector.hasPreviousRecord());
          source.setButtonNext(dbConnector.hasNextRecord());

     source.setCurrentPosition(Integer.toString(dbConnector.getCu
rrentPosition()));

     source.setTotal(Integer.toString(dbConnector.getTotalRecords
()));
     }

     /**
      * Performs search; resets on true
      * Displays message on error
      */
     private void search(boolean reset)
     {
          String sSQL;
          boolean ok;

          sSQL = "SELECT * FROM PartnersVendors WHERE ";

          if (reset)
          {
               source.setNameField("");
               source.selectCountryName(0);
               source.setAll(true);
               source.setPartners(false);
               source.setVendors(false);
               source.setBrowsePane("");
          }

          if (source.isVendorsSelected()) sSQL += "TYPE = 1";
          else if (source.isPartnersSelected()) sSQL += "TYPE =
0";
          else sSQL += "(TYPE = 0 OR TYPE = 1)";

          if (source.getNameField() != "") sSQL += " AND NAME
LIKE ’%" + source.getNameField() + "%’";
          if (source.getCountryName() != "") sSQL += " AND
COUNTRY = ’" + source.getCountryName() + "’";

          sSQL += " ORDER BY PCODE;";
```

```
            try
            {
                  ok = dbConnector.search(sSQL);
                  setNavButtons();
                  if (!ok)
                  {
                        source.setBrowsePane("");
                        new
JOptionPane().showMessageDialog(source,"No records
found","Information",(new JOptionPane().INFORMATION_MESSAGE));
                  }
                  else getNextRecord();
            }
            catch(Exception except)
            {
                  new
JOptionPane().showMessageDialog(source,"Error:
"+except,"Error",(new JOptionPane().ERROR_MESSAGE));
            }
      }
}
```

PartnerVendorRecordObject.class

Same as in "Appendix A.1"

PartnerVendorConnectorInt.class

Same as in "Appendix A.1"

APPENDIX B

default.asp

```
<html>
<head>
<title>Database Access using ASP</title>
<!--
          Defines frame layout
-->
</head>
<frameset rows="130,*" border="0" frameborder="0"
framespacing="0">
<frame name="topMenu" src="topmenu.asp" scrolling="no" noresize
marginheight="0" marginwidth="0">
<frame name="main" src="main.asp" scrolling="yes" noresize
marginheight="0" marginwidth="0">
</frameset>
</html>
```

topmenu.asp

```
<html>
<head>
<!--
          Filter menu.
          Connects to database to get a list of unique country
names
-->
<link type="text/css" rel="stylesheet" href="styles/main.css">
<script language="javascript">
<!--
//Submits FIlterForm form
//Reset form on Reset button

function SubmitForm(v) {
     if (v == "Reset") document.FilterForm.reset();
     document.FilterForm.submit();
}
//-->
</script>
</head>
<body bgcolor="#ffffff">
<!--
          Form returns results in main frame
-->
<form name="FilterForm" action="main.asp" method="post"
target="main">
<table border=0>
<tr><td colspan=4><span class="heading">Filter
options:</span></td></tr>
<tr>
     <td><span class="textNormal">Name:</span> </td>
     <td><input type="text" name="Name"
maxlength="60">&nbsp;&nbsp;&nbsp;</td>
     <td><span class="textNormal">Record type:</span> </td>
     <td>
          <input type="radio" name="RecordType" value="1"
checked> <span class="textSmall">All</span>
          <input type="radio" name="RecordType" value="2"> <span
class="textSmall">Partners</span>
          <input type="radio" name="RecordType" value="3"> <span
class="textSmall">Vendors</span>
          </td>
</tr>
<tr>
     <td><span class="textNormal">Country:</span> </td>
     <td><select name="Country">
          <option selected></option>
          <%
          ’Opening connection to database
          Set conPartnersVendors =
Server.CreateObject("ADODB.Connection")
          conPartnersVendors.Open "PartnersVendors"

          ’Opening recordset
          Set rdsCountry =
Server.CreateObject("ADODB.Recordset")

          ’Getting country names
          With rdsCountry
               .ActiveConnection = conPartnersVendors
               .Source = "SELECT DISTINCT COUNTRY FROM
PartnersVendors ORDER BY COUNTRY;"
               .Open

               Do Until .eof
          %>
                    <option><%=rdsCountry("COUNTRY")%></option>
          <%
                    .MoveNext
               Loop

               .Close
          End With
          %>
          </select>
     </td>
     <td><span class="textNormal">Results per page:</span> </td>
     <td>
          <input type="radio" name="Results" value="5" checked>
<span class="textSmall">5</span>
          <input type="radio" name="Results" value="10"> <span
class="textSmall">10</span>
          <input type="radio" name="Results" value="20"> <span
class="textSmall">20</span>
     </td>
</tr>
<tr><td colspan=4><img src="picts/empty.gif" width="1"
height="6"></td></tr>
<tr><td colspan=4 align="center"><input type="button"
value="Apply" onclick="SubmitForm(’Apply’)"> <input type="button"
value="Reset" onclick="SubmitForm(’Reset’)"></td></tr>
</table>
</form>
</body>
</html>
```

main.asp

```
<%
’Main application part
’Searches database, displays results, maintains connection state

’Opening connection to database
Set conPartnersVendors = Server.CreateObject("ADODB.Connection")
conPartnersVendors.Open "PartnersVendors"
’Creates recordset
Set rdsMain = Server.CreateObject("ADODB.Recordset")

’Connection state
’1. Record per page
PageSize = CInt(Request("PageSize"))
If PageSize = 0 Then PageSize = 5
’2. Last record index
RecordIndex = CInt(Request("RecordIndex"))
’3. Last number of records shown
LastPageSize = CInt(Request("LastPageSize"))

Results = CInt(Request("Results"))
If Results > 0 Then PageSize = Results

’Constructing SQL string
sSQL = "SELECT * FROM PartnersVendors WHERE "

SearchName = CStr(Request("Name"))
SearchCountry = CStr(Request("Country"))
SearchRecordType = CInt(Request("RecordType"))

Select Case SearchRecordType
      Case 3
            sSQL = sSQL & "TYPE = 1"
      Case 2
            sSQL = sSQL & "TYPE = 0"
      Case Else
            sSQL = sSQL & "(TYPE = 0 OR TYPE = 1)"
End Select

If Len(SearchName) > 0 Then sSQL = sSQL & " AND NAME LIKE ’%" &
SearchName & "%’"
If Len(SearchCountry) > 0 Then sSQL = sSQL & " AND COUNTRY = ’" &
SearchCountry & "’"

sSQL = sSQL & " ORDER BY PCODE;"
’End constructing SQL string

’Setting recordset parameters and opening recordset
With rdsMain
      .ActiveConnection = conPartnersVendors
      .Source = sSQL
```

```
     .CursorType = 1
     .Open
End With

’Checking if Next/Previous button was pressed
ActionPerformed = Request("ActionPerformed")
Select Case ActionPerformed
     Case "Next"
          RecordIndex = RecordIndex + 1
     Case "Previous"
          RecordIndex = RecordIndex - (LastPageSize+PageSize) +
1
End Select
%>

<html>
<head>
<link type="text/css" rel="stylesheet" href="styles/main.css">
<script language="javascript">
<!--
//Setting value of the button pressed
function SubmitForm(v)
{
     document.browseForm.ActionPerformed.value = v;
     document.browseForm.submit();
}
//-->
</script>
</head>
<body bgcolor="#ffffff">
<%
’Getting records from database and formatting them
n = 0
Do Until rdsMain.eof Or n > RecordIndex + PageSize - 1
If n >= RecordIndex Then
’Checking record type
     If rdsMain("TYPE") = 0 Then
          RecordType = "Partner"
          Set rdsPartnerType =
Server.CreateObject("ADODB.Recordset")
          With rdsPartnerType
               .ActiveConnection = conPartnersVendors
               .Source = "SELECT PTYPE FROM PartnerTypes WHERE
PCODE = ’" & rdsMain("PCODE") & "’;"
               .Open
               ’Checking partner type
               Select Case rdsPartnerType("PTYPE")
                    Case 2
                         PartnerType = "Monographic and serial"
                    Case 1
                         PartnerType = "Serial"
                    Case Else
```

```
                        PartnerType = "Monographic"
                  End Select
                  .Close
             End With
        Else
             RecordType = "Vendor"
        End If
        %>
        <p class="textNormal"><b><%=rdsMain("PCODE")%></b></p>

        <p class="textNormal"><u>Type:</u> <i><%=RecordType%></i>.
<i><%=PartnerType%></i>.</p>

        <p class="textNormal"><b><%=rdsMain("NAME")%></b></p>

        <p
class="textNormal"><u>Address:</u><br><%=rdsMain("STREET")%><br><
%=rdsMain("CITY")%> <%=rdsMain("STATE")%>
<%=rdsMain("ZIP")%><br><%=rdsMain("COUNTRY")%></p>

        <%If Not IsNull(rdsMain("PHONE1")) Or Not
IsNull(rdsMain("PHONE2")) Then%>
             <p
class="textNormal"><u>Phone:</u><br><%=rdsMain("PHONE1")%>&nbsp;&
nbsp;&nbsp;&nbsp;<%=rdsMain("PHONE2")%></p>
        <%End If%>

        <%If Not IsNull(rdsMain("FAX")) Then%>
             <p
class="textNormal"><u>Fax:</u><br><%=rdsMain("FAX")%></p>
        <%End If%>

        <%If Not IsNull(rdsMain("EMAIL1")) Or Not
IsNull(rdsMain("EMAIL2")) Then%>
             <p
class="textNormal"><u>Email:</u><br><%=rdsMain("EMAIL1")%>&nbsp;&
nbsp;&nbsp;&nbsp;<%=rdsMain("EMAIL2")%></p>
        <%End If%>

        <%If Not IsNull(rdsMain("CPERSON")) Then%>
             <p class="textNormal"><u>Contact
person:</u><br><%=rdsMain("CPERSON")%></p>
        <%End If%>

        <%
        ’Checking profile
        If rdsMain("TYPE") = 1 Then
             Set rdsVendorProfile =
Server.CreateObject("ADODB.Recordset")
             With rdsVendorProfile
                  .ActiveConnection = conPartnersVendors
```

```
                .Source = "SELECT PROFILE FROM VendorProfiles
WHERE PCODE = ’" & rdsMain("PCODE") & "’;"
                .Open
                If Not .bof And Not .eof Then
                %>
                     <p class="textNormal"><u>Vendor
profile:</u><br><%=rdsVendorProfile("PROFILE")%></p>
                <%
                End If
                .Close
           End With
     End If
     %>

     <%
     ’Checking notes
     Set rdsNotes = Server.CreateObject("ADODB.Recordset")
     With rdsNotes
           .ActiveConnection = conPartnersVendors
           .Source = "SELECT NOTES FROM PartnerVendorNotes WHERE
PCODE = ’" & rdsMain("PCODE") & "’;"
           .Open
           If Not .bof And Not .eof Then
           %>
                <p class="textNormal"><u>Vendor
profile:</u><br><%=rdsNotes("NOTES")%></p>
           <%
           End If
           .Close
     End With
     %>

     <%
End If
rdsMain.MoveNext
n = n + 1
’End of getting and displaying records
%>
<p><hr width="50%" style="text-align:left;"></p>
<%
Loop
’Setting number of records on the last page
LastPageSize = n - RecordIndex
’Setting last record index
RecordIndex = n - 1
’Returning connection state information
%>
<form name="browseForm" action="main.asp" method="post">
<input type="hidden" name="PageSize" value="<%=PageSize%>">
<input type="hidden" name="RecordIndex" value="<%=RecordIndex%>">
<input type="hidden" name="LastPageSize"
value="<%=LastPageSize%>">
```

```
<input type="hidden" name="ActionPerformed" value="">
<%
’Setting navigational buttons
If RecordIndex - PageSize > 0 Then
%>
     <input type="button" name="Previous" value="Previous"
onclick="SubmitForm(’Previous’)">
<%
End If
If RecordIndex + 1 < rdsMain.RecordCount Then%>
<input type="button" name="Next" value="Next"
onclick="SubmitForm(’Next’)">
<%
End If
rdsMain.Close
conPartnersVendors.Close
%>
</form>
</body>
</html>
```