Web Mashups using Firefox plugin

Vijay Rao

CS297 Report

for

Prof. Pollet

Web Mashups using Firefox plugin

1 Introduction

The term Web Mashups was coined to describe applications that rely on data that is externally available; sometimes freely in terms of RSS feeds or webservices and can be used in another application thus creating a new type of application that is of tremendous value. As more users use the web there is a need for the user to be able to see the data in a multitude of different ways. A user viewing school API ranking information would like to see them with the house listings to see a house in a neighborhood of a good school. Another user would like to see the Megan's law database plotted on Google Maps to see sex offenders in his or her neighborhood. The options to see the data in context of other data is becoming more obvious and necessary.

I propose a Firefox application using XULRunner installed on the user's desktop and a firefox plugin that works in tandem allowing users to save portions or snippets of a web page. It will also allow users to add events to their calendar or add contacts by right clicking and selecting the text on a web page. Saved snippets which have addresses or locations on them can be viewed on Google Maps. The users can tag the snippets they want to save and categorize them. This will allow users to mashup various sources of information together and view them in a consistent manner. The pages that previously had to be viewed independently can now be viewed in context of other pages.

2 Technology Involved

Client side technologies have improved a lot and modern browsers such as Firefox have a great framework in place to develop and deploy client side applications. Firefox has an elaborate framework in place to develop web based applications as well as client side applications.

Javascript

JavaScript is a client side scripting language that runs in all modern browsers today. It is run on the client machine and can be used to massage, validate and transfer data to the server. With the new way of sending data to the server called AJAX JavaScript has become more popular. Object Oriented JavaScript is also getting more mainstream as programmers are now more comfortable than before.

XUL

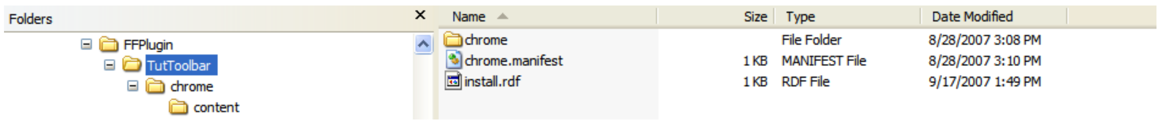
Firefox has a rich architecture where custom extensions and additions can be written with its extension framework. The framework uses various components such as XUL (XML User Interface Language), XPCOM (Cross Platform Component Object Model) and XPConnect (allows connecting to XPCOM objects). XUL is an XML based user interface language that can be customized with custom graphics and layout that can be used to develop any internet or non-internet based applications. XUL can be controlled and manipulated using JavaScript and DHTML.

3 Deliverable 1 – Sample Firefox plugin

The first deliverable was to create a sample Firefox test plugin. I created a test plugin that allowed the user to enter a search criteria and the results were displayed. The following steps were done to create a test firefox plugin

3.1.1 Step 1 – Creating the necessary directory structure

The first step to creating a firefox plugin is to create the right directory structure.



Name	Size	Type	Date Modified
chrome		File Folder	8/28/2007 3:08 PM
chrome.manifest	1 KB	MANIFEST File	8/28/2007 3:10 PM
install.rdf	1 KB	RDF File	9/17/2007 1:49 PM

Fig 1 – Directory Structure for Firefox Plugin

3.1.2 Step 2 – Creating the install.rdf

The installer manifest is read by Firefox to get details about our extension. Here is the install.rdf

```
<?xml version="1.0"?>
```

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
    <Description about="urn:mozilla:install-manifest">
        <!-- Required Items -->
        <em:id>tuttoolbar@borngeek</em:id>
        <em:name>Tutorial Toolbar</em:name>
        <em:version>1.0</em:version>
        <em:targetApplication>
            <Description>
                <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
                <em:minVersion>1.5</em:minVersion>
                <em:maxVersion>2.0.0.*</em:maxVersion>
            </Description>
        </em:targetApplication>
        <!-- Optional Items -->
        <em:creator>Vijay Rao</em:creator>
        <em:description>An example toolbar extension.</em:description>
<em:homepageURL>http://www.borngeek.com/firefox/</em:homepageURL>
    </Description>
</RDF>
```

Listing 1 – install.rdf

3.1.3 Step 3 – Creating the chrome.manifest

The chrome manifest tells Firefox what packages and overlays is provided by our extension. Overlays are layers and extensions added to Firefox.

```
content tuttoolbar chrome/content/
```

```
overlay chrome://browser/content/browser.xul

chrome://tuttoolbar/content/tuttoolbar.xul
```

Listing 2 – chrome.manifest

3.1.4 Step 3 – Creating the xul files

The xul file describes the layout and position of the toolbar. It contains various navigation and menu items that provide functionality to the toolbar

The following is the root element of the overlay file

```
<?xml version="1.0"?>

<overlay id="TutTB-Overlay"

xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Listing 3 – tuttollbar.xul – overlay

The following describes a toolbar item with menu dropdowns

```
<toolbaritem flex="0">

      <toolbarbutton id="TutTB-MainMenu" type="menu"

                               tooltiptext="Tutorial Toolbar Main

Menu">

      <menupopup>

            <menuitem label="Google Home Page" accesskey="G"

                                  tooltiptext="Navigate to Google"

oncommand="TutTB_LoadURL('http://www.google.com/')" />

                        <menuseparator />

                        <menuitem label="Born Geek Website"

accesskey="B"

   tooltiptext="Navigate to Born Geek"
```

```
oncommand="TutTB_LoadURL('http://www.borngeek.com/')" />
                    </menupopup>
                </toolbarbutton>
</toolbaritem>
```

Listing 3 – tuttoolbar.xul

4 The various XUL elements are added to XUL file to make up the UI component.

4.1.1 Step 4 – Scripting the Toolbar

The following line at the top of the toolbar attaches a javascript file to the xul overlay. Javascript can now be used to select the xul elements with id the same way as DHTML.

The following piece of code gathers the search terms and passes it to google

```
function TutTB_Search(event, type)
{
    // This variable will hold the URL we will browse to
    var URL = "";
    // This variable will tell us whether our search box is empty or
not
    var isEmpty = false;

    // Get a handle to our search terms box (the <menulist> element)
    var searchTermsBox = document.getElementById("TutTB-SearchTerms");
// Get the value in the search terms box, trimming whitespace as
```

```
necessary
    // See the TutTB_TrimString() function farther down in this file
for details
    // on how it works.
    var searchTerms = TutTB_TrimString(searchTermsBox.value);

    if(searchTerms.length == 0) // Is the search terms box empty?
        isEmpty = true;         // If so, set the isEmpty flag to true
    else                        // If not, convert the terms to a URL-
safe string
        searchTerms = TutTB_ConvertTermsToURI(searchTerms);
    // Now switch on whatever the incoming type value is
    // If the search box is empty, we simply redirect the user to the
appropriate
    // place at the Google website. Otherwise, we search for whatever
they entered.
    switch(type)
    {
    // Build up the URL for an image search
    case "image":
        if(isEmpty) { URL = "http://images.google.com/"; }
        else        { URL = "http://images.google.com/images?q=" +
searchTerms; }
        break;
    // Build up the URL for a web search
    case "web":
    default:
        if(isEmpty) { URL = "http://www.google.com/"; }
        else        { URL = "http://www.google.com/search?q=" +
searchTerms; }
        break;
    }
    // Load the URL in the browser window using the TutTB_LoadURL
function
    TutTB_LoadURL(URL);
}
```

Listing 4 – tuttoolbar.js

5 Deliverable 2 – DOM Test

The second deliverable was to create a test HTML DOM and capture various mouse events such as mouseover and mouseout. A test HTML page was created with various block level elements such as <div>, <p> and <table>. A test javascript function was called onLoad event of the HTML page. This function registered the mousedown and the mouseup event so that users selection block that includes the start element and the end element is captured. The DOM nodes were then iterated over selecting the parent node till the <body> node was encountered.

The path was then alerted to the user. The test was done to determine the uppermost block level element that can be saved. The following listing shows the piece of code that traverses the DOM looking for block level elements and then prompts the user

```
                  function calculatePath(tgt) {

                        arr = new Array();

                        //tgt.setCapture();

                        //alert("int tgt " + tgt);

                        path=tgt.tagName+" ";

                        //alert(path);

                        while(tgt.tagName != 'BODY') {

                              //alert("inside");

                              arr.push(tgt.tagName);

                              path=path+ tgt.parentNode.tagName +" --
";

                              tgt = tgt.parentNode;

                              //alert(path);

                        }

                        arr.push('BODY');

                        //alert(arr);

                        return arr;


                  }


                  function capturePathUp(upNode) {

                        var selection;

                        if(document.selection) {

                              selection = document.selection

                        } else {

                              selection = window.getSelection();

                              alert(selection.getRangeAt(0));

                        }
                        //uparr = calculatePath(upNode);

                        //alert("path " + startarr.join(' | '));

                        /*var lastpop;

                        if(downarr.length > uparr.length) {

                              for(i = downarr.length; i >=
uparr.length; i--) {

                                    lastpop = downarr.pop();

                              }

                        } else if(downarr.length < uparr.length){

                        } else {

                              pickParentBlockElem(uparr);

                        }*/

                  }
```

Listing 5 -

6 Deliverable 3 – Block Element Highlight

The third deliverable was to create a test HTML DOM and highlight block sections such as div and table elements on mouse movements. The block elements hovered over by the mouse were given a red solid border to highlight the block element. The following listing shows the piece of code that added this functionality

```
                        var all = document.getElementsByTagName("div");

                        for ( var i = 0; i < all.length; i++ ) {

                              // Watch for when the user moves his
mouse over the element

                              // and add a red border around the
element
                              all[i].onmouseover = function(e) {
                                    this.style.border = "2px solid
red";
                                    stopBubble( e );
                              };
                        // Watch for when the user moves back out of
the element
                        // and remove the border that we added
                              all[i].onmouseout = function(e) {
                                    this.style.border = "0px";
                                    stopBubble( e );
                              };
                        }
```

Listing 6 – highlighting block elements

7 Deliverable 4 – Saving web page contents on desktop

The fourth deliverable was to save some component of the web page on the users desktop. A lot of inbuilt Firefox services already exist as components which can be utilized. I found a utility at [6] which had the interfaces for writing the contents to a specified location. Here is a piece of code that writes the passed content to a file. This was embedded in the sample plugin to save the contents of the <body> tag.

```
function save(mydata) {
      //mydata = 'hELLO VIJAY';
      alert("in save " + mydata);

      try {


      netscape.security.PrivilegeManager.enablePrivilege("UniversalXPCo
nnect");

      } catch (e) {

            alert("Permission to save file was denied.");

      }

      var file = Components.classes["@mozilla.org/file/local;1"]

            .createInstance(Components.interfaces.nsILocalFile);

      file.initWithPath( savefile );

      if ( file.exists() == false ) {

            alert( "Creating file... " );

            file.create(
Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 420 );

      }

      var outputStream = Components.classes["@mozilla.org/network/file-
output-stream;1"]

            .createInstance( Components.interfaces.nsIFileOutputStream
);

      /* Open flags

      #define PR_RDONLY       0x01

      #define PR_WRONLY       0x02

      #define PR_RDWR         0x04

      #define PR_CREATE_FILE  0x08

      #define PR_APPEND      0x10

      #define PR_TRUNCATE     0x20

      #define PR_SYNC         0x40
      #define PR_EXCL         0x80
      */
      /*
      ** File modes ....
      **
      ** CAVEAT: 'mode' is currently only applicable on UNIX platforms.
      ** The 'mode' argument may be ignored by PR_Open on other
platforms.
      **
      **   00400   Read by owner.
      **   00200   Write by owner.
      **   00100   Execute (search if a directory) by owner.
      **   00040   Read by group.
      **   00020   Write by group.
      **   00010   Execute by group.
      **   00004   Read by others.
      **   00002   Write by others
      **   00001   Execute by others.
      **
      */
      outputStream.init( file, 0x04 | 0x08 | 0x20, 420, 0 );
      //var output = document.getElementById('blog').value;
      alert( "B4 write... " );
      var result = outputStream.write( mydata, mydata.length );
      alert( "result ... " +  result );
      outputStream.close();


}
```
Listing 7 – saving a html block to file on client machine

8 Conclusion

This semester the four deliverables helped solidify the technology and the resources required to accomplish the final deliverable. This semester involved exploring various technologies and doing a proof of concept to test each technology. All four deliverables were successfully accomplished.

The next semester will involve integrating all the deliverables to create a final product. The final product will be a Firefox plugin that will enable users to save snippets of a web page and then view them or display them on a Google Maps page.

9 Bibliography

1. Kenneth C. Feldt (2007).

   Programming Firefox: Building Rich Internet Applications with XUL.
   Oreilly.

2. Google (n.d.)

   Google Maps API Version 2 Reference Retrieved Aug 30, 2007 from
   http://www.google.com/apis/maps/documentation/reference.html

3. Andre Lewis, Michael Purvis, Jeffrey Sambells, Cameron Turner (2007).

   Beginning Google Maps Applications with Rails and Ajax:From Novice to Professional (Paperback)
   Apress.

4. Jonah Bishop

   Firefox Toolbar tutorial Retrieved Sep 8, 2007 from
   http://www.borngeek.com/firefox/toolbar-tutorial/

5. Joy of XUL Retrieved Sep 8, 2007 from

   http://developer.mozilla.org/en/docs/The_Joy_of_XUL

6. Captain's Mozilla XUL LOG

   Firefox Toolbar tutorial Retrieved Dec 15, 2007 from
   http://www.captain.at/programming/xul/