

# MyEnv Virtual Desktop Environment

Joshua Daniel

May 2008

Department of Computer Science  
California State University, Fresno

# MyEnv Virtual Desktop Environment

Joshua Daniel

May 2008

Department of Computer Science  
California State University, Fresno  
Fresno, CA 93740

Project Advisor  
Shih-Hsi Liu

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science

## **Abstract**

With the price of hardware decreasing and the processor and memory technology increasing, the world of computers is moving back to the thin client model. It is a more practical model for most companies to be able to put their money into a single server and to be able to purchase low powered machines for each employee. However, this is not the only benefit to the thin client model. The fact that one's session is available at any terminal one is logged into is the real power behind the thin client. This would be especially valuable to a student who has to use a different machine to log in each time the student visits the on campus lab. Keeping their environment intact would mean that they could create their environment at home or at school and access it from any machine that has Internet access.

This master project is to provide a single web site that will provide a user the ability to log in and retrieve their session, which includes open browsers, saved videos and bookmarks, using DHTML and AJAX technologies. One of the major challenges in this implementation will be keeping a breadcrumb trail of where the user has navigated to with each of the virtual browsers in the user's desktop environment. Since modern JavaScript does not allow access to the URL location of an iframe pointing a remote location, which is what the virtual browser in this project are comprised of, the HTML source of the websites browsed to will have to be copied to the local server and then

rendered to the user. This action of copying remote HTML locally is commonly referred to as tunneling.

## Preface

This master project was conducted in order to experiment with what could be done using simple HTML and JavaScript. The project could have been created using Java or Flash. However, there is a certain attraction to using DHTML because more and more web companies are moving towards using the combination of DHTML, JavaScript, and AJAX. Moreover, it does not require the host operating system to include any additional plug-ins, since most modern browsers are capable of running JavaScript. Not to leave out the fact that there is some server side scripting in this project. However, some scripting is necessary in order to save and retrieve data from the web server/database. Even more so, in the case of this project, the web server itself is a server side script rather than a compiled binary. It is enough to note that the server side script used in this project is Python which was a first for this author however the server side code, or in other words back end, could have been just as easily been written in ASP or PHP. Python did however keep the server side code very clean with its built in tab policy, which forces the coder to use appropriate number of tabs per code block. The database was chosen to be SQLite since it saves its data to file, which made it easy to move the entire project from one machine to another for testing and development. However, any database could be easily substituted for SQLite in this project. It is this author's wish that this project could one day be used in the college lab environment or at least a good place for YouTube users to store their favorite videos.

# CONTENT

<b>ABSTRACT</b>	<b>3</b>
<b>PREFACE</b>	<b>5</b>
<b>CONTENT</b>	<b>6</b>
CHAPTER 1	<b>9</b>
<b>PROJECT OVERVIEW</b>	<b>9</b>
Introduction	9
CHAPTER 2	<b>10</b>
<b>RELATED WORKS</b>	<b>10</b>
Yahoo! User Interface Library	10
YouOS	11
CHAPTER 3	<b>12</b>
<b>BACKGROUND</b>	<b>12</b>
JavaScript/DHTML	12
Python	13
JSON	14
<b>IMPLEMENTATION</b>	<b>15</b>
User Profile	15
Page Canvas	15
Canvas Div	15
Coordinate System Using CSS	16
User Interface	18
Browser Objects	18
Bookmarks and AJAX	18
Why use AJAX?	19
JavaScript Browser	20
Resulting Look and Feel	21
Other Types of Windows	22
<b>MyEnv Database Strategy</b>	<b>23</b>
Summary	23
Database Schema	24
User Table	24
Environment Table	24
Bookmarks Table	26
Video Bookmarks Table	26
Summarize	27

CHAPTER 5	<b>28</b>
<b>MyEnv XML Specification</b>	<b>28</b>
Overview	28
AJAX Request/Response XML	30
Add a Bookmark	31
Add a Video Bookmark	32
Retrieve bookmark URL	33
Retrieve Video Bookmark Embed Code	34
Request to Save Environment	35
Request to retrieve last environment	36
Request to Remove Browser	38
Error response	38
 CHAPTER 6	 <b>39</b>
<b>MyEnv XML to SQL Mapping</b>	<b>39</b>
Overview	39
Add a Bookmark	41
Add a Video Bookmark	42
Retrieve Bookmark URL	42
Retrieve Video Bookmark	43
Request to Save Environment	43
Request to Retrieve Last Environment	44
Request Last Browser URL	45
 CHAPTER 7	 <b>46</b>
<b>Project Problems and Solutions</b>	<b>46</b>
Overview	46
JavaScript Security	46
Hyperlink Injection	49
The Browser History Problem	51
The Form POST and Session Problem	53
 CHAPTER 8	 <b>56</b>
<b>Conclusion</b>	<b>56</b>
 <b>REFERENCES</b>	 <b>58</b>



## CHAPTER 1

### ***PROJECT OVERVIEW***

#### **Introduction**

The idea of this master project is to create a browser desktop environment within a single webpage. The environment would allow a user to open multiple web browsers and position them according to personal preference. They may then end their session at any time and log in at a later time from any machine with their environment still intact. The end result will be a mobile environment that a user may use from any JavaScript enabled browser. It also allows mobile devices to point to a single page and require nothing else but JavaScript in order to use multiple browser windows. The motivation of this project is to learn how much can be accomplished using JavaScript to provide a desktop-like environment and to also find out what the limitations of web browsing will be when using such an environment. The project will be called MyEnv short for “My Environment”.

## CHAPTER 2

### ***RELATED WORKS***

#### **Yahoo! User Interface Library**

The Yahoo! User Interface Library referred to as YUI is an API based on a set of JavaScript controls built to provide JavaScript programmers a tool to create a rich web environment<sup>[11]</sup>. The API includes calls that support displaying modal dialogs, movable on screen windows and Document Object Model (DOM) utilities. All of the mouse interactions and events are hidden behind the scenes. The developer does not need to worry about recording mouse position based on event or manually inserting or removing objects from the HTML DOM. It provides the end user with a website that feels more like a desktop or local application environment. Many of the techniques used behind the scenes by YUI are implemented in this experiment from the ground up and will be covered in later chapters. The dynamic HTML (DHTML)<sup>[6]</sup> portion of this experiment which uses raw JavaScript could have been substituted by using the YUI API package. However part of the experiment was to learn how to develop raw DHTML and AJAX (Asynchronous JavaScript and XML) using JavaScript. At the time this experiment started YUI was also in a beta state.

For the sake of learning in this experiment we will reinvent the wheel in order to better understand how the wheel is made. YUI however would have been a great platform to

base this experiment on if it had existed in a stable state at the time this experiment started.

## **YouOS**

YouOS <sup>[12]</sup> is a web-based operating system. It does not provide any open source or API for other developers to use. However it does provide a good example of how a full JavaScript desktop environment should work. It incorporates its own word processor, spreadsheet program, email program and other utility type programs (i.e., calculator). All of the functionality is rendered on a single page much like the experiment that is covered in this document. All events triggered by the user are handled via JavaScript, and resulting actions by the OS are rendered to the user by manipulating the HTML DOM via JavaScript.

What is important to this experiment is that it has the ability to open a browser window in a single iframe. However, it does not support multiple windows or bookmarking of URLs. The reason being is that its browser is simply an iframe window and has no knowledge of what URLs the browser is visiting. Such a query to the iframe window would be a JavaScript security issue which will be covered in later chapters. This is the problem that MyEnv will attempt to solve.

## CHAPTER 3

### ***BACKGROUND***

#### **JavaScript/DHTML**

This project will be developed completely on two web pages (HTML pages). The first will be the login page, which will use a simple HTML post and server-side authentication. The second page will be the canvas page, which will be the final page the user's browser accesses when using the MyEnv program. The dynamic content and user interface on the canvas page will be provided completely by JavaScript.

Modern JavaScript has the ability to access the HTML DOM object and manipulate it in real time. The DOM is accessed client side, which means that it is altered in the user's browser not on the server. The server is then unaware of any changes, and therefore there is no need for the browser to request anything from the server. As far as the server is concerned it has already served the HTML page to the browser.

The ability to do this has led to development of new terminology called Dynamic HTML or DHTML.<sup>[5]</sup> By using DHTML, we can programmatically move HTML tag positioning on the screen while capturing mouse events, which will give the user the impression that they are moving items within the web page.<sup>[5]</sup> Each potentially movable item on the page is given CSS attributes of left and top. By using these attributes, the

items can be positioned on the page using the distance (in pixels) from the top left corner. By using DHTML, the top and left attributes can be changed in real time. This will give the user the feel of a desktop type environment where they can move, open and close windows in real time all in a single HTML page. Alternate technologies that could have been used here are Adobe Flash or a Java Applet.

## **Python**

The Python programming language was originally developed as a scripting language. Its purpose was to be easy to read and provide an API which was short and to the point. Because of these two attributes Python is widely used today in the software industry. One of Python's most avid users is the dot-com giant Google. Python has grown tremendously since its original release; it now includes a multitude of freely downloadable packages and APIs.

One of modern Python's free packages is a web server called webpy<sup>[10]</sup>; it is a powerful development environment for server side development in Python. Another is a package called BeautifulSoup<sup>[2]</sup>, which converts HTML content into a DOM object the same way JavaScript does. This allows for manipulation of the DOM, which will be used to alter the web pages that user is browsing to when using MyEnv.

Python also provides an efficient built in function calls, which means less code to write for the developer. All this and its ability to be portable across operating systems made it the language of choice for server side development in this project. To be perfectly clear,

this does not mean that another server side language could not have been used. ASP, PHP, Java or any other server side scripting would have worked just as well as server side code in this experiment.

## **JSON**

The “x” in Ajax stands for XML. This is because Ajax communication has traditionally been handled by passing XML back and forth between the client side JavaScript and the server side code. XML is compiled on the client side and passed to the server where it is parsed and then the server responds with XML in which the client side JavaScript then has to parse. Using JSON<sup>[13]</sup>, which is a string representation of a JavaScript object, can eliminate the step of parsing. By passing this string back from the server to the client side JavaScript, the client side script can then begin to utilize the data returned from the server as an object. Of course, JSON is not required for this project it was an optional step that eliminated the extra step of parsing.

## **IMPLEMENTATION**

### **User Profile**

Each user logging into the MyEnv system will require a user profile. This will allow for the system to save the user's information in the database, which includes the user's alias,  $x$  and  $y$  positions of currently active windows and any saved bookmarks. The user can retrieve their profile and log into their environment using a simple HTML form provided. Once they submit a login and password and are authenticated they will be forwarded to their saved environments.

### **Page Canvas**

The main page, which is the only visible HTML page other than the login page for this desktop environment, will be called the *canvas*. The canvas will house all open browser windows; no browser window will protrude outside of the area given by the canvas.

### **Canvas Div**

The canvas will be implemented using an HTML div tag. This will be the parent tag that will house multiple div tags, which will contain the browser windows. In order for the user to be able to place a browser at any position on the canvas, the canvas will have to support a coordinate system in which the HTML objects may be placed anywhere

on the screen rather than the traditional table format in which elements must be placed inside of table cells.

## Coordinate System Using CSS

The coordinate system can be obtained by utilizing a cascading style sheet or CSS. Using CSS we can apply certain attributes to any div tag on the page. The following would be an example of a CSS entry for the canvas and browser tags:

---

```
div.button div.canvas{ position:absolute }  
div.browser{ position:absolute; z-index: 2; border:1px solid #ccc }
```

---

Figure 1: CSS Entry Example

The div tags will be used to house iframes, which will represent browser windows on the canvas, much like the windows on an OS desktop. The z-index is put in place so that we can make sure that no div is hidden behind the canvas, the canvas should be the heaviest object on the screen. However, the z-index for the browser windows should change dynamically depending on which browser was last selected by the user, the default is a z-index of 2 since we assume that the canvas would have a z-index of 1<sup>[5]</sup>.

By using the simple CSS shown above in the page head tag, the following HTML code will then provide us with a canvas and two div tags positioned using the coordinate system:

```
<div class="canvas" style="width:100%;height:100%">
  <div class="browser"
style="left:30px;top:50px;width:300px;height:150px">window 1</div>
  <div class="browser"
style="left:100px;top:150px;width:300px;height:150px">window 2</div>
</div>
```

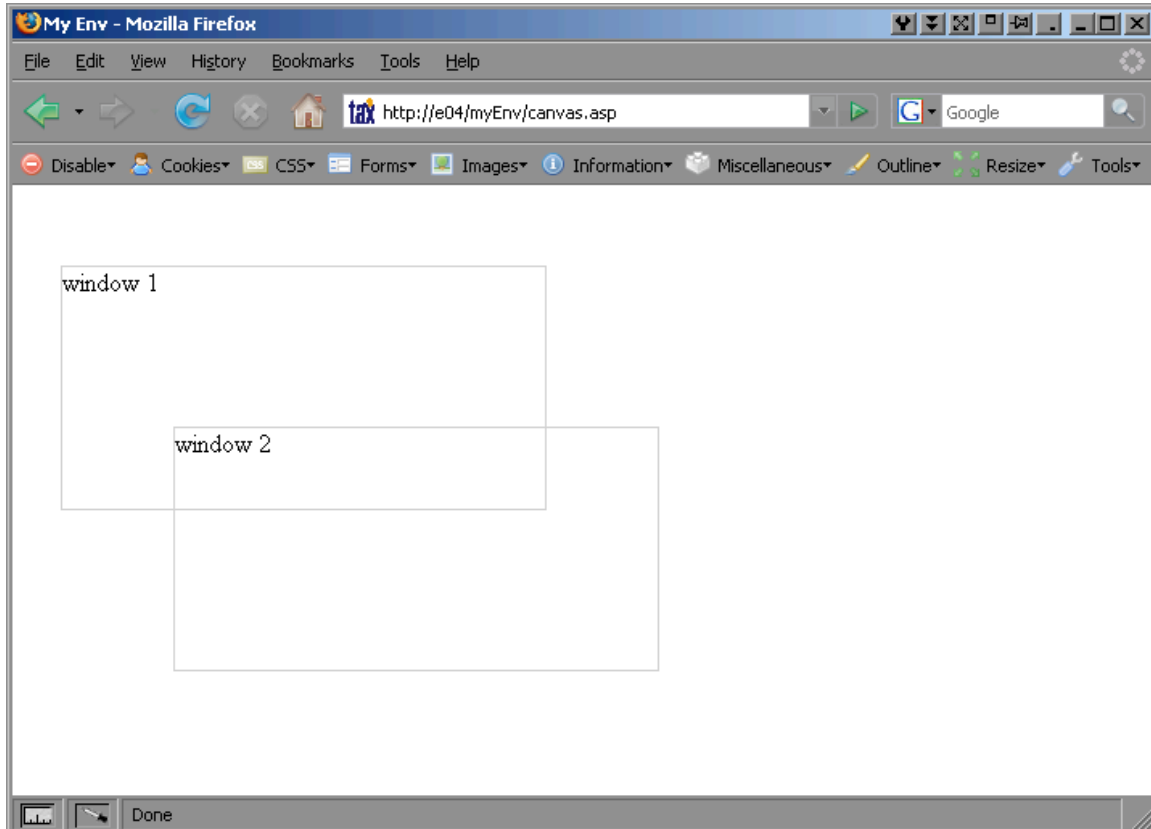


Figure 2: Simple CSS coordinate example

## **User Interface**

Once logged into the environment the user will have the ability to create browser windows and move them around and resize them. The best way to accomplish this will be to use JavaScript to capture mouse events and dynamically change the location of the div tags using DHTML.

## **Browser Objects**

Browser objects are iframes wrapped in div tags. By utilizing DHTML we can move these objects around the screen dynamically. In this experiment, an object representation of each browser window will be stored in an array JavaScript objects that are dynamically written to the page. Each object will hold the attributes of its own window: position, width, height and URL. The position in the array of browser objects will also note its z-index. Therefore, every time a user clicks one of the browser windows, that object will be moved to the top of the browser array and therefore take precedence when drawn on the screen. The positions of any remaining windows sustain their relative locations in the array. Therefore, the browser object that was originally at position zero in the array would be pushed down to position one and two to position three and so on.

## **Bookmarks and AJAX**

At any time the user may decide that the page they are currently browsing in a particular window is interesting and they would like to bookmark it. Since these browser windows are dynamically created by JavaScript, the only way to store the bookmark in the database, without posting to the page, is to use AJAX. AJAX will allow us to send a

message via HTTP POST containing the URL to be bookmarked to a script on the server, which will then insert that URL into the database.

### **Why use AJAX?**

Since the dynamics of the user interface are handled by JavaScript and therefore running on the client's machine, it would be inefficient and possibly a security issue for the JavaScript to make a direct connection to the database. Since JavaScript is client side it would be a risk for the database connection to be made directly since a mischievous user may alter the connection or alter the queries that sent to the database. Therefore, by using AJAX what we are essentially doing is making a request to a server side page which only purpose is to listen for GET and POST requests coming in via AJAX. Either way the script will now be doing its work with the database on the server side, which will make it more efficient. Furthermore, since we are providing an XML or JSON format for the message, it is also less subjective to SQL injection. SQL injection is a problem that may occur when a malicious user may want to alter or retrieve data from a database they should not have access to by altering SQL statements that a webpage is using to retrieve or update data. By using AJAX instead of a direct SQL query the server side will only accept certain commands, which eliminates the ability of a malicious user to use SQL injection.

## JavaScript Browser

There are a couple of different JavaScript browsers available for download. Most are free to use. All however fill their own window namely; they take up all real-estate in the browser window in which they are invoked. In this project the JavaScript browser will have to be encapsulated in a div tag and take up part of the canvas area. In order to do so an open source JavaScript browser must be modified or a new browser must be written from scratch. In this project the browser was written from scratch using an iframe; however the Accent JavaScript Browser<sup>[1]</sup> would have worked just as well. The Accent JavaScript Browser is freeware and available for use by anybody in their own projects. An example screen shot of the browser is shown below:

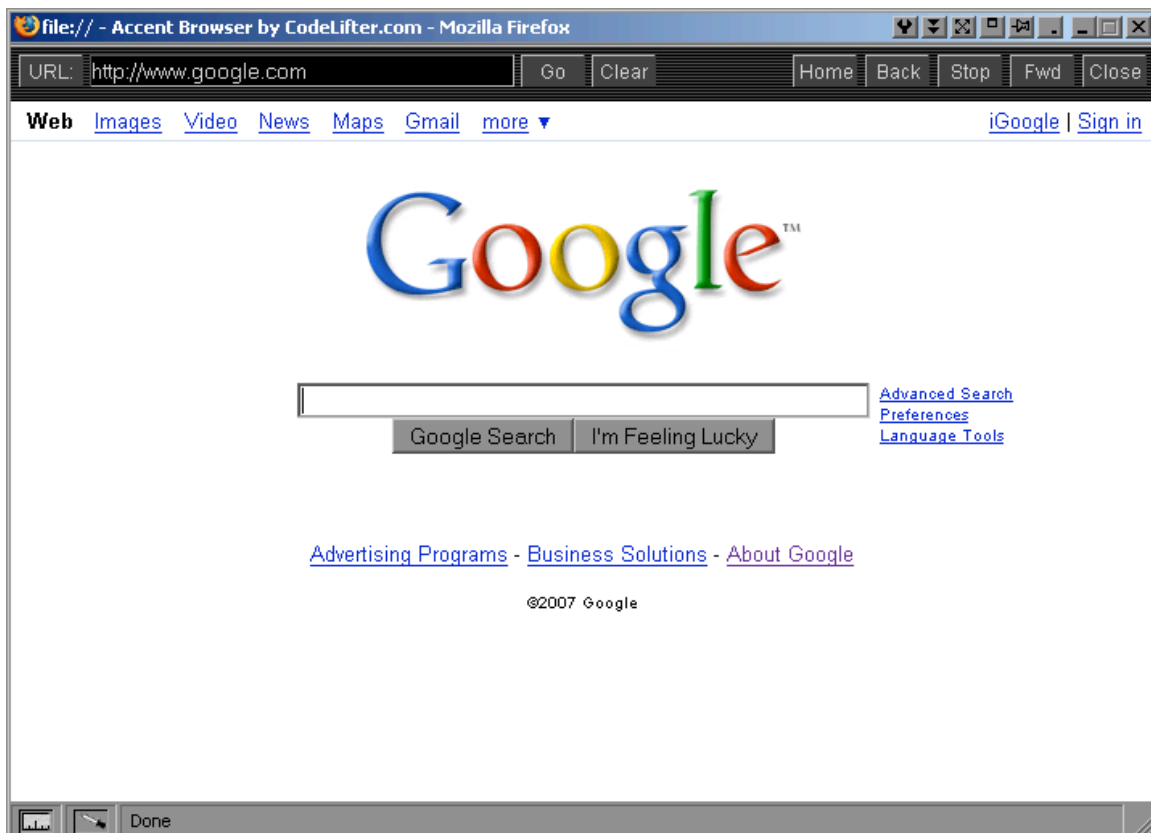


Figure 3: The Accent JavaScript Browser

Notice that the browser takes up the whole window. This is typical of a JavaScript browser and is also what needs to be modified in order for the browser to fit in with the requirements of this project. The browsers that were created for this project consist of a JavaScript navigation bar, which controls the location of an iframe below it. The navigation and iframe are wrapped up in a single div tag, which is what is considered a virtual browser. For the scope of this report, the browsers that are opened in side of the MyEnv canvas will be referred to as a virtual browser.

### **Resulting Look and Feel**

The resulting look and feel should mimic a desktop environment on a single HTML page. It therefore should allow the user to rearrange windows however necessary. The example below shows two virtual browser windows open to different websites and also a bookmarks pane in the upper right hand corner. The user is not limited to the number of windows open or the number of bookmarks. The virtual browsers are floating and therefore can be moved around the page and resized.

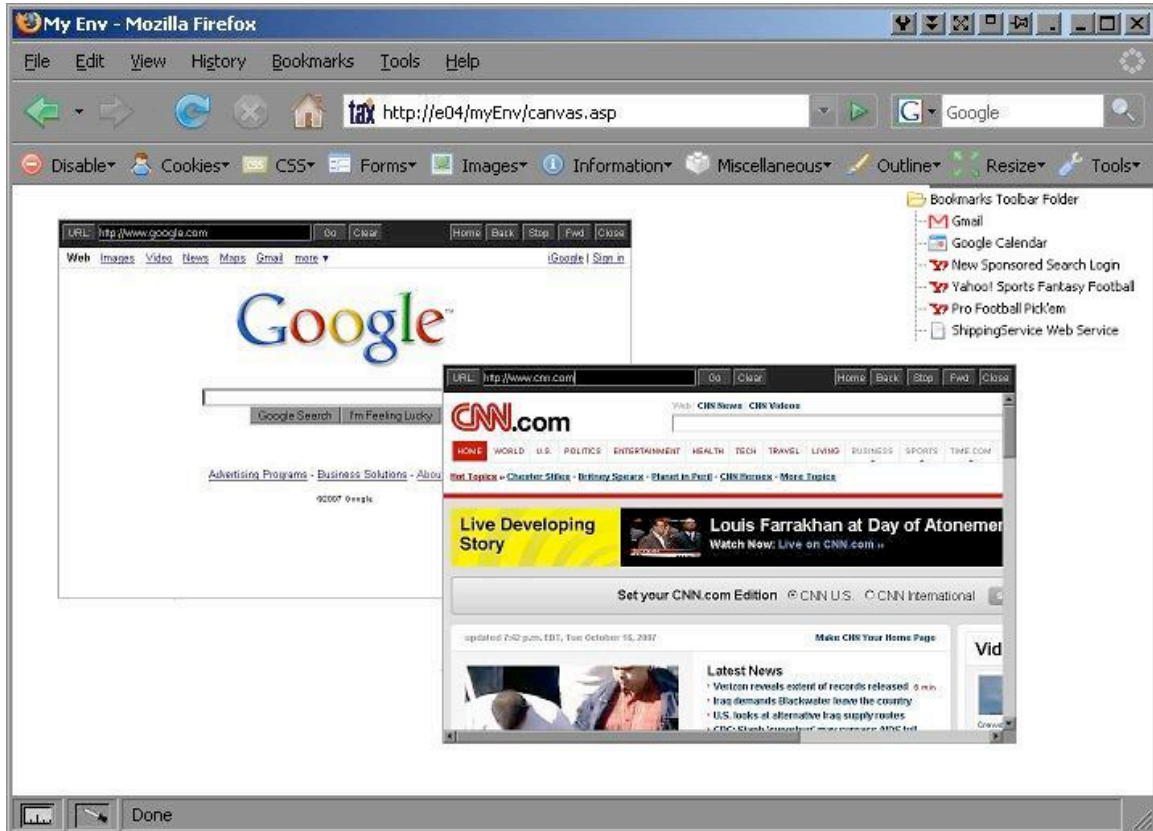


Figure 3: Example of resulting look and feel

## Other Types of Windows

This platform is easily expandable to include other types of windows that the user may use. For example, the user may want to store their documents and images on the server and therefore view them as well. This can be achieved by using a web based document viewer like Macromedia Flash Paper <sup>[2]</sup> embedded in one of the JavaScript browser objects. Another idea would be to embed HTML object tags inside the JavaScript browser objects, which would allow the user to view flash videos inside their virtual environment.

## CHAPTER 4

### *MyEnv Database Strategy*

#### **Summary**

MyEnv will utilize a database in order to save the state of a user's environment. In order to do so, it will require a user's table to store user profiles. A user id will index the profiles; this user id will be a foreign key in all tables associated with the user. Some of the tables include but are not limited to: bookmarks, video bookmarks and the environment table which includes the last known window positions of the user's environment.

Since the MyEnv program is client side JavaScript, the communication between the application and the database would normally have to take place via JavaScript database connection. However this is not a valid solution. Since the client's machine will not have direct access to the database, we will need a bridge to connect the front-end JavaScript to the backend database. The bridge will be a web page comprised of server side script, i.e., ASP/PHP/Python dependant on the host operating system. The JavaScript will send XML/JSON containing the information that needs to be stored or retrieved from the database via post to the server side script and the server side script will then query the database as necessary and return the resulting XML back to the client side JavaScript.

## Database Schema

### User Table

The user table will house the user profile. It will be the main point of entry into the system. The user will enter his/her credentials username/password, which will then be cross-referenced against the information in the table. If the credentials check out, the user will be sent to the canvas page, which will immediately start their environment. The id column will be the identity column of the table and will act as a foreign key in all other tables of this database.

---

	Column Name	Data Type	Allow Nulls
	id	int	<input type="checkbox"/>
	username	varchar(20)	<input type="checkbox"/>
	password	varchar(20)	<input type="checkbox"/>
	fname	varchar(20)	<input type="checkbox"/>
	lname	varchar(20)	<input type="checkbox"/>
	email	varchar(50)	<input type="checkbox"/>
	lastLogin	datetime	<input checked="" type="checkbox"/>

---

Figure 4: The User Table

### Environment Table

The environment table will encapsulate the user's environment. The state of all open windows associated with a particular user id will be recorded in this table. It will include the  $x$  and  $y$  positions along with the current height, width and URL location if applicable. Since there can potentially be different types of windows (browser window, video window, photo window) the windowType column is also included to differentiate

between window types. The userID column is a foreign key from the user table. The id column from the user table to the userID column in the environment table is one to many. Each user will have one environment, however the environment table houses information about active windows on the user's desktop. Therefore, one userID could own many rows in this table. Using another parent table this could potentially be expanded to one user to many desktops in the event that MyEnv would support multiple desktops per user. In that scenario a table, possibly called Desktop, could house many environments. The other rows of this table, i.e., xPosition, yPosition, zPosition, width, height, URL and ObjectEmbed, are all attributes of a window. The first five are size and position attributes. URL and ObjectEmbed store the current URL that the window is pointed to. If the window is a video window then the ObjectEmbed will store the video's embedded data.

---

	Column Name	Data Type	Allow Nulls
	id	int	<input type="checkbox"/>
	userID	int	<input type="checkbox"/>
	windowType	int	<input type="checkbox"/>
	xPosition	float	<input type="checkbox"/>
	yPosition	float	<input type="checkbox"/>
	zPosition	float	<input type="checkbox"/>
	width	float	<input type="checkbox"/>
	height	float	<input type="checkbox"/>
	URL	varchar(255)	<input checked="" type="checkbox"/>
	ObjectEmbed	text	<input checked="" type="checkbox"/>

---

Figure 5: Environment Table

## Bookmarks Table

The bookmarks table will store all user bookmarks. It will have a one user to many bookmarks relationship just like the environment table. The bookmarks may be added or removed by the user. The communication to add or remove a bookmark will be handled by an XML message sent via JavaScript to the backend script.

---

	Column Name	Data Type	Allow Nulls
	id	int	<input type="checkbox"/>
	userID	int	<input type="checkbox"/>
	title	varchar(50)	<input checked="" type="checkbox"/>
	URL	varchar(255)	<input type="checkbox"/>
	browserID	int	<input type="checkbox"/>

---

Figure 5: Environment Table

## Video Bookmarks Table

The Video Bookmarks table will work exactly like the bookmarks table except instead of storing the bookmark URL it will store the embedded object tag as seen on many flash video sites. For example:

---

```
<object width="425" height="355">
  <param name="movie" value="http://www.youtube.com/v/KCU8L9hs-Hg&rel=1"></param>
  <param name="wmode" value="transparent"></param>
  <embed src="http://www.youtube.com/v/KCU8L9hs-Hg&rel=1" type="application/x-shockwave-flash"
    wmode="transparent" width="425" height="355"></embed>
</object>
```

	Column Name	Data Type	Allow Nulls
	id	int	<input type="checkbox"/>
	userID	int	<input type="checkbox"/>
	title	varchar(50)	<input type="checkbox"/>
	ObjectEmbed	text	<input type="checkbox"/>

---

Figure 6: Embedded Object Example and Video Bookmarks Table

The XML tags shown in Figure 6 will be stored in the ObjectEmbed column in this table.

## Summarize

To summarize the database strategy, MyEnv will be using four simple database tables in order to store the users profile and environment. The database will be interfaced using a server side ASP/PHP script. The script will receive commands from the client side JavaScript in XML format and respond in XML format. In this fashion, the client side JavaScript need not have direct access to the database but rather via AJAX (Asynchronous JavaScript and XML). The AJAX documentation will be explained in the next chapter.

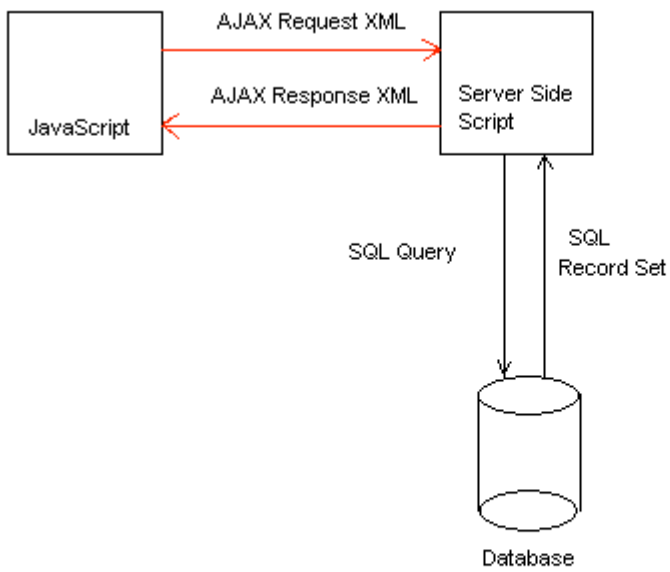
## CHAPTER 5

### ***MyEnv XML Specification***

#### **Overview**

MyEnv will utilize AJAX to communicate with the database using a server side script as the translator. The client side JavaScript will send XML messages to the server side script via HTTP POST; the server side script will then make the appropriate database calls and return a confirmation or a result set back to the client side JavaScript in XML format (*please see Figure 7 below*). The term XML is used loosely here so that it falls within the definition of AJAX. However, the system can and will utilize JSON messages. JSON is much more efficient when interpreting on the client-side via JavaScript. For clarity, the following document will specify the XML format that MyEnv will use. However, the implementation may use JSON in its place.

The diagram below will serve as a reference for the rest of the document. The SQL query is a query to the database. The SQL record set is the filtered results the database would return to a calling process based on the query. The transformation of the XML into SQL query will be covered in the next chapter.



---

Figure 7: System diagram showing AJAX communication

## **AJAX Request/Response XML**

There will be seven different XML requests and its corresponding response that could be sent and received respectively by the JavaScript client and finally an error response that the server side script may return.

1. Add a bookmark
2. Add a video bookmark
3. Retrieve bookmark URL
4. Retrieve video bookmark embed code
5. Save environment
6. Retrieve last environment
7. Error response

## Add a Bookmark

---

Request:

```
<myENV>
  <request>
    <addBookmark>
      <userID> userID </userID>
      <title> bookmark Title </title>
      <url> bookmark URL </url>
    </addBookmark>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <addBookmark>
      <bookmarkID> bookmarkID </bookmarkID>
      <title> bookmarkTitle </title>
    </addBookmark>
  </response>
</myEnv>
```

---

Figure 8: XML request and response for adding a bookmark

## Add a Video Bookmark

---

Request:

```
<myENV>
  <request>
    <addVideoBookmark>
      <userID> userID </userID>
      <title> bookmark Title </title>
      <objectEmbed><![CDATA[ Object Embed Text ]]</objectEmbed>
    </addVideoBookmark>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <addVideoBookmark>
      <videoBookmarkID> video bookmarkID </videoBookmarkID>
      <title> video bookmarkTitle </title>
    </addVideoBookmark>
  </response>
</myEnv>
```

---

Figure 9: XML request and response for adding a video bookmark

## Retrieve bookmark URL

---

Request:

```
<myENV>
  <request>
    <getBookmark>
      <userID> userID </userID>
      <bookmarkID> bookmarkID </bookmarkID>
    </getBookmark>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <getBookmark>
      <title> bookmark Title </title>
      <url> bookmark URL </url>
    </getBookmark>
  </response>
</myENV>
```

---

Figure 10: XML request and response for retrieving bookmark URL

## Retrieve Video Bookmark Embed Code

---

Request:

```
<myENV>
  <request>
    <getVideoBookmark>
      <userID> userID </userID>
      <videoBookmarkID> video bookmarkID </videoBookmarkID>
    </getVideoBookmark>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <getVideoBookmark>
      <title> video bookmark Title </title>
      <embedCode> embedCode </embedCode>
    </getVideoBookmark>
  </response>
</myENV>
```

---

Figure 11: XML request and response for retrieving video bookmark

## Request to Save Environment

---

Request:

```
<myENV>
  <request>
    <saveEnv>
      <userID> userID </userID>
      <windowList>
        <window> **
          <type> window type </type>
          <browserID> browser ID </browserID>
          <xPos> window x Position </xPos>
          <yPos> window y Position </yPos>
          <zPos> window z Position </zPos>
          <width> window width </width>
          <height> window height </height>
          <URL> current URL </URL>
          <code> current embed code </code>
        </window>
      </windowList>
    </saveEnv>
  </request>
</myENV>
```

\*\* The window node occurrence is 1-n

---

Response:

```
<myENV>
  <response>
    <saveEnv>
      <userID> userID </userID>
      <browserID> browserID </browserID>
      <status> save status </status>
    </saveEnv>
  </response>
</myENV>
```

---

Figure 12: XML request to save environment

*Note: In the case of save environment request any response other than an error would conclude that the save went without error however returning the userID and a status doesn't hurt.*

## Request to retrieve last environment

---

Request:

```
<myENV>
  <request>
    <getEnv>
      <userID> userID </userID>
    </getEnv>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <getEnv>
      <userID> userID </userID>
      <windowList>
        <window> **
          <type> window type </type>
          <xPos> window x Position </xPos>
          <yPos> window y Position </yPos>
          <zPos> window z Position </zPos>
          <width> window width </width>
          <height> window height </height>
          <URL> current URL </URL>
          <code> current embed code </code>
        </window>
      </windowList>
    </getEnv>
  </response>
</myENV>
```

---

Figure 13: XML request and response for retrieving last environment

## Request to Retrieve Last Browser URL

---

Request:

```
<myENV>
  <request>
    <getURL>
      <userID> userID </userID>
      <browserID> browserID </browserID>
    </getURL>
  </request>
</myENV>
```

---

Response:

```
<myENV>
  <response>
    <getURL>
      <userID> userID </userID>
      <URL> current URL </URL>
    </getURL>
  </response>
</myENV>
```

---

Figure 14: XML request and response for retrieving last browser URL

## Request to Remove Browser

---

```
<myEnv>
  <request>
    <remove>
      <windowList>
        <window>
          <browserID> browserID </browserID>
        </window>
      </windowList>
    </remove>
  </request>
</myEnv>
```

---

Figure 15: XML request to remove a browser

## Error response

---

```
<myENV>
  <response>
    <error>
      <code> error code </code>
      <desc> error description </description>
    </error>
  </response>
</myENV>
```

---

Figure 16: XML error response

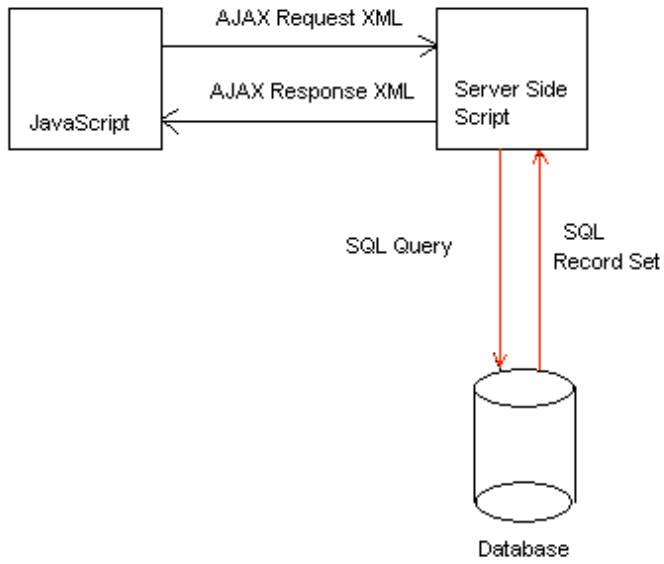
## CHAPTER 6

### ***MyEnv XML to SQL Mapping***

#### **Overview**

The AJAX communication between the client side JavaScript and the server side script will be in the form of XML, which is specified in Chapter 3. This chapter shows how the server side script will translate the incoming XML into SQL to enter and retrieve data from the database. The XML schema has been set up to allow an easy translation into the SQL query statement and therefore will not require the server side script to do much work to translate over to SQL.

In the diagram below it is the link between the Server Side Script and the database:



---

Figure 16: Diagram showing server side script communication with database

## Add a Bookmark

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <addBookmark>
      <userID> userID </userID>
      <title> bookmark Title </title>
      <url> bookmark URL </url>
    </addBookmark>
  </request>
</myENV>
```

SQL translation:

```
insert into bookmarks (userid, title, url) values( userID, bookmark Title, URL)
```

---

Figure 17: Add bookmark xml to sql

*Note: the primary key for the bookmarks table is an increment column, so we will not specify it here. But it will be added automatically into the database for indexing purposes.*

Assuming that the insert succeeded, the server side script will return a success message back to the JavaScript client.

## Add a Video Bookmark

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <addVideoBookmark>
      <userID> userID </userID>
      <title> bookmark Title </title>
      <objectEmbed><![CDATA[ Object Embed Text ]]</objectEmbed>
    </addVideoBookmark>
  </request>
</myENV>
```

SQL translation:

```
insert into videoBookmarks (userID, title, ObjectEmbed) values ( userID, bookmark Title,
Object Embed Text)
```

---

Figure 18: Add video bookmark xml to sql

## Retrieve Bookmark URL

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <getBookmark>
      <userID> userID </userID>
      <bookmarkID> bookmarkID </bookmarkID>
    </getBookmark>
  </request>
</myENV>
```

SQL Translation:

```
select title, url from bookmarks where userID="userID" and id="bookmarkID"
```

---

Figure 19: Retrieve video bookmark XML to SQL

## Retrieve Video Bookmark

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <getVideoBookmark>
      <userID> userID </userID>
      <videoBookmarkID> video bookmarkID </videoBookmarkID>
    </getVideoBookmark>
  </request>
</myENV>
```

SQL Translation:

```
select title, objectEmbed from videoBookmarks where userID=" userID" and id=" video bookmarkID"
```

---

Figure 20: Retrieve video bookmark XML to SQL

## Request to Save Environment

The save environment XML request will require some extra processing on the server side script. Since this is not a one to one function, meaning a single XML block in this case translates into one-to-many insert statements. The multiple insert statements will vary depending on what type of window we are saving. Both cases are shown below:

For window type browser:

---

```
insert into environment
( userID, windowType, xPosition, yPosition, zPosition, width, height, URL, ObjectEmbed)
values
( userID, window type, window x Position, window y Position, window z Position, window width, window height, current URL, null )
```

---

Figure 21: SQL statement to save environment (web browser)

Notice that we send null for the last parameter. The reason is the window type is of browser, so it will not contain any video embedded code and therefore will have to pass a

value of *null* to the database. The next query will do the exact opposite in order to bypass having to send a URL.

---

```
insert into environment
( userID, windowType, xPosition, yPosition, zPosition, width, height, URL, ObjectEmbed)
values
( userID, window type, window x Position, window y Position, window z Position, window width, window
height,null,current embed code )
```

---

Figure 22: SQL statement to save environment (video browser)

## Request to Retrieve Last Environment

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <getEnv>
      <userID> userID </userID>
    </getEnv>
  </request>
</myENV>
```

SQL translation:

```
select * from environment where userID="userID"
```

---

Figure 23: Retrieve last environment XML to SQL

In this case 0-n rows will be returned back from the database and therefore the server side script will have to compile those rows into a single XML statement to return to the front side JavaScript.

## Request Last Browser URL

---

Client side JavaScript XML Request:

```
<myENV>
  <request>
    <getURL>
      <userID> userID </userID>
      <browserID> browserID </browserID>
    </getURL>
  </request>
</myENV>
```

SQL translation:

```
select URL from bookmarks where userID="userID" and browserID="browserID" where  
id=(select max(id) from bookmarks where userID="userID" and browserID="browserID" )
```

---

Figure 24: Request last browser URL

## CHAPTER 7

### ***Project Problems and Solutions***

#### **Overview**

This chapter will discuss some of the problems that were encountered when implementing this project. It will go over different methods of solving each and the best possible solution will be selected. Since the MyEnv virtual desktop is implemented using DHTML, we are limited in what can be done to emulate a desktop environment. For example, JavaScript Security on modern browsers, this does not allow us to poll the current location of any iframe. If security does not allow us to record the location of the iframes then we cannot save a user's desktop state. Different solutions to these problems will be discussed in this chapter.

#### **JavaScript Security**

One of the major problems of this project is the ability to track the URLs that a user visits using the iframe browsers. If you take a look at the JavaScript API for an iframe, it looks trivial we simply have to reference the location attribute of the particular iframe in question. There are multiple ways to try and reference the iframe location; the following shows three examples:

1. `frames['myiframe'].location.href`
2. `frames['myiframe'].location`
3. `frames['myiframe'].src`

The first example is the traditional way of getting the href, or hyperlink reference; this has been deprecated in recent browsers to use the second method, which is simply referencing the location attribute to return the URL. If the iframe's content is on the same web server as the parent frame, then we can access the iframe location. For example, if the parent frame, which is the browser frame, is pointing to <http://www.myenv.com> and the child frame, the iframe in question, is pointing to <http://www.myenv.com/subpage.html> then JavaScript allows us to either use method 1 or method 2 from the list above to retrieve the iframe's location. However, if the parent frame is pointing to <http://www.myenv.com> and the iframe's location is <http://www.google.com/subpage.html> then we will receive a JavaScript security error. This is referred to as a cross domain security violation<sup>[4]</sup>. This security violation protects us from being at the mercy of web hackers. If for example a website full real-estate is taken up by a single iframe then a all URLs that a browser visits can potentially be tracked by the iframe's creator using JavaScript without the user ever knowing that they were being spied on.

The third example above simply returns to use the default location of the iframe. Take the following example of an iframe instantiation in HTML.

---

```
<iframe src="iframe_page.html" id="myiframe"></iframe>
```

---

Figure 25: iFrame Example

The iframe's default location is set to "iframe\_page.html" and therefore any request to the third method; frames['myiframe'].src would then return us this value. Any subsequent clicks in the iframe that navigates the user to different pages would not affect the src attribute; it will always return the default location.

Therefore, since we cannot access the current location of an iframe, this causes a major problem with the development of the MyEnv system. Since the system must be able to emulate a desktop of multiple web browsers, it must not only know the location of one browser window but all browser windows. Furthermore, it must also know the current location and previous locations if they exist. This functionality would give the user the ability to use the back and forward buttons on each virtual browser. However, since we cannot access the iframe location, we will have to intercept the user's path as they click through the iframe. The solution in this project will be to make sure that every link in the iframe points to our domain (in this example www.myenv.com), by pre-pending a relative URL to the beginning of every URL on the page. For example, if a URL on the page points to <http://www.amazon.com/page1.html> then we can simply add to the beginning a path to our domain for example <http://www.myenv.com/pagegetter/http://www.amazon.com/page1.html> the fact that the URL in the iframe window and the parent window are of the same domain is enough for JavaScript to allow us to query the iframe location. Notice that in the URL we have a subfolder called "pagegetter". This is in fact a script which lives on our web server that takes one argument which is the remainder of the URL. So in our example above we have the script pagegetter taking in an argument of <http://www.amazon.com/page1.html>.

The pagegetter script will retrieve the HTML from its argument URL and then have to go through the page and make sure to prepend our domain URL to the beginning of any hyperlink reference on the page. The process of prepending our URL to every link on the page is covered in the next chapter.

## Hyperlink Injection

The solution proposed by this project to get around the JavaScript iframe security is to build a script to modify all the hyperlinks on an HTML page to reference the local domain. What this means is that if a user browses to a site using one of the virtual browsers, we first have to intercept the event and use a server-side script to perform a GET on that page and pull the HTML text back in order to then modify it. This technique is often referred to as HTTP tunneling. Only certain tags will have to be dealt with. The following is a list of the tags in question:

- Image <img>
- CSS include <link>
- Anchor Tag <a>
- JavaScript Include <script>

A Python package called BeautifulSoup<sup>[2]</sup> will be used to read in the HTML text into a DOM object in order to loop through the tags and modify them. Regular expressions could have also been used here, however would be more difficult to read and maintain, since BeautifulSoup provides a more object oriented approach. The image, CSS and JavaScript include tags can be dealt with using the same logic; both may or may not be referencing their source documents with an absolute URL. If they are referenced with an

absolute URL then we can leave them alone, since they will still be able to find their respective files. However, if they reference a relative URL, we must make sure to prepend the correct domain and path in order to make the URL absolute so that it can be referenced on the user's machine. The following is an example of the logic for the anchor tag:

---

```
for a in soup.findAll('a'):
    #print >>sys.stderr, a['href']
    try:
        if a['href'].startswith(" http://"): #or ftp afp!
            a['href'] = "/pagegetter/" + a['href']
        else:
            addSlash = "/"
            if a['href'].startswith("/"):
                addSlash = ""
            path, src = self.parentDir(path, a['href'])
            a['href'] = "/pagegetter/" + scheme + "://" + domain + path + addSlash + a['href']
    except KeyError, e:
        print >>sys.stdout, ""
```

---

Figure 26: Logic for anchor manipulation

The anchor tag must be dealt with differently, since the system must record clicks through the anchor tags. We must prepend not only the absolute path of its href attribute but also prepend a relative path to our local domain as well. What this does is point the link to our local server side script called pagegetter. When the user then clicks on the link it will call the pagegetter script with the absolute URL of the destination page as an argument. The pagegetter script will then take the absolute URL and retrieve the page making all of the changes described above and then serve it to the user's virtual browser. By going through this process we solve two problems: One is that we can record where the user has been and is going; and the other is that we fool the client side JavaScript into giving us the iframe's location without throwing a security violation. The next step is to

store the browsers past and current location in the database so that the virtual browser's back and forward buttons will have references on where to set the iframe location when they are clicked.

## **The Browser History Problem**

This chapter will deal with the functionality of the back and forward buttons on the virtual browsers. The idea is to mimic the functionality of a regular desktop browser. In other words, the browser's back button would point "back" to the referring page in which the user left to enter their current page and in the same way the browser's forward button would move them "forward", which is really another form of moving back just in the opposite direction, to the page they were on before they clicked the back button.

On a desktop web browser, the list of the browsers referring pages is kept in memory and therefore in order to mimic this behavior directly each virtual browser must have its own list of pages in history. Since the virtual browser is not a desktop application, it is functioning inside a web page which means it is stateless. Therefore, we cannot simply use the parent browser's back and forward buttons because that will destroy the browser's state. There are two methods to solve this problem: One is to store an array of links for each browser in JavaScript; and the other is to save the history of visited links in the database.

The first option, which is to store the history in JavaScript, would mean that we would not have to make a trip to the database immediately to save the referring link. As the user clicks through the hyperlinks the array of history will grow in memory and therefore the user can then click the back or forward buttons to move through the history array. The problem with this method is that at some point the history array needs to be saved in the database. If the user leaves the MyEnv website and logs in at a later time that history has to be available. However, with the nature of JavaScript if we do not save the history on the server side then when the user logs into a new machine a new JavaScript session is created and hence a new empty history array. Therefore, in order to make sure that we save the history array per browser the JavaScript must transfer the history array content to the database on logout and on the event that the browser window is closed. And in the same way the JavaScript must retrieve the history information when a user logs back into their session. The benefit of this is that the client side JavaScript will not have to make a round trip to the database each time the user clicks a hyperlink, the disadvantage is that upon login and logout the JavaScript will have to store the array history for each browser in the database.

The second option is more trivial. It states that each and every time a user clicks through a hyperlink and thus adds to the history array that the JavaScript makes a request to the database to update the array history. This means that instead of the array history being stored on the front end in JavaScript memory that it be stored exclusively in the database. This will require the client side JavaScript to make an AJAX call each and every time the user clicks a hyperlink or either of the back or forward buttons. This method will put

more strain on the server since the number of calls to the database will be increased. However, the amount of information in each request will be much smaller on average than the amount that needs to be transmitted in the first method. This method will also assure the array history will be in tact even if the user's browser accidentally crashes or if the JavaScript window close event doesn't run. And since the JavaScript already needs to make AJAX requests to the database every time a browser is resized or moved, this method stays within the theme of the project, which is the server side database always has the latest environment.

To summarize, this project will be initially implemented using the second method, for reason's stated above. The client side JavaScript will make requests upon each back and forward button request in order to update and retrieve each browsers history array. This will cause some extra overhead for the server. However, it will ensure that the user's session will stay intact.

### **The Form POST and Session Problem**

The previous section explained how each anchor link presented in the iframe browsers will be intercepted and interpreted by our server side script. Therefore, all HTML will therefore reside locally on the MyEnv web server before it is rendered to the user's browser. This works great for a static HTML page however poses a problem for dynamic pages that rely on server side scripting. Not all dynamic pages will be a problem. For example, if an ASP page is set to render a select box in HTML, the select box would have already been rendered into HTML before the pagegetter script downloads it locally.

However let's assume that an ASP page on load assigns variables to its session and we point our pagegetter script to retrieve the page in question. Once pagegetter makes a request to the page, the session variables would then reside on the remote web server and not on our local MyEnv web server. Therefore any subsequent page that makes use of these Session variables will defiantly fail. This may not be a problem if the remote server identifies the pagegetter script as a regular browser and handles its session correctly and if only one user on the MyEnv server is browsing the same domain. However, if multiple users are logged into MyEnv and browsing the same remote server then the sessions will be shared among these separate users, since all users are using the same instance of the pagegetter script to retrieve HTML from the remote server. A solution to this may be to fire off a new instance of the pagegetter script for each user or to get more granular so sessions are not shared between iframe browsers a new instance could be created for each browser in the user's environment.

Dynamic server side scripting is not limited to sessions. Another large problem stands in the way of MyEnv being a complete browser environment, which is the inability to handle form submits. An HTML form tag includes an attribute called "action." The action attribute tells the browser where to forward the information collected on the form when the user clicks the submit button. The action attribute value is normally a URL which includes a server side script file that would then know what to do with the submitted information. For example: <http://www.mydomain.com/handleSubmit.asp> .

In the above example handleSubmit.asp would contain logic to be run on the form variables that were submitted, it may for example insert those values into a database or

perform some calculation on the values and return the results to the screen. In either case the logic that takes place is all contained within the server side script. Therefore if the pagegetter script was set to download the page the form resided on it would not then be able to also download the form with the logic since the code is not visible it cannot be reproduced locally. This is where MyEnv fails, when submitting to a dynamic form, which is the case of most forms on the internet.

## CHAPTER 8

### ***Conclusion***

As the world moves back to the thin client, the idea of a web based virtual desktop may prove useful. It may be most useful in a school environment in which each student has their own environment, which includes bookmarks, web and video browsers. This model will not require the student to have full access to any operating system desktop, rather they can simply log in via a web form to retrieve their environment.

In order to create a useful environment the single webpage would have to mimic a desktop. With the use of the modern browser and JavaScript this could be accomplished. The difficulty lies in the ability to present the user a state full web-browsing environment. A breadcrumb trail would have to be kept for each virtual browser and in order to do so each virtual browser, which is comprised of an iframe, would have to report its current location each time the user browses to a new URL. Since retrieving the URL of an iframe is a JavaScript security violation a work around is required. Since JavaScript does allow retrieval of a local URL within the host's domain the work around is to retrieve the remote html and render it locally, and thereby we can create a breadcrumb trail.

By saving the users session and saving the browser history for each browser we can then provide users the ability to log in create an environment and expect that environment to be available to them at any terminal which has a useable internet connection and a modern browser that supports JavaScript.

The problem lies in server side scripting on remote web servers. Since the dynamic server side code is only available to the server it runs on we cannot download this code and run it locally. Furthermore the server side code may have access to a local database which the MyEnv server will not have access too. Therefore any type of form post cannot be supported in the MyEnv environment and hence in order to mimic a desktop browser environment users will be limited to static html pages. This may be useful for school environments in which only static text articles and videos are served for students to research on for example a library or online school newspaper.

## References

- [1] CodeLifter.com - Accent JavaScript Browser 1.0 Kit  
<<http://www.codelifter.com/main/javascript/accentbrowser1.html>>
- [2] Beautiful Soup  
<<http://www.crummy.com/software/BeautifulSoup/>>
- [3] Benjamin, Dan. How to install Mysql on Mac OSX Leopard 10 November 2007  
<<http://hivelogic.com/articles/installing-mysql-on-mac-os-x/>>
- [4] D, Steven. Beating the Browser's Iframe Security. 11 November 2006  
<<http://www.elctech.com/2006/11/2/javascript-communication-between-2-iframes-from-different-domains>>
- [5] Eshaghi, Edvin. Personal Interview. 13 March 2008.
- [6] JavaScript xml DOM parsing  
<[http://www.w3schools.com/xml/xml\\_parser.asp](http://www.w3schools.com/xml/xml_parser.asp)>
- [7] McCullough, Peyton. Using SQLite in Python. 12 April 2006  
<<http://www.devshed.com/c/a/Python/Using-SQLite-in-Python/>>
- [8] Ratter, Robert. SQL object tutorial for consolidating library into objects. 18 Mar 2008.  
<<http://www.sqlobject.org/SQLObject.html#using-sqlobject-an-introduction>>
- [9] Swartz, Aaron. Python Session Setup.  
<<http://webpy.org/sessions/install>>
- [10] Swartz, Aaron. Webpy Tutorial the development web server.  
<<http://webpy.org/tutorial2.en>>
- [11] The Yahoo! User Interface Library (YUI)  
<<http://developer.yahoo.com/yui/>>
- [12] YouOS A Web Operating System  
<<https://www.youos.com/>>
- [13] Teague, Jason Cranford. CSS, DHTML & AJAX Fourth Edition.