New Technologies for Rapid Development of Web Orientated Database Applications

Nick Hatzigeorgiu ***and*** **Apostolos Syropoulos**

Abstract

Many new Internet technologies are available to Web programmers. Using these technologies it is now easy to develop dynamic Web based applications that use databases. We describe some of these new technologies and we briefly present two examples showing how we can use these technologies in real world applications.

1. Introduction

The development of the Hypertext Markup Language (HTML) and the Web browser has spurred a tremendous growth of the Internet over the past 5 years. Many traditional multimedia or database developers are now interested in programming for the World Wide Web. Our personal experience leads us to believe that such applications can be useful in educational environments. Furthermore, using the latest Web technologies it is quite easy for the inexperienced programmer to develop dynamic Web applications.

HTML is an interpreted markup language based on tags within regular ASCII text. The Web browser contains an interpreter for this language and a viewer for the resulting formatted text. The HTML documents (Web pages) are being kept on a Web server. The Web server and browser communicate with each other using the HTTP protocol. The client-server model of the HTTP protocol is simple: the client requests a document, the server sends the document to the client and closes the connection.

The simplicity of the original form of both the HTML language and the client-server model of the HTTP protocol are largely responsible for the success of the Web. Almost everyone can write Web pages. However, this simplicity is also the major problem in developing Web applications. Some of the original handicaps are

- HTML is a markup language, not a programming language. It doesn't even offer some type of loop construct.
- HTTP requests from the client can only be for "whole" documents, not for "chunks" of data. Thus, the client can only show or change a whole document, not parts of it.
- HTTP is stateless. After the server has fulfilled a client request, no memory of the transaction is being kept.

Obviously, it is quite difficult to develop dynamic Web applications under these restrictions. "Dynamic" has a two-fold meaning within this paper: "dynamic content" and "dynamic user interface".

Dynamic content is contrasted to the static content of the regular Web pages. It can be the result of an interaction with a database of some short, which leads to the construction (on the fly) of an HTML page.

Usually this is achieved by server-side programming techniques. Till very recently, the only way of achieving dynamic content was the use of CGI scripts on the server. This is a valid technique but somewhat difficult to learn. Also, it posses a higher performance tax to the Web server that the alternatives we are describing later (this is because the server has to initialize a new process for every CGI script it is running).

A dynamic user interface is one that allows the user to do something more than flipping though Web pages. It is the result of client-side programming. The classical way to do this was the use of a scripting language such as Java. In this case the source code is transmitted to the client where it is interpreted by a Java compiler that resides on the client. The only alternative was the use of precompiled Java applets that run on the client.

In our opinion, CGI scripts on the server-side and Java scripts and applets on the client side can achieve a lot towards dynamic Web applications but they have a stepper learning curve that the technologies we will describe. Furthermore, they pose some restrictions that are the result of security concerns. However, there are cases, such as the development within the bounds of Intranets or educational establishments, were those security constraints are not necessary or even desirable. (This is especially true when the developer of those applications also belongs to the team of the local network managers!)

1. Client Side Programming

HTML has grown considerably since its first implementation. The original HTML language, developed by Tim Berners-Lee, allowed only hyperlinks and some basic formatting of ASCII text. Later we have the addition of forms (which allow the client to transmit data to the browser), tables, scripts and frames. Scripts allow us to use an interpreted high level language (such as JavaScript or Visual Basic Scripting Edition) on the client, provided that the client includes the necessary interpreter for the scripting language. Frames allow us to view simultaneously more than one hypertext document and consequently give us the ability to change only a part of the viewable area (which, however, remains a discrete document).

Nowadays, the World Wide Web Consortium (W3C, http://www.w3.org) is responsible for the standardization of the HTML language. The latest W3C Recommendation is HTML 4.0 (sometimes its also called Dynamic HTML, DHTML) [1]. It is composed of various elements and, among else, provides us with two powerful programming tools:

- Cascade Style Sheets (CSS). CSS allow the programmer to format the hypertext documents in a more flexible way than ever before. It is a presentation tool, useful for a better aesthetic appeal of our applications.
- The ability to "name" and manipulate each and every tag. These names can be used to change the various properties or the content of the document while the client is viewing the document. For example, we can hide or show text and pictures, we can change parts of the document when the user performs some action (such as moving the mouse over an object) and much more. Practically every tag in our document becomes a named object and can

be manipulated using client side scripting.

Here, we have to point out that the two most popular Web browsers today, Netscape Navigator and Microsoft Internet Explorer (I.E.), have implemented dynamic content in non-compatible ways. Furthermore, they provide interpreters for different languages: Navigator supports JavaScript while I.E. supports Jscript (derived from Java) and VBScript [2] (derived from Visual Basic). Thus it is possible that some dynamic Web pages work fine in one browser and do not work at all when viewed by another browser. There is some hope that this will soon end, since the next generation of both browsers will probably support the HTML 4.0 standard. Also, the recent standardization by W3C of ECMAScript, (proposed by both Netscape and Microsoft) will provide us in the future with a Java-like scripting language that all browsers can understand.

Beyond HTML and scripts, the programmer has in his/her disposal another useful element of dynamic client-side content: precompiled objects. These can be either Java applets or ActiveX controls. ActiveX controls are objects based on Microsoft's COM technology. They currently work only with I.E. and some very useful ones (e.g. providing database access) are already included with the I.E distribution (for more information on the use of the technologies that I.E. provides, see [3]). If the developer wants to create some other ActiveX controls he can easily do that using most of the RAD tools (e.g. Visual Basic, C++, Delphi). If a developer includes an ActiveX control in a Web page and the client browser doesn't have this particular control, the browser will automatically download the control.

It is now evident that the Web application developer has to make some hard choices at the planning stage of his project. There are many kinds of dynamic client-side content but those that are the most useful or necessary can only be determined by the particular requirements of every project. The first question that every developer should ask is whether he can forego the inclusion of any dynamic client side content at all. The existence of new technologies such as Java, applets, CSS, VBScipt et. al. does not mean that we have to use them when we don't need them! It is also important to remember that most of the older browsers do not support some (or all!) of those technologies. Finally, we have to consider the performance penalty our application might suffer due to object downloads.

If the developer decides that he/she has to use some kind of dynamic client-side content, then he/she has to consider what the audience is. Sometimes, it is known in advance that the audience will be using a certain type of browser and this makes things easier. For example, in one of our projects we knew that all the clients were high schools using I.E. So we decided to use VBScript and standard ActiveX controls. This allowed us to deploy our application sooner than if we had to use Java applets and/or strictly server side programming.

1. Server Side Programming

Server side programming can provide us with dynamic content, usually in conjunction with a database residing on the server. When the client requests a document, the server runs a script, performs a query on the database if this is required and then formulates an HTML page which is transmitted to the client via HTTP.

Server-side programming depends on both the platform and the particular Web Server we are using. Sometime ago, most Web servers were running on Unix and server-side programming meant a CGI script (probably written in Perl). However, the Windows NT platform is now a viable alternative and it is often easier to use. Furthermore, there are plenty of Web servers available, both for Windows NT and Unix, and each of them provides different tools and technologies to the developer.

Again there are some choices that have to be made during the planning stage of a project, namely what the operating system and what the Web server will be. Here we will focus on some tools that are available on Windows NT using the Internet Information Server version 4.0 (IIS) [4], which is available free of charge from Microsoft (http://www.microsoft.com). The latest version of IIS includes the Active Server Pages (ASP) technologies [5]. Writing ASP pages is simple: we just write our regular HTML (or DHTML) and we include some server-side code (e.g. in VBScript or Jscript). The server executes the script, constructs an HTML page and sends it to the client. This sounds like CGI scripts, but there are many differences. ASP pages are faster to run than CGI scripts (the Web server doesn't create a separate process for each ASP script), they are easier to write (we can write the HTML part in the usual way) and, most of all, they provide us with plenty of build-in functionality.

ASP is composed of several server-side objects that can be used by the programmer. For example, there is the "application object" that can store application-wide variables, objects and procedures (an application, in this context, consists of all the web pages that reside in a particular directory). There is also the session object. A session begins when a user requests any page of the application. Many sessions can be opened simultaneously, depending on the number of users that are connected to our application. Using the session object we can store and retrieve information for events, keeping track of the state of the application and thus, partially overcoming the "stateless" property of the HTTP protocol. Finally, another group of objects we have in our disposal is the ActiveX Data Objects (ADO) and their database functionality.

In general, there are two kinds of relational databases: single-user (such as Access, Paradox etc.) and multi-user (such as Oracle, SQL Server etc). Internet applications are inherently multi-user applications but nothing prevents us from using single-user databases in situations when we know that we won't have too many simultaneous users (for example, when a database is accessed only by a small workgroup). Using ADO makes it is easy to build tree-tiered database applications where the Web server (ASP) plays the role of the middle-tier and the Web browser is the client front end. The Web server can communicate with the database through the ODBC (Open Database Connectivity) using the ADO build-in objects. The front end communicates with the Web server through HTTP using HTML forms (or even ActiveX objects or Java applets).

Let us give a simple example to show how easy this is. Suppose we have in our server an Access database called "schools" and we want to retrieve and present the field "names" from a table called "students" which resides in the database. First, we create an ODBC System Data Source (System DSN) giving it a name (here the name will be "schools". Then we create the following ASP code:

```
<%
set StudentsConn = server.CreateObject("ADODB.Connection")
StudentsConn.Open "schools"
Set RS =Server.CreateObject("ADODB.Recordset")
sql="SELECT names FROM students"
RS.Open sql, StudentsConn
%>
<HTML><HEAD></HEAD>
<BODY>
<B>These are the student names:</B>
<% Do while Not RS.EOF %>
<%=RS("names")%>
<% RS.MoveNext
Loop
RS.Close
StudentsConn.Close%>
</BODY></HTML>
```

As we see, the Visual Basic code resides within "<%" and "%>" tags. The code is simple to read: we open a connection with the database, we send an SQL query and then we present the resulting recordset using a "While...Loop" construct. Practically, what we do in ASP is to use a combination of HTML code interwoven with VBScript (or Jscript) code.

2. Remote Data Services (RDS)

The techniques we have mentioned so far address most of the original limitations of the HTML-HTTP model, except one: we cannot perform a query to the database (resulting probably in an interactive modification of the displayed data) without reloading the whole page.

The Remote Data Service (RDS, http://www.microsoft.com/data/rds/) objects, included with I.E. 4.0 and IIS 4.0, address this issue. Using these ActiveX objects, the server can send some data to the client, the client can perform some modifications to the data and the changes can be saved in the server's database without having to reload the page at all! It is obvious that using RDS we come very close to the regular three-tiered client-server model used to access databases, only that now we don't have to install a front-end to the client since all the necessary objects are included in the I.E. installation.

RDS technology includes both server-side and client-side components and requires the synergy or many of the technologies we discussed so far: IIS, ActiveX, client and server-side scripting, ADO, ODBC. All of these technologies must be present which means that we have to be using Windows NT with IIS for a Web server and IE 4.0 (or later) as the client browser. When we use RDS, we include an invisible Advanced Data Connector (ADC) ActiveX control in our ASP page and then we use its methods to access the data, present, manipulate or update it, without having to conduct any reloading of the pages.

3. Development Considerations and Examples

It is obvious that the manager of every Web application project has to make some important choices at the very beginning of the project. Those concern both the development platform and the technologies to be used. The managers have to take into account the in-house skills, the specific requirements of each project, and the target audience. Of course, this is true in the development of any client-server application, with the exception of having to take into account the extreme speed of evolution of new Web technologies. Just a year ago, RDS was not an option, and right now we have no browsers with complete HTML 4.0 or ECMAScript support although this might change six months from now.

We have successfully applied some of the above technologies in two of our projects. Both projects need about another year to complete but they are already functional. The fact that they are long-term, non-commercial projects influenced our decisions.

The first project involves the development of a database filled with regional data, both for scientific and public use. At the beginning of the project we chose to develop in Access because of easy of use and integration with MS-Word. The writers and researchers filled special forms in MS-Word which were processed by scripts and macros in Visual Basic to extract the information that filled the Access database tables. A simple Web interface was developed to access this data. During the project it became evident that we need Unicode data types for some fields, so we are planning an upsize to MS-SQL Server 7.0 by the end of the year. We have already experimented with a partial upsize and it has posed no special problems either to the database or to the interface, since we have been using ODBC calls with standard SQL statements.

For this project we knew that the initial transaction volumes would be low, so using an Access database was sufficient, at least for the beginning. We also knew that the researchers who would be using the interface worked in a variety of platforms and that it would eventually become public, so we decided to use nothing more than HTML 3.2 (HTML forms and tables). On the server side we have the freedom to use any platform we liked, so we choose Windows NT 4.0 with IIS 4.0 and Visual Basic server-side scripting (ODBC and ADO).

The second project involved the creation of a small database and an interface to be used as a pilot project in some public schools. The interface would be used to teach some aspects of the Greek language to students and people with almost no computer skills should be able to benefit from it. Also, it had to include some kind of bulletin board where students of different schools could pose questions and get answers concerning the teaching material.

For this project we knew that the schools had Windows NT LANs and were using IE 4.0 as a Web browser and that only these particular schools would use the interface (it wouldn't become public). Also that the schools would prefer a simple to use but attractive interface. So we decided to use the full spectrum of technologies we described along this paper. The database was developed in MS-Access and we used ODBC calls. We developed an MS-Access Web interface for the teachers to enter student-related data into Access (students' names and groups) using HTML 4.0. The user interface accessed by students used DHTML, RDS and ActiveX objects.

4. Conclusions

The development of Web technologies continues with an unprecedented speed, even by computer industry standards. The Web application developer has plenty of technologies in his/her disposal today and even more Web technologies are considered for standardization (such as XML, see [6]). There are still major problems such as the incompatibility of the various Web browsers, the relative immaturity or some of these technologies and the absence of satisfactory debugging tools or even development tools (a simple text editor like notepad remains the best development tool for creating complicated scripts). Yet, despite those problems, using the latest available technologies can often make a difficult project much easier.

Bibliography

1. Elizabeth Castro, HTML 4 for the World Wide Web: *Visual QuickStart Guide*, Addison-Wesley 1998.
2. Paul Thurrott and Nolan Hester, *VBScript for the World Wide Web*, Addison-Wesley 1997.
3. Alan Simpson, *Official Microsoft Internet Explorer Site Builder Toolkit,* Microsoft Press 1997
4. *Microsoft Internet Information Server Resource Kit,* Microsoft Press 1998
5. Brian Francis, et al., *Professional Active Server Pages 2.0*, Wrox Press Inc. 1998.
6. Scott Mace et al., Weaving a Better Web, *Byte* (International Edition), March 1998, pg. 58.

Author Contact Information

Nick Hatzigeorgiu, Ph.D. Institute of Language and Speech Processing
Xanthi Branch Vas. Sofias 8 671 00 Xanthi Greece
nikos@xanthi.ilsp.gr

Apostolos Syropoulos Department of Civil Engineering
Democritus University of Thrace 671 00 Xanthi Greece
apostolo@obelix.ee.duth.gr