GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

AUTOMATIC TESTING OF MODERN WEB APPLICATIONS

K.Lavanya*[1] & Dr.V.Shankar[2]
[*1]M.Tech Student, Department of CSE, Kakatiya Institution of Technology and Science, Warangal District, Telangana, India

[2]Assoc.Professor, Department of CSE, Kakatiya Institution of Technology and Science, Warangal District, Telangana, India

ABSTRACT

AJAX-based internet a pair of applications considers state full asynchronous client/server communication, and client-side run-time manipulation of the DOM tree. This not solely makes them basically totally different from ancient internet applications, however conjointly a lot of erring and tougher to check. We tend to suggest a technique for testing Ajax applications mechanically, supported a crawler to infer a state-flow graph for all (client-side) program states. We tend to determine AJAX-specific faults which will occur in such states (related to e.g., DOM validity, error messages, discoverability, and back-button compatibility) additionally as DOM-tree invariants which will function oracles to observe such faults. Our method, raised to as ATUSA, is required in an exceptionally tool giving generic invariant examination elements, a plugin-mechanism to feature application-specific state validators, and generation of a check suite covering the methods obtained throughout crawl. we tend to describe 3 case studies, consisting of six subjects, evaluating the kind of invariants which will be obtained for Ajax applications additionally because the fault revealing capabilities, quantifiability, needed manual effort, and level of automation of our testing approach.

Keywords*:* *automated testing, web applications, Ajax.*

I. INTRODUCTION

Web applications are becoming increasingly important in recent years as people more and more tend to interact with Internet to carry out their everyday activities, e.g. browsing, shopping, gaming, working and socializing. Compared to traditional desktop applications, the bar of developing and deploying web applications is relatively lower. To some extent, anyone could rent some space from an Internet Service Provider and post his/her own web applications. On one hand, this substantially facilitates the prosperity of Internet and web applications. On the other hand, it welcomes many web applications that are not sufficiently validated or tested. These applications may cause all sorts of problems, with some of them having financial impact. Web application testing is hence a very important challenge for the health of web application engineering. Despite its importance, testing web applications poses many new challenges. First, web applications are often highly dynamic. A server-side script could generate a huge no. of client-side pages that may seem different and carry out different functionalities, depending on the client side user inputs and the internal states on the server-side. AJAX (Asynchronous JavaScript and XML), a popular client side technique, allows on-the-fly page updates by interacting with server scripts. Such page updates do not need loading a fresh page, but rather self-modify part of the current page (e.g. resizing a frame, repositioning a button, replacing part of the page with an image). Such updates also change both the interface and the functionality of the current client page. These active features make web application testing hard. Recently, researchers have developed various web application testing techniques that aim to address the challenges. Particularly, S. Artzi et al. proposed a test generation technique for dynamic web applications by combining concrete and symbolic executions [8]. Their tool Apollo can automatically generate test inputs to expose faults in web applications, guided by similarity of code artifacts. G. Wassermann et al. developed a test generation technique to handle dynamic features in web applications [29]. Their technique also controls console execution to maximize test coverage of certain code artifacts in server scripts, such as string values, objects, and arrays. While the aforementioned techniques have substantially advanced web application testing, they more or less evolve from traditional software testing techniques and inherit some solutions that may become sub-optimal in the back ground of web application testing due to its dynamic nature. One of such sub-optimal solutions is coverage criterion, which often serves as the stop condition for testing, answering the question whether a test suite is sufficient for a subject program. It is a critical cornerstone for test selection, test prioritization, and test generation. The most popular traditional coverage criterion is code coverage, or more specifically, statement coverage, which measures the percentage of statements that are executed by a given test suite. It is simple, actual and hence widely used in many testing methods, including many of the aforementioned web application testing techniques. However, we observe that statement coverage is sub-optimal for web application testing. Web applications are usually heterogeneous systems, including HTML pages, server-side scripts (e.g. PHP scripts), and client side scripts (e.g. JavaScript) that may be embedded in server scripts or HTML pages, and even SQL scripts sometimes. It is very difficult to measure code coverage of client side scripts or HTML pages as they are usually dynamically generated by server scripts. As such, different user interactions may induce different client-side pages and scripts. It is unclear what would be the universal set of client-side code artifacts that an ideal test suite should cover. Indifference, computing code reporting for server scripts is a controllable problem. In fact, many current web application testing techniques adopt such a criterion [8]. However, it is unfortunately insufficient for web application testing. In particular, one line of server-side script could induce many client-side code artifacts that may exercise different functionalities. A test case cover a server-side line could not promise that the corresponding client-side logic is sufficiently tested.

II. RELATED WORK

Web Crawling. Web crawling is a well-studied problem with still ongoing challenges. A survey of the field of Web archiving and archival Web crawling is available in [5]. A focused, or goal-directed, crawler, crawls the Web according to a predefined set of topics [8], and thus influences the crawler behaviour not based on the structure of Web applications as is our aim, but on the content of Web pages. Our approach does not have the same purpose as focused crawling: it aims at better archiving of known Web applications. Both strategies for are thus complementary. Content in Web applications or content management systems is arranged with respect to a template (which may include left or right sidebar of the Web page, navigation bar, header and footer, main content, etc.). Among the various works on template extraction, Gibson et al. [9] have performed an analysis of the extent of template-based content on the Web. They have found that 40–50% of the Web content (in 2005) is template-based (i.e., part of some Web application), which is growing at the rate of 6–8% per year. This research is a strong hint at the benefit of handling Web application crawling in a specific manner. Forum Crawling. However application-aware packed in general has not yet been addressed, there have been some efforts on content extraction from Web forums. One such approach [10], dubbed Board Forum Crawling (BFC), leverages the organized structure of Web forums and simulates user behaviour in the extraction process. BFC deals with the problem effectively, but is still confronted to limitations as it is based on simple rules and can only deal with forums with some specific organized structure.

III. APPROACH

An overview of our approach is depicted here. First, we extract feature vectors from the collected DOM elements to build a training corpus. Each vector in the corpus has to be labelled by its topic if the corpus is for input topic identification. However, we may label a group of vectors as the same topic at a time instead of one by one. The detail will be provided in Section III-C. Afterwards, for encountered DOM elements, we extract and transform them to project them into the vector space constructed by training data, and do inference based on their similarities to vectors in the training corpus. Here we provide a running example and detailed explanation about how to exploit the proposed technique on input topic identification. Once the topics are identified, the corresponding values can then be selected from a pre-established databank [28] or generated by data models such as smart profile [18]. The same concept can be applied to state comparison and clickable detection without significant changes. A. Feature Extraction A novelty of this paper comparing with other crawling based techniques for web application testing is that we consider not only the attributes but also the nearby tags or descriptions of DOM elements for input concept permit. Algorithm 1 shows how it is achieved. First, we specify DOM attributes such as id, name, placeholder and max length which concern input topic identification in an attribute list, and the values of matched attributes of the DOM element will be put into the feature vector (line 2 to 4). Moreover, to find the corresponding descriptions, we search the siblings of the DOM element for tags such as span and label in a tag list and put the texts enclosed by the tags into the feature vector (line 11 to 18). If no such tags are found, the search will continue on the DOM's parent recursively for several times (line 20). In addition, we perform a couple of normalizations such as special character filtering and lowercase conversion to the words in the extracted feature vector.

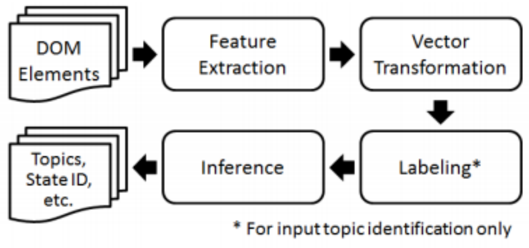
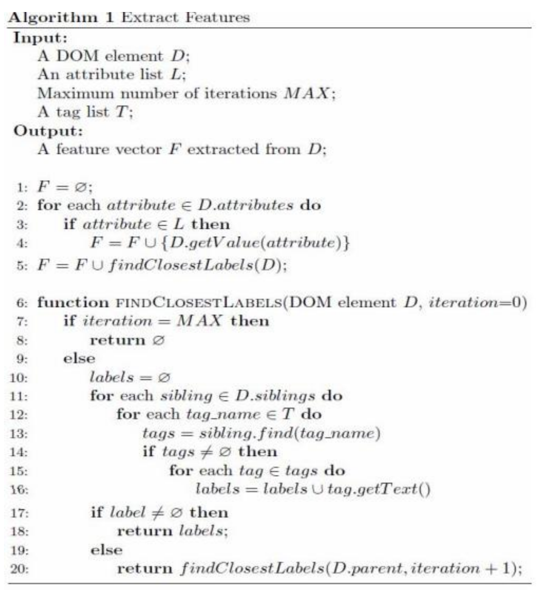


Fig. 1. An overview of the proposed approach



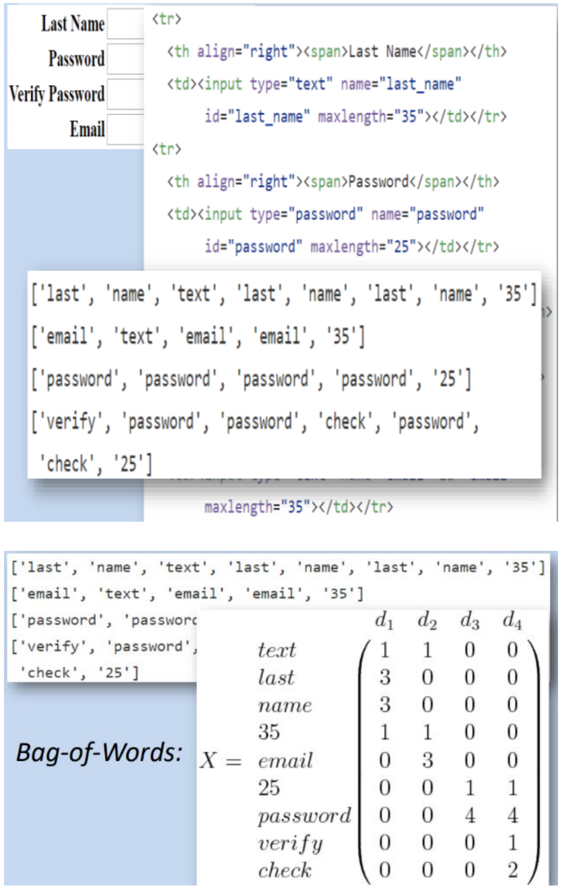
```
Algorithm 1 Extract Features
Input:
    A DOM element $D$;
    An attribute list $L$;
    Maximum number of iterations $MAX$;
    A tag list $T$;
Output:
    A feature vector $F$ extracted from $D$;

1: $F = \varnothing$;
2: for each $attribute \in D.attributes$ do
3:     if $attribute \in L$ then
4:         $F = F \cup \{D.getValue(attribute)\}$
5: $F = F \cup findClosestLabels(D)$;

6: function FINDCLOSESTLABELS(DOM element $D$, $iteration$=0)
7:     if $iteration = MAX$ then
8:         return $\varnothing$
9:     else
10:        $labels = \varnothing$
11:        for each $sibling \in D.siblings$ do
12:            for each $tag\_name \in T$ do
13:                $tags = sibling.find(tag\_name)$
14:                if $tags \neq \varnothing$ then
15:                    for each $tag \in tags$ do
16:                        $labels = labels \cup tag.getText()$
17:        if $label \neq \varnothing$ then
18:            return $labels$;
19:        else
20:            return $findClosestLabels(D.parent, iteration + 1)$;
```

Human interacts with web applications through the text in natural language – but not the DOM structures or attributes

- In mark up language (e.g. HTML and XML), the reserved words for DOM attributes are limited – id, name, type…
- While the words used in text and attributes for input fields of the same topic may be different among web applications, they are usually semantically similar – "last name", "surname", "family name"

[Lavanya, 5(6): June 2018]
DOI- 10.5281/zenodo.1262195

ISSN 2348 – 8034
Impact Factor- 5.070



```
<tr>
  <th align="right"><span>Last Name</span></th>
  <td><input type="text" name="last_name"
      id="last_name" maxlength="35"></td></tr>
<tr>
  <th align="right"><span>Password</span></th>
  <td><input type="password" name="password"
      id="password" maxlength="25"></td></tr>
```

```
['last', 'name', 'text', 'last', 'name', 'last', 'name', '35']
['email', 'text', 'email', 'email', '35']
['password', 'password', 'password', 'password', '25']
['verify', 'password', 'password', 'check', 'password',
 'check', '25']
```

```
      maxlength="35"></td></tr>
```

```
['last', 'name', 'text', 'last', 'name', 'last', 'name', '35']
['email', 'text', 'email', 'email', '35']
['password', 'passworc
['verify', 'password',
 'check', '25']
```

Bag-of-Words:

$$X = \begin{array}{c} \\ text \\ last \\ name \\ 35 \\ email \\ 25 \\ password \\ verify \\ check \end{array} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \\ \end{array}$$

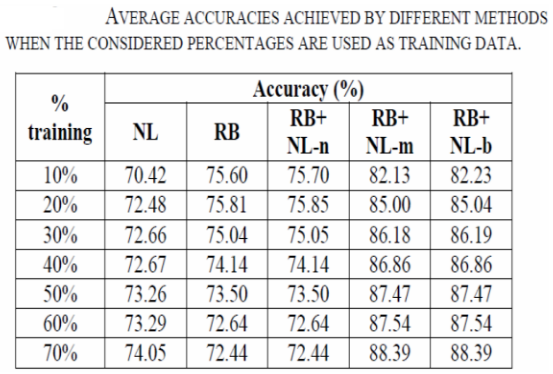
$$X = \begin{array}{c|cccc} & d_1 & d_2 & d_3 & d_4 \\ \hline text & 1 & 1 & 0 & 0 \\ last & 3 & 0 & 0 & 0 \\ name & 3 & 0 & 0 & 0 \\ 35 & 1 & 1 & 0 & 0 \\ email & 0 & 3 & 0 & 0 \\ 25 & 0 & 0 & 1 & 1 \\ password & 0 & 0 & 4 & 4 \\ verify & 0 & 0 & 0 & 1 \\ check & 0 & 0 & 0 & 2 \end{array}$$

Fig 2: Vector Transformation

A. Experiment Result

Steps

- Randomly choose x% of the forms as training data (corpus) – x = 10, 20, 30, 40, 50, 60 , 70
- Generate rules (i.e. mappings from feature strings to topics) using the training forms
- Infer the rest forms with: – The proposed approach (NL) – Rule-based approach (RB) – RB+NL-n (no-match) – RB+NL-m (multiple-topic) – RB+NL-b (both)
- Repeat 1000 times


AVERAGE ACCURACIES ACHIEVED BY DIFFERENT METHODS WHEN THE CONSIDERED PERCENTAGES ARE USED AS TRAINING DATA.

% training	Accuracy (%)				
	NL	RB	RB+ NL-n	RB+ NL-m	RB+ NL-b
10%	70.42	75.60	75.70	82.13	82.23
20%	72.48	75.81	75.85	85.00	85.04
30%	72.66	75.04	75.05	86.18	86.19
40%	72.67	74.14	74.14	86.86	86.86
50%	73.26	73.50	73.50	87.47	87.47
60%	73.29	72.64	72.64	87.54	87.54
70%	74.05	72.44	72.44	88.39	88.39

IV. CONCLUSION

In this paper, we proposed a natural-language technique to improve the effectiveness of crawling-based web application testing. By considering semantic similarities between a training corpus and a DOM element to be inferred, input topic identification, GUI state comparison and clickable detection can be performed with the proposed approach. In the future, we plan to evaluate how the proposed techniques impact overall crawling efficacy with more data and other topic model alternatives such as LDA. Moreover, the proposed feature extraction algorithm could be improved with more information about DOM elements such as comments.

V. FUTURE WORK

The impact overall crawling efficacy with more data and other topic model alternatives such as LDA

- Information retrieval from, e.g., comments, of DOMs
- Mobile apps ?

REFERENCES

1. *M. Burner. Crawling towards eternity: Building an archive of the World Wide Web. Web Techniques Magazine, 2(5), 1997.*
2. *J. Cho and H. Garcia-Molina. Parallel crawlers. Technical report, 2001.*
3. *A. Christensen, A. Feldthaus, and A. Moller. Precise analysis of string expressions. In SAS'03, pages 1–18, 2003.*
4. *C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou. Ajax crawl: making AJAX applications searchable. In ICDE'09, pages 78–89, 2009.*
5. *P. Frankl and E. Weyuker. An applicable family of data flow testing criteria. IEEE Transactions on Software Engineering, 14(10):1483–1498, 1988.*
6. *E. Geay, M. Pistoia, B. Ryder, and J. Dolby. Modular string-sensitive permission analysis with demand-driven preciion. In ICSE'09, pages 177–187, 2009.*
7. *Jupp, E.: Obama's victory tweet 'four more years' makes history. The Independent (November 2012), http://ind.pn/RF5Q6O*
8. *Coleman, S.: Blogs and the new politics of listening. The Political Quarterly 76(2) (2008)*
9. *Mulvenon, J.C., Chase, M.: You've Got Dissent! Chinese Dissident Use of the Internet and Beijing's Counter Strategies. Rand Publishing (2002)*
10. *Giles, J.: Internet encyclopaedias go head to head. Nature 438 (2005)*
11. *Masanès, J.: Web archiving. Springer (2006)*
12. *Sigurðsson, K.: Incremental crawling with Heritrix. In: IWAW (2005)*