Proc. of the 11th Software Enginnering and Knowledge Enginnering Conference (SEKE), June 1999.

Constructing Control-Flow-Based Testing Tools for Web Application

Ji-Tzay Yang, Jiun-Long Huang, Feng-Jian Wang
{jjyang,jlhuang,fjwang}@csie.nctu.edu.tw

Computer Science and Information Engineering
National Chiaotung University
Hsinchu City , Taiwan 300

&

William. C. Chu
chu@cis.thu.edu.tw

Computer Information Science
Tunghai University
Taichung City, Taiwan 400

Abstract

Flexible and rich application frameworks of Web-based software design make Web-based applications more prevalent in both Internet and Intranet environments. Programmers enjoy various of Web application frameworks whose support ranging from simple user interactions based on plain client-server model, to complicated distributed-object computations based on CORBA. The varity gives user the flexibility to decide a proper framework, and leads to the demands of new support tools and testing framework to test and maintain Web applications.

In this paper, a architecture is proposed to support the testing of web-based applications. The architecture covers application model extraction, test execution automation, and test design automation. In addition, a graph-based application model is also presented to model the behavior of web-based applications. With the graphic presentation, several traditional software testing techniques are extended to test Web-based applications.

1. Introduction

Current Web model and its related improvement give Web-based application designers flexibility at choosing proper development products for their implementation. One Web-based application usually provides interactive services by mrans of a sequence of Web pages containing text, images, and user-fillable form associated with computation on the Web server. To reduce load on application server and the programming effort for user interaction, user may moves part of computation to the Web browser side by means of embedding browser-executable components such as Java Applet or Microsoft ActiveX components inside Web pages.

When Web apploications become large, more advanced Web application frameworks, such as Cold Fusion, are needed to implement many desired features such as database connection, n-tire Web application, load-balancing, session management, and security management. In these frameworks, lots of software components be applied directly or customized are provided to perform interaction at Web browser and database connection at the Web server. Behavior and glues of components are described in scripting language such as JavaScript or Microsoft VBScript.

Current developers of large Web-based application do not have sufficient and powerful tools to debug or test their Web applications. [7] also addressed the necessity of software testing support to handle the complexity of Web applications Existing Web testing tools on Internet are usually made for verifying the syntax in HTML documents, checking the hyperlink integrity in a set of HTML documents, testing GUI components embedded in browsers, and measuring the performance of the Web application. Few testing tools consider Web applications's control and data flow via static and dynamic analysis. There are still some significant characteristics of Web applications need to be studied, such as the control and data dependency of Web application's programming modules. This paper proposes a Web application model by extending control flow graph to model Web application's control. The extended model is further used to generate test cases by applying traditional flow-based test cases generation techniques.

This paper is organized as follows. Section 2 reviews the current approaches to implement Web applications. Section 3 describes the behavior model for Web applications. Section 4 describes the testing environment and its architecture, which includes test case recording tool, test case composition tool, execution tools, and load testing tools. Finaly, section 5 gives conclusion and future works of the web application testing environment.

2. Characteristics of Web Applications

Web applications are composed of static HTML documents and programs run at both server and client sides. HTTP (Hyper-Text Transfer Protocol) is used for communication between Web browsers and Web servers. Web servers and database servers are usually connected by *de facto* database access protocols such as ODBC (Open Database Connection) or JDBC (Java Database Connection). The following subsections introduce the architecture overview and activities involved in the Web applications implementation. Meanwhile, the desired testing support are also addressed.

2.1. Web Application Architecture

Web application architecture shown in Figure 1 is supported by current Web-related techonologies. It consists of three major tiers, the Web browser, the Web server, and optional database servers. The information process in the application is passed throught each tier. The user interaction is performed at the Web browser tier. The program logic is done at the Web server tier. The database processing is done at the database server tier. Hence, the Web application architecture is also known as a *three-tier* application architecture. When the database server tier is omitted, it is known as a *two-tier* application architecture.

The Web browser is capable of retrieving hyper-text documents, as requested by the application users, from the Web server via HTTP protocol. It renders the hyper-text document in HTML (Hyper-Text Markup Language) format on the screen. Contemporary Web browsers also embed Java VM and interpreter to execute the Java Applets or Java Scripts specified in the documents. The browser also allows users to extend its functionality by installing additional software modules such as Netscape Communicator's *plug-in* modules and Microsoft Explorer's ActiveX objects.

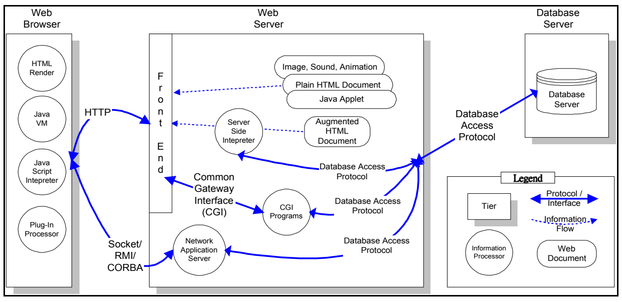


Figure 1. Web Application Architecture

The Web server has a frontend (HTTP daemon) to accept the HTTP requsets from the browsers. Accroding to the server's configuration, it may directly serve the stored HTML documents, Java Applets, or multimedia files, or forwards the requests to CGI programs, by which the HTML documents to be returned are generated dynamically. Some Web servers also have modules [4][8] to interprete the augmented HTML documents before sending them to browsers.

In advanced Web applications, components in Web browsers may communicate with other components in the Web servers with protocols which are not HTTP. These emerging protocols such as CORBA are more fitting when developing distributed Web application in object-oriented technology.

2.2. Programming at Web Clients and Servers

Web client programming focuses on the visual presentation and user interaction. The program guides the user to input data and validates user's input before submitting it to the Web server. For more complicated interaction, software compnents such as Java Applets or Active X objects can be embedded in Web pages. Script languages such as JavaScript or VBScript can be applied to glue the components.

Besides deploying static documents on the Web servers, desingers write augmented HTML documents or CGI programs to achieve more functionalities by dynamically generating HTML documents at run-time. The augmented HTML documents are interpreted at server side by interpretation modules such as ASP interpreter or Perl interpreter. Programs (or scripts) within the augmented HTML documents or CGI programs are usually passed the well-encapsulated HTTP requests which originates from the Web browsers. They are also provided with session managment and database connection facilities. Examples of the server technologies are Microsoft Active Server Page(ASP) [8], Sunsoft Servlet [11] and Cold Fusion's CFML [1]. Database connections can be provided by Microsoft IDC(Internet Data Connection), ADO(Active Data Object), ODBC(Open DataBase Connection), or JDBC (Java DataBase Connection).

Testing supports at the server usually help either to test a set of documents to check the integrity of the Web links among the documents, or to test the CGI programs as traditional application. However, few tools help to generate http transactions as test cases based on the application behaviors which include application logics performed at server side and user's decision made at browser side.

3. Web Application Models

In this section, a method is proposed to model the constitutents of Web application in order to apply traditional test case generation strategies.

3.1. Extracting an Application Model

A Web application consists of a set of programming modules which are executed at the servers or the browser. To build the control flow graph for the Web application, our proposed method maps each entity in the Web application to a component of the control graph, i.e. nodes, branches, edges. The extended control graph for the Web application contains the following symbols as appeared in Figure 2: programming module (node), user branch, application branch, hyper-link edge, HTTP-redirect edge, and intra-module edge.

A *node* in the control flow graph represents a programming module. A programming module is usually implemented in a single file such as *.html* file in HTML, *.cgi* file in Perl, or *.asp* file in ASP. There may exist a more detailed control flow graph residing in the single programming module, but it is not discussed when considering the inter-module control graph.

During the mapping of control flow branch, the extended model classifies the *branches* into the *user branch* and the *application brach*. The *user branch* models that the user selects one of hyperlinks from the browsed document at browser side. The application branch models that the current programming module forwards the control to other programming modules for further processing based on the application logic.

When a programming module *M* generates an HTML document containing hyperlinks to modules M_1, M_2 and M_3 for user's selection, node *NM*, NM_1, NM_2 and NM_3 are created to represent programming module *M*, M_1, M_2 and M_3 respectively in the control flow graph. In addition, an user branch symbol *UB* is placed among *NM*, NM_1, NM_2 and NM_3. An intra-module edge is used to associate *NM* and *UB*. Each branch alternative from *UB* to NM_1, NM_2 and NM_3 respectively is represented by one hyper-link edge.

When a programming module *M* needs to transfer the control to module M_1, M_2, *or* M_3 for furthur processing based on the application logic, an *intra-module* link is used to connect the node for module *M* with an *application brach* symbol to represent the control transfer made by application logic. Each application branch alternative is connected by an *HTTP-redirect edge*. The name '*HTTP-redirect*' comes from the technique which achieves the control transfer between Web programming modules by sending HTTP redirect command to the browser. When a module *X* needs to transfer control to module *Y*, it sends to the browser an HTTP-redirect command containing the URL referring to module *Y*. On receiving the URL, the browser sends a new HTTP request which invokes module *Y* to the server. Thus, the application's control transfers from module *X* to module *Y*.

The control flow graph in Figure 2 can be constructed by analyzing its programming modules. In the figure, there are two HTTP-redirect edges from *orderStart.asp* to two independent modules. In other programming practices where no http-redirect is applied, the application logic implemented in *availOffer.asp* and *incomplete.html* are merged into one single *orderStart.asp*. Hence, the node for *orderStart.asp* would contain four hyper-links *memberInfo.html*, *availDrinks.asp*, *availPizza.asp* and *confirm.asp*. The practice can reduce the number of programming modules, hoever it still increases the difficulties of program understanding and testing. In fact, many links are not easily determined by simple analysis on the programming modules. Following subsections introduce static and dynamic analysis methods, which can obtain a control flow graph more precisely.

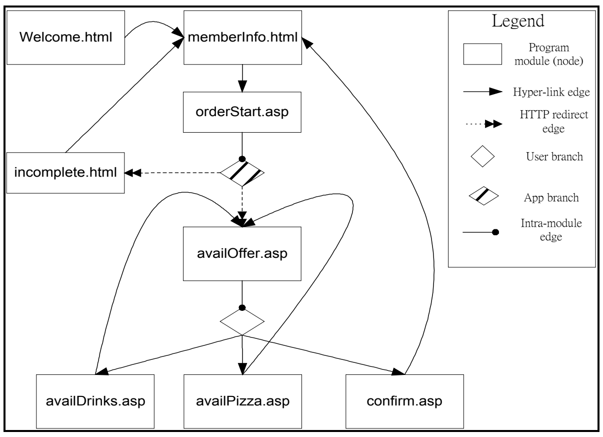


Figure 2. A sample control flow graph of Web application

3.2. A Model Obtained by Static Analysis

In static analysis, the source code of the programming modules are analyzed to extract the inter-module relations. For Web applications implemented in ordinary Web application frameworks, the application logic is scattered in several files. There are two popular approaches to specify application logic. One is based on document content, it specifies application logic inside some blocks of augmented HTML files. The other is based on application logic, it specifies logic in script languages like PERL and lets the script output the desired HTML file.

When constructing a model through static analysis, the application designed in the content-oriented approach can be extracted richer link information due to its format simplicity. Analyzers can extract inter-module information from the augmented HTML file (or HTML template) by extracting attributes from certain HTML tags. For example, the following fragment of an augmented HTML file can be extracted hyper-links shown in ***bold-italic*** typeface.

```
<a href=memberInfo.html …>
…
<form action=orderStart.asp …>
…
```

```
<frame src=leftPane.html>
…
<image src=roadmap.gif  usemap=#mapDef>
…
<script>
  …
  WindowObject.href="foo.html"
  …
</script>
…
<map name=mapDef>
  <area shape=rect ... href=site1.html>
  <area shape=rect ... href=site1.htm2>
  <area shape=rect ... href=site1.htm3>
</map>
```

Figure 3. Hyperlinks inside a programming module

However, for the following fragment written in Microsoft ASP, it is difficult to extract the five links generated by the fragment with static analysis only. The dynamic analysis in the next subsection can help to extract the links generated dynamically.

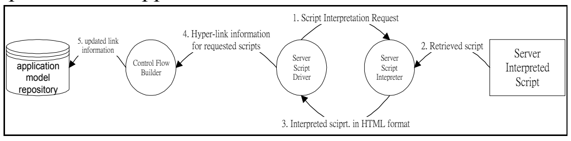
```
<ul>
<%
for i = 1 to 5
  ref = "option" & i & ".html"
  Response.write("<li> <a href=" & ref & ">"
& i & "</a>" )
next
%>
</ul>
</html>
```

Figure 4. A sample *.ASP* file which needs dynamic analysis on the Web server

3.3. A Model Obtained by Dynamic Analysis

The dynamic analysis on the Web application can extract the link information by driving (loading) the program modules to its interpreting engine. As mentioned above, programming modules may be executed at servers or browsers. Supporting tools may be desinged to analyze the following information:

(1) The link information of the programming module after the server interpretation: For example, the ASP scripts shown in Figure 4 contains no link information at static analysis. After the supporting tool drives it into the server's interpretation engine, five links (i.e. option1.html, option2.html, ... and option5.html) can be extracted from the interpreter's HTML output. A server script driving tool shown in Figure 5 may be provided to support the extraction.



1. Script Interpretation Request
5. updated link information
4. Hyper-link information for requested scripts
2. Retrieved script
application model repository
Control Flow Builder
Server Script Driver
Server Script Intepreter
Server Interpreted Script
3. Interpreted sciprt. in HTML format

Figure 5. Dynamic analysis for server interpreted scripts.

(2) The link information of the programming module after the browser interprets the client-side scripts: For example, by drving the following JavaScript to the browser's interpreter, two more links (i.e. site1.html and site2.html) can be obtained. It is similiar to case (1) but the difference is in the way to get the links information, as shown in figure 6.

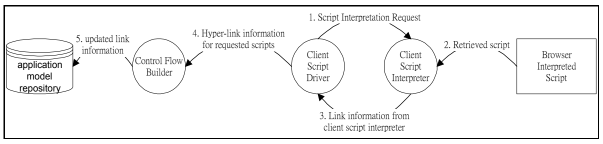

1. Script Interpretation Request
5. updated link information
4. Hyper-link information for requested scripts
2. Retrieved script
application model repository
Control Flow Builder
Client Script Driver
Client Script Interpreter
Browser Interpreted Script
3. Link information from client script interpreter

Figure 6. Dynamic analysis on browser interpreted scripts.

(3) The link information generated during user interaction on the Web page: The programming module may dynamically generate hyper-link request based on user's interaction on the Web page. The type of link information extraction requires a person or automatic GUI driving tools to drive the user interface. Thus, a link monitor in Figure 7 can extract the link information.

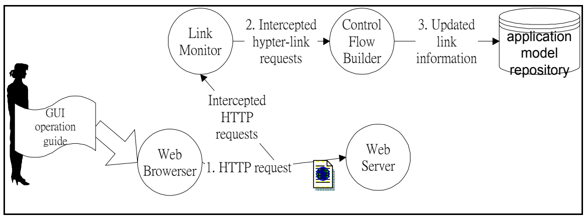

Link Monitor
2. Intercepted hypter-link requests
Control Flow Builder
3. Updated link information
application model repository
Intercepted HTTP requests
GUI operation guide
Web Browerser
1. HTTP request
Web Server

Figure 7. Analysis on hyper-links generated by user interactions.

3.4. Generating Test Case for Web Applications

With the application model constructed for the Web applications, test case generation techniques based on control-flow graphs, such as path testing, can be applied to test Web applications directly [6][9][10]. Two path testing strategies, statement and branch coverage, are adopted in the environment. Practical program testing requires both the statement and branch coverage. IEEE software test standard [2] regards the statement coverage as the minimum testing requirement. In the Web application testing, the term *statement* is replaced with *programming module*.

For example, based on the statement coverage strategy, one set of test cases generated for the sample application shown in Figure 2 are listed as the following:

TestCase#	1	2
Nodes In Path	-welcome →memberInfo →orderStart →incomplete	-welcome →memberInfo →orderStart →availOffer →availDrinks
TestCase#	3	4
Nodes In Path	-welcome →memberInfo →orderStart →availOffer →availPizza	-welcome →memberInfo →orderStart →availOffer →confirm

If the branch coverage stragtegy is used to generate test cases, more paths are included in addition to above four paths. Paths containing edges *availDrinks→availOffer*, *availPizza→ availOffer*, and *confirm→memberInfo* must be included to make every branch alternative exercised.

4. A Web Application Testing Architecture

An application testing environment helps to automate the testing process and integrate testing tools to support testing during the test process. [5][13] have evaluated the architectures and capabilities of application testing environments. The Web application testing environment introduced here is to provide integrated tools to support the testing requirments as mentioned in section 3.

4.1. An Architecture Supporting Web Application Testing

As shown in Figure 3, the Web Application testing tool architecture consists of five subsystems, Source Document Analysis, Test Management, Test Development, Test Failure Analysis, Test Execution, and Test Measurement. Each subsystem cooperate with each other to achieve testing activties. A set of unified access interfaces for each subsystems are provided to separate the interfaces from their implementations, and it makes each subsystems more reusable and easy maintained.

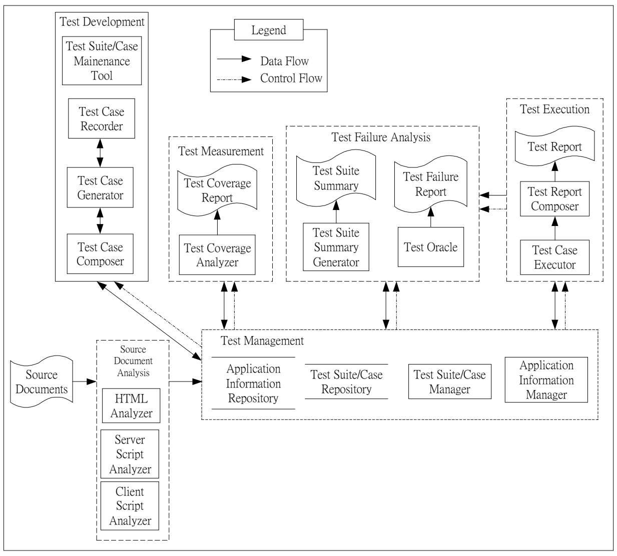


Figure 8. The Web Application Testing Architecture

The location and related parameters of the Web application under testing must be provided to proceed with subsequent testing activities. The testing activties are grouped into the following four major steps: (1) application model construction,, (2) test case construction and composition, (3) test case execution, and (4) test result validation and measurement.

Application model construction is the responsibility of Source Document Analysis subsystem. The *HTML Analyzer* fetches the programming modules of the application and performs static link analysis on the source documents of these modules. The S*erver Side Analyzer* performs dynamic link analysis on the HTML output by interpreting the source code with the WWW server.s The C*lient Script Analyzer* performs dynamic link analysis at client side by driving the WWW browser to fetch the programming modules and interprete the client-side scripts. Application models are built by these analyzer and stores in *Application Repository* in Test Management sunsystem.

To construct test cases, *Test Case Generator* can generate test cases automatically with user-selected test criterion mentioned in section 3.4 and other parameters. Besides, users can create test cases manually by using *Test Case Recorder* or *Test Case Composer*. *Test Case Recorder* intercepts and records HTTP communications between Web browser and servers to form test cases based on Web application user's real use-case. *Test Case Composer* is an environment for users to create test cases by writing test scripts directly.
To specify test cases in a more flexible way the test case designer can resort to *handcrafting test scripts* by himself. Because the testing environment stores the outcome of test case recorders and test case generators in the test repository in the form of test scripts. Test desingers can change test cases by modifying test scripts.

4.2. The Execution of Testing Cases

In test cases execution, the T*est Case Executor* interpretes designated testing scripts and sends corresponding HTTP requests. Testing staffs can choose to run the test cases in batch mode or in interactive mode. In the batch mode the test cases are executed in background without rendering the HTML documents. In the interactive test case execution mode, testing staffs watch the Web pages under testing on the Web browser and control each step of the test case execution. For each test case the control panel of the test case player provides testing staffs with information about the number of Web pages to be visited, the data values to enter in HTML forms. Testing staffs lead the test case to next step or specific step by pressing the '*next step*' button or 'Goto N-th step' button at the control panel.
For Web pages requiring data input, testing staffs can request the test case player to fill data in the form fields on the Web page by pressing the '*fill data*' button. The functionalities save testing staffs from manually filling testing data or locating hyperlinks to invoke on the Web page. It is especially useful when testing HTML forms with lots of data fields to fill.

4.3 Test Result Validation and Measurement

The test case validation is made by the testing staffs or *Test Oracle*, and the validation result is inserted into the test repository for generating testing report.
Test Oracle is an component with expected patterns in the returned HTML documents, and validates the test results of application. A language is used to specify the expected test results. However, for complicated Web applications, it needs human attention to validate the test results. No matter what method is applied, test results are store in *Test Suite/Case Repository* in Test Management subsystem.
Test measure includes test coverage measurement and analysis. *Test Coverage Analyzer* is designed to measure whethre and how much a test criterion is adequately satisfied. Since it is expensive to achieve 100% coverage, testers can decide one proper rate of coverage based on their constraints in testing.

4.4. Implementation of the Architecture

A prototype of the testing architecture have been developed and demonstrated. The client script driver is written in JavaScript running at Web browser. The server script driver is implemented as a Java application. Hyper-link information obtained from script drivers are analyzed by PERL to form application models. Test case recorder/player are implemented as a Web proxy server with additional features.
Users of the testing environment can manipulate the tools from Java Applets embedded in Web pages. The controlling interface implemented as Java Applet provides GUI for user interactions and communicates with the testing center via Java RMI.

5. Conclusion

This paper models the inter-module relations of Web applications in terms of control flow graph and data flow graph. In addition, an architecture is provided to help construct application model by analyzing the programming modules statically and dynamically. Test cases of the Web application can be generated based on the application model or composed through the test case composition interface. Automatic testing tools such as *Test Case Executor* execute testing scripts to automate Web application test. The prototype of the testing tools has been implemented and integrated in the Web environment.

Bibliography

[1] Allaire Corp., *Cold Fusion*, in http://www.allaire.com/products/COLDFUSION/.

[2] ANSI/IEEE Std 1008-1987, "*IEEE Standard for Software Unit Testing"* in Collection of ANSI/IEEE standards on software engineering, IEEE Computer Society Press, 1987.

[3] ANSI/IEEE Std 829-1983, "*IEEE Standard for Software Test Documentation"* in Collection of ANSI/IEEE standards on software engineering, IEEE Computer Society Press, 1987.

[4] Apache Server Project, "*Module mod_include*", in http://www.apache.org/docs/ mod/mod_include.html.

[5] Eickelmann N.S. and Richardson D.J., "*An evaluation of software test environment architectures",* IEEE Proceedings of ICSE-18, p.p. 353—364, 1996.

[6] Frankl, P.G., and Weyuker, E.J., "An applicable family of data flow testing criteria", IEEE Transactions on Software Engineering, Vol 14, p.p. 1483—1498, 1988.

[7] Fromme B., *Web Software Testing – Challenges and Solutions*, InterWorks '98 Conference, 1998. Also available in http://www.interworks.org/conference/IWorks98/sessions/sn135/paper.html

[8] Homer A., et al. "Professional Active Server Pages", WROX publishing, 1997.

[9] Ntafos, S. C., "*A comparison of some structural testing strategies*", IEEE Transactions on Software Engineering, Vol 14, p.p. 868—874, 1988.

[10] Rapps, S., and Weyuker, E.J., "*Selecting software test data using data flow information*", IEEE Transactions on Software Engineering, Vol 11, p.p. 367—375, 1985.

[11] Sun Microsystem, *Java Servlet*, in http://java.sun.com/products/java-server/servlets/index.html.

[12] *Testing and Testing Management Tools*. Available in http://www.methods-tools.com/tools/testing.html.

[13] Vogel P.A., "*An Integrated General Purpose Automated Test Environment*", ACM ISSTA '93, p.p. 61—69, 1993.