AJAX Crawler

Paul Suganthan G C
Department of Computer Science and Engineering
CEG, Anna University, Chennai, India
Email: paul.suganthan@gmail.com

***Abstract*—This paper describes the implementation of an AJAX(Asynchronous Javascript And XML) Crawler built in Java. With the advent of Web 2.0 , AJAX is being used widely to enhance interactivity and user experience. Also standalone AJAX applications are also being developed. For example, Google Maps, Gmail and Yahoo! Mail are classic examples of AJAX applications. Current crawlers ignore AJAX content as well as dynamic content added through client side script. Thus most of the dynamic content is still hidden. This paper presents an AJAX Crawler and also discusses about the optimizations and issues regarding crawling AJAX.**

I. Introduction

In a traditional web application, every page has a unique URL , whereas in a AJAX application every state cannot be represented by a unique URL. A particular URL may have a lot of states with different content. Dynamic content is added to the DOM(Document Object Model) through Javascript. Thus an AJAX Crawler requires the ability to execute Javascript. Traditional crawlers doesn't require a Javascript engine. Thus for crawling AJAX we need to simulate the behavior of a browser. The numerous limitations in crawling AJAX is overcome by the fact that large amount of hidden web need to be crawled and made searchable. This paper describes the implementation of an AJAX Crawler built using HtmlUnit Java library. The challenges that exist in crawling AJAX are

- Javascript execution
- Constructing the navigation model
- DOM Analysis

II. Event Model

In an AJAX application, client side events trigger the change in DOM structure of a webpage. For crawling the numerous states in a particular page, these client side events need to be invoked. We consider only click event. First, we need to identify the HTML elements which need to be clicked. Then the click event has to be invoked on those elements.

A. Identification of Clickables

Clickables are those HTML elements on which click event can be invoked. Identification of Clickables is the first phase in an AJAX Crawler. It involves identifying events that would modify the current DOM. The main issue regarding this is that events may be added to an HTML element in many ways. A number of ways to add event listener are shown below.

-
- test.onclick=test_function;
- test.addEventListener('click',test_function,false);
- Using Jquery javascript library,
  $('#test').click(function()
  {
  test_function();
  });

All the above 4 methods, perform the same function of adding the event onclick on element test. Thus event generating elements cannot be identified in a standard way because of the numerous Javascript libraries that exist and each has its own way of defining event handlers. So the approach of clicking all the clickable elements is being followed. Though this approach is time consuming and can cause sub elements to be clicked repeatedly, it has the advantage of all the possible states being reached. XPath is used to retrieve the clickable elements. The XPath expression to retrieve all clickable elements in a webpage is shown below.

$$//a \mid //address \mid //area \mid //b \mid //bdo \mid //big \mid //blockquote \mid //body \mid //button \mid //caption \mid //cite \mid //code \mid //dd \mid //dfn \mid //div \mid //dl \mid //dt \mid //em \mid //fieldset \mid //form \mid //h1 \mid //h6 \mid //hr \mid //i \mid //img \mid //input \mid //kbd \mid //label \mid //legend \mid //li \mid //map \mid //object \mid //ol \mid //p \mid //pre \mid //samp \mid //select \mid //small \mid //span \mid //strong \mid //sub \mid //sup \mid //table \mid //tbody \mid //td \mid //textarea \mid //tfoot \mid //th \mid //thead \mid //tr \mid //tt \mid //ul \mid //var$$

B. Event Invokation

HtmlUnit library provides the ability to invoke any event on an HTML element. The event is invoked on all elements retrieved using XPath expression. After an event is invoked on an element, we need to wait for background Javascript execution.

III. AJAX Website Model

A. State Machine

An AJAX webpage can be represented by a State machine [1]. Thus the navigation model of an AJAX driven webpage can be visualized as a State Machine. The state machine can be viewed as a Directed Multigraph. JUNG (Java Universal Network/Graph Framework) [2] is used for building the state machine. The state machine is stored in GraphML [3] format.

978-1-4673-2149-5/12/$31.00 ©2012 IEEE

Nodes represents application states
Edges represent transition between states

A sample GraphML file depicting a state machine is shown below.

```
<?xml version="1.0" encoding="UTF-8"? >
<graphml
xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml"
>
<key id="event" for="edge" ><desc >Event type</desc ></key
>
<key id="target" for="edge" ><desc >Event generating
element</desc ></key >
<graph edgedefault="directed" >
<node id="0"/ ><node id="1"/ >
<node id="2"/ ><node id="3"/ >
<edge source="0" target="1" >
<data key="event" >onclick</data >
<data key="target" >/html/body/div/table/tbody/tr[1]/td[3]/div[1]
</data >
</edge >
<edge source="0" target="2" >
<data key="event" >onclick</data >
<data key="target" >/html/body/div </data >
</edge >
<edge source="0" target="3" >
<data key="event" >onclick</data >
<data key="target" >/html/body/div/table/tbody/tr[1]/td[1]/div/div[4]
</data >
</edge >
<edge source="1" target="2" >
<data key="event" >onclick</data >
<data key="target" >
/html/body/div/table/tbody/tr[1]/td[1]/div/div[8]/strong </data >
</edge ></graph ></graphml >
```

From the above GraphML file, the following inferences can derived,

- Number of nodes(application states) = 4
- The application state changes from source to target on clicking the element derived by the Xpath expression stored in a key named target. Thus from the GraphML file, the path from one state to another can be obtained.

IV. Crawling AJAX

A. Crawling Algorithm

The first step in crawling is to load the initial state and then wait for background Javascript execution [4] (this handles the case when an AJAX call is made using onload event). Then all clickables in the initial state are found and the event is invoked. The clickables are extracted using XPath expression. Those elements matching the XPath expression are clicked. If there are DOM changes, then the state machine is updated. The crawling is done in a breadth first manner. First all states originating from the intial state are found. Then each state is crawled in a similar fashion. HtmlUnit [5] Java library is used for implementing the AJAX Crawling Algorithm. A WebClient object can be viewed as a Browser instance. This covers the requirement of an AJAX Crawler to be capable of executing Javascript.

Algorithm 1 Ajax Crawling algorithm

```
1:  procedure CRAWL(url)
2:      Load url in HtmlUnit WebClient
3:      Wait for background Javascript execution
4:      StateMachine ← Initialize state machine
5:      StateMachine.add(initial state)
6:      while still some state uncrawled do
7:          current_state ← find some uncrawled state to
        crawl
8:          webclient ← get_web_client (current_state,
        StateMachine, url)
9:          while current_state still uncrawled do
10:             crawl_state(webclient,current_state,StateMachine)
11:             webclient ← get_web_client (current_state,
        StateMachine, url)
12:         end while
13:     end while
14:     save the StateMachine
15: end procedure
```

Algorithm 2 Ajax Crawling algorithm (Continued)

```
1:  procedure GET_WEB_CLIENT(state, StateMachine, url)
2:      webclient ← Load url in HtmlUnit WebClient
3:      Wait for background Javascript execution
4:      path ← Find shortest path from initial state to
        current state
5:      while current state not reached do
6:          xpath ← Get Xpath to traverse to next state in
        path
7:          Generate the click event on the element retrieved
        by xpath
8:          Wait for background Javascript execution
9:      end while
10:     return webclient
11: end procedure
```

One of the problems with the HtmlUnit Webclient is that, once a DOM change occurs and another state is reached, we can't be able to go back to the source state to continue the breadth first crawling process. We need to again traverse from the initial state to the source state , to continue the crawling process. This is done by the function GET_WEB_CLIENT. Here we find the path from the intial state to the current state to be crawled. Then invoke the events along the path to reach the current state. Another issue with the WebClient is that it cannot be serialized

and stored. Thus each time when there is a DOM change, there is a need to traverse from the initial state to current state.

The algorithm for crawling a individual state is described by the function CRAWL_STATE. Special care must be taken in order to avoid regenerating states that have already been crawled (i.e., duplicate elimination). This is a problem also encountered in traditional search engines. However, traditional crawling can most of the time solve this by comparing the URLs of the given pages - a quick operation. AJAX cannot count on that, since all AJAX states have the same URL. Currently, we compare the DOM tree as a whole to check if two states are same.

Algorithm 3 Ajax Crawling algorithm (Continued)

```
1: procedure CRAWL_STATE(webclient, state, StateMachine)
2:     elements ← Get all clickable elements using Xpath
3:     while still an element remaining do
4:         xpath ← Get Xpath of the current element
5:         if current element is already clicked in the current state then
6:             continue
7:         end if
8:         if current element is an anchor element then
9:             href ← Get href attribute of the current element
10:            if href is null then
11:                Generate the click event on current element
12:                Wait for background Javascript execution
13:                if dom is changed then
14:                    if new state is not already present in StateMachine then
15:                        Add the new state to StateMachine
16:                        Add a transition from current state to new state
17:                    end if
18:                    return
19:                end if
20:            end if
21:        else
22:            Generate the click event on current element
23:            Wait for background Javascript execution
24:            if dom is changed then
25:                if new state is not already present in StateMachine then
26:                    Add the new state to StateMachine
27:                    Add a transition from current state to new state
28:                end if
29:                return
30:            end if
31:        end if
32:    end while
33: end procedure
```

B. Reconstruction of a particular state after crawling

After crawling the states of a particular URL, the states should be indexed to be able to be searched by the search engine. Thus a state needs to be reconstructed for being displayed in search results. A web browser can load only the initial state of a URL. But we need to load subsequent states which actually occur in browser after a sequence of Javascript events are invoked. Thus we use Selenium Web Driver [6] to load a particular state in browser directly. A Web Driver can be viewed as a browser which can be controlled through code. Hence we find the path from the initial state to the state to be loaded. At the beginning, the initial state is loaded in the Web Driver. Then the Javascript events along the path are invoked in the Web Driver until the required state in reached. Thus the required state is loaded in the browser to be viewed by the user. From then the user can continue browsing from the required state like a normal browser.

Algorithm 4 Reconstruction of a particular state after crawling

```
1: procedure RECONSTRUCT_STATE(state)
2:     Read the graphML file of the corresponding URL and construct a Directed Multigraph
3:     path ← Find shortest path from initial state to the state to be constructed (Dijkstra Algorithm)
4:     Load the initial state in a Web Driver like Selenium
5:     while state not reached do
6:         xpath ← Get Xpath expression of the element to be clicked next
7:         Generate the click event on the element retrieved by xpath
8:         Wait for background Javascript execution
9:     end while
10:    The required state is currently loaded in the Web Driver
11: end procedure
```

V. Analysis

A. Results

Table I contains the list sample test cases used for evaluating the performance of AJAX Crawler.

Case	AJAX Site
C1	http://test.thurls.com/ajax/home.php
C2	http://spci.st.ewi.tudelft.nl/demo/aowe/
C3	http://www.itrix.co.in/
C4	http://demo.tutorialzine.com/2009/09/simple-ajax-website-jquery/demo.html
C5	http://test.thurls.com/ajax/home1.php

TABLE I
Test Cases

Table II contains the experimental results obtained for the sample test cases. Probable Clickables are those elements in the DOM which can be clicked. Detected Clickables are those elements that actually trigger change in DOM structure.

Case	Maximum DOM String Size(bytes)	Probable Clickables	Detected Clickables	Number of States
C1	5829	24	8	8
C2	6378	61	11	11
C3	17422	167	27	27
C4	2159	23	5	5
C5	8233	58	26	26

TABLE II
EXPERIMENTAL RESULTS

Table III contains the crawling time for each test case. Also crawl time per state is also shown.

Case	Number of States	Total Crawling time (in mins)	Crawling time per state (in mins)
C1	8	11.44	1.43
C2	11	216.45	19.68
C3	27	607.5	22.5
C4	5	34.9	6.98
C5	26	103.13	3.97

TABLE III
CRAWLING TIME

B. Performance of the AJAX Crawler

The performance of an AJAX Crawler depends on the following factors.

- AJAX Request time
- Network Latency
- Web Server Request/Response time
- Selection of clickables

C. Optimizations

The following are some of the possible extensions or optimizations that can be added to the AJAX Crawler.

- **Using the DOM change statistics between states**

  Consider a transition from state 1 to 2. Now there may be many static elements common to states 1 and 2. So the effect of invoking the events on these static elements in both the states is same. Thus, the elements which get added or changed in DOM when the state changes from 1 to 2 needs to be found out. The events need to be invoked only on those changed elements in state 2. The remaining transitions are same as in state 1.

- **Multi Threading**

  This can be done by having separate HtmlUnit webclient in each thread. To make sure that multiple threads dont crawl the same path, the state machine needs to be synchronized between the threads.

- **Avoid invoking events on nested elements**

  When the event has already been invoked on a element, the event need not been invoked again on the enclosing parent element. For example, Consider the following HTML element,

  ** test **

  Here the clicking the element testhas the same effect as clicking test . Thus event need not be invoked on the enclosing parent element. This would reduce the number of duplicate transitions in the state machine.

VI. CONCLUSION

The implementation of a basic AJAX Crawler has been discussed in this paper. Also possible optimizations have been identified. Crawling AJAX is a difficult problem that is not addressed by current search engines. But crawling AJAX results in improved search result quality. To integrate AJAX Crawler with current search engines, the AJAX crawling has to be highly optimized. The crawling time of a webpage by traditional crawlers is in the order of milli seconds where as the crawling time of a URL by AJAX Crawler is normally in order of minutes. Thus the AJAX crawling time has to be significantly reduced to be integrated with current search engines.

REFERENCES

[1] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Trans. Web*, vol. 6, no. 1, pp. 3:1–3:30, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2109205.2109208

[2] "JUNG," http://jung.sourceforge.net/, 2010.

[3] "GraphML," http://graphml.graphdrawing.org/, 2007.

[4] C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou, "Ajax crawl: Making ajax applications searchable," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ser. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 78–89. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2009.90

[5] "HtmlUnit," http://htmlunit.sourceforge.net/, 2011.

[6] "Selenium Web Driver," http://seleniumhq.org/docs/03_webdriver.html, 2011.