WebVizOr: A Fault Detection Visualization Tool for Web Applications

Barbara Hazelwood
Mathematics and Computer Science
Xavier University
Cincinnati, OH 45207
hazelwoodb@xavier.edu

Holly Esquivel
Computer Science and Information Systems
University of Nebraska at Kearney
Kearney, NE 68845
esquivelhm@unk.edu

1 Introduction

Businesses, governments, and consumers increasingly rely on the stability, security, and usability of web applications. However, the scale of such applications can make the verification process both time-consuming and laborious. To reduce the overhead of the testing process and to ensure proper application behavior, testers need automated, cost-effective test strategies to develop, execute, and analyze the success of test cases.

In this paper we present WebVizOr (Web Application Fault Detection Visualization with Oracles), an open-source tool which aids in the analysis of test case results. Our tool takes as input test cases, which are a series of HTTP requests sent to a web application, and the HTML responses generated on executing those test cases. In its simplest usage, WebVizOr provides a means for navigating through and viewing the HTML responses, which comprise the test case results. Beyond visualization, WebVizOr harnesses the power of various oracles to automatically analyze and compare the HTML responses from different versions of the same application in order to locate symptoms of possible faults. This last usage is especially pertinent in a world of ever-evolving applications, in which the first version is almost never the last. As applications grow and change, the need for regression testing is apparent.

2 Background

In this section we discuss capture/replay testing techniques, which were the motivation behind the development of WebVizOr. We also discuss four previously defined web application testing oracles which have been integrated into the WebVizOr system.

2.1 Capture/Replay Testing

WebVizOr was originally conceived to facilitate the use of a capture/replay testing framework. In capture/replay testing the actual deployed behavior of a previous version of the application is captured by logging each HTTP request made to the system. These requests are then organized into test cases by some standard (e.g. all requests made by a given user or in a given time frame) and are then replayed against a new version of the application to verify correctness of the new version. For a more detailed look at a capture/replay framework based on user sessions, see [4].

2.2 Oracles: HTML Comparator Algorithms

When a test suite (a collection of test cases) is executed against a previous version of the application it evokes the **expected** results. When the same test suite is executed against a new version of the application it evokes the **actual** results. Expected and actual results can be compared for the purposes of detecting faults in the new application version. In [4] four basic comparator algorithms - oracles - for validating HTML output were defined. All of these algorithms involve executing a simple diff command to compare the expected output with a corresponding actual output. The oracles diverge in the ways in which they pre-process the outputs before comparing them.

The first basic oracle is called **Raw** because it executes a diff on the unfiltered, source HTML. This oracle will detect any changes that have occurred in the HTML. While inexpensive, the Raw oracle returns a high rate of false positives due to its sensitivity. It will

[0]This paper is under submission. Do not reproduce without permission.

detect every change in layout, dynamic content, font color, etc., even when those changes do not constitute actual faults.

The second basic oracle is a content-based oracle which we call **Text** ("Content" in [4]). Before the diff is run, the raw HTML is filtered of all tags and tag attributes, leaving only the textual content of the output. This oracle avoids detecting false positives that are due to formatting changes, however it will still detect changes to dynamic text content which are not always faults.

The third basic oracle is a structure-based oracle which has two variations that we call **Tag** ("Structure" in [4]) and **Tag++**. They are similar in that they exclude the textual content of the output and focus on the tags within the HTML. Tag filters out everything except for the tag names. Tag++ goes beyond this and includes certain tag attributes. Structure-based oracles avoid false-positives do to dynamic text content but can incorrectly fail pages that have only slight changes to the GUI which do not affect application behavior.

The fourth basic oracle examines the results of the test suite at a higher level to ensure that all of the responses expected were actually returned. This oracle, which we call **Path** ("Flist" in [4]), executes a diff on the list of HTML responses in the expected and actual test suite results. Path catches errors that resulted in URL redirection or in the server not generating HTML responses. It is useful as a precursor to the other oracles by identifying which responses in the expected results have no counter-part in the actual results.

2.2.1 Other Diff Algorithms

There is some literature in data managements about how to diff semi-structured data [1, 2]. They used their algorithms to diff two web pages, but the algorithms seemed slow and would not likely be practical for our purposes.

3 Tool Features

This section will describe in more detail the features of WebVizOr that are key to our unique automated design.

3.1 View Saved HTML Responses

WebVizOr presents HTML responses to the user via a single view or comparison view. Single view displays a single set of HTML responses, presumably from the execution of a single web application version. Comparison view displays two sets of HTML responses side-by-side, presumably from the execution of a single test suites against two different web application versions (the **actual** and **expected** results). Single view allows the user to quickly check an HTML response in its rendered or HTML source code form. Comparison view allows the user to compare responses, which should ideally be the same, with the aid of oracle comparator results. In both single and comparison view, the user may choose to view the HTML responses in rendered form. This rendered form does not reference any style sheet information that may be a part of the actual web application. However, the user should be able to effectively analyze the responses even without the style sheet rendering.

3.2 Navigate through HTML Responses

Efficiently navigating through HTML responses is crucial to the automation of fault detection. Navigating through responses has primarily become a problem because of the hundreds of various responses that can produced by a single web application [3]. When viewing responses in WebVizOr in either the single or comparison view the user has the ability to iterate forward and backward through the entire test suite. This includes the ability to navigate to the previous or next response and to the previous or next test case. The user may also choose to utilize WebVizOr's navigation frame to jump to any response in the test suite. All test cases are displayed in the navigation frame as folders. If WebVizOr encounters a test suite with more than 30 test cases it automatically sorts the test cases in alphanumerical order and places the test cases into sub-folders. If each sub-folder contains more than 40 test cases then the test cases within each folder are again split up into sub-folders and so on. The sub-folders greatly reduce the loading time of the test suite directory in the navigation frame as well as reduce the time for the user to find and jump to a specific test case or response.

3.3 View Detailed Test Case Information

Web applications often respond differently based on the parameters that accompany an HTTP request. In order to analyze why certain HTML responses have been received from a web application, users may wish to view the details of the original request. When viewing a response from a test case, all of the HTTP requests that made up that test case can be displayed in a lower pane. To further utilize the detailed request pane, WebVizOr allows the user to click on one

Apps	Description	# Test Cases	Total # Requests	Avg. Case Size
Masplas	Workshop registration/management	169	1,107	6.6 requests
DSpace	Digital publications library	1,800	22,129	12.2 requests

Table 1. Example Web Applications

App	Usage	Init Time	Open Browse	Switch Test Case	Switch Requests
Masplas	Single	1 sec	1 sec	1 sec	1 sec
	Comparison	5 sec	1 sec	4 sec	4 sec
DSpace	Single	5-15 sec	1 sec	1 sec	1 sec
	Comparison	22 sec	2 sec	5 sec	4 sec

Table 2. Time Costs

of the detailed requests displayed and jump to the corresponding response in the test case.

3.4 View Oracle Comparator Results

Viewing rendered and raw forms of an HTML response can be an inefficient way to analyze possible faults between two responses, thus an essential part of our tool is the ability to view sets of individual HTML responses after they have been analyzed by various oracles. Each oracle comparator is run on the two responses being analyzed when a user navigates to that pair of responses. WebVizOr displays the oracle processed actual and expected response in individual panes and highlights the differences between them based on the results returned by the comparator. The buttons to view individual oracles are themselves highlighted if the oracle comparator detected differences. The user can then select which oracle results they would like to view to determine if they believe the fault detected is truly a fault or simply a false positive. None of the oracles we have implemented can make this distinction themselves, thus by viewing a combination of filtered oracle results in WebVizOr the user may be able to determine if a real fault was found.

3.5 View Fault Reports

WebVizOr is designed to utilize the vast amount of information collected during test suite execution to generate fault reports for a given suite. Fault information can only be utilized when using WebVizOr's comparison view, because two sets of HTML responses are required for oracle comparator results to be created. The information that WebVizOr utilizes to create the fault reports must be generated beforehand by running the oracles against the entire test suite. This can be a lengthy - but automated - process, after which fault reports can be immediately accessed through WebVizOr.

WebVizOr's fault report includes oracle results from all of the response sets in the test suite. The user may choose to view the fault report as a suite where all test cases are listed with a summary of the number of responses with possible faults in them per oracle. The suite view also includes oracle result details for each response in the test suite in which a possible fault was detected. The user may also choose to view a fault report for a specific oracle. Both fault report views allow the user to jump to an individual response or test case, which is then displayed in the main WebVizOr window.

4 Implementation

WebVizOr is itself a web application. Our implementation allows WebVizOr to have a very friendly GUI front end with a modular back-end. The GUI front end is built using JSPs and HTML that communicate with a Java servlet back-end. Oracles are executed from the Java back-end and are written in either Perl or Java. Bash scripts are used to gather a list of information about possible faults when various oracles are performed, and this information is then used to generate the fault report [4].

5 Evaluation

To evaluate the effectiveness of our tool and of the four basic oracles we put WebVizOr to work on two example web applications. See Table 1.

Usage	Dynamic HTML Files	Filtered Text Files	Comparison Files
Single	2	0	0
Dual	2	2 per oracle	1 per oracle

Table 3. Space Costs

5.1 Time and Space Costs

There are lag times associated with different functions of WebVizOr, some of which are affected by the size of the test suite. These differences are apparent in the comparison between DSpace and Masplas results; DSpace is a much larger application and has a larger test suite. Table 2 summarizes the timed results for various tool functions.

WebVizOr creates various temporary files during run-time. The number of files is static for a given usage of the tool (single view vs. comparison view) and does not fluctuate with test suite size. Dynamic HTML files are generated once for each new test suite. In comparison view, new filtered text files and comparison text files are written over old filtered and comparison files with each response set that is viewed. If the tool is exited correctly, these temporary files will be removed. Table 3 summarizes details of the temporary files.

5.2 Ease of Navigation

As described above, WebVizOr provides various navigation options. The most versatile option is the test suite navigation frame, which allows the user to jump to any response in the test suite. To manage potentially large test suites, test cases within the test suite navigation frame are grouped into folders and sub-folders. When running WebVizOr for the DSpace test suite of 1,800 test cases it never took more than five mouse clicks to arrive at a given response. For Masplas it never took more than four. Both of these upper limits were the extreme cases, in which all of the folders and sub-folders were collapsed.

5.3 Oracle Integration

Four basic oracles have been integrated into WebVizOr that are executed and utilized to produce a fault report that organizes the oracle results into an easily accessible form. The execution of all oracles on an HTML response is completely automated and occurs in current time when utilizing the tool. The fault report requires that a specifically formatted file be generated prior to tool usage that contains a summary of the oracle results. To accomplish this, the oracles must be executed against the expected and actual results of the entire suite. While this execution is time costly, it has be automated by our research group.

In addition to the four basic oracles, we have begun to experiment with new more complex oracles. Some of these oracles have been successfully integrated into a new version of the tool.

5.4 Extensibility

Adding a new Perl or Java-based oracle to the run-time functionality of WebVizOr is as simple as adding a single line to the oracle configuration file. This will not, however, integrate the new oracle into the fault report. As mentioned above, the fault report is dependent on a file that must be generated in advance. To integrate a new oracle into the fault report, steps must be taken to integrate the oracle into this pre-process and to include the new results in the generation of the fault report.

5.5 Portability

The issue of portability is still under investigation. This tool was developed within the Linux environment and is fully functional within that environment.

6 Conclusion and Future Work

In this paper we have presented a new tool, WebVizOr, for the automation of fault detection in web applications. WebVizOr supplies a way for web application testers to verify the correctness of a new application version by comparing its responses with the responses that were obtained from a previous version of the application. We have discussed the motivational background of our work including capture/replay testing as well as HTML oracle comparator algorithms, our implementation of WebVizOr and its features, and an evaluation of WebVizOr current capabilities. Our future work includes increasing WebVizOr's portability on various platforms and investigating additional oracles or improving our current oracles.

References

[1] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *International Conference on Management of Data*, pages 26–37. ACM SIGMOD, May 1997.

[2] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchicaally structured information. In *International Conference on Management of Data*, pages 493–504. ACM SIGMOD, June 1996.

[3] James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *International Conference on Software Engineering*, pages 467–477. ACM Press, May 2002.

[4] Sara Sprenkle, Emily Gibson, Sreedevi Sampath, and Lori Pollock. Automated replay and failure detection for web applications. In *International Conference on Automated Software Engineering (ASE)*, pages 253–262. IEEE/ACM, November 2005.