Interactive session where everyone gets to have a word

How Does WebDriver ‘Drive’ the Browser Compared to Selenium-RC?

This would be the base of this discussion, later on let’s move on to check the Pros and Cons of using WebDriver over RC on different browsers

Selenium-WebDriver makes direct calls to the browser using each browser’s native support for automation. How these direct calls are made, and the features they support depends on the browser you are using.

That’s the reason why there are separate drivers for each browser

Selenium-RC works the same way for each supported browser. It ‘injected’ JavaScript functions into the browser when the browser was loaded and then used its JavaScript to drive the AUT

RC passes the Selenium command to the browser using Selenium-Core JavaScript commands, the browser using its JavaScript interpreter, executes the Selenium command.
What mechanism does Selenium RC use to do user simulations on different browsers -

As mentioned before they aren’t different, Its only that the way it locates the elements can slightly vary.

Well how does RC locate elements ?

It uses a file called selenium-browserbot.js , here’s a snippet of it

```javascript
/*
 * Find the element with id - can’t rely on getelementbyid, coz it returns by name as well in IE..
 */
BrowserBot.prototype.locateElementById = function(identifier, inDocument, inWindow) {
  // HBC - use 'this' instead of 'BrowserBot.prototype'; otherwise we lose
  // the non-prototype fields of the object!
  return this.locateElementByXid(identifier, inDocument, inWindow)
  || BrowserBot.prototype.locateElementByXname(identifier, inDocument, inWindow)
  || null;
};

/*
 * Find the element with id - can’t rely on getelementbyid, coz it returns by name as well in IE..
 */
BrowserBot.prototype.locateElementByXid = function(identifier, inDocument, inWindow) {
  if (element & element.getAttribute("id") === identifier) {
    return element;
  }
  else if (browserVersion.isIE || browserVersion.isIeopera) {
    // SEL-484
    var elements = inDocument.getElementsByTagName("");
    for (var i = 0, n = elements.length; i < n; ++i) {
      element = elements[i];
      if (element.nodeName.toLowerCase() == 'form') {
        if (element.attributes["id"].nodeValue == identifier) {
          return element;
        }
      } else if (element.getAttribute("id") == identifier) {
        return element;
      }
    }
    return null;
  } else {
    return null;
  }
};
```

Does Selenium RC have an order when it uses the JavaScript functions to locate elements?

Yes it does and as we see in the image if we have not given specified the attribute ( for id/name) it does check for an Identifier and failing which it would check for name attribute. The other locating strategies do specific calls to JavaScript functions (CSS, DOM, Xpath)

Well is Selenium1’s locating strategies vary across browser ?

Back to the snippet!!  As you can see within the locating functions there are separate conditions for different browsers. And it takes a lot of looping for browsers like IE – hence the time to execute in IE is drastically more!!!
Different modes Selenium RC uses to launch browsers and why?

- *iexplore - Launches IE
- *firefox - Launches Firefox
- *safari - Launches Safari
- *opera - Launches Opera
- *googlechrome - Launches Google Chrome
- *chrome - Launches special Firefox profile with elevated security privileges, to bypass Javascript security errors
- *iehta - Launches IE with elevated security privileges
- *piexplore - Launches IE in proxy injection mode
- *pifirefox - Launches Firefox in proxy injection mode
- *custom - Launches your own custom browser executable

We say that selenium RC injects JavaScript into the Browser to manipulate the isn’t it a security issue??

Yes, injecting XSS or cross site scripting is a security issue.

But RC overcomes the Same Origin Policy(which is using multimode browsers to execute) by Its use of the Selenium Server as a proxy to avoid this problem. It, essentially, tells the browser that the browser is working on a single “spoofed” website that the Server provides.

A very detailed description of this –

http://seleniumhq.org/docs/05_selenium_rc.html#selenium-rc-architecture

Heightened Privileges Browsers

This workflow in this method is very similar to Proxy Injection but the main difference is that the browsers are launched in a special mode called Heightened Privileges, which allows websites to do things that are not commonly permitted (as doing XSS, or filling file upload inputs and pretty useful stuff for Selenium).

Any Questions Regarding Selenium RC’s Interactivity with Browsers??

Yes we all had problems with Safari 😊
Moving on to WebDriver and its interactivity with Browsers

The most widely used Firefox Driver

The driver comes in the form of an xpi (firefox extension) which is added to the firefox profile when you start a new instance of FirefoxDriver.

By default, the Firefox driver creates an anonymous profile, but we can customize it as well as the binary and so can many properties of the Firefox browser will run the AUT.

Ex:

```java
File profileDir = new File("path/to/top/level/of/profile");
FirefoxProfile profile = new FirefoxProfile(profileDir);
profile.addAdditionalPreferences(extraPrefs);
WebDriver driver = new FirefoxDriver(profile);
```

Issues:

Patches have been applied to suppress some abnormalities in Firefox

One of the patches includes the handling of subsequent pop-ups, that is if within a single browser session you try to deal with more than 1 popup the selenium code crashes when you try to interact with the second pop-up. This was a barrier for us which we eventually sorted by introducing a patch directly to the web driver source code and recompiling the driver.

One more issue that we had to face was, in certain instances the Firefox driver hanged indefinitely without the code being continued. Initially this was a major barrier for us which wasted a lot of regression time for us. Ultimately we came up with the solution being a simple timeout introduced to the framework using TestNG.
IE Driver:

As the InternetExplorerDriver is Windows-only, it attempts to use so-called "native", or OS-level events to perform mouse and keyboard operations in the browser. This is in contrast to using simulated JavaScript events for the same operations.

Some required configurations:

- Protected Mode settings for each zone to be the same value. The value can be on or off, as long as it is the same for every zone.
- The browser zoom level must be set to 100% so that the native mouse events can be set to the correct coordinates.

Issues:

If the onChange event of the `<select>` element contains JavaScript code that calls alert(), confirm() or prompt(), calling WebElement’s click() method will hang until the modal dialog is manually dismissed and loses window focus.

There were issues reported on Hovering over elements but we can’t relate to any.

HTMLUnitDriver:

Currently the fastest and most lightweight implementation of WebDriver. As the name suggests, this is based on HtmlUnit. HtmlUnit is a java based implementation of a WebBrowser without a GUI. For any language binding (other than java) the Selenium Server is required to use this driver.

Safari Driver:

Anyone can contribute, We haven’t tasted it yet 😊
**ChromeDriver:**

Developed in collaboration with the Chromium team, the ChromeDriver is a standalone server which implements WebDriver's wire protocol.

The ChromeDriver consists of three separate pieces. There is the browser itself ("chrome"), the language bindings provided by the Selenium project ("the driver") and an executable downloaded from the Chromium project which acts as a bridge between "chrome" and the "driver". This executable is called "chromedriver".

Chrome does have some features so that we can use extensions and some advanced properties. For ex: Chrome by default opens in new tabs but through adding an extension it can be opened in a new window for the automated tests

```java
else if(seleniumBrowser.contentEquals(InvokeSeleniumGrid.wdGoogleChrome)) {
    DesiredCapabilities capabilities = DesiredCapabilities.chrome();
    capabilities.setCapability("chrome.screens", Arrays.asList("--load-extension=" + extChromeNewWin.getAbsolutePath()));
    try {
        driver = new ChromeDriver(InvokeSeleniumGrid.cds, capabilities);
    } catch (Exception e) {
        try {
            InvokeSeleniumGrid.cds.start();
            driver = new ChromeDriver(InvokeSeleniumGrid.cds, capabilities);
        } catch (Exception e) {
            System.setProperty("webdriver.chrome.driver", InvokeSeleniumGrid.chromeDriverExe);
            driver = new ChromeDriver(capabilities);
        }
    }
}
```

**Issues:**

This particular case is involved with Drop downs. When you have ‘optgroup’ element inside a select element (as shown in the pic below) chrome driver is unable to select the correct option from the drop down. This was a bit tricky to handle because of its nature and being specific only to one browser. However ultimately we overcome this using jQuery.

```html
<select id="group_selector" name="groupselector">
  <optgroup label="autAbsenceStudent">
    <option selected="selected" value="2047729404">autAbsenceStudent</option>
    <option value="1556750964">autAbsenceVocStudents</option>
  </optgroup>
</select>
```
Enough about what the drivers are made up of and types of!! What really is “native” events ???

Click, Select, etc... It’s that the browser has specific drivers for triggering the native events

What’s New in Selenium2 ? What’s so exiting about it??

- For starters it’s faster,
- more reliable,
- don’t have to have the selenium server running
  and Kishan has done an interesting presentation in the previous meet-up on WebDriver which explains the features of it.

Locating Strategies of web driver

We had a look at browserbot.js for RC, how will Web Driver do it

**How XPath Works in WebDriver**

At a high level, WebDriver uses a browser’s native XPath capabilities wherever possible. On those browsers that don’t have native XPath support, we have provided our own implementation. This can lead to some unexpected behaviour unless you are aware of the differences in the various xpath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Names</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnitDriver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>InternetExplorerDriver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>No</td>
</tr>
<tr>
<td>FirefoxDriver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML:

```html
<input type="text" name="example" />
<input type="text" name="other" />
```

The following will happen:

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>Number of Matches In</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnitDriver</td>
<td>FirefoxDriver</td>
</tr>
<tr>
<td>/input</td>
<td>1 (&quot;example&quot;)</td>
</tr>
<tr>
<td>//input</td>
<td>0</td>
</tr>
</tbody>
</table>
New Windows, Pop Ups . Frames, Actions and JavaScript Executors.

This has made interactivity with browsers much easier. It’s very simple to Switch frames, windows
Snippet of switching to a particular frame

```java
try{
    if(!byWebElement)
        driver.switchTo().frame(frameName);
    else{
        driver.switchTo().frame(driver.findElement(By.name(frameName)));
    }
}catch(WebDriverException we){
    logFailScenario("Switching to frame " + frameName + ":");
    throw we;
}
```

Switching to a particular alert

```java
try{
   waitForAlert(timeout, failureText);
    Alert alerttoaccept = driver.switchTo().alert();
    alerttoaccept.accept();
    successful = fronterAlertClickSuccesscode;
}catch(WebDriverException we){}
```

Switching to a particular Window

```java
String originalWindow = driver.getWindowHandle();
frontermethods.Click(FronterXPath.linkInsertInternalLink);

driver.switchTo().window(toolName);
frontermethods.WaitForElementPresent(divEditingArea);

driver.switchTo().window(originalWindow);
```

Etc.. etc.. It’s a piece of cake , Well not all the time, there are a few misfits, but eventually you can find a workaround.

Let’s round up the Pros and Cons of RC and WebDriver when working on different browsers!!

..........

Questions ??