Total Recall for AJAX applications – Firefox extension

A Writing Project

Presented to

The Faculty of Computer Science

San Jose State University

In Partial Fulfillment of the Requirement for the

Degree

Master of Science

By

Smita Periyapatna

May 2009

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett

Dr. Mark Stamp

Dr. Araya Agustin

ABSTRACT

Ajax, or AJAX (Asynchronous JavaScript and XML), is a group of interrelated web development techniques used to create interactive web applications or rich Internet applications[9]. Web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of an existing web page. [9]

One of the biggest problems with Ajax applications is saving state and accommodating the succession of the history controls, (Back/forward buttons). Ajax allows documents to become stateful, but when the user intuitively goes for the history controls in the browser window, things go wrong. The user expects to see the previous state of the document and is surprised to see a webpage they were on 20 minutes ago, before they arrived at the Ajax application. Our project aims to solve this problem. We have implemented an extension to the Firefox Mozilla browser that caches different states of web pages at regular intervals and displays all the different states of the page as the user navigates through the history.

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF LISTINGS

INDEX OF TABLES

INTRODUCTION

Overview

In AJAX, most of the action takes place inside a single page. When an AJAX page is loaded, new instances of Javascript objects are created. When you leave that page and go to some other page like say Yahoo, the Javascript objects are completely wiped out. When you hit the back button, the page actually reloads completely. All the objects are lost and this can be pain.

First, it is something that not all end users are aware of, which can lead to errors. Second, users see their state completely wiped out; when they go back to their AJAX application with the back button, they see the original state of their program, not the last place they left it. Third, this can affect performance, since the AJAX application has to re-retrieve everything from the server rather than use its local state.

The Project

Our project aims to solve the above-mentioned problems. Our project involves extending the functionality of the browser to save the states of an Ajax page in something similar to storing normal history items. This will allow users to go back to the last place they left it, instead of going to the original state of their AJAX application. It also checks if some states were already saved in the history item, if they were, then those states would not be saved. Our extension will allow users to set a time interval at which they want to save the states of a page. It will also

allow users to save states of a Flex application when developed using our jsChromeBridge API.

Report Overview

This report starts with describing various technologies used to develop the product. The next section gives details about design and implementation of every component of the extension. The design and implementation involves creating the right directory structure, creating the user interface, the preference system and finally implementation of the caching mechanism. It also includes the implementation details of caching on Flex application. The next section gives details about usability testing and testing of the extension on various websites. The report ends with a conclusion.

Technology Used

Our project aims to build an extension to Mozilla Firefox browser.
For our extension, we have used eXtensible User Interface Language (XUL) for creating widgets and written Javascript functions to bind user actions. XUL is a XML grammar to add/modify user interface widgets of the browser. We have also used Mozilla's XPCOM interfaces.

XUL

XUL is an XML based user interface markup language developed by Mozilla. XUL provides a rich set of UI components. XUL can be used to build feature rich cross platform applications. XUL also allows the use of existing web standards and technologies like CSS, JavaScript and DOM.

For my extension, XUL is used to create the user interface (buttons on the tool bar) and the preference system.

Javascript

Javascript is a scripting language used mostly for client side web development. It is also the core scripting language in Mozilla browser. Javascript is used in various levels in Mozilla. A user interface level, which manipulates content through the DOM. A client layer, which calls on the services provided by XPCOM. An application layer is available, in which Javascript is used to create an XPCOM component.

Javascript is used to handle all the user generated events and to communicate with the XPCOM interfaces used in our extension.

XPCOM

XPCOM is a standard cross-platform object model provided by Mozilla that exposes a core set of components and interfaces for component management, file abstraction, object message passing and, memory management. [6]

For our extension XPCOM interfaces were used for the preference system, storing and retrieving of pages from the cache and for resource description framework.

CSS

Cascading Style Sheets (CSS) are used to describe the look and formatting of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can be applied to any kind of XML document, including SVG and XUL.

For our extension CSS was used to style the icons on the toolbar. The icons were taken from “http://www.icons-gallery.com/”

Flex and ActionScript

Our extension supports saving states of a page in a Flex application, written by using our jsChromeBridge API. A test application is developed using Flex and show the saving of states.

Flex is used for development and deployment of cross-platform rich Internet applications. [8] ActionScript is used primarily for the development of websites and software using the Adobe Flash Player platform. [11]

Implementation Details

To save the states of an AJAX page in our extension, we wanted to use a mechanism similar to how normal history items are stored.

Following section gives detailed description of the steps followed to create an extension.

Setting up the directory structure

It is required to setup a proper directory structure before creating an extension. The figure below shows the directory structure

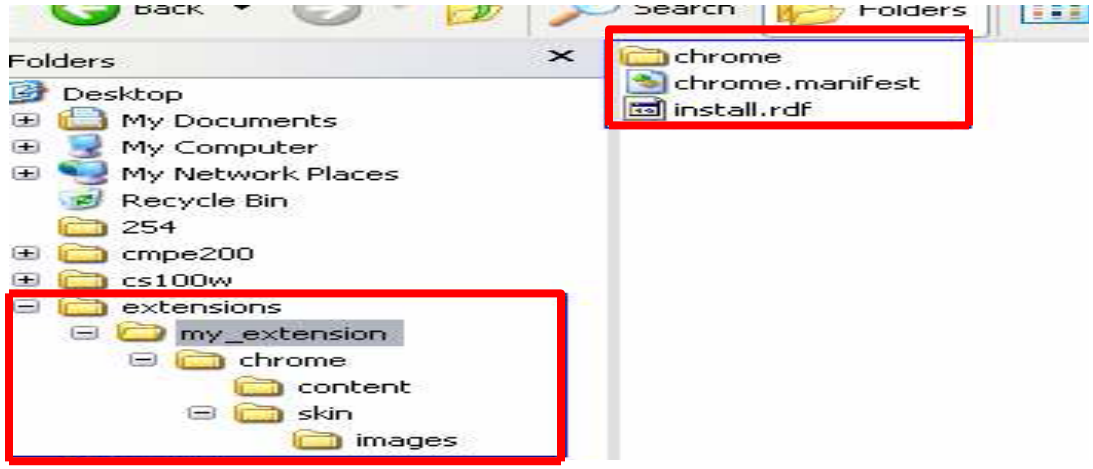

Figure 1: Directory Structure

The install.rdf file is used to determine information about an extension as it is being installed. It includes the metadata identifying the extension, providing information about the author, version, etc. The following is the sample install.rdf file:

```
<?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">
        <em:id>test@mail.com</em:id>
        <em:version>1.0</em:version>
        <em:type>2</em:type>
        <em:targetApplication>
                <Description>
                        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
                        <em:minVersion>1.5</em:minVersion>
                        <em:maxVersion>3.0.*</em:maxVersion>
                </Description>
        </em:targetApplication>

      <! -- Front End MetaData -->
      <em:name>Sample</em:name>
      <em:description>A test extension</em:description>
      <em:creator>Smita</em:creator>
      <em:homepageURL>http://www.mozilla.org</em:homepageURL>
      <em:iconURL>chrome://sample/skin/images/pic.jpg</em:iconURL>
       <em:optionsURL>chrome://sample/content/prefSample.xul</em:optionsURL>
  </Description>
</RDF>
```

Listing 1: Install.rdf

The install.rdf file begins with description of the extension the first one being a string formatted id for the extension. The next one is the version, which identifies the version of the extension being supplied. The type, an integer value, represents the type of add-on. For example (2 for extension, 4 for themes, 8 for locales etc). Next is the target application, an object specifying an application targeted by this add-on. This means that the add-on will work with the application identified by the id property (<em: id>) specified from the minimum version (<em: minVersion>) up to and including the maximum version (<em:maxVersion>). The

above-mentioned properties were not optional. However, there are few properties, which are optional. Such as the name of the addon, description, creator, etc. The <*optionsURL*> specifies the path to extension's options dialog box. This is only useful to extensions. If this property is specified, when the extension is selected in the extensions list, the Options button is enabled and will show this.

The chrome.manifest tells Firefox the location of the chrome packages files and overlays. Overlays attach other UI widgets to XUL documents at run time. The chrome.manifest files also contains the location of the content directory which has the XUL and JavaScript files, Skin which has the images and CSS files and Locale which has the DTD and .properties files. The following is the sample chrome.manifest file:

```
<!--maps chrome://sample/content/ to the content folder, registering
the content provider-->
content sample chrome/content/

<! -- allows modifying Firefox's mail window UI xul file from sample.
xul file -->
overlay chrome://browser/content/browser.xul
chrome://sample/content/sample.xul

<! -- applies test.css file to the customizeToolbar.xul-->
<! -- This is required to create new toolbar buttons -->
style chrome://global/content/customizeToolbar.xul
chrome://sample/skin/test.css

<! -- This is the default skin provider -->
skin sample classic/1.0 chrome/skin/
```

Listing 2: Chrome.manifest

The next step was to create the XUL file. The XUL creates the micro-back/micro-forward, play and stop buttons. All events were handled in Javascript files.

Creating User Interface

The user interface consists of four buttons: Start, Stop, microForward, and microBack.

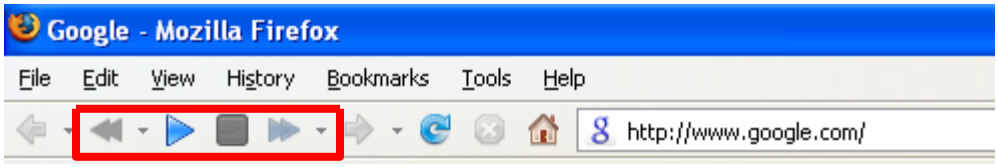

Figure 2: User Interface

Implementation

The following Listing shows the snippet of code used to style the microBack button.

```
#myextension-backbutton {
  list-style-image: url("chrome://sample/skin/back.png");
}
#myextension-backbutton[disabled = true]{
  list-style-image: url("chrome://sample/skin/backbw.png");
}
toolbar[iconsize="small"] #myextension-backbutton {
  list-style-image: url("chrome://sample/skin/back_small.png");
}
toolbar[iconsize="small"] #myextension-backbutton[disabled = true] {
  list-style-image: url("chrome://sample/skin/backbw_small.png");
}
```

Listing 3: CSS file

In the above Listing, myextension-backbutton is the id of the microBack button. The *url("chrome://sample/skin/back.png")* gives the path of the image file used

for microBack button. When the microBack button is disabled the *disable* attribute is set to *true* and a new image file is used for microBack button. The *toolbar[iconsize="small"]* specifies that the icon size is set to “small” and uses respective image file for microBack button.

In the similar manner, all other buttons are stylized.

The microBack button and the microForward button have popup menus similar to the history Back/Forward controls. The Listing 4 shows attributes set for the microBack button and for the popup menu.

```
        //setting attributes of the microBack Button
        back.setAttribute('id',"myextension-backbutton");
        back.setAttribute('label',"myBackButton");
        back.setAttribute('class',"toolbarbutton-1 chromeclass-toolbar-
additional");
        back.setAttribute('tooltiptext',"cacheBackButton");
        back.setAttribute('type',"menu-button");
        back.setAttribute('disabled',"true");

        //Popup menu attributes
        backpopup.setAttribute('id',"backpopup");
        backpopup.setAttribute('context',"");
        back.appendChild(this.backpopup);

        //inserting the microBack button in the toolbar
        var node1 = document.getElementById("myextension-playbutton");
        navbar.insertBefore(back,node1);
```

Listing 4: Code for microBack Button

The menu items in the popup menu are created at runtime when the caching starts.

Preferences

A preference is any value or defined behavior that can be set by the user. Preferences set via user interface, usually a preference dialog, takes effect immediately.

Implementation

The listing below shows the preference file used to create the preference dialog box. The path to the preference XUL file should be given in the install.manifest file.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<prefwindow id="myExtensionOptions" type="prefwindow"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
buttons="accept, cancel" title="My Options">
  <prefpane id="myPane" flex = "1" >
    <preferences id="tabsPreferences">
      <preference id="extensions.my_extension.myOptions"
name="extensions.my_extension.myOptions" type="int"/>
    </preferences>
   <script type="application/x-javascript"
src="chrome://sample/content/js/pref.js"/>
         <radiogroup id="myOpt"
preference="extensions.my_extension.myOptions">
                  <radio label="Three sec" value="2" />
                  <radio label="Five sec" value="3"/>
                  <radio label="Ten sec" value ="4"/>
          </radiogroup>
  </prefpane>
</prefwindow>
```

Listing 5: Preference XUL file

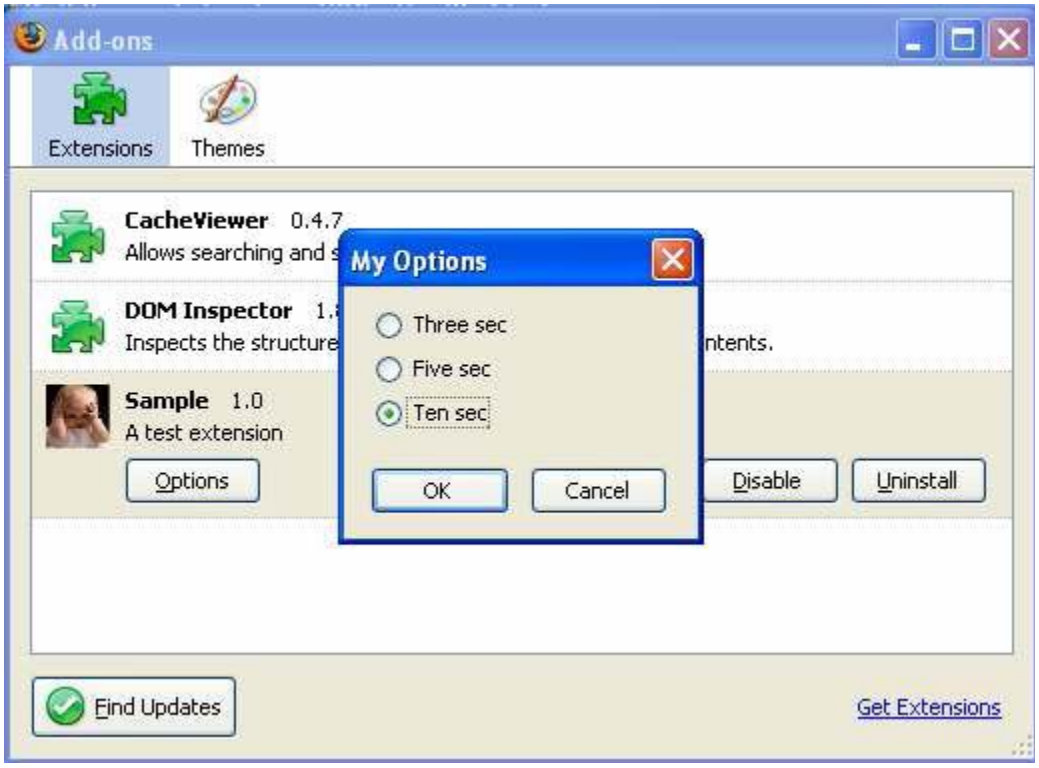

Figure 3: Preference Dialog Box

To implement the above feature, our extension uses Mozilla's nsIPrefService and nsIPrefBranch2 XPCOM interface. The nsIPrefService interface is the main entry point into the back end preferences management library. The preference service is directly responsible for the management of the preferences files and facilitates access to the preference branch object, nsIPrefBranch2. The nsIPrefBranch2 interface is used to manipulate the preferences data. It is used to get and set default and/or user preferences across the application. nsIPrefBranch2 allows clients to observe changes to preference values. An observer is added to receive notification of the changes made through the Preference dialog box. As soon as the notification is received the observer calls the caching function, which is explained in the next section, with the new time interval.

```
//extension registers to the Preference service
this.prefs = Components.classes["@mozilla.org/preferences-service;1"]
        .getService(Components.interfaces.nsIPrefService)
        .getBranch("extensions.my_extension.");
 this.prefs.QueryInterface(Components.interfaces.nsIPrefBranch2);
 this.prefs.addObserver("",this, false);

//gets called when notifications are received.
observe: function(subject, topic, data)
{
     this.myOpt = this.getOpt();
     this.setOpt(this.myOpt); //calls caching function
     if (topic != "nsPref:changed")
     {
         return;
     }
 }
```

Listing 6: Preference Service

Caching

Caching starts when a user clicks the play button on the toolbar. Clicking on the stop button will stop the caching. Users can use the microBack and microForward button to browse the cached pages.

The figures below show different cached states of a page. As seen in the Figure 4, the Back button is disabled; whereas, the microBack button's popup menu has three menu items. These menu items correspond to the different states of the same page.

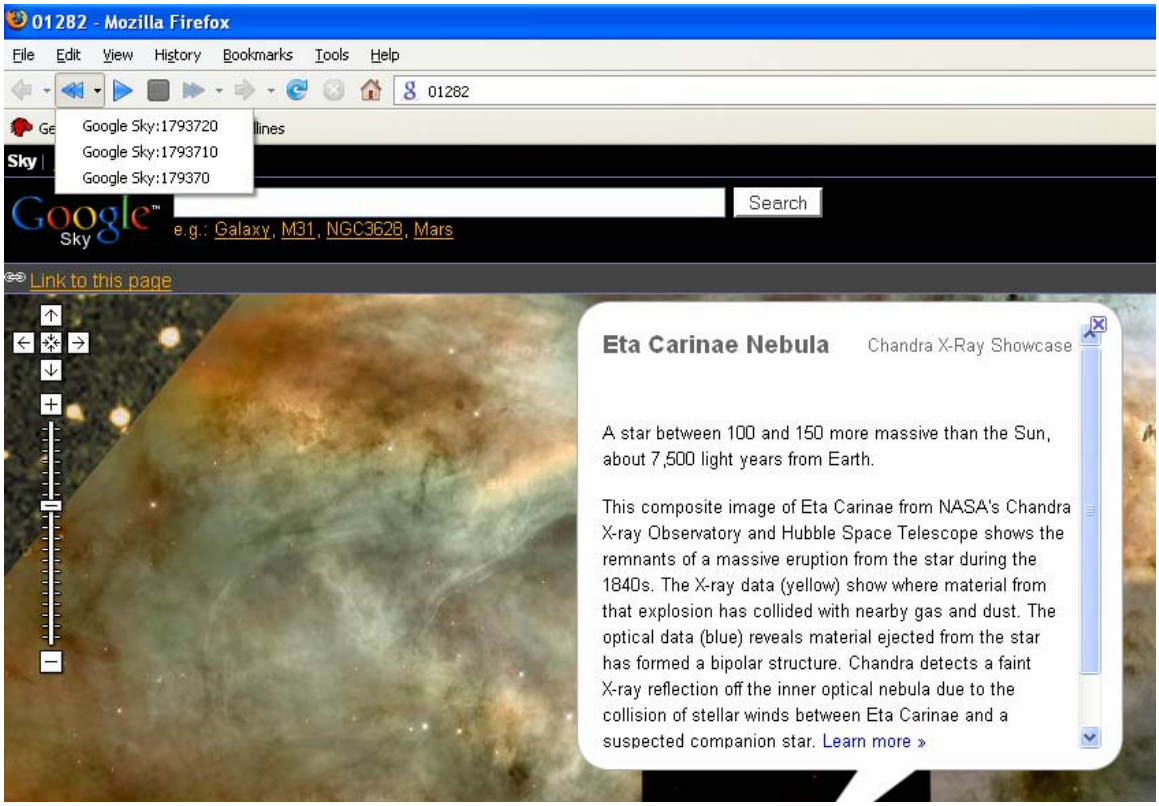

Figure 4: microBack popup menu

The Figures 4, 5 and Figure 6 are completely different from each other even though it is the same web page. Our extension is caching all the different states of the web page and putting it in micro history. Google sky uses AJAX to get different information on to the web page; hence the History controls are not able to save those states as history items.

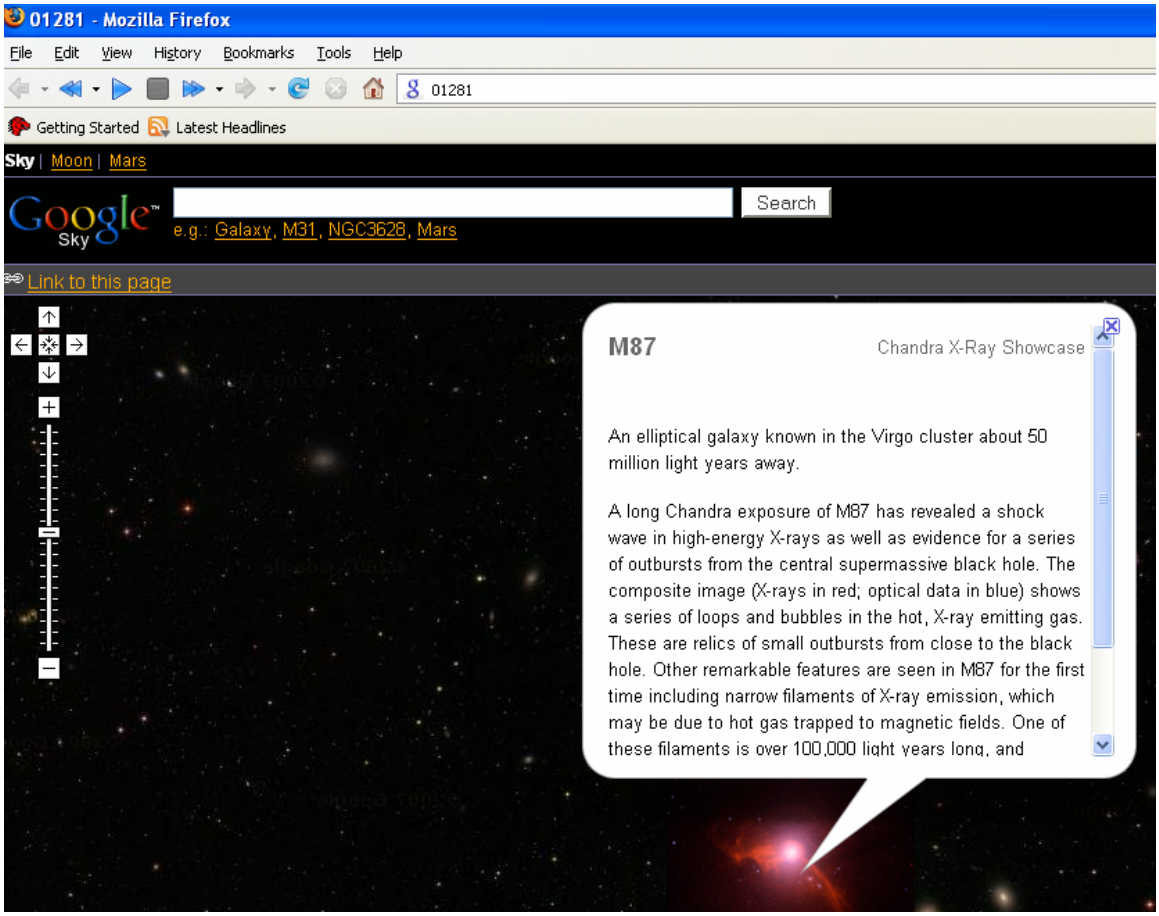

Figure 5: Cached page 1

The Figure 6 shows, the microForward button having three popup menu items, while the microBack button is disabled. Whenever microBack button is clicked a new menu item is added to the microForward button's popup menu, while deleting that menu item from microBack button's popup menu. Similar action takes place when microForward button is clicked.

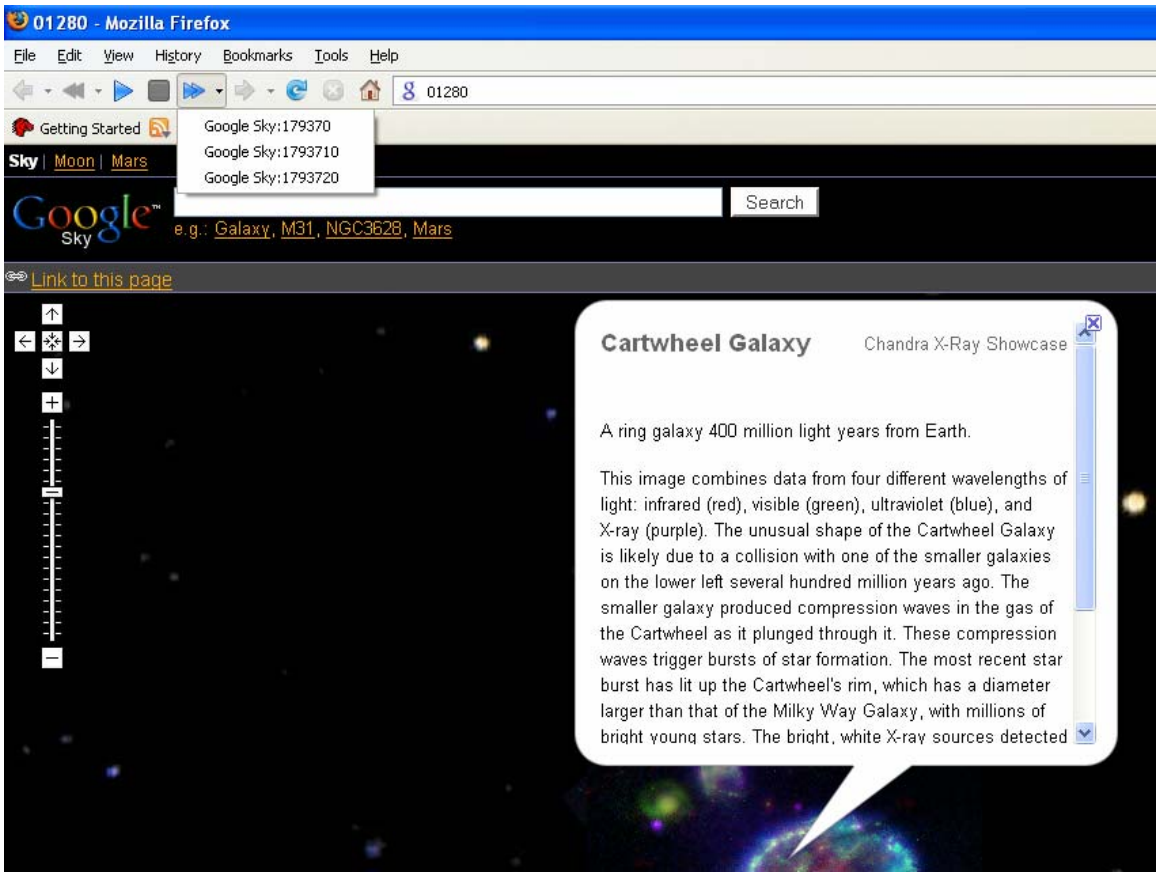

Figure 6: Cached page 2

Implementation

Our extension captures the page's DOM tree's body.innerHTML property. The captured body.innerHTML is then written to the disk using Mozilla's nsICacheService and nsICacheSession XPCOM interfaces. In order to write/read data from cache, we have to create a session. The nsICaceService interface provides functions to create a session that represent a client's access into the cache.

The nsICacheSession interface handles open synchronous and asynchronous cache entry operations, like openCacheEntry. Te openCacheEntry method gives

a synchronous cache access. It returns a unique nsICacheEntryDescriptor descriptor each time it is called, even if the same key is specified. A cache session can only give out one descriptor with WRITE access to a given cache entry at a time. This method opens blocking input stream to cache data. This method opens blocking output stream to cache data.

The listing below shows the way nsICache service is called and how pages are written and read from the cache.

```
var nsCacheService = Components.classes["@mozilla.org/network/cache-
service;1"];
var service =
nsCacheService.getService(Components.interfaces.nsICacheService);

var outputEntry =
buttonsObj[countT].myHist.openCacheEntry(this.nsICache.ACCESS_WRITE);
var output = outputEntry.openOutputStream(0, -1, 0);
if (output.write(stg, stg.length) != stg.length)
      alert("disk cache write broken!");

var inputEntry =
buttonsObj[countT].myHist.openCacheEntry(this.nsICache.ACCESS_READ);
var input =
buttonsObj[countT].myHist.wrapInputStream(inputEntry.openInputStream(0, -1, 0));
var d = input.read(input.available());
```

Listing 7: nsICache Interface

Resource Description Framework

The Resource Description Framework (RDF) is a simple, cross-platform database for small data stores. Bookmarks, global history in Mozilla use RDF.

Before writing to cache, the content, the URL and the title are stored as RDF resources. Before writing any new content to the cache the content is compared with all the previously written content to check if it is equal. This is done with the help of RDF. The RDF service is used only to check if the data already exists in the cache.

```
var resource = buttonsObj[countT].myHist.rdf.appendResource(rdfKey,
this.root);
buttonsObj[countT].myHist.rdf.setLiteralProperty(resource,
buttonsObj[countT].myHist.rdf.NS_CACHEVIEWER+"key",
buttonsObj[countT].myHist.key);
buttonsObj[countT].myHist.rdf.setLiteralProperty(resource,
buttonsObj[countT].myHist.rdf.NS_CACHEVIEWER+"data",
buttonsObj[countT].myHist.data);
buttonsObj[countT].myHist.rdf.setLiteralProperty(resource,
buttonsObj[countT].myHist.rdf.NS_CACHEVIEWER+"title",
buttonsObj[countT].myHist.title+ ":" +
buttonsObj[countT].myHist.getTimeStamp());

var res = buttonsObj[countT].myHist.ds.GetAllResources();
while(res.hasMoreElements())
{
        var n = res.getNext().QueryInterface(Ci.nsIRDFResource);
        var k = buttonsObj[countT].myHist.rdf.getLiteralProperty(n,
buttonsObj[countT].myHist.rdf.NS_CACHEVIEWER + "data");
        if(buttonsObj[countT].myHist.data == k )
        {
                flag = 1;
                break;
        }
}
```

Listing 8: RDF snippet

When the content is cached a new menu item is added to the microBack button's popup menu, with the title of the page as its label. The following listing shows the code snippet to create the popup menu item.

```
function createBackChild(n,tit,flashArr)
{
	var bb = buttonsObj[countT].back;
	buttonsObj[countT].back.setAttribute("disabled","false");

	buttonsObj[countT].back.addEventListener('mousedown',buttonsObj[co
	untT].myHist.eventBack,false);
	buttonsObj[countT].back.addEventListener('mouseup',buttonsObj[count
T].myHist.stopTimer,false);
	buttonsObj[countT].back.addEventListener('click',buttonsObj[countT].m
yHist.goBack,false);

	var bChild =  buttonsObj[countT].backpopup;
	var idTag = "menuitem" + n;
	buttonsObj[countT].menuitem[n]=document.createElement("menuitem")
;
	buttonsObj[countT].menuitem[n].setAttribute("id",idTag);
	buttonsObj[countT].menuitem[n].setAttribute( "label", tit );
	buttonsObj[countT].menuitem[n].setAttribute( "index",n );
	for(key in flashArr)
	{
		buttonsObj[countT].menuitem[n].setAttribute(key,flashArr[key]);
	}
	if(buttonsObj[countT].backpopup.hasChildNodes())
	{
		var node = buttonsObj[countT].backpopup.firstChild;
		buttonsObj[countT].backpopup.insertBefore(buttonsObj[countT].
	menuitem[n],node);
	}else
	{
		buttonsObj[countT].backpopup.appendChild(buttonsObj[countT].
	menuitem[n]);
	}
}
```

Listing 9: Create popup menu item

When the microBack button is clicked the most recent menu item gets appended to the microForward button's popup menu while it gets deleted from the microBack button's popup menu. A similar function as shown in Listing 9 is written to create popup menu items for microForward button.

```
function goBack()
{
	var bChild = buttonsObj[countT].backpopup;
	var node = buttonsObj[countT].backpopup.firstChild;
	buttonsObj[countT].index = node.getAttribute("index");
	tit = node.getAttribute("label");
	flashVar = new Array();
	for( var i = 0;i < buttonsObj[countT].myHist.flashArray.length;i++)
	{
		var fid = buttonsObj[countT].myHist.flashArray[i];
		flashVar[fid] = node.getAttribute(fid);
	}
	var i =  parseInt(buttonsObj[countT].index);
	buttonsObj[countT].backpopup.removeChild(node);

	if(!bChild.hasChildNodes())
	{
		var bb = buttonsObj[countT].back;
		buttonsObj[countT].back.setAttribute("disabled","true");
	}
	buttonsObj[countT].myHist.createForwardTree(i,tit,flashVar);
	buttonsObj[countT].myHist.loadUrl(buttonsObj[countT].index,flashV
ar);
}
```

Listing 10: Clicking microBack

Similar function, as shown in the Listing 10, is written when microForward button is clicked.

Keeping the microBack/microForward button pressed will display the cached pages as a slide show. Following Listing shows the vents captured for doing the same.

```
//mousedown will start displaying the pages as slide show
buttonsObj[countT].back.addEventListener('mousedown',buttonsObj[countT].myHist.
eventBack,false);

//mouseup will stop the slideshow
buttonsObj[countT].back.addEventListener('mouseup',buttonsObj[countT].myHist.sto
pTimer,false);
```

Listing 11: Events for creating slideshow

Multiple tabbed browsing

The functionalities provided by our extension, which is mentioned in the above sections, have been implemented in case of multiple tabs also. The tab count is used as one of the combinations to create the key, which is used as an index to retrieve data from cache.

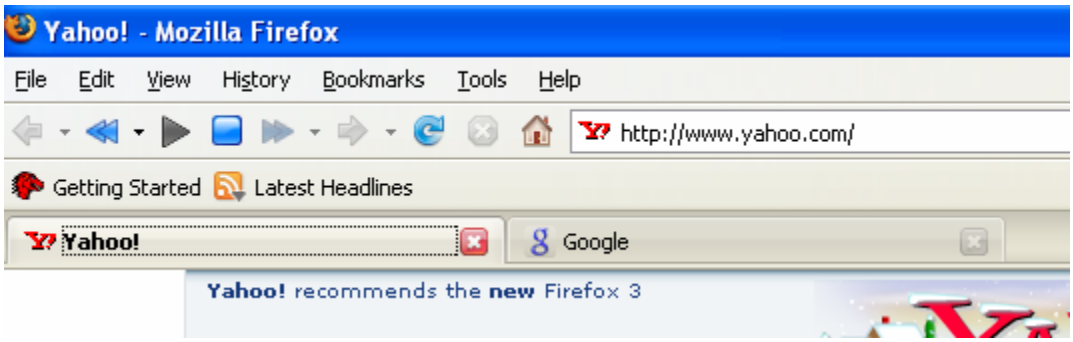

Figure 7: Caching started on Tab 1

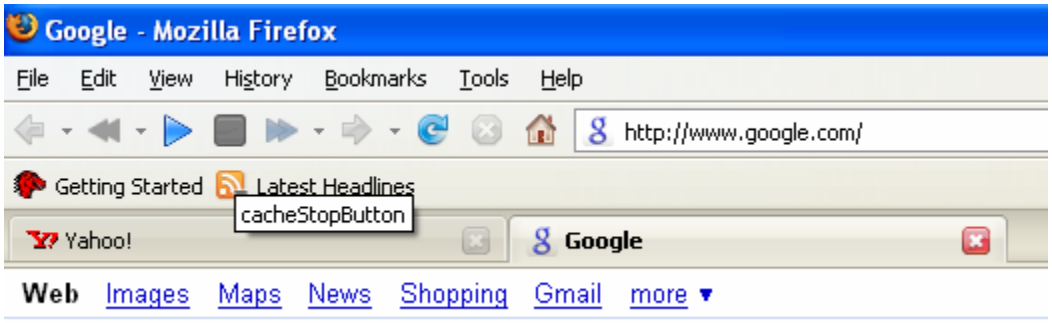

Figure 8: No caching started on Tab 2

Implementation

The “TabOpen” event is captured when a new tab is opened, while “TabSelect” event is captured when we move from one tab to another tab. Following Listing shows how these events are captured. Whenever a new tab is opened, a new object is created. This object has its own set of buttons and micro history. Whenever a tab is focused, that particular tab’s object’s button and micro history gets focused.Window mediator is a Mozilla component that keeps track of open windows. It is accessed through the nsIWindowMediator interface. The two most common uses of nsIWindowMediator are:

1. Getting the most recent / any window of a given type.
2. Enumerating all windows of a given type.

For our extension we have used getEnumerator() function to get the count of all open tabs.

```
var wm = Components.classes["@mozilla.org/appshell/window-
mediator;1"].getService(Components.interfaces.nsIWindowMediator);


var browserEnumerator = wm.getEnumerator("navigator:browser");
var tabbrowser = browserEnumerator.getNext().getBrowser();


//Events to captured TabOpen,TabClose and TabMove
window.getBrowser().addEventListener("TabOpen", function
e(){callbackLoad();}, false);
window.getBrowser().addEventListener("TabClose",callTabClose,false);
window.getBrowser().addEventListener("TabSelect",callTabMove,false);
```

Listing 12: Tabs

Flex application caching

Flex application do not allow users to navigate through the various states of the application by using the web browser's back and forward navigation commands. In our project we have implemented an API, jsChromeBridge, which will allow users to navigate through the applications previous states. For example, a user can set the horizontal slider position at different values, and then click the browser's microBack button to return to its previous states i.e. to the previous position of the horizontal slider.

In the figure below the initial position of the slider is shown and the caching has also begun.

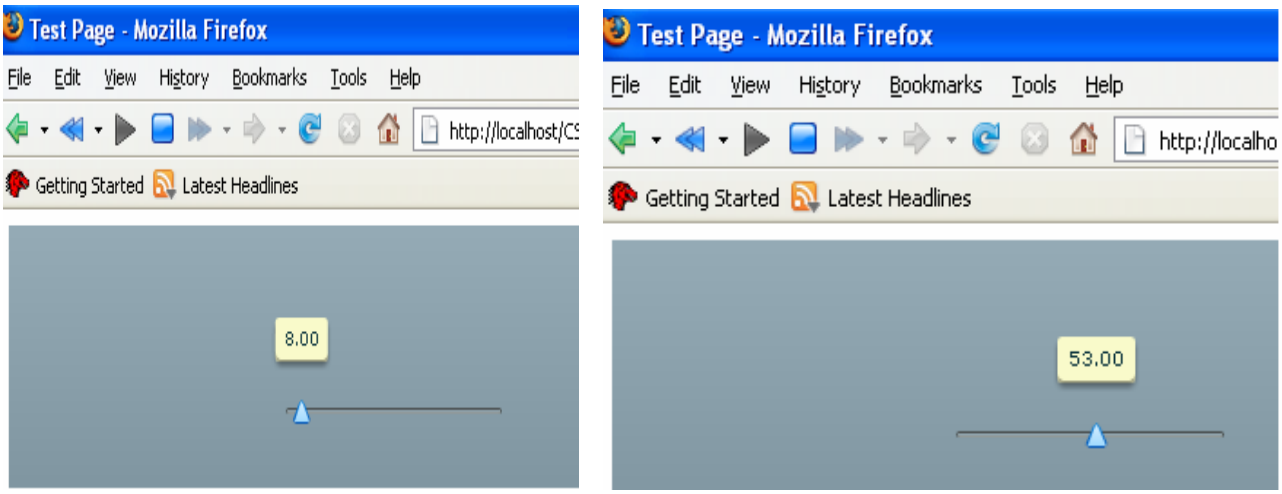

Figure 9: Saving states of Flex application

The microBack button is activated as it has already captured different positions of the slider.

The Figure below shows that when microBack button is clicked, the browser displays the initial state of the slider.

The two figures below show the different positions of the horizontal slider when the microBack and microForward buttons are clicked respectively.

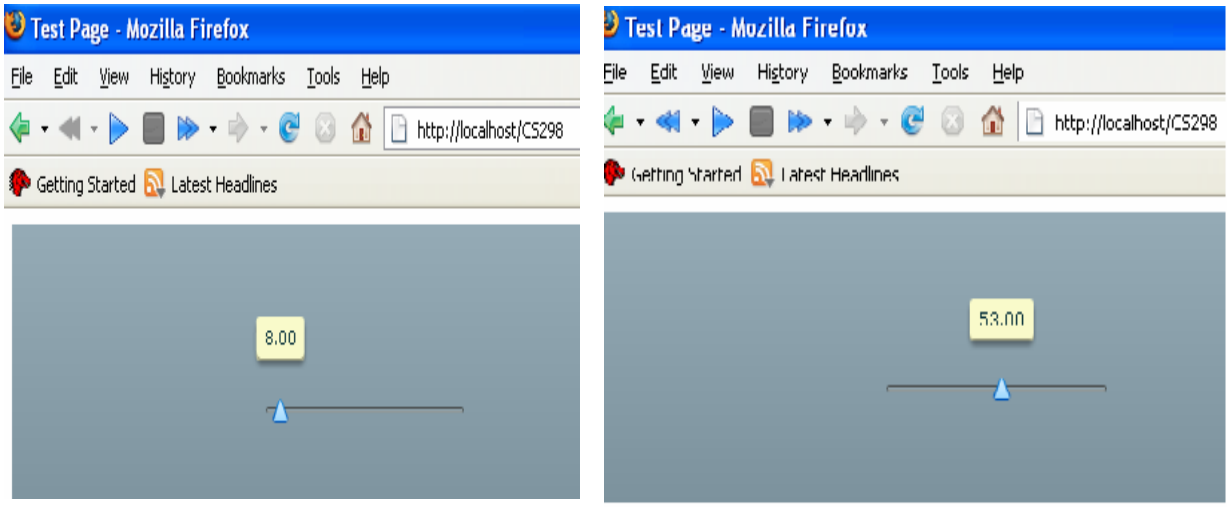

Figure 10: Navigation to the previous state of Flex application

Implementation

Whenever any Flex application, written using our API, is loaded in the browser our extension parses the HTML DOM tree to get the ***object*** tags. These object tags are used to embed Flex application in the HTML files. Once the object tags are found, new event is created and dispatched to our API, which in turn communicates with the Flex application to get all the Flex variables and their values. Once the Flex application returns, the API creates a new event and dispatches the event to our extension with all the Flex variables and their values.

```
// event created and dispacted to the API to get the Flex variables
var elm = htmlDoc.getElementById(flashID);
if (elm && "createEvent" in htmlDoc)
{
        elm.setAttribute('flashvar', " ");
        elm.setAttribute('flashid',countFlash);
        var evt = htmlDoc.createEvent("Events");
        evt.initEvent("myEvent", true, false);
        elm.dispatchEvent(evt);
}
```

Listing 13: Caching Flex parameters

These values are then cached and retrieved from cache when the user clicks micro-back or micro-forward button.

When the microBack or microForward button is clicked our extension again creates and dispatches an event to the API with the values of the Flex variables. The API in turn sets the values of the variables in the Flex application. This allows the users to navigate to different states of the Flex application using microBack and microForward button.

```
//event created and dispatched to the API with flex variables
var elm = htmlDoc.getElementById(flashId);
if (elm && "createEvent" in htmlDoc)
{
        elm.setAttribute('flashvar',flashvar[flashId]);
        var evt = htmlDoc.createEvent("Events");
        evt.initEvent("myEvent1", true, false);
        elm.dispatchEvent(evt);
}
```

Listing 14: Setting Flex parameters through API

The Listing below shows the code snippet of our API, jsChromeBridge.js.

```
var arr = new Array();
var count = 0;
function setFlashParams(val,id)
{
	arr[id] = val;
}
function myeventListener(event)
{
	ele = event.target;
	var id = ele.getAttribute('id');
	document.getElementById(id).getFlashParams(id);
	if ("createEvent" in document)
	{
		ele.setAttribute("flashvar", arr[id]);
		var evt = document.createEvent("Events");
		evt.initEvent("MyExtensionEvent", true, false);
		ele.dispatchEvent(evt);
	}
}
function myeventListener1(event)
{
	ele = event.target;
	var id = ele.getAttribute('id');
	var flashvar = new Array();
	flashvar[id] = ele.getAttribute('flashvar');
	document.getElementById(id).setFlashParams1(flashvar[id]);
}
window.addEventListener("myEvent",myeventListener,false);
window.addEventListener("myEvent1", myeventListener1,false);
```

Listing 15: jsChromeBridge.js

Writing the jsChromeBridge API was challenging, as there were not many documents available explaining the interaction between Javascript and chrome. The Mozilla documentation had details about interaction between chrome and Javascript,but not from Javascript to chrome. We spent time implementing it in different ways before we got it right.

Extension Packaging

Extensions are a form of Installable Bundle, which can be downloaded and installed by a user. Extensions use a directory structure, which can provide chrome, components, and other files to extend the functionality of a XUL program. [7]

Every extension must provide an install.rdf file, which contains metadata about the extension, such as its unique ID, version, author, and compatibility information.

Making an Extension XPI

An XPI (XPInstall) file is a ZIP file containing the extension files, with the install.rdf file at the root of the ZIP. Rename the ZIP to .XPI before uploading in to the add-ons website. Users can download and install XPI files from the Mozilla's add-ons website or can install the XPI file locally.

Submission of extension to AMO

The extension, the .xpi file, is submitted to Mozilla's add-ons website as an experimental add-on. The add-on works for all platforms and it works for Mozilla Firefox 3.0.* as well.

Figure 11: Mozilla Addons Page

Testing on AJAX based websites

Testing was done on various AJAX based websites like Google Maps, Google Sky, Netflix, A9.com and Netvibes.

Results of testing on Google Sky are shown in the previous section.

The following paragraph shows the results of testing done on Netflix and A9 website.

Netflix's website uses AJAX to display the movies details on mouse-over. Achieving this without AJAX would be very impractical, because the graphics text adds takes long to download. AJAX makes this page interesting by requesting the data for each movie from the server and putting it onto the page without re-loading. These graphics text addups are not saved in the normal history items. Our extension caches these graphics text add ups. As shown in the following figures, navigating through the micro-back and micro-forward buttons shows the cached graphics text add ups.

Figure 12: Netflix 1

Figure 13: NetFlix 2

A9 is Amazon.com's search engine. A9 uses AJAX to display the result of the search by dynamically adding column according to the selected resources. One can easily toggle these search areas on or off without researching or reloading the page. The following figures show how the tabs are added/removed dynamically without reloading the page.

These dynamically added columns are not cached by the History. Our extension caches these dynamically added columns. The following figures show that navigating through the micro-back and micro-forward buttons allows the users to view those dynamically added/removed columns.

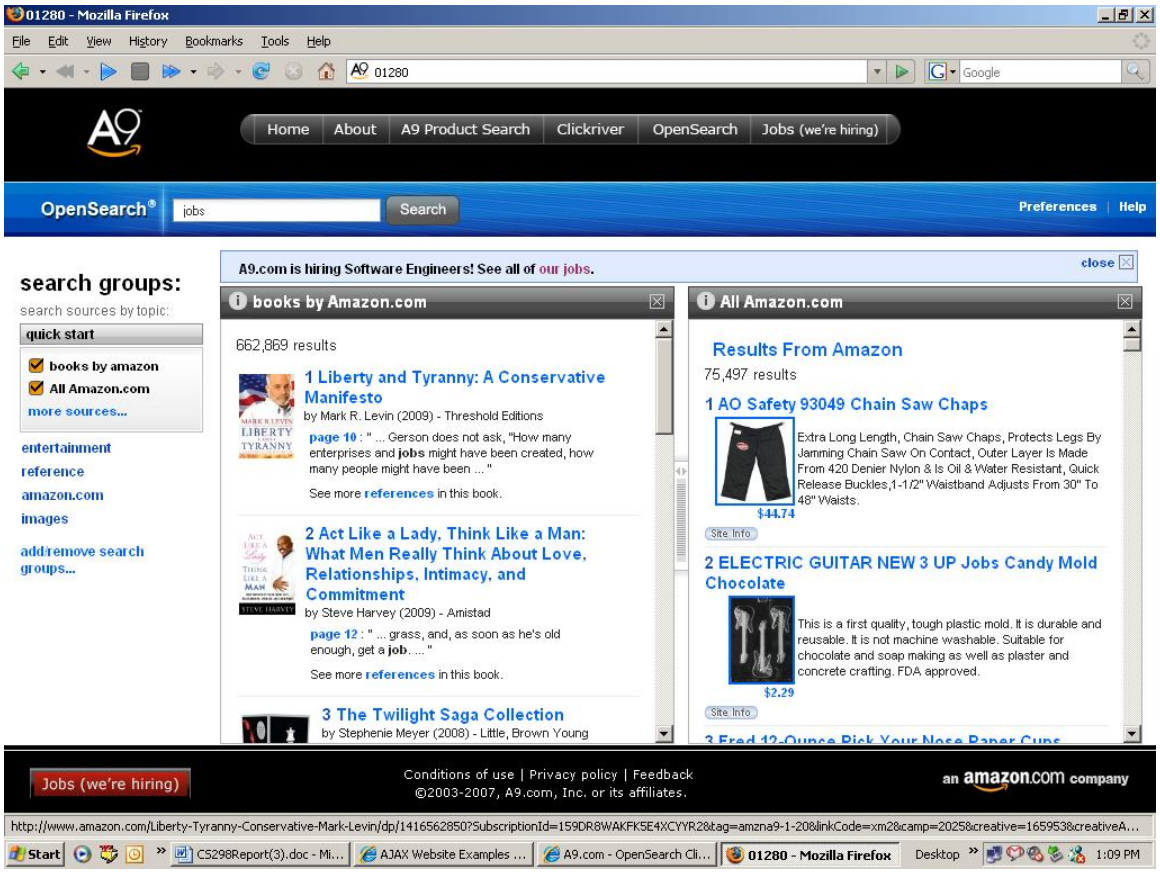

Figure 14: A9 -1

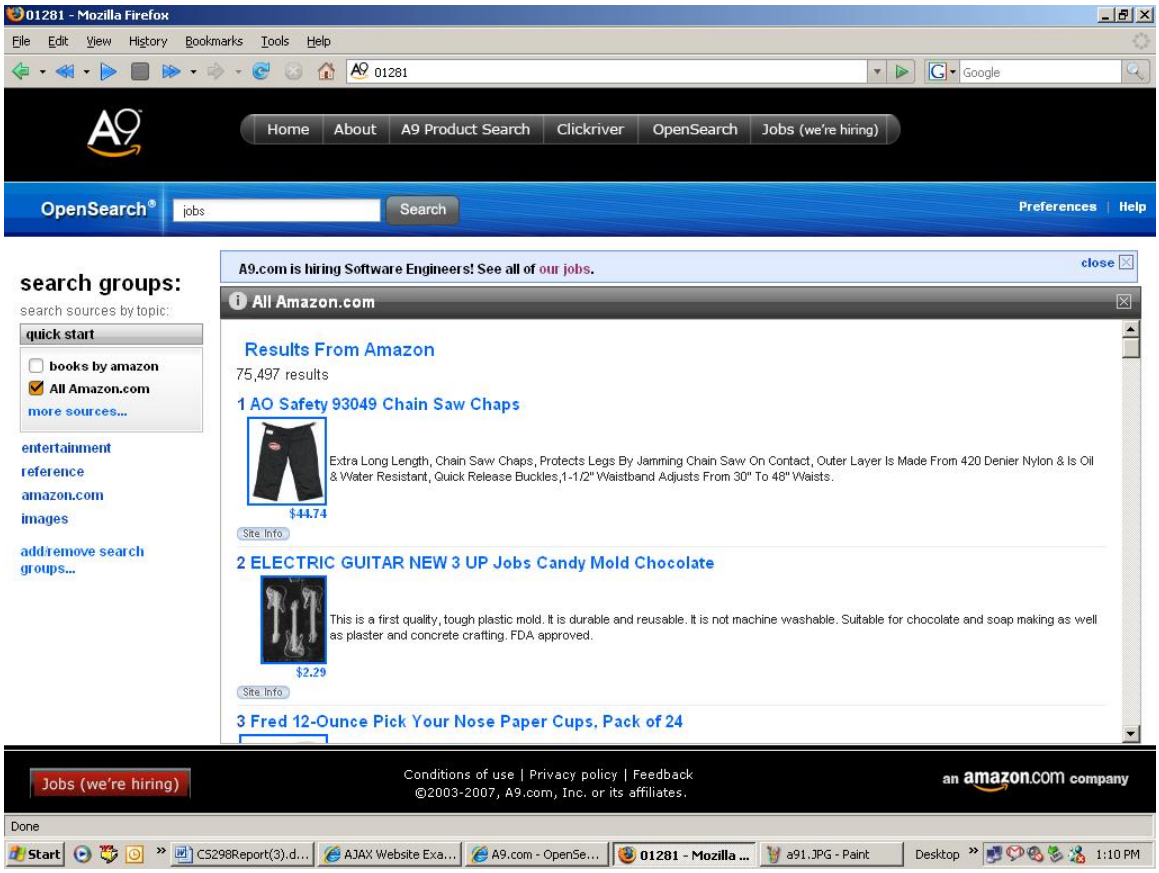

Figure 15: A9-2

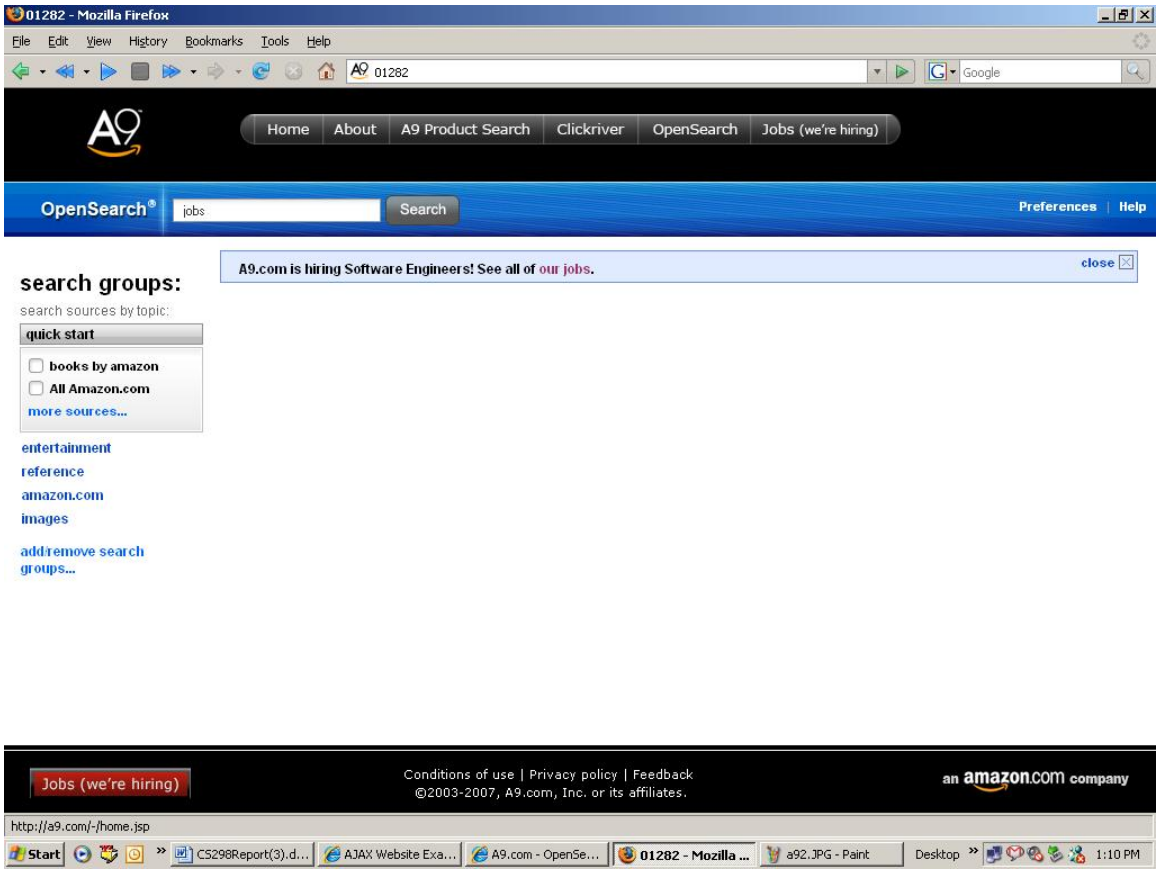

Figure 16: A9-3

Netvibes is an AJAX based website, where in users can add new widgets or delete widgets. The page's layout is in form of tabs, each tab containing some user defined modules. Navigating between tabs, adding and deleting of widgets is done using AJAX.

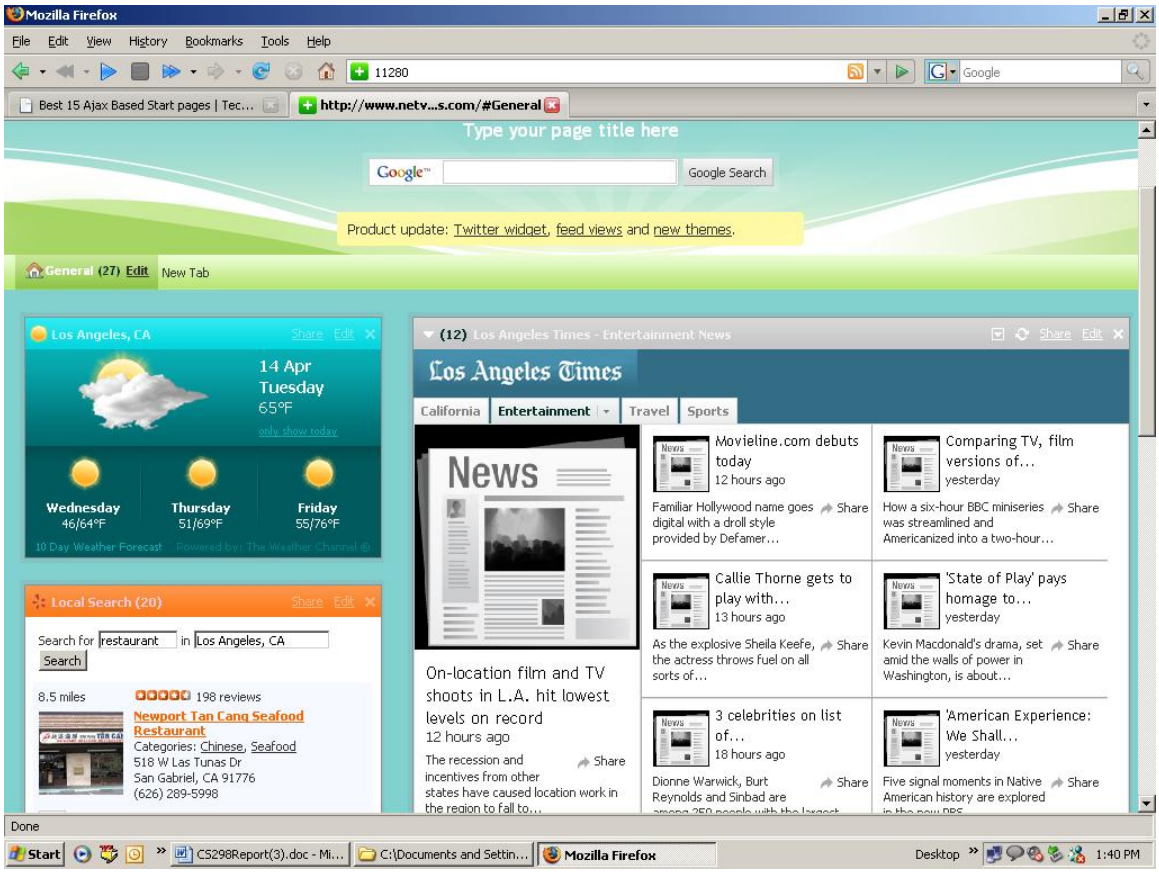

Figure 17: Netvibes -1

Adding/Removing of widgets from the website is not cached in the normal history. The above figure shows a weather widget in the website. The following figure shows the widget delete from the website.

Using the back button, does not allow the user to see the widget. But using the micro-back button, one can see the weather widget being present.

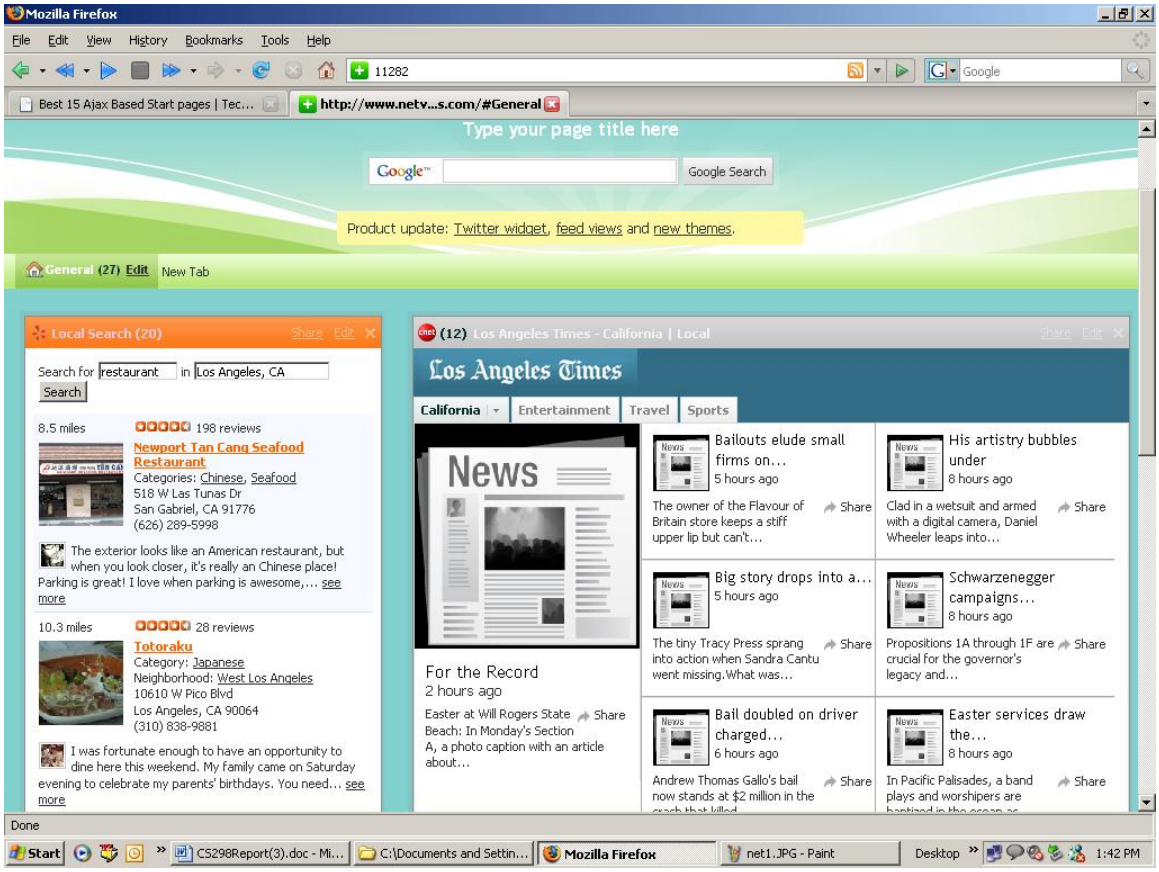

Figure 18: Netvibes -2

Following table gives the summary of the websites our extension was tested with the results.

Website	Results
Netflix's website uses AJAX to display the movies details on mouse-over. Achieving this without AJAX would be very impractical, because the graphics text adds takes long to download. AJAX makes this page interesting by requesting the data for each movie from the server and putting it onto the page without re-loading.	Passed
A9 is Amazon.com's search engine. A9 uses AJAX to display the result of the search by dynamically adding column according to the selected resources. One can easily toggle these search areas on or off without researching or reloading the page.	Passed
Netvibes is an AJAX based website, where in users can add new widgets or delete widgets. The page's layout is in form of tabs, each tab containing some user defined modules. Navigating between tabs, adding and deleting of widgets is done using AJAX.	Passed

Table 1: Summary of websites tested

Usability Testing

The users were not aware of the fact that Ajax clicks are not saved as the normal history items. Initially they were not comfortable using the extension, as they were getting confused about the actual history items and the history stored by our extension. After giving them a demo, they visited few websites like Google Maps and Google Sky. They thought the extension was useful.

Cyrus Poja, Joseph Tusoy and Abdurrahman Nurhasan, third year students from Ateneo de Davao University, Philippines have downloaded my extension from the Mozilla addons website. They are working on project, which is similar to my project. They tried my application with Yahoo mail and Google map. Here is what they have to say, “I already tried the extension. I tried it with yahoo mail and google map. I see no problem with the back button, but when I hit the forward button, it causes the URL to change all into numeric. As I observed, there are alerts that popped up. May be those were testers alerts. It is working.”

Response to their feedback:

The alerts that popped out were debug statements. I have removed all the debug statements. The numeric number that comes on the address bar while navigating through the microBack and microForward buttons will be changed to the actual URL of the page.

The following figure shows the dashboard, which shows that users are installing the addon from the Mozilla’s addons website.

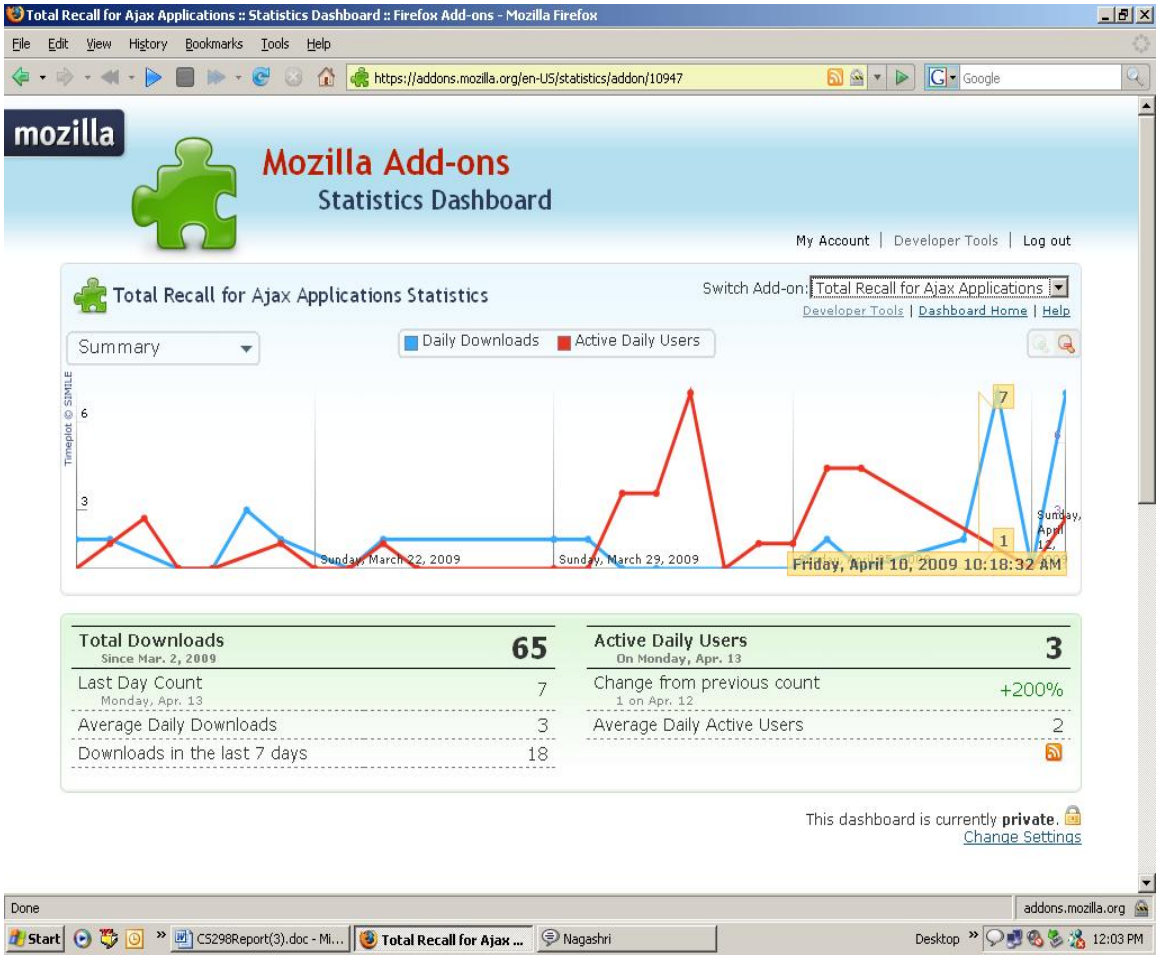

Figure 19: Statistics Dashboard for my extension on addons website

The dashboard also shows how many users have downloaded the extension. It gives the count of the average daily downloads and the active daily users.

Conclusion

The goal of the project was to develop an extension which would allow the users to capture the states of an AJAX page. This goal was accomplished by developing user interface something similar to the normal Back/Forward buttons. Users can start and stop caching pages with the start and stop buttons or browse the saved items with the back and forward buttons. Keeping the back/forward button pressed will display the cached pages as a slide show.

The initial plan of the project was to save all the AJAX clicks of a web page. When we tried doing that, it was getting difficult to figure out if the clicks were AJAX calls or would it redirect to a different page. Hence we decided to take snapshots of the page at regular intervals and save it in the cache. Thus, users were given an option to set the time interval they want caching to occur.

Our extension extended this functionality to multiple tabbed browsers. Another goal established was to capture the states of a Flex application. This goal was accomplished by implementing an API which acts a bridge between the chrome and Javascript, which in turn communicates with Flex. The caching of states of a Flex is limited to the applications which are developed using our jsChromeBridge API.

We faced several challenges because of poor Mozilla documentation on XPCOM interfaces. We faced difficulties for extending our functionality to multiple tabbed

browser. We also faced challenges when extending the functionality to Flex application again because of lack of documentation on interaction between Chrome and JavaScript. We had to go through the Firefox code to figure out stuff.

Bibliography

1. Creating Applications with Mozilla. David Boswell. O'Reilly. 2002.

2. Official page of Mozilla.
   http://developer.mozilla.org/en/docs/Building_an_Extension

3. XUL Tutorial and XPCOM Reference.
   http://www.xulplanet.com/

4. JavaScript Reference.
   http://www.w3schools.com/jsref/default.asp

5. RDF reference
   http://developer.mozilla.org/en/docs/RDF_in_Fifty_Words_or_Less

6. Mozilla Developer Center XPCOM Retrieved May 01, 2008
   http://developer.mozilla.org/en/docs/XPCOM

7. Extension Packaging
   https://developer.mozilla.org/en/extension_packaging

8. Flex Developer's Guide
   http://livedocs.adobe.com/

9. Icons gallery
   http://www.icons-gallery.com/

10. AJAX
    http://en.wikipedia.org/wiki/Ajax_(programming)

11. ActionScript
    http://en.wikipedia.org/wiki/ActionScript