School of Physical Sciences and Engineering
MSc in Advanced Software Engineering
2005

Automated Regression Testing of Web Applications

Project Report

by

Nadia Alshahwan

Supervised by

Professor Mark Harman

2nd September 2005

Acknowledgments

I would like to express my gratitude to the following people who provided me with help and support through out this project.

Professor Mark Harman for his unique way of supervision that increased the learning outcome of this project and his valuable advice, constant support and helpful insight.

Andrew Macinnes from the BBC for taking an interest into this project.

The British Council and BEA Systems for sponsoring me and giving me the opportunity to be here.

My friend Gihan Thanushka for his help with the format of the report.

My friends Gassan Almansour and Sara Alghabban for volunteering as external testers for the evaluation.

And last but not least, my family – my mother and father, Iman, Hatem, Moayad, Issam and Salah - for their ongoing love, care and support.

Abstract

Considering web applications expanding use in different critical fields such as businesses and medical and governmental systems, the need for quality assurance techniques that take into account the special characteristics of web applications is pressing. Web applications change more frequently than traditional software which makes regression testing a key factor in their success. However, with the high pressure nature of the development environment, well planned thorough testing is often sacrificed in favor of meeting deadlines. Applications released with bugs and defects could result in the loss of customer trust and potential profit.

The answer to this problem is automated testing. Although many tools exist for aspects such as load and security testing or link and HTML validators, automated functional testing is limited. Existing tools are capture/replay tools or test case frameworks which are both not fully automated. Employing user session data in testing has been suggested and proved to be effective. However, if the structure of the web application or the format of the requests has been altered, the previously recorded session data would be of no use and would fail if run as a test suite.

This project attempts to find a solution that would make automated regression testing of web applications possible. The technique relays on using session data for testing but finds a solution to the problem of the affect of web application modifications on their usefulness by automatically adjusting the logged data according to the changes. Locating the changes will be done by semi white box analysis and automated form filling and submission. The solution should be as generic and automated as possible. The project proved by practice that this technique is most useful with applications where the changes are mostly to form fields and structure. However, if file names are changed or pages are deleted the amount of reused old data decreases proportionally to the changes made. The results accomplished by this tool point to a new direction in automation of testing that could lead to faster and effortless quality assurance.

Table of Contents

List of Figures and Tables

1 Introduction

User behavior while visiting a web application is the best representation of its most common use. Therefore, logging user's actions and using them later for testing would result in a very effective and almost effortless way of testing and insuring web application quality. However, web applications change rapidly and if those changes affect the structure of requests, the recorded data becomes invalid. Finding a way to automatically adjust this data to those changes would solve the problem and result in a step forward for automation of web application testing.

The importance of web applications has been growing in recent years. Nowadays they are widely used for businesses, scientific and medical purposes such as diagnostic based expert systems and for government organizations. Many legacy systems have been linked to a web front end to take advantage of functionalities provided by the web. E-banking, e-billing, online stock exchange and other critical systems raise the demand for more reliable and bug free web systems.

The challenge here is greater for several reasons; Usage of web systems is unpredictable and can change drastically from one day to the next and hits can rise from several hundreds to several thousands if the website for example was caught by a search engine. Web applications change more frequently in the form of incremental changes over short periods of time. Also, they are usually composed of multiple intermingled languages. Finally, web applications have complex multi tiered architectures.

Web application testing and verification is a relatively new field. Existing testing tools mostly cover navigation and configuration testing, usability testing, security and performance testing, link checking tools and HTML validators. However, with the escalating amount of interactive web applications, that expect input from the user and produce output, the demand for functional testing tools has risen. Existing functional testing tools can be grouped into capture/ replay tools and frameworks and wizards to create and run test cases.

A new direction in web testing is to use previously recorded user session data for regression testing. Session data is the set of actions a user executes from entering a web application until leaving it. This data is in the form of a sequence of URLs with

their related field-value pairs if any. This data can be recorded easily by minor modifications to the application's code or by enabling the log option on the server.

The advantage of using session data in testing is that since it represents authentic user behavior it would be more realistic and more likely to expose bugs. Usually after a system goes online, all the hidden defects that were not caught during the testing period would be detected and reported either by users complaints or on the servers logs. Therefore, if this can be duplicated during testing it could result in a more effective test process. However, whenever the web application is modified in a way that affects its structure and input parameters large parts of this session data become useless.

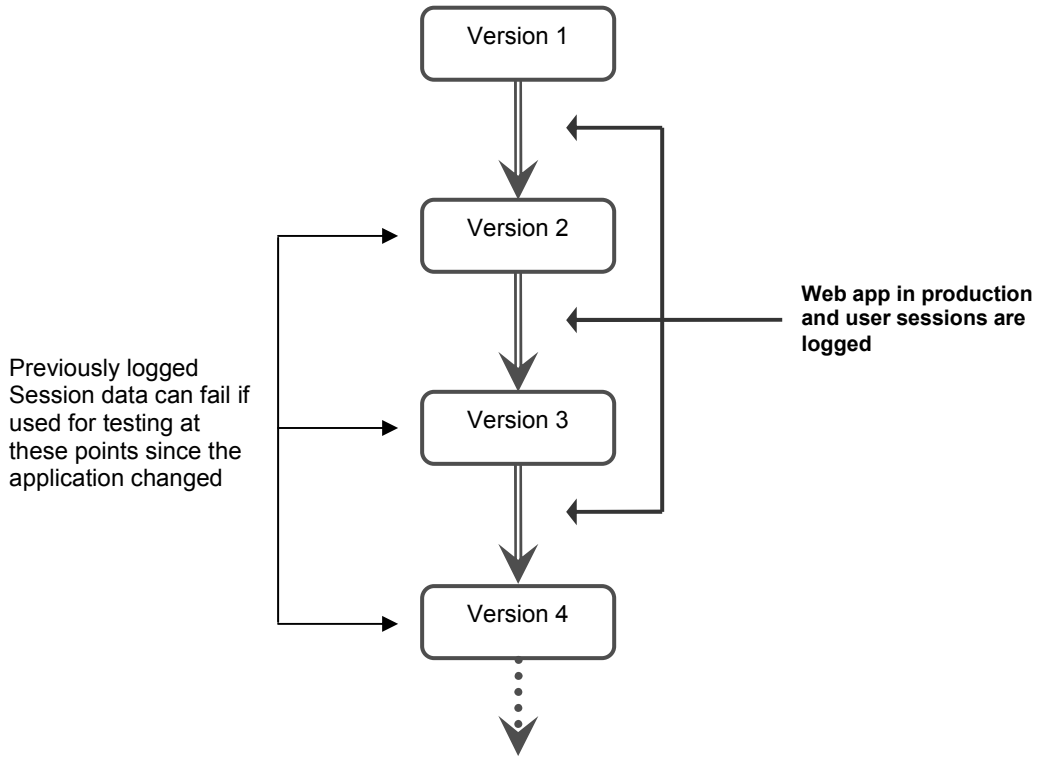

Figure 1: Web Application Lifecycle and its Effect on the Usefulness of Session Data

With every release of a new version of a certain web application, any data collected prior to the release date becomes invalid *(figure 1)*.

1.1 Objective

The objective is to find a time saving, cost effective way of performing regression testing of web applications with minimal user intervention. The solution should be as generic as possible and independent of any web architecture technology. It should

make use of user session data as it has been proved to be effective in error detection **[1,2]**.

1.2 Project Definition

The proposed project is a user session based fully automated regression testing tool for web applications. This tool would be used every time a new version of a web application is released to automatically produce valid session data that can be used to test the web application using existing user session data that has been recorded over the lifetime of the web application. Organizations that change their web applications frequently and already use some technology to record user interactions will benefit from it in achieving thorough regression testing with minimal effort.

The approach followed by this tool would be divided into two main steps. The first step would be assessing the new website to construct all possible test cases. This is needed to discover changes in the structure and the name-value pairs. The second step is to iterate through all the previously recorded sessions and rewrite them in the appropriate format by comparing them to the result of the analysis of the web application in step one.

1.3 Road Map

This document is divided into the following sections:

- Background gives a detailed review of related research papers and existing tools and techniques.
- Design and Specification describes this tools approach and design.
- Implementation gives a comprehensive explanation of the way the design was implemented
- Evaluation explains the evaluation strategy and the obtained results.
- Future work suggests how this tool can be expanded and enhanced in the future.
- Conclusion sums up what has been achieved and discovered by this project.
- There is also a glossary of term and appendices of code and design details and an index to facilitate use of this document.

2 Background

Testing is the most important analytical quality assurance method **[3]**. Although awareness of software testing importance is growing, it is usually the first development step to get sacrificed or reduced whenever a system is behind schedule. Exposing and fixing bugs becomes more expensive as the development lifecycle goes on. Moreover, a system with a large number of defects will lose the users trust and cause them to shift to another product resulting in a loss of potential economical profit and customer trust **[4]**.

The obvious solution to this problem is to introduce automation to the testing process to save time and resources and still make it possible to produce high quality software.

2.1 Web Quality Assurance and Testing

Web development in general and web testing is going through an evolution similar to the one traditional software has gone through in the past. Web application testing is still a young and fragmented market. The fact that web applications differ from traditional ones has to be studied and dealt with to adjust traditional testing techniques taking into account those differences.

A definition of quality aspects of a web application can lead to a better understanding of what to target in the testing process. The main aspects are:

- Structural Quality: The website should be well connected for easy navigation and all external and internal links should be working **[4]**.
- Content: HTML code should be valid and content matches what is expected **[4]**.
- Timeliness: a web application changes rapidly. The change has to be identified and highlighted and tested **[4]**.
- Accuracy and Consistency: content should be consistent over time and data should be accurate **[4]**.
- Response Time and Latency: Server response times to user's requests should be within the accepted limits for that particular application **[4]**.
- Performance: performance should be acceptable under different usage loads **[4]**.
- Security: with the expanding amount of applications such as e-commerce sites and e-banking, security has become a major issue.

- Underlying complex architecture: Testing with elements of the architecture such as web servers, application servers and database server in mind are necessary **[1]**.
- Usability: a site needs to be easy to use and accessible to different types of users.

Many tools exist that address some of these concerns automatically **[5, 6]**. Structural quality can be insured by using Link Checkers and web crawlers. Content can be checked with HTML Validators. Response time and performance can be tested by performance and navigation testing tools. Special tools exist for security and usability testing. WEBXact **[7]** for example examines broken links as well as compliance with accessibility guidelines. Architecture can be tested by configuration testing tools.

2.2 Functionality Testing Importance

Pages of a web application can be classified to static and dynamic pages **[8]**. For websites consisting only of static pages the use of the previously mentioned tools would be sufficient to insure quality. However, with the dominating amount of electronic businesses and other online applications most applications have large dynamic content. *Figure 2* shows the typical structure of a dynamic request. The application server has to do some processing and possibly query the data base in order to create the appropriate response. This calls for a focus on functional testing.

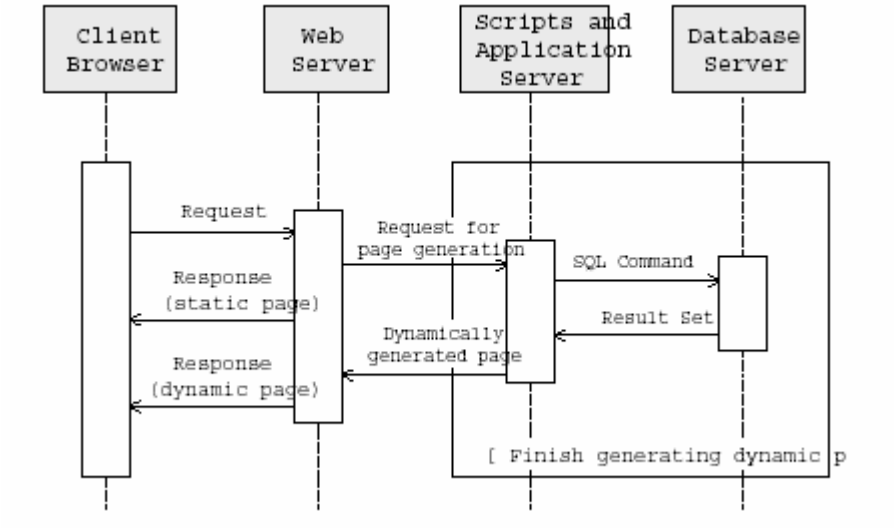

Figure 2: Typical Web Application Structure [2]

Unfortunately not a lot of tools exist for that domain and most of the existing tools are either capture replay tools or test script frameworks and executers. When it comes to automated testing, we are often faced with the "fragile Test" **[9]** problem that could

result in the failure or ineffectiveness of the tool. Behavior, Interface, Data and Context Sensitivity have to be taken into account. Behavior sensitivity is the change in a system's behavior in case it was modified. Naturally every recorded test case affected by the change will fail. Interface sensitivity is a concern when the test tool runs through the system's interface rather than directly communicating with the program itself. Data sensitivity is the state of the system before running the tests which has to be reset every time to its initial state to insure that the tests don't fail. Context sensitivity deals with changes in the environment such as servers, printers and other factors that might affect testing.

2.3 Capture/Replay Tools

Capture replay tools provide a way of testing websites with dynamic content. Using the tool, the test engineer goes through the website recording various testing scenarios as needed. The tests can later be replayed for regression testing. However, since the test scenarios are recorded from a user's manual actions, the resulting test suit would not be thorough and high coverage might not be achieved. Also, changes in the structure of the web application would make the recorded test suit fail to run and make it necessary to rerecord all or part of the test scenario.

[9] Takes a closer look at using record and replay for regression testing. Behavior sensitivity was avoided by freezing the system during the testing phase. Interface sensitivity was avoided by building the tool into the applications code instead of externally. Data sensitivity was avoided by getting a snap shot of the system before running the tests and then resetting it every time the test needs to be rerun.

A lot of parallels can be drawn between the concept of capture/replay and the use of session data in testing; studying what aspects affect the test and how they have been addressed can give an insight on how to deal with the same problems in this project.

2.4 Test Script Frameworks

Another group of available tools aids in the creation and run of test scripts. These tools provide a set of functions to help simplify this process. One example is HttpUnit, it emulates the relevant portions of browser behaviour, including form submission, JavaScript, basic http authentication, cookies and automatic page redirection. It allows java test code to examine returned results **[10]**. Apache JMeter is a 100%

pure Java desktop application designed to load test functional behaviour and measure performance **[11]**. Another tool is SimpleTest like HttpUnit and Jmeter it is a framework of classes that can be used to simulate a browser with methods such as get and post request or click link, set field value…etc they also have methods to validate the response by looking for certain text **[12]**.

Although using these tools might not be ideal when it comes to regression testing, they open up a whole new world of possibilities. The methods offered by these tools can be used to create a tool that can run automatically and without user intervention provided that the test cases and data can be also generated automatically using different kind of tools or techniques. An application to this could be writing a generic script using one of these tools that reads a file of session data and executes the requests it contains. The script will be independent from implementation and would not be affected by the change in code. Only session data will need to be updated and this would be covered by the proposed tool.

2.5 Structural / White Box Analysis

2.5.1 Definition and Criteria

Structural analysis is examining the code to understand the behavior of the application. This analysis can then be used to create test suites that would cover certain paths. White box testing criteria for web applications can be defined based on what is used in traditional software as follows **[13]**:

- Page testing: every page in the site is visited at least once in some test case.
- Hyperlink testing: every hyperlink from every page in the site is traversed at least once.
- Definition-use testing: all navigation paths from every definition of a variable to every use of it, forming a data dependence, is exercised.
- All-uses testing: at least one navigation path from every definition of a variable to every use of it, forming a data dependence, is exercised.
- All-paths testing: every path in the site is traversed in some test case at least once.

For the propose of this project we need to use the All-paths testing criteria as the objective is to validate paths in a certain session therefore we need to know every valid path. However, this criteria is impractical since there are infinite paths in a site

unless we add some constrains. A restriction of only considering independent paths can be used.

2.5.2 Related Work

Two aspects of generating graphs of web sites were discussed by **[14]**: first, generating graphs of dynamic pages with varying data values i.e. identifying two variants of the same dynamically created page with varying content based on user input. Since we are not running the test data and validating the result of our execution, there is no need to take this into account. All we are interested in is whether a certain page is called from another certain page when we attempt to verify the validity of a sequence of calls in one of the old existing sessions. Second, Developing appropriate modeling techniques for web sites with complex frame structure.

A research project at the University of Delaware made an ambitious attempt at studying the possibility of creating an "automatic test case generator that would take in source code as input and create feasible test cases". The idea relayed on white-box analyses of source code with the ability to recognize different scripting languages to create test cases **[15]**. However no details were given about the outcomes.

2.5.3 Ricca and Tonella's Approach

In Ricca and Tonella's *Analyses and Testing of Web Applications* **[13]**, automatic support for analyses and testing of web applications was proposed. Two tools where developed for this purpose. ReWeb which downloads a web application and creates a UML graph of the relationships between its different parts representing web pages as nodes and links as edges between the nodes. Special cases such as frames and forms are covered by this tool. *Figure 3* shows part of the constructed model of a web application.

The second tool is TestWeb which uses the graph produced by ReWeb to generate test suites. One of the limitations though is that the whole process is semi-automatic with user interventions required at several points. Most noticeably, the user has to provide the set of required fields and their values for every form encountered by each of the tools. Also, there is no released version of either tool on the net or source code that can be used and modified to integrate any of those tools functionalities in a different project.

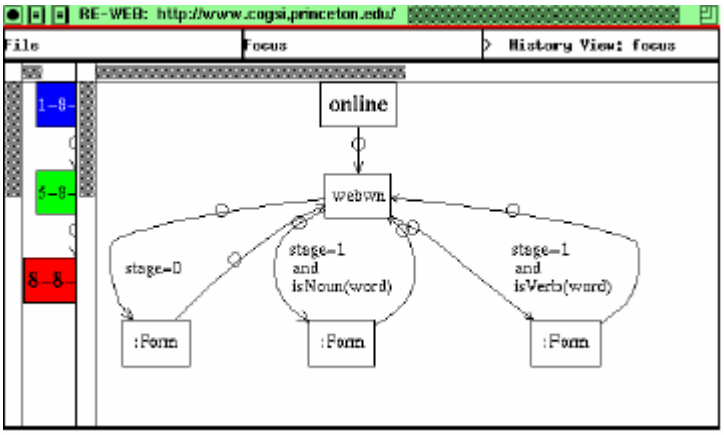

Figure 3: Sample ReWeb Output[13]

In their *Building a Tool for the Analyses and Testing of Web Applications: Problems and Solutions* **[16]** more details about those tools are given. ReWeb consists of a spider, analyzer and viewer. A structural white box technique is used to determine data dependence when constructing the test suite. The parser recognizes HTML and Javascript. The viewer displays the constructed graph representation of the web site. This graph not only shows the structure but also displays in different colors which parts where modified, added or deleted in different point in time.

2.5.4 Conclusion

This puts light on an interesting approach to the problem we have. A tool like ReWeb can be used to discover the structure of a web application. If TestWeb is used afterwards to create a test suit, this test suit can be compared to the old session data we have to discover where the adjustments need to be made. The problem is that this tool is not fully automated and not open source so that amendments can be made. Moreover, the tools were not put on the public domain. However this gave reason to examine open source web spiders and crawlers to try to find one that could be modified and customized for this project's needs.

2.6 Web Spiders

The aim of the analysis of the web application has to be clear and focused on through out development. It is tempting to attempt to turn it into an automated test generation tool. Although it has the potential to fulfill that with some expansions, the sole purpose for it is to provide a way of validating old session data and showing in what way it should be changed to be valid again.

One interesting tool is MOMspider **[17]**. A web robot created to act as a maintenance tool. It traverses a web application periodically to uncover any problems. Although it only looks for broken links and expiring web pages and there is no mention of it handling links with parameters (forms..etc) , the technique of spidering through links and collecting data is similar to what needs to be done in this project to explore a web application's structure.

[18] Describes a method for robust multilingual parsing of multi language programs and applies it to parsing ASPs. Parsing is the way to understanding and analyzing web systems as a necessary step for automating the testing process and possibly maintenance and modification.

[19] Attempts to find a solution for automated testing of web applications with dynamic content. Examining client side scripts and automated form submission.

WebSPHINX **[20,21]** is a customizable spider with a GUI interface. The project also provides a class library that can be used to implement spiders in java. The only problem is that form handling isn't covered by the provided classes. However, since it is an open source project changes and amendments can be done to cover any missing requirements. The spider would have to be built from scratch though and it could turn out to be time consuming and would shift attention from the main goal of this project.

Jspider **[22]** is another open source crawler that has the advantage of an event dispatcher, plugins and most importantly the option to store the result of the spidering into a database. However, documentation is lacking with the developers guide not being available online. On the other hand the user manual is thoroughly descriptive and examples of different configurations and sample pugins also exist.

Jspider supports cookies, user agent headers, redirects and all other HTTP standards as defined by the RFC 2616 **[23]**. It also is configured to be a “well behaving” web robot obeying the instructions of the webmaster in robot.txt. However, it has no facilities to handle forms neither automatically or manually. This feature has to be added before being able to use this tool.

2.7 Form Filling

Currently available crawlers only traverse hyperlinks ignoring forms and more importantly the large amount of content hidden behind them **[8].**
In Ricca and Tonella's ReWeb **[13, 16]** form field values were simply requested from the user. After an initial analysis the user has to go through forms one by one on the graph providing the set of fields together with their values.

Generally speaking, for every page with a form a number of distinct successive states exist. Even when all possible input values are known (checkboxes, radio buttons..etc), it is impossible for an automated tool to determine which of those values are sufficient to cover those states. On the other hand it is impractical to use all possible values. Even a simple form like the one on *figure 4* will result in a large number of possible inputs since there are 36 different ways to fill up the form.

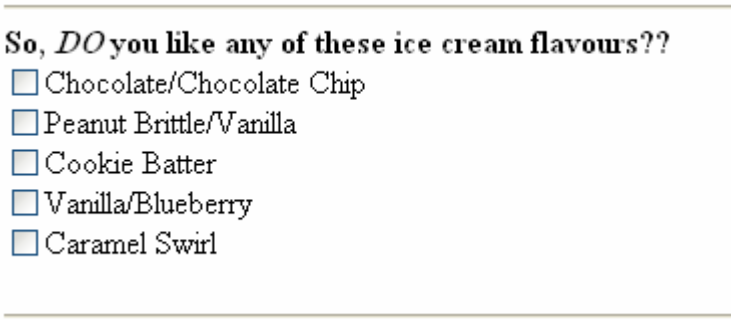

Figure 4: Simple Form Example

VeriWeb **[19]** goes a step further by introducing the smart-profiles concept. SmartProfiles "represent sets of attribute-value pairs that are used to automatically populate forms" **[19]**. These are pre-entered by the user once before running and used whenever a form is encountered. Their strength lies in the fact that they are independent of the site's structure.

The concept of using pre-saved values from a database to automatically fill forms could be the ideal solution.

2.8 User Session Data

2.8.1 Definition and Uses

User session data is the set of user actions performed on a certain web application from entering the site until leaving it. It can be logged by minor configuration changes to the server or less transparently by adding snippets of code to the application itself. Many websites already have some form of logging. The reason for this varies and could be one or a combination of the following:

- A certain user's actions when visiting a website can be used to customize it according to his/her needs and preferences (e.g. Amazon).
- Monitoring requests can give information about traffic on the website during different times of day.
- Detecting defects by examining errors recorded on the log.

2.8.2 Issues

One issue that needs to be resolved is resetting the state of the application to the original state it was in when the requests were actually made to insure that the results would be the same. This could be hard in practice since applications don't usually keep a backup of their databases before each release. However this can be ignored since repeating the sessions can be just as beneficial in testing without them yielding the same results.

2.8.3 Related Work

In **[1]** the use of session data in testing was proposed with the focus on fault detection. A comparison between different techniques was made to prove its effectiveness. An e-commerce site with realistic scripting, webpage and database query faults seeded into it was used by people who were made to behave like typical users by giving them a multi-step assignment and providing an incentive for them to take the task seriously.

The following five testing techniques were applied thereafter:

- An implementation of Ricca and Tonella's white box technique with the assumption of one input value for each field encountered.
- An implementation of Ricca and Tonella's technique with boundary values.
- User sessions repeated as test cases.
- Creating new test cases from mixing different user session.

- A hybrid technique that uses the test cases from the first technique but fills out the field values from session data.

Metric	WB-1		WB-2		US-1		US-2		HYB	
	abs	%	abs	%	abs	%	abs	%	abs	%
Test Suite Size	28	–	64	–	85	–	84	–	1089	–
Block Coverage	263	66	306	76	263	66	255	64	260	65
Function Coverage	65	97	66	99	65	97	64	96	65	97
Faults Detected	22	51	25	58	23	53	23	53	23	53

Figure 5: comparison of the effectiveness of different white box and session data techniques [1]

The table in *figure 5* shows the result of this experiment. The second technique (WB-2) provided the greatest fault detection power and coverage while session data techniques (US-1, US-2) performed better than the first technique (WB-1) but not as good as the second. The hybrid technique (HYB) with the largest number of cases didn't provide extra detection or coverage. However, it was suggested the replaying user sessions as they were recorded could've performed better and given the highest fault detection percentage if the number of sessions was larger. Since sessions collected for this study were simulated rather than authentic, it is fair to say that session data would perform better. Considering the low effort of gathering those test cases and the minimal user intervention needed, the results are impressive.

Further analyses of the results showed that the faults discovered by white box techniques and user session techniques were different. This leads to the conclusion that the two techniques are complementary.

In **[2]** the problem of web application state was further studied. Repeating the whole experiment with the state saved and reset before re-running the test cases didn't result in significant differences in results. However, there are special cases were this wouldn't be valid.

The only other study found that deals with session data was **[24]** which focuses on the analyses of user session data for the purpose of enhancing software development and maintenance tools. Concept analysis and common subsequence analysis were used to understand usage patterns of web applications. As a demonstration of the effectiveness of this type of analyses, it was applied for automatic test generation.

3 Specification and Design

3.1 General Approach and Scope

3.1.1 General Approach

Putting the goal of this tool in mind the following steps have to be achieved to reach the desired result:

- Analyze the modified web application in order to identify the change.
- Compare the result of the analyses to the recorded session data and adjust accordingly to restore its validity.

The tool should be generic and independent of the web applications architecture and technologies used. Also, it should simulate white box analysis by examining the applications structure but doing it through the web server's responses rather than the code itself.

3.1.2 Scope

The class of web applications considered is defined as following:

- Most scripting languages are included: ASP, PHP, JSP as well as HTML.
- Applications with dynamic content whether it is temporal (news pages), client dependent (Amazon) or input dependent (Forms) are all included.

3.2 Analyzing the Modified Web Application

3.2.1 Main Concerns

The main concern in this step is to be able to include dynamic pages in this analysis with minimal user intervention. This should be achieved by a mechanism to automatically fill and submit forms.

3.2.2 Approach

a web crawler that has the ability to handle forms and store the result of the analysis in some sort of reusable format will be used.

We are faced with two possibilities:

- Building a crawler using one of the available frameworks and helping classes such as webSPHINX.

- Using one of the available open source crawlers and adding to its functionalities as needed.

Customizing an available tool would insure that the basic functionalities are working properly and avoid wasting time and effort into something already done and established. There are a number of tools that can be used but Jspider is our choice of tool for the following reasons:

- It is highly flexible and expandable.
- Extensive information about the result of the analysis can be configured to be saved in a database.
- An event dispatcher is used and plug-ins are supported.

On the other hand, support and documentation are somewhat lacking. However, this can be overcome by manually assessing the code to understand how it works and by using the available user manual.

3.2.3 JSpider

JSpider is a highly flexible, configurable web robot engine implemented entirely in Java. An over view of its main components is given in *figure 6*.

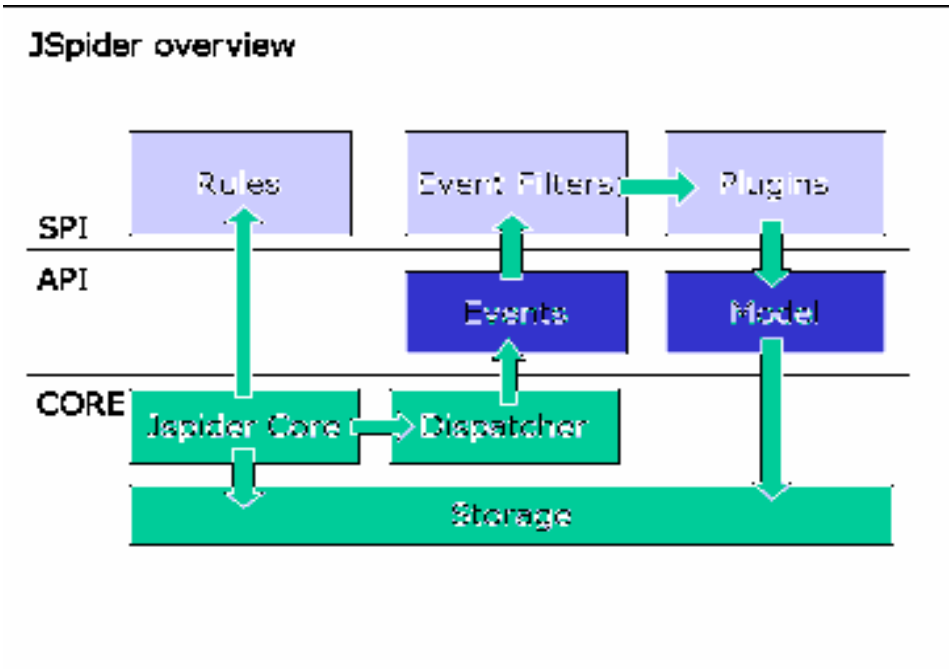

Figure 6: JSpider Structure [25]

The main functionalities are carried out in the core. The design is based on making every piece of work carried out by Jspider into a task. Fetching a web page, parsing a web page, deciding whether a page should be fetched or not or parsed or not are all made into tasks and added to the scheduler.

Therefore, we need to modify the parsing task executer in order to expand its functionalities to include forms.

3.2.4 Form Filling

To keep the process automated, a pre-filled database will be used to find the appropriate value. Also, fields with a limited number of options such as menus, checkboxes and radio buttons can be handled automatically.

3.3 Constructing Test Cases from the Analysis

3.3.1 Approach

The aim of the analysis of the web application and construction of test cases is to provide a way of validating the old session data. It is tempting to attempt to turn it into an automated test generation tool since it has the potential to fulfill that with some expansions. However, the goal is simple, given a sequence of requests, the tool needs to determine whether it's valid or not. Therefore, the chosen criteria should be *All independent paths* coverage.

3.3.2 Test Case Generation

Figure 7 shows a graph representing the structure of a web application. Nodes represent pages and edges represent links.

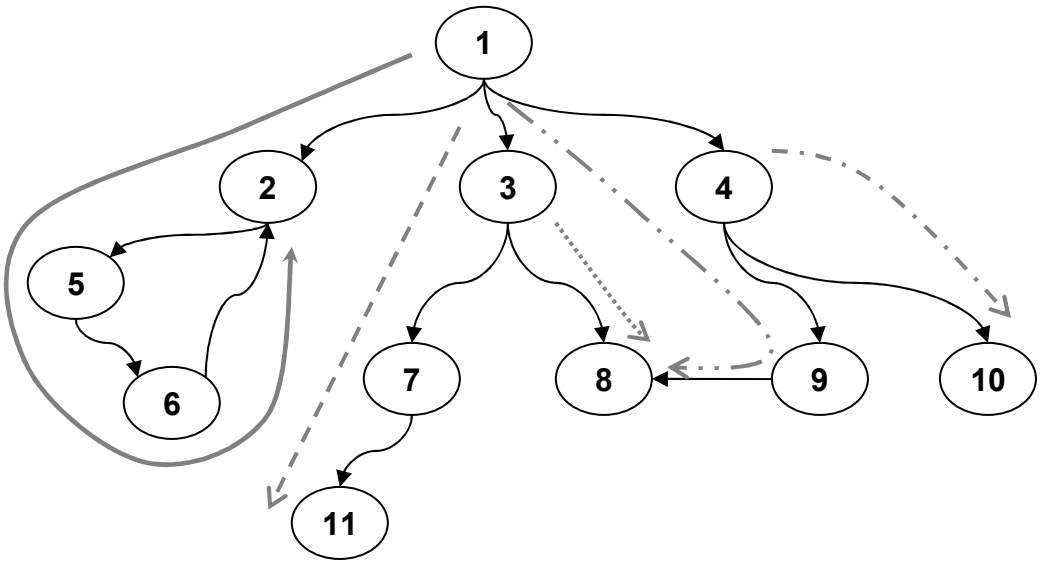

Figure 7: Graph Structure of a Web Application Showing Consructed Paths

Only independent paths are considered. In the first path (1, 2, 5, 6, 2) the loop is only traversed once. The third path (3, 8) starts from node 3 rather than node 1 since the sequence (1, 3) was already covered by the second path (1, 3, 7, 11). This is possible in web applications since they can be entered from any point by making the desired request. The same case is faced in path (4, 10) and path (1, 4, 9, 8).

3.4 Converting User Session Data

3.4.1 General Assumptions

The only assumption made is that if a URL for a page changes, it will be considered as a new page. This is because to discover such cases elaborate string matching techniques would be needed that could even lead to faulty results since two pages can have similar URLs while being completely different.

3.4.2 Matching Sessions to Cases

Two basic tasks have to be accomplished:

- Adjusting any invalid requests.
- Adjusting invalid sequences of requests

3.4.2.1 Adjusting invalid requests

Figure 8 shows two URL requests the first representing a request from an old session. The second is the same request but from the result of the analysis.

```
www.test.com/register.html?name=nadia&email=nadia@yahoo.com&pho
ne=077777777&submit=submit

www.test.com/register.html?name=test&email=test&address=test&su
bmit=submit
```

Figure 8: URL Before and After Web Application was Modified

The tool should examine both and remove any fields in 1 no longer present in 2, in this case phone. It should also add any new fields in 2 to 1 (address) resulting in the following request which is both valid and has realistic values:
www.test.com/register.html?name=nadia&email=nadia@yahoo.com&address=test&submit=submit

3.4.2.1 Adjusting Invalid Sequences

The second part deals with validating the sequence of requests. *Figure 9* shows the state chart of a system before and after it was modified. While a call from the command page to the edit page was valid in the previous version of the application, it is no longer valid in the new version. The call has to go through the list page first. If a session contained that sequence it should be adjusted and a request to the list page should be added in between.

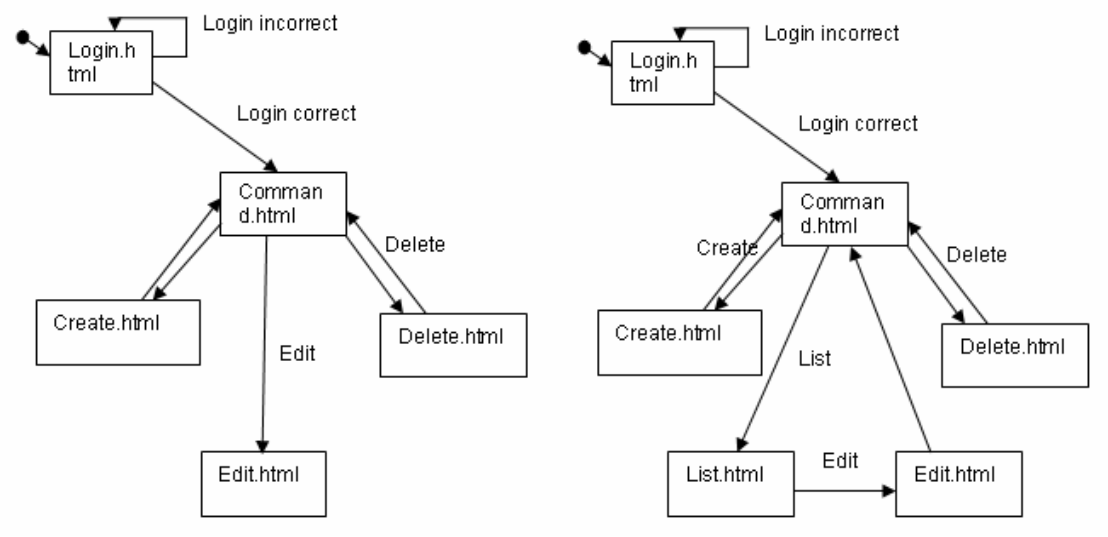

Figure 9: State Chart of a Web Application before and after Modifications

Three classes of changes to the sequence should be considered; a page could be deleted from the web application or a new page or sequence of pages could be added between two previously consecutive pages or a link to a page could be simply deleted from another page.

In the first case *(figure 10)* page 5 was dropped in the newer version. The solution should be to delete it from the sequence and link the previous request to the next providing the link is valid.

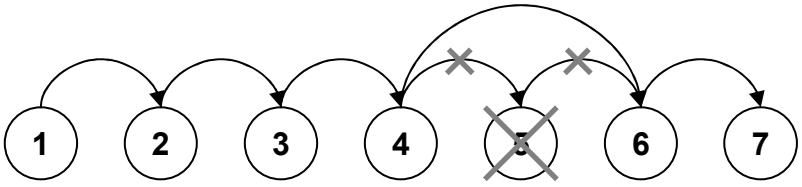

Figure 10: Sequence Adjusting – dealing with a deleted page

The second case is when, as in the example above *(figure 11)*, new pages were added between two requests in the previous version. The link should be broken and the gap between the two requests filled with the appropriate subsequence derived from the test cases.

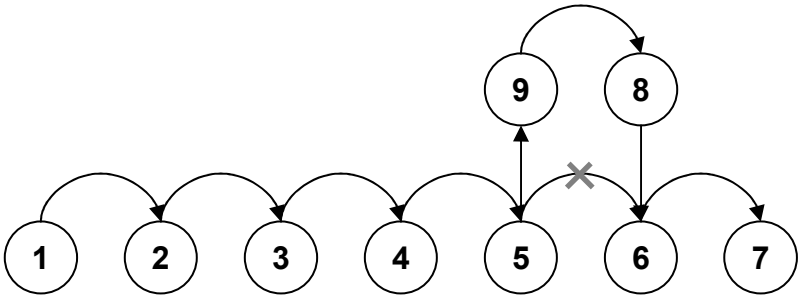

Figure 11: Sequence Adjusting – dealing with request added between two previously consecutive requests

The third and final case is when a page is no longer accessible from another page but is still valid and could be accessed in an alternative way. The difference between this case and the previous case is that no valid sequence could be found to link the two requests. For example *(figure 12)* the link to page 6 was deleted from page 5 but page 6 and the following sequence of requests is still valid. Here the original sequence will be split into two new sequences.

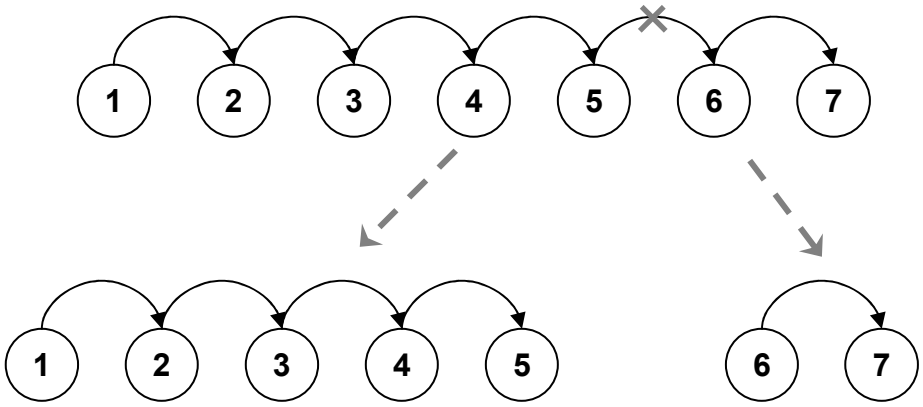

Figure 12: Sequence Adjusting – dealing with a deleted link where no sequence of requests can be found to connect the two

3.5 User Interface

The question of whether or not to have a user interface can be argued on both ways. Since all the components can be run from the command line, it might be easier to create a batch file. However, it is better to keep each function completely independent in order to avoid the need to restart the whole process if a later function failed. Also, this would give the user of the system flexibility to rerun any certain part as needed. However, we can't expect the user to learn and remember all the commands and enter them separately. In conclusion, the best approach would be to have a user interface with a button for each function and an option to run all functions together.

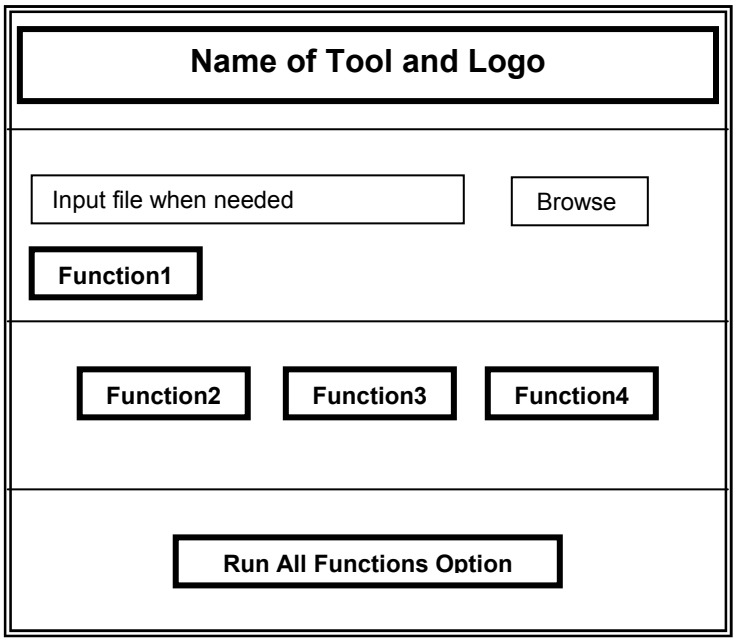

Figure 13: Screen Layout of User Interface

The outline of the interface's appearance is given in *figure 13*.
Function buttons are provided for each component such as analyzer and converter.
Input file dialogs are provided for every component that needs an input file to run.
Run all Functions option is provided for convenient running of the whole tool.

3.6 Data and Attributes

Data will be collected about sessions, test cases and field-value pairs. *Figure 14* shows how the system interacts with each type of collected data.

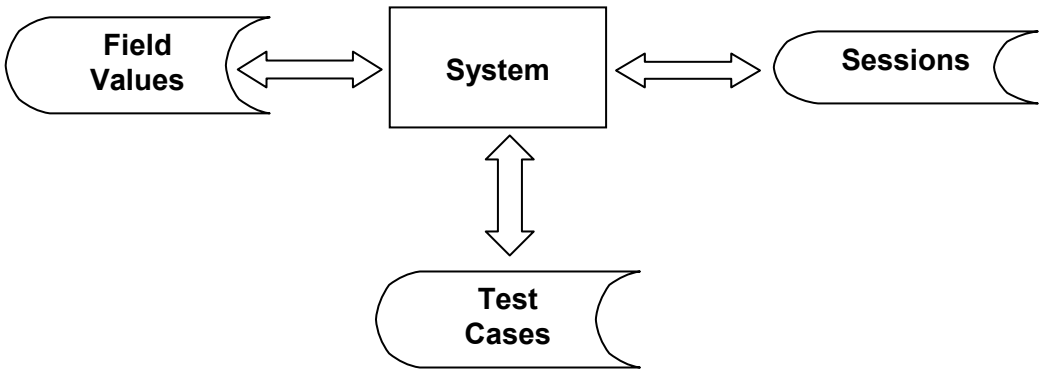

Figure 14: Interaction between the System and Data

Sessions are the data collected over the lifetime of the application that needs to be modified.
Test cases are requests constructed from the analysis and used as a reference for the conversion**.**
Field-value pairs are collected from the sessions and other sources and used to fill forms.

The structure of the attributes collected for the first two is similar but needs to be kept separately for semantic and implementation reasons. Attributes are: URLs, case/session number and step number. For field-value pairs we need to collect field names and values.

A detailed ER-Diagram that was created using the concepts in **[26, 27]** and attribute descriptions can be found in Appendix A.

3.7 General Architecture

3.7.1 System Architecture

The system consists of three tiers: user interface, application components and the database. *Figure 15* shows a general platform independent model of the system.

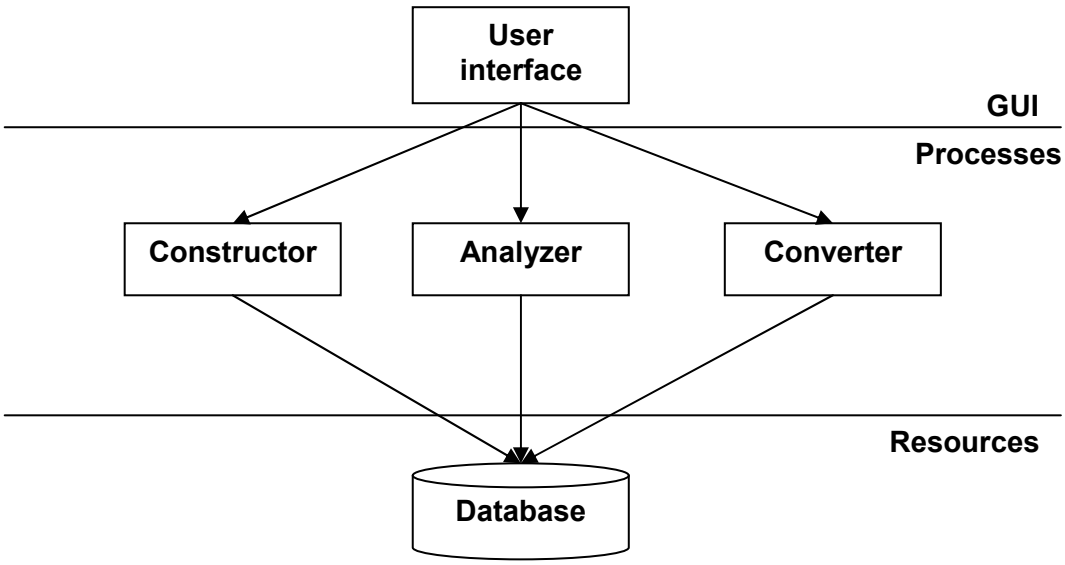

Figure 15: General Architecture of the System

3.7.2 Tools and Techniques

3.7.2.1 Programming Language and Database

Components of the tool will be written in Java and the database is MySql for the following reasons:

- JSpider which is the tool adapted and used in the system is in Java and uses MySql. Since code changes to it will naturally be in Java it is better to make the whole system uniform for smoother integration.
- Having worked on Java many times before I have a good foundation that would make implementation faster and avoid unnecessary waste of time in learning.
- Since Java is the dominating programming language and has a good future it would be beneficial to get a deeper understanding of it.

3.7.2.2 Requirements for Running the Tool

To try and test a tool it should be used on a web application on a local server. The server used for this is Apache Tomcat. Although not part of the implementation, it is

still necessary to have web applications hosted locally to simulate conditions for which the tool will be used.

- Apache Tomcat has a configurable logging facility that can be used to collect session data to work with.
- Having used it before, I have a good understanding of its abilities and functionalities.

3.7.3 Development Methodology

A mix between the waterfall model and what can be classified as a modular approach. The waterfall model is used in general except for the part concerning implementation.

3.7.3.1 The Waterfall Model

Advantages of the waterfall model *(figure 16)* are **[28]**:

- Progress can be tracked easily due to clear development stages.
- Deliverables can be easily identified.

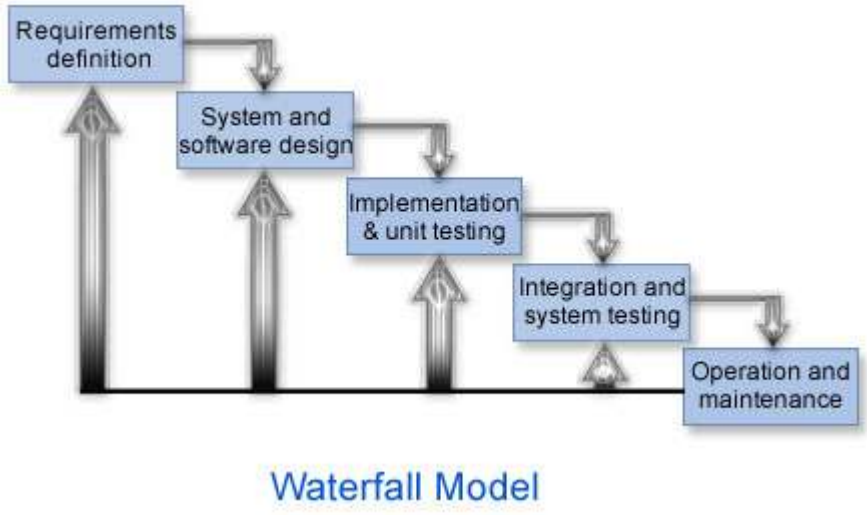

Figure 16: Waterfall Model [29]

Disadvantages are all related to project management and large scale projects. In conclusion, the waterfall model is suitable for small project with a small team of developers (in this case one).

3.7.3.2 Modular Development

The system is divided into functions. Each function is developed into a component and tested separately.

4 Implementation

4.1 Overview

The internal structure of the system is shown in *figure 17*. Components are run from the user interface. Each component has a certain function and functions are implemented to be completely independent and each component can be run separately. Component query the database as needed.

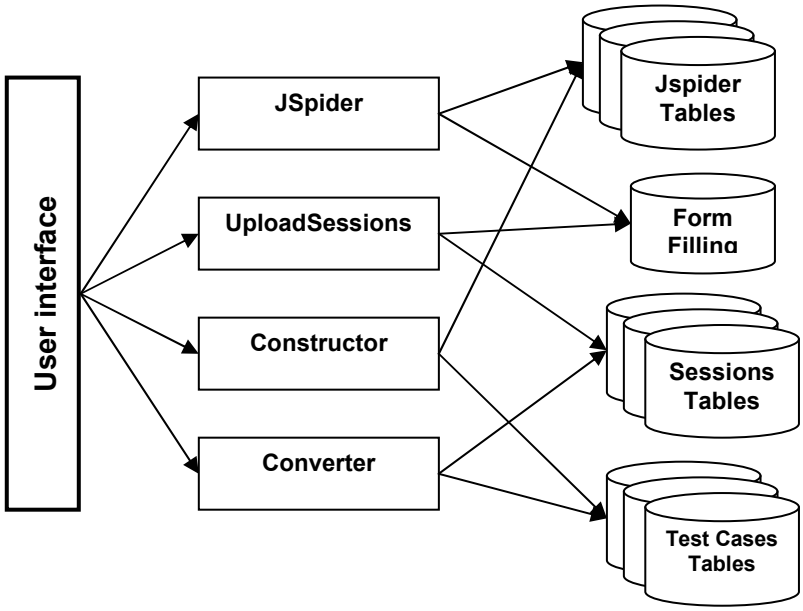

Figure 17: Internal Structure of the System

UploadSessions uploads the sessions into the sessions' database table and the field-value pairs to the form filling table.

JSpider is the modified version of the web crawler. It analyzes the new version of the web application and stores the result of the analysis in the database.

Constructor examines the result of the analysis in the data base and produces a set of test cases based on an all independent paths testing criteria. These test cases are saved then to the test cases table in the database.

Converter uses the sessions and test cases tables to rewrite the sessions and save them back to the sessions' table.

4.2 JSpider

4.2.1 Objective

As was mentioned earlier, we need to modify JSpider to introduce a form exploring capability to it resulting in a new version.

4.2.2 Analysis

Due to lack of a class diagram or developer support for this tool the first step to identify where to make the changes is to analyze the code and understand the internal workings of the tool.

The way parsing of a page is done in JSpider is as follows:

- When a resource is discovered it is scheduled for parsing using *InterpreteHtmlTask*.
- When *InterpreteHtmlTasks* is executed, the resource is examined line by line and every line will then be passed to the *FindUrls* class.
- *FindUrls* looks for patterns associated with links such as *"herf="*, if a pattern is found it is added scheduled again for fetching and parsing.

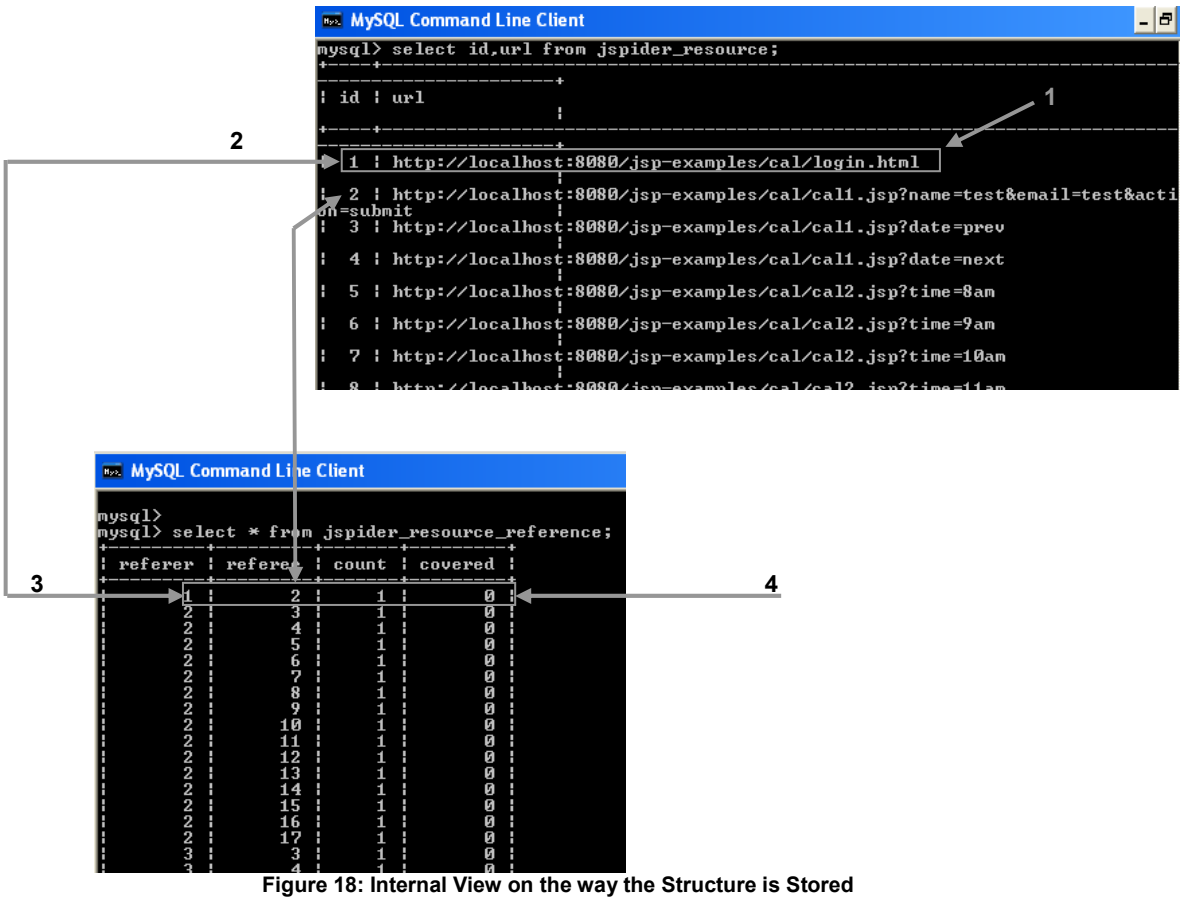

Figure 18: Internal View on the way the Structure is Stored

Figure 18 shows the way the website structure is stored into the data base. A row in *jspider_resource (arrow 1)* stores a URL together with other related info not shown here and gives it a unique ID. *Arrow 2* and *3* show the relationship between *Jspider_resource_reference* and *jspider_resource*. A row in *jspider_resource_ reference (arrow 4)* represents a reference between two pages i.e. page with ID 1 (login.html) has a reference or a link to page with ID 2 (cal1.jsp).

4.2.3 Modifications

4.2.3.1 General Modifications to the Process

The process needs to be modified in the following way:

- Before passing a line to *FindUrls*, a check is performed to determine whether it contains a form by searching for the keyword "*action=*".
- If the keyword was found, the whole form block will be passed instead of just the line. The end of the form will be determined by the keyword "*</form>*"
- In *FindUrls*, each input field is extracted and passed to the *extractField* method.
- *extractField* will extract the field name and find the appropriate value, either by querying the database or in case of field types with limited choices randomly selecting a value.
- Each field-value is then appended to the URL.
- After all fields are appended the URL will be treated like any other URL.

4.2.3.2 Form Filling

Table 1 shows how different input types are handled by the program. Radio buttons, checkboxes and select inputs are all selected randomly. *Textarea* is always set to 'test' to save space and shorten the URL since no validations are usually performed on it. The default type is text if field type isn't specified.

Field Type	Handling Mechanisim
text & number	Get value from database or default or 'test'/ any number
Radio	Select random
Checkbox	Set randomly to checked or unchecked
Select	Select random
Password	Get from database
Hidden	Default value
Textarea	'test'
Reset	Ignore
Submit	If more than one submit button (e.g. add, remove) select random

Table 1: handling different input types in forms

4.2.3.3 Main Methods

JSpider methods affected by enhancements:

FindURLs this method parses a line of code and looks for URLs. It was changed to parse a form element and look for input fields.

Main added methods:

extractField This method is called from *FindURLs*. One input field is passed to it delimited by '*<input*' and '*input>*'. It detirmines the type of the input field and returns it's name and value based on the guidelines in table 1.

countOccour This method is called from *extractField.* It takes an input field of type *select* as an argument and returns the number of options. This is used to select a value randomly.

getValue This method is called from *extractField.* It takes the name of a field as an argument and returns the field's value stored in the table *jspider_form_filling*.

4.3 Constructor

4.3.1 The Process

The test case constructor will examine the *jspider_resource_reference* table and construct all independent paths. Test cases are then stored in the *jspider_test_cases* table.

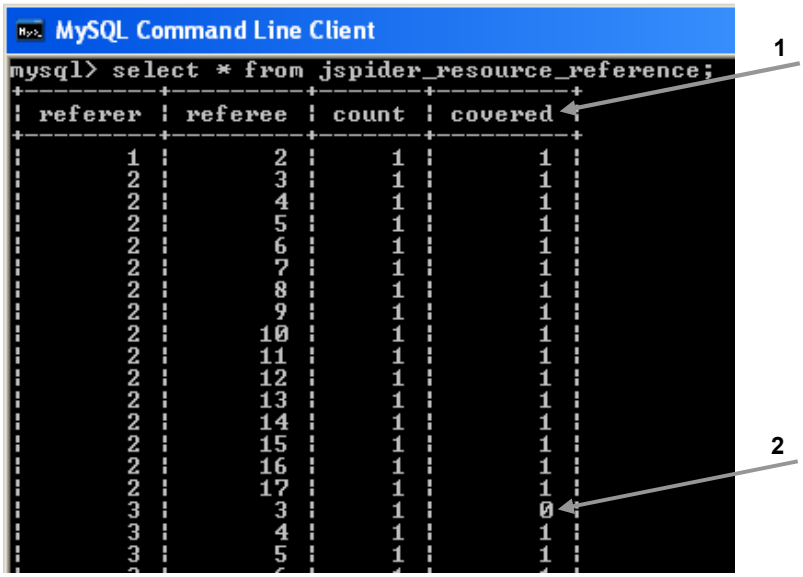

Figure 19: The Way Reference Table is Used to Create Test Cases

A flag field *(arrow 1)* to indicate when a referrer-referee row was covered had to be added to *jspider_resource_reference* to insure the paths generated are independent. *Figure 19* shows the tables after the component was run. Resources referencing themselves *(arrow 2)* are completely ignored to avoid infinite loops.

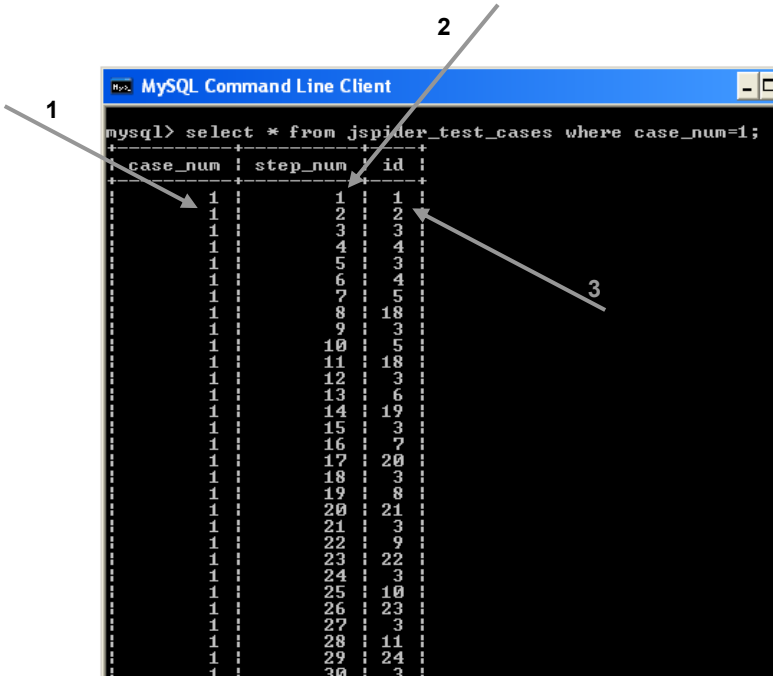

Figure 20: The way Constructed Test Cases are Stored

Figure 20 shows part of test case 1 that demonstrates the way it is stored. The order of steps is maintained in *step_num (arrow 2)* and the URLs *(arrow 3)* executed for each step are referenced here by their unique ID which is linked back to *jspider_resource* as mentioned in section *4.2.2.* All steps are associated to a certain test case through the *case_num (arrow1)*.

4.3.2 Main Methods and Classes

TestCase This class keeps track of anything related to the currently processed test case.

incrementCaseNum This method increments case number whenever a new test case is being handled.

incrementStepNum This method increments step number when ever a new step needs to be added.

findFirstNotCovered This method is part of the *ResourceReferenceDB* class. It is called by the construct program. It finds the first row in that hasn't been covered yet in jspider_resource_reference (i.e. flag covered is set to 0 or false).

4.4 UploadSessions

4.4.1 The Process

4.4.1.1 Pre-Processing

To make the upload of sessions generic and adaptable to any format of log file, an assumption was made that users of the tool will convert their log files to a standard format. This can be easily done by a script or small program that extracts requests from a session file and rewrites it using the format given by this tool. An XML Schema was developed and validated using an online validater **[30].** A copy of the schema and information about its validation can be found in Appendix B.

4.4.1.2 Uploading Sessions

UploadSessions parses the session files given as arguments when running the component and does the following:

- Extracts URLs and saves them in the table *converter_session_urls*.
- Extracts filed-value pairs from each URL and saves them to the table *jspider_form_filling*.
- Stores information about the sequence of requests and to which session they belong in the table converter_sessions in the same structure used for storing test cases.

4.4.2 Parsing Session Files

SAX and DOM are used to parse the session file since it is in XML. The program parses the XML file and returns a DOM document. The DOM document is queried later using xpath **[31]** to find nodes representing requests and their related parameters. Code examples in **[32]** where used to write the code.

4.4.3 Main Methods and Classes

Session This class keeps track of anything related to the currently processed session.

incrementSessionNum This method increments session number whenever a new session is being handled.

SessionDB This class is used to communicate with the database tables *converter_sessions* and *converter_session_urls*.

4.5 Converter

The converter has to be the last step run on the system. Two steps have to be performed to achieve its function:

- Rewriting URLs in a valid format using *jspider_resource* and *converter_session_urls*.
- Adjusting invalid sequences using tables *jspider_test_cases* and *converter_sessions*.

4.5.1 Pre Processing

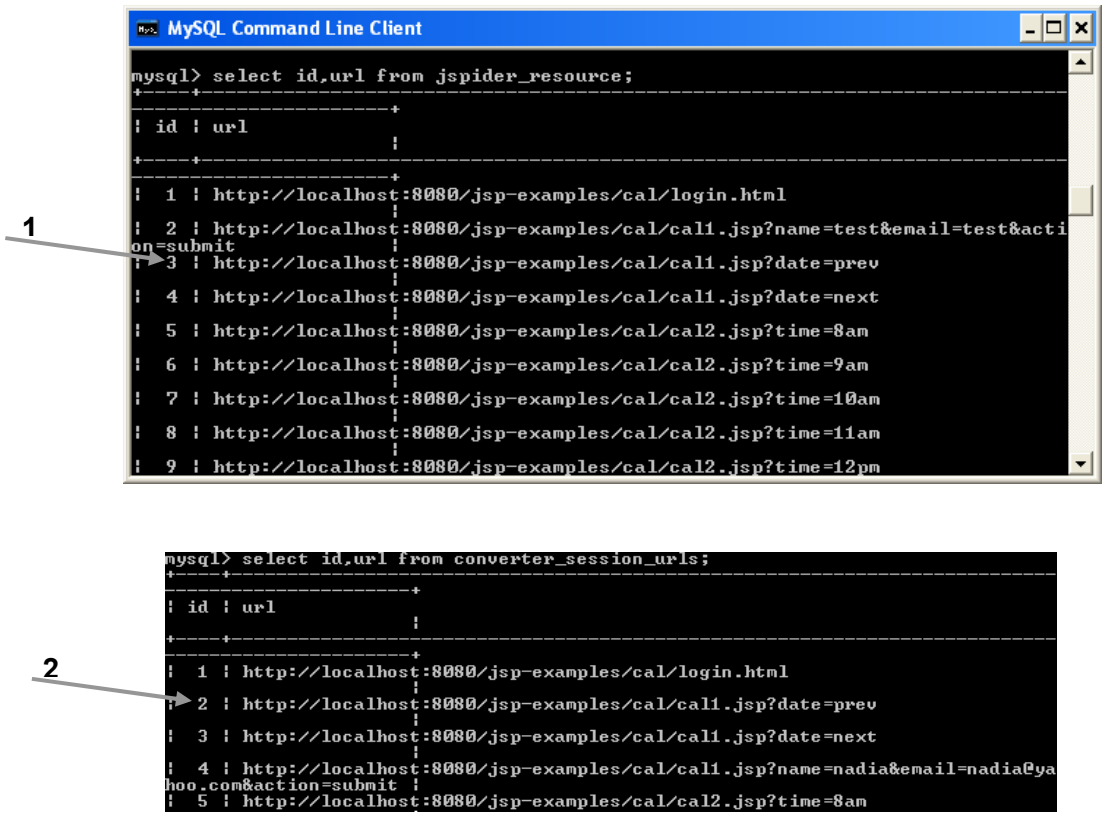

Figure 21: ID Mapping between Test Case and Session URLs

The number of test case URLs will be considerably less than the number of session URLs since test cases are all possible request formats and sessions are all requests collected over a period of time. In other words, a number of session URLs can be associated to one test case URL. It would save processing time to retrieve all test case URLs and have them ready in a table instead of re-querying the database several times for the same URL.

Another issue to consider is that URL IDs for sessions and test cases are not synchronized. This is done in this way to keep different components independent and the sequence of running the Constructor and Converter irrelevant. *Figure 21* illustrates an example: while the URL with ID 3 *(arrow 1)* is the *cal1.jsp?date=prev* page in test cases, in sessions it is *cal1.jsp?date=next* while *cal1.jsp?date=prev* has ID 2 *(arrow 2)* .

4.5.2 Rewriting URLs

The easiest and safest way is to rewrite the values of fields in the matching test case URL to keep manipulation of strings to a minimum and avoid errors. Urls are matched based on the part without the parameters. *Figure 22* demonstrates how this is implemented.

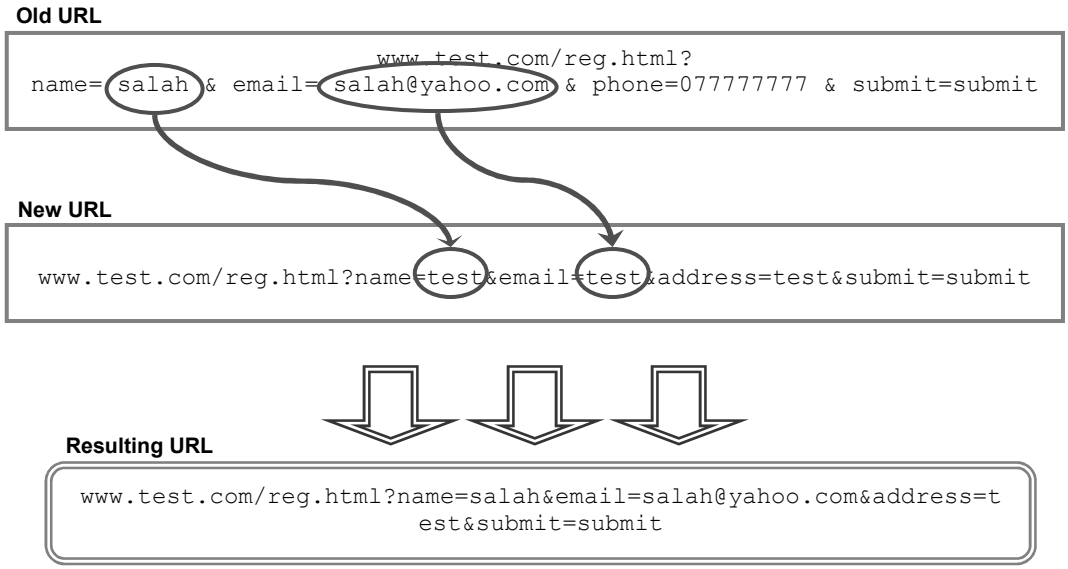

Figure 22: Implementation of URL Rewriting

4.5.3 Adjusting Sequences

Validating and adjusting sequences of session requests is done in the following steps:

- The first request in the session is validated (i.e. checked if it still exists) and if valid added to the final valid sequence.
- Every pair of consecutive session requests will be validated against the pairs in the *jspider_reference_resource* table.
- If the pair is valid the request's IDs will be added to the final valid sequence.
- If it's not valid the second request in the pair is dropped if it is not valid anymore (i.e. doesn't exist) and the process is repeated.
- If the second request is valid, a gap filler has to be found to connect the two requests.

4.5.4 Main Classes and Methods

The converter consists of the following classes:

MatchingUtil This class handles any functions related to preparation for the converter processes.

- ***getCorrependingID*** This method is used in the mapping process. It takes a session URL as an argument and returns the ID of the matching test case URL.
- ***stripUrl*** This method takes a URL as an argument and returns the URL stripped of it's parameters (field-value pairs).

URLConstructor This class handles rewriting of URLs in the correct format.

- ***rewriteUrl*** this method takes two URLs as arguments one session URL and one test case URL and returns a new URL that has all the fields in the test case URL with the values from the session URL.

URLExtractor This class handles any functions related to referencing test case URLs.

- ***fillURLs*** This method returns a vector of all test case URLs. This is done to save processing time since the list of test case URLs have to be traversed every time a match for a session URL is needed.
- ***getURL*** This method takes an ID as an argument and returns the corresponding test case URL from the vector produced from *fillURLs*.

URLAdjuster This class uses *URLConstructor* and *URLExtractor* and *MatchingUtil* to go through all session URLs and rewrite them in a valid format that corresponds with the changes done to the web application.

TestCaseSeqConstructor This class handles test cases in the process of adjusting sequences.

- ***execute*** This method is run from the class constructer. It creates a list of a string representation of all available test case sequences in the database.
- ***findGapFiller*** This method takes two request IDs and returns a valid sequence that would link the two requests.

SequenceAdjuster This class handles the validation and adjustment of session requests sequences.

- ***validateUrlId*** This method takes a session Id and checks whether or not it is still valid by comparing its value to the last valid ID. It returns 1 if the URL is valid and 0 if it's not.

It uses the following method from the *ResourceReferenceDB* class that handles interactions with the jspider_resource_reference table:

- ***validateSequence*** This method takes two session IDs as input and checks if the pair is valid by looking it up in jspider_resource_reference.It returns 1 if the sequence exists and 0 if it doesn't.

SessionPrinter This class prints the sequences produced by *SequenceAdjuster* into a file using URLs in place of the IDs.

4.6 Database

4.6.1 Database Tables

Figure 23 shows all the tables in the database. Tables referenced by *arrow 1* are session tables. *Arrow 2* points at the pre-existing JSpider tables and *arrow 3* and at the Form Filling table *arrow 4* at test case tables.

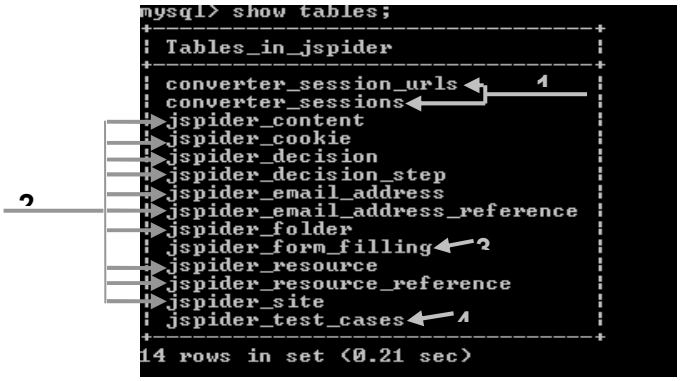

Figure 23: Database Tables

Appendix A has descriptions of new and modified tables' fields.

4.6.2 Classes and Methods

DBI This class handles establishing and safely closing the connection to the database.

FormFillingDB This class handles any interactions with the table *jspider_form_filling*.

ResourceDB This class handles interactions with *jspider_resource*. Methods include *getURL* by ID and *getAllURLs*.

ResourceReferenceDB This class handles interactions with *jspider_resource_reference*. Methods include *findFirstNotCovered*, *getByReferer* and *getNumOfReferenced* for a specified page. It also initializes the table when running the constructor by setting the flag covered to zero for all rows.

ResourceReferenceRow This class defines the structure of a row in *jspider_resource_reference*. It is used when the whole row needs to be passed back as a return value from a method.

SessionDB This class handles any interactions with *converter_sessions* and *converter_session_urls*. Methods include *create* which creates a row in *converter_sessions*, *createSessionUrl, updateSession* and *getNextSession* which takes a session number as an argument and returns the next session number. It also initializes the two tables if a flag that is passed to the constructor is set to 1.

TestCaseDB This class handles any interactions with jspider_test_cases. Methods include create and initialize.

4.7 User Interface

A simple user interface was created using Java swing with the help of tutorials in **[33]**. *Figure 24* shows the user interface for running the tool. *UploadSessions* needs the session file as an input. If no file was entered into the input box *(arrow 1)* an error message would appear for the user *(figure 25)*. Run all will run the components in the following order: uploadSessions, JSpider, Constructor and finally the converter.

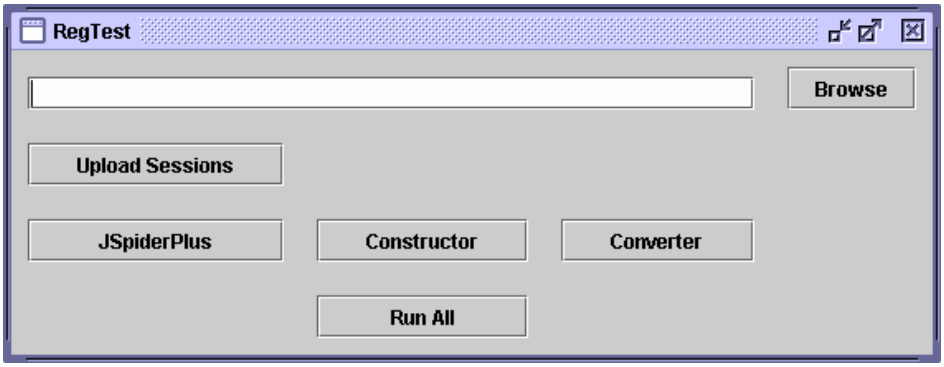

Figure 24: User Interface

Figure 25: User Interface – Error Message

5 Testing

5.1 Test Strategy

The tool has to be tested thoroughly to insure that all its functionalities are working properly.

5.1.1 JSpider Testing

5.1.1.1 Regression and Technical Testing

JSpider can be configured to run a technical and functional test suite that makes sure that the modifications did not affect any part of the system.

For functional regression testing the spider will be run on a well-known resource http://j-spider.sourceforge.net so that results can be automatically compared to what is expected. JUnit test results can be then generated and verified.

5.1.1.2 Added Functionality Testing

To test the added functionality small sites with 2-3 pages were spidered. These sites were chosen for having all the different input types currently handled by the spider. Both scenario and negative testing approaches were used.

5.1.2 Testing the System as a Whole

The system was tested as a whole on a simple calendar application provided with Apache Tomcat server as an example. The resulting file was manually examined and compared to the expected results.

5.2 Results and Analysis

Testing JSpider's ability to handle forms was successful. A copy of the resulting set of URLs can be found in Appendix C.

Running the tool on an application was successful. Some cases that had characteristics not handled by this project produced errors. Pages that have more than one set of input parameters or cases where HTML forms code was written using file writers within the code could not be successfully handled. This points to the direction of future enhancements that can be done.

6 Evaluation

6.1 Aim

The evaluation of the tool should try to measure its effectiveness in terms of the percentage of user session data that was reused in comparison to the amount of change that was done to the web application. Naturally, if most of the old session data was discarded the usefulness of the tool is reduced. The effectiveness of the produced test data in defect detection is irrelevant since an assumption was made that it is indeed effective based on previous studies **[1,2]**.

Another variable that should be considered is time saving. If the tool takes a considerably long time to run, its value is also decreased. However, the alternative is to create test cases manually which would take longer and require more resources and effort and wouldn't provide the same level of coverage and error detection as mentioned previously in the background section.

6.2 Approach

Andrew MacInnes from the BBC new media team has kindly attended a number of presentations during the curse of this project. A demonstration of the tool was conducted on the 26th of August to get real feedback and suggestions on how to improve this tool. His comments showed an interest in what the tool can achieve and he kindly offered an invitation to visit the BBC and try to adapt the tool to the development environment their and test its effectiveness.

To conduct the evaluation the following will be done:

- A web application will be logged and user session data will be collected.
- Modifications will be done to the web application.
- The tool will be run to adjust the recorded session data according to the changes.
- Data will be collected from the results to determine the amount of requests and sequences reused.
- The original web application will be changed several times with the amount of change increasing gradually. Running the tool and analyzing the results will be repeated.
- Results will be analyzed to determine the range where the use of the tool is most effective.

6.3 Variables and Metrics

The different versions of the web application that will be evaluated constitute the independent variables. The dependent variables are usefulness of tool and time.

Usefulness of tool will be measured by the ratio of session requests used to changes to the code. Changes to the code will be estimated based on the number of changes made; deletions, additions and modifications will be all counted. Changed file names will be considered as new files.

Time will be measured by the relationship between amount session data to run time and the relationship between web application size and run time.

6.4 Recording Session Data

6.4.1 Set Up of the Web Application

An online book shop *(figure 26)* was used as the web application employed for the evaluation. It was provided as an open source application by **[34]**. *Table 2* provides information about the size of the application.

Number of files	30
Lines of code	8413
Number of database tables	7

Table 2: Book Shop Application Information

Only user functionalities of the web application were considered. The book shop operates similar to any commercial web site. It provides the customer with the ability to register, login, search, browse, purchase books and use a shopping cart.

Figure 26: Online Book Shop

6.4.2 Logging of Requests

The site was set up on an Apache Tomcat server and users were asked to explore and use it while having a logging tool enabled. Emphasis wasn't put on the type of use and how realistic it is since the aim is not to measure session data effectiveness. Sessions with an average of 35 requests were collected.

A tool that records user requests was used **[35]**. A description of this tool and more information can be found in Appendix D. A program was written to convert the recorded data into the format specified for this tool. The program name is *FormatSessionFile.java* and can be found in Appendix B. The resulting session log can be found in Appendix E.

6.5 Modification of the Web Application

Two volunteers were asked to modify the web application several times with an increasing percentage of change each time. This was done to insure that changes aren't biased to make the tool more affective.

The Types of changes made were as follows:

- Structure: The way pages were linked together was changed.
- Forms: fields were added or deleted and names of fields were changed.
- Pages were deleted and others were added.
- File names were changed.

This has been done six times by incrementing changes every time. *Table 3* displays how the changes will be done.

Version	Change (%)	Modified Pages	Added pages	Deleted Pages	File name changes
V1	10%	3	1	0	0
V2	20%	6	2	1	1
V3	30%	9	3	2	2
V4	40%	12	4	2	3
V5	50%	14	5	3	4
V6	60%	16	6	3	4

Table 3: modification Strategy for web application

6.6 Execution and Results

SessionUploader and *converter* will be evaluated based on session data size. *JSpider* and *constructer* will be evaluated based on application size.

Num of Sessions	Uploading Time (secs)	Converting Time (secs)
1	4.506	7.406
3	10.325	12.322
10	17.635	20.005
20	33.918	37.103
30	47.028	42.338
40	61.369	65.901

Table 4: Execution Times of Uploader and Converter based on Number of Sessions

Table 4 shows execution times for *sessionUploader* and converter with varying number of sessions. We can see that the time in seconds goes up when the number of sessions is increased. However, the rise decreases in proportion with the rising number of sessions.

Application Size	JSpider Time (secs)	Constructor Time (secs)
Small	3,314	1,001
Medium Small	31,194	26,368
Medium	35,543	30,201
Large	182,462	76,339

Table 5: Execution Times of JSpider and Constructor based on Application Size

Table 5 shows execution times for *JSpider* and the constructer on different application sizes. Medium small and medium sized web applications produced similar times. The large application took considerably long to be spidered but didn't take as long to construct paths.

Application Version	Used Session Requests	Used Session Requests(%)
V1	50	100%
V2	46	92%
V3	41	82%
V4	36	72%
V5	29	58%
V6	29	58%

Table 6: Number of Session Requests Reused in the New Log File

Table 6 shows the number of session requests reused to produce the new log file. After running *sessionUploader* the number of unique requests was 50. The results are somewhat unclear although changes in the code went up to 60% for the sixth version, the percentage of is reused data is still high. For the final two versions the percentage didn't change although the two versions differ by 10%. However, they both have the same number of deleted pages and file name changes. Also, the number of reused sessions fell by 5 from version 3 and version 4 although the changes in deleted pages and file name changes only increased by one.

6.7 Analysis and Conclusion

From *Figure 27* we can see that the running time of both the uploader and the converter in proportionately related to the number of sessions processed. As can be seen the slop of both lines representing the uploader and converter increases after the point of ten sessions. This is probably because of start up and initialization which makes smaller session similar in the time they consume.

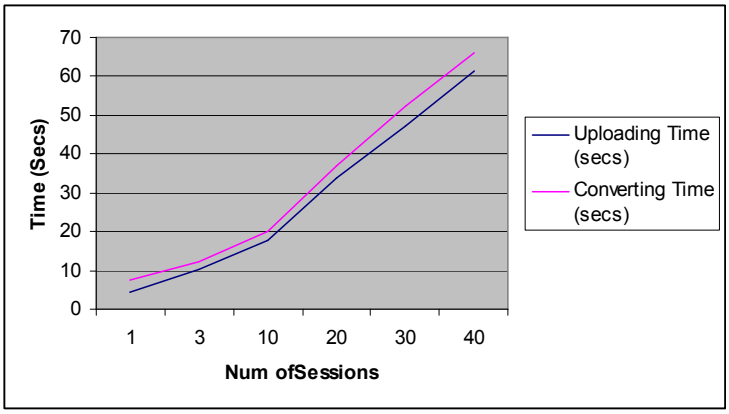

Figure 27: Relationship between Execution Time and Number of Sessions for the Uploader and Converter

Figure 28 shows the relationship between application size and Jspider and constructor execution times. The slop increases dramatically because the difference in size between a medium and a large application is big. However, it is worth noticing that the difference in execution time between medium small and medium applications is hardly noticeable. Deeper analysis of all applications used showed that the complexity in which pages of an application are linked and the number of dynamic pages and forms has greater affect on the time Jspider and the constructer take to execute.

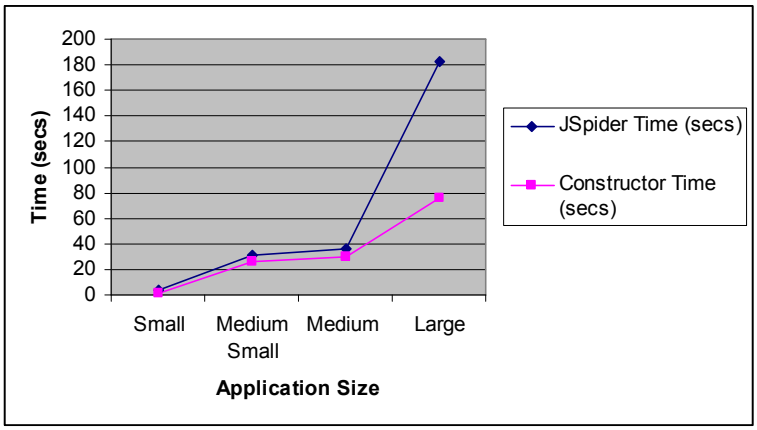

Figure 28: Relationship between Execution Times and Application Size for JSpider and Constructor

Figure 29 shows the number of session requests used for each of the modified versions of the application. Further analysis showed that changes in the code and form fields does not affect the number of session requests reused. This can be seen

by comparing the last two versions where changes to the whole application were increased by 10% but the percentage of requests reused stayed the same. However, the number in deleted pages and pages with changed file names was the same in both versions.

If a page was deleted or its name changed, all of it's occurrences with different parameter in the session log will be discarded. The percentage of discarded session requests can't be predicted because it depends on the number of times a certain request appears in the log file.

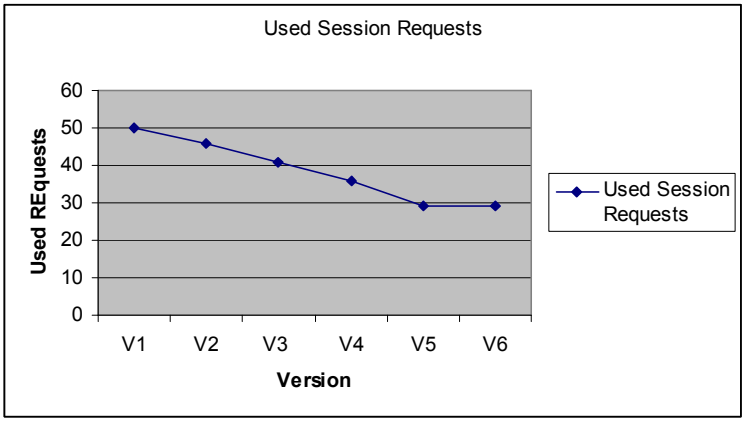

Figure 29: Reused Session Requests

The graph was reproduced *(figure 30)* only taking into account the changes that affect the number of session requests reused. The two lines are symmetric; when changes were 0% reused sessions were 100%. However this is not always valid since the number of occurrences of an eliminated page also affects the outcome.

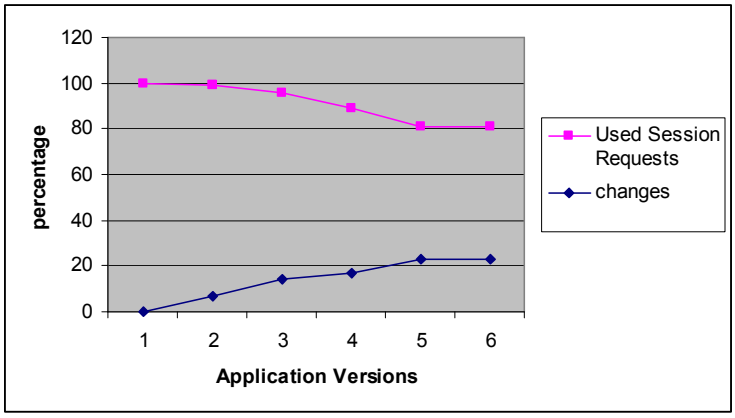

Figure 30: Relationship between Changes Made and Sessions Reused

In conclusion the tool's effectiveness can be hugely increased by adding functionality to recognized pages with changed names. However, deleted pages can naturally not be avoided.

7 Future Work

This tool can be enhanced and expanded in the following way:

- Form filling can be extended to cover more complex field types such as files and buttons.
- The rule for choosing values can be changed from one value to boundary values or all values found in session requests. However this could result in repetitive paths that don't enhance coverage. A technique has be developed to identify values that could lead to higher coverage and only include them. A possible way of doing this can be by examining the path in sessions and checking if an input value would expose a previously unexercised path.
- For new fields the way values are generated can be enhanced to be more realistic.
- URLs with different possible sets of parameters can be considered and dealt with.
- A tool to turn the resulting log file into a test script can be developed. Expected results would have to be entered manually by the test engineer.

The analyzing component (JSpider) has the potential to be expanded into a stand alone automatic test generator if the above mentioned enhancements were added. However, for it to be truly practical and valuable in generating test suites, analysis of the code and servlets behind the HTML front and client side scripts such as javascrit have to be done to provide sufficient coverage and high error detection.

Also, the same technique can be applied to other fields. Instead of session data a similar approach can be used for adjusting and rewriting test case or maintaining and updating test scripts automatically. However, test scripts usually have a test oracle or expected results to be checked within their code. The problem that should be considered then is how to adjust the test oracle/expected results. Usually an added field to a form wouldn't result in a completely different expected outcome but there are many cases where this is true. A solution has to be thought out and implemented. This can be a big challenge since it is hard if not impossible to automatically predict expected results.

8 Conclusion

8.1 Achievements and Interpretation of Results

This project attempted to find a fully automated approach to regression testing of web application by making use of the history of logged real user requests previously recorded since the last release. The approach was producing new session data by adjusting the old log files. The adjustments are done based on an analysis of the new version of the web application that recognizes any changes affecting the way pages interact with each other or the structure of forms.

Analysis of the application after it was modified was done by adapting an existing web crawler. Enhancements were made to handle forms and construct all independent paths from the result of the analysis. The resulting component was able to analyze web application successfully.

One of the main challenges was automated form completion and submission. The employed solution was relaying on values extracted from the old session data and on default values and random choices when the set of options is limited (e.g. checkboxes and lists). By testing several websites that contained forms with various input types, this solution proved to be successful in providing input that resulted in successful submission. Being able to submit the form and get the expected response is sufficient for the aim of this project since only the structure of the application needs to be discovered. Producing accurate inputs that would return realistic responses would be needed if the aim was actual testing.

The tool was able to intelligently reconstruct test sequences and reformat requests producing a new set of test data that can be used to produce test scripts.

The design and implementation were systematically created to make the tool as generic and independent of the fast changing web technologies as possible.

8.2 Extensions and Enhancements

In the analysis process the tool can be extended for a more thorough exploration of possible paths that might not have been discovered by the used inputs. Currently the strategy is to use one value for each field but expanding this to a number of values

carefully selected from the old sessions could result in better coverage of more complex applications. Also, enhancements can be made to the way values are chosen for new fields. The current system is flexible in allowing the test engineer to add manually add values to the database table used to fetch values. However, an automated approach would be more beneficial and in line with the objective of this project. Analysis of labels which usually hold some information for a real-life user can be a possible solution **[8]**.

Rewriting of URLs and adjusting of sequences can be enhanced by making the tool able to distinguish changed file names and differentiate between requests with the same URL but different parameter sets.

8.3 Discussion

The idea of rewriting data used for testing when it becomes invalid instead of discarding it and reproducing new data can be applied to other types of files. Test script, test suites amongst other things can be handled in the same way. However, as with session data running the test will insure that the system's old functionalities are still running correctly but coverage of completely new functionalities has to be tested separately. Moreover, the approach in general can probably be applied to testing of traditional software by analyzing input and output parameters. However, thorough research needs to be done to identify concerns and issues.

The developed regression tool provides an effortless, time-saving way of testing web applications. However, it should be combined with other forms of traditional testing to insure that the new application is bug free. Although it provides authentic testing of the system that is more likely to expose defects caused by the new modifications, the coverage for parts hugely affected by the changes is not as thorough. This is because the amount of test data for these is not as huge and varied since no matching cases exist.

In today's highly competitive environment, user's trust is the most valuable commodity. Making sure a system can still provide them with the functions they usually use and relay on after it has been changed or enhanced will help maintain their trust and loyalty. The strength and importance of this approach is in its ability to provide this.

9 References

[1] S. Karre S. Elbaum and G. Rothermel. Improving web application testing with user session data. In Proceedings of the International Conference on Software Engineering, pages 49–59, May 2003.

[2] S. Karre S. Elbaum and G. Rothermel. Leveraging User Session Data to Improve Web Application Testing. November 2003.

[3] J. Viega and J. McManus. THE IMPORTANCE OF SOFTWARE TESTING. http://www.cutter.com/research/2000/crb000111.html. January 2000.

[4] E. Miller . WebSite Testing. 2005. http://www.soft.com/eValid/Technology/White.Papers/website.testing.html

[5] R.Hower. Web Site Test Tools and Site Management Tools. http://www.softwareqatest.com/qatweb1.html.

[6] Software Testing and Test Tools Resources. http://www.aptest.com/resources.html

[7] WebXACT. http://webxact.watchfire.com/

[8] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In Proceedings of the 27th International Conference on Very Large Data Bases, Pages: 129 - 138 . 2001.

[9] G. Meszaros. Agile Regression Testing Using Record & Playback

[10] HttpUnit - http://httpunit.sourceforge.net/index.html

[11] Jmeter - http://jakarta.apache.org/jmeter/index.html

[12] SimpleTest - http://www.lastcraft.com/simple_test.php

[13] F. Ricca and P. Tonella. “Analysis and Testing of Web Applications.” In IEEE, pp. 25-34. 2001.

[14] B. Jonsson. ASTEC project: Automated Testing. http://user.it.uu.se/~bengt/ASTEC/Planning00/testing-01-03-01/

[15] L. Pollock. Web Testing Project. http://www.cra.org/Activities/craw/dmp/awards/2001/mcglade/final_paper.htm

[16] F. Ricca and P. Tonella. Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions. Proc. of TACAS'2001, Tools and Algorithms for the Construction and Analysis of Systems, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS'2001, LNCS 2031 pp 373-388. April , 2001.

[17] R. T. Fielding. Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web. Presented at the First International World-Wide Web Conference (WWW94) in Geneva, Switzerland, May.

[18] N. Synytskyy, J. R. Cordy and T. R. Dean .Robust Multilingual Parsing Using Island Grammars.

[19] M. Benedikt, J. Freire and P. Godefroid, VeriWeb: Automatically Testing Dynamic Web Sites

[20] Robert C. Miller and K. Bharat. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. Appeared in Proceedings of the Seventh International World Wide Web Conference (WWW7), Brisbane, Australia, April 1998. Printed in Computer Network and ISDN Systems v.30, pp. 119-130, 1998. Brisbane, Australia, April 1998.

[21] WebSPHINX - http://www.cs.cmu.edu/~rcm/websphinx/

[22] JSpider, http://j-spider.sourceforge.net

[23] RFC 2616 - http://www.w3.org/Protocols/rfc2616/rfc2616.html

[24] S. Sampath, A. L. Souter, L. Pollock. Towards Defining and Exploiting Similarities in Web Application Use Cases through User Session Analysis.

[25] Jspider User Manual - http://j-spider.sourceforge.net/doc/index.html.

[26] Entity Relationship Modeling Technique
http://sysdev.ucdavis.edu/WEBADM/document/td_entityrel-guidelines.htm

[27] Entity-Relationship Diagrams (ERD) - http://www.umsl.edu/~sauter/analysis/er/er_intro.html

[28] The Standard Waterfall Model for Systems Development
http://asdwww.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm

[29] Vcustomer - http://www.vcustomer.com/quality-software.htm

[30] XSD Schema Validator - http://apps.gotdotnet.com/xmltools/xsdvalidator/

[31] XPath Tutorial - http://www.w3schools.com/xpath/default.asp

[32] The Java Developers Almanac 1.4 - javaalmanac.com

[33] Creating a GUI with JFC/Swing - http://java.sun.com/docs/books/tutorial/uiswing/

[34] GotoCode - http://www.gotocode.com/

[35] Badboy! - http://www.badboy.com.au/

10 Glossary

White box	A testing technique where test data is selected based on the tester's knowledge of the system. The data is chosen to excises different paths in the code satisfying a certain criteria. The results are compared to the behavior expected from the program.
Black box	A testing technique where the user has no knowledge of the internal workings of the program. The tester only knows the input and what the expected output should be
Independent path	A path through a graph that has an edge not in any other path
ER-Diagram	Entity-relationship diagram
Regression Testing	Tests executed after an application has been modified to insure that it retains its validity and that the changes have no conflict with the rest of the system.
Functional Testing	Testing whether the application achieves the functionality for which it was designed and created.
Web Spiders	Programs that visit web sites and go through their pages to read and record information. Usually used in search engines. Also known as crawlers and robots.
Waterfall development model	A linear development method where development phases are completed sequentially without overlapping.
Dynamic pages	Pages that are created during run-time based on the input of a user or temporal variables like time of day.
User Session Data	A user's actions and requests to a web server from entering a certain web application until leaving it.

11 Appendix A

11.1 ER-Diagrams

11.1.1 Test Case Data ER-Diagram

This diagram shows the relationship between test cases and resources. A resource is any part of the web application typically a web page. The relationship is one to many; a resource can appear as part of many test cases. A resource can reference or be referenced by many other resources.

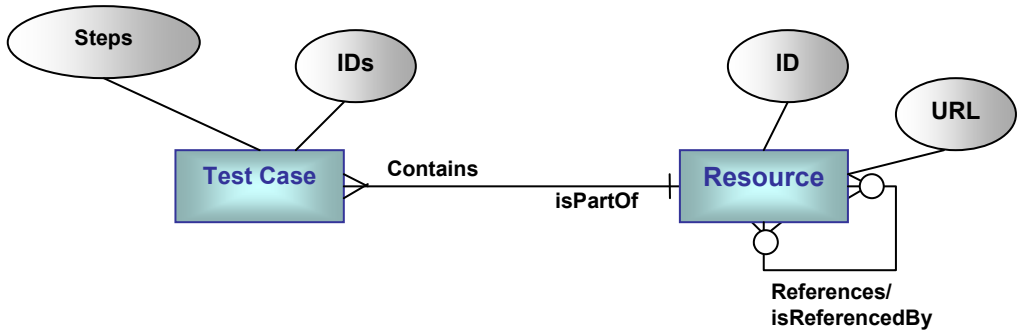

Figure A1: ER-Diagram for Test Cases and Resources

11.1.2 Session Data ER-Diagram

This diagram shows the relationship between sessions and requests. A request is the URL sent to the web server by the user. The relationship is one to many; a request can appear as part of many sessions.

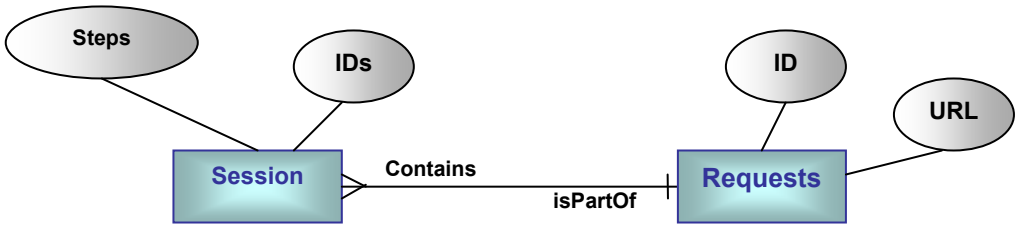

Figure A2: ER-Diagram for Sessions and Requests

11.2 Table Attribute Descriptions

11.2.1 JSpider Tables

Tables *jspider_resource* and *jspider_resource_reference* are the main JSpider tables used in different components of the RegTest system. *Jspider__resource_reference* was modified by adding the field *used* that keeps track of whether or not the row was used when constructing test cases to keep generated paths independent.

```
mysql> desc jspider_resource;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      |      | PRI | NULL    | auto_increment |
| url        | longtext     |      |     |         |                |
| state      | int(11)      |      |     | 0       |                |
| httpstatus | int(11)      |      |     | 0       |                |
| site       | int(11)      |      |     | 0       |                |
| timems     | int(11)      |      |     | 0       |                |
| mimetype   | varchar(255) | YES  |     | NULL    |                |
| size       | int(11)      |      |     | 0       |                |
| folder     | int(11)      |      |     | 0       |                |
+------------+--------------+------+-----+---------+----------------+
9 rows in set (0.01 sec)
```

Figure A3: jspider_resource

```
mysql> desc jspider_resource_reference;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| referer | int(11) |      | PRI | 0       |       |
| referee | int(11) |      | PRI | 0       |       |
| count   | int(11) |      |     | 0       |       |
| covered | int(1)  |      |     | 0       |       |
+---------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Figure A4: jspider_resource_reference

11.2.2 New Tables

11.2.1.1 Session URLs

Table *converter_session_urls* table stores all unique URLs found in session data. Field *used* is used to keep track of which requests were used and which were completely invalid for the purpose of the evaluation. Field *ID* is the primary key.

```
mysql> desc converter_session_urls;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| id    | int(11)  |      | PRI | NULL    | auto_increment |
| url   | longtext |      |     |         |                |
| used  | int(1)   |      |     | 0       |                |
+-------+----------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Figure A5: converter_session_urls

11.2.1.2 Sessions

Table *converter_sessions* stores the structure of sessions extracted during the *UploadSessions* process. One session identified by a number has many steps and each step corresponds to a certain request referenced by its ID. The primary key is a compound key consisting of *session_num* and *step_num*.

```
mysql> desc converter_sessions;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| session_num | int(11) |      | PRI | 0       |       |
| step_num    | int(11) |      | PRI | 0       |       |
| id          | int(11) |      |     | 0       |       |
| mapped_id   | int(11) |      |     | 0       |       |
+-------------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Figure A6: converter_sessions

11.2.1.3 Test Cases

Table *jspider_test_cases* stores the structure of test cases constructed by running the constructer. One test case identified by a number has many steps and each step corresponds to a certain URL referenced by its ID. The primary key is a compound key consisting of *case_num* and *step_num*.

```
mysql> desc jspider_test_cases;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| case_num | int(11) |      | PRI | 0       |       |
| step_num | int(11) |      | PRI | 0       |       |
| id       | int(11) |      |     | 0       |       |
+----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Figure A7: jspider_test_cases

11.2.1.4 Form Fields

Table *jspider_form_filling* stores fields and their values extracted from session files or entered by the user. Field name is the primary key.

```
mysql> desc jspider_form_filling;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| field | varchar(50) |      | PRI |         |       |
| value | varchar(50) |      |     |         |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Figure A8: jspider_form_filling

12 Appendix B

12.1 Log File Standard Schema

```
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0">
    <xsd:element name="Logfile" type="SessionType" />
  - <xsd:complexType name="SessionType">
    - <xsd:sequence>
        <xsd:element name="Session" type="StepType" minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  - <xsd:complexType name="StepType">
    - <xsd:sequence>
        <xsd:element name="SessionNum" type="xsd:string" />
        <xsd:element name="Step" type="xsd:string" minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
```

Figure B1: log-file-schema.xml

12.2 Schema Validation

The created XML schema was validated using an online tool. A session file was then validated against the schema to test if the defined format matches what is required. *Figure B2* shows the result of the validation.

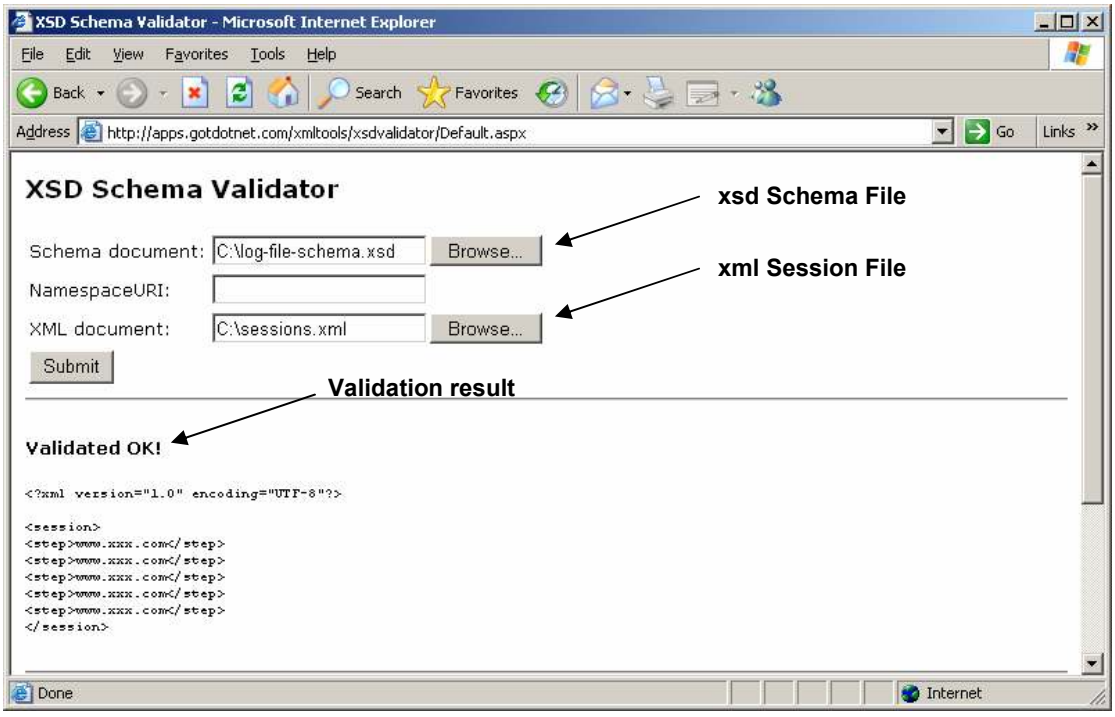

Figure B2: Validating the Schema

13 Appendix C

13.1 Testing Outcomes

```
http://localhost:8080/jsp-examples/cal/login.html
http://localhost:8080/jsp-examples/cal/cal1.jsp?
name=nadia&email=nadia207%40yahoo.com&action=submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?date=prev
http://localhost:8080/jsp-examples/cal/cal1.jsp?date=next
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=8am
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=9am
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=10am
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=11am
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=12pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=1pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=2pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=3pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=4pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=5pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=6pm
http://localhost:8080/jsp-examples/cal/cal2.jsp?time=7pm
http://localhost:8080/jsp-examples/cal/cal1.jsp?
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=8am&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=9am&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=10am&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=11am&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=12pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=1pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=2pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=3pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=4pm&description=meeting&submit
```

```
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=5pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=6pm&description=meeting&submit
http://localhost:8080/jsp-examples/cal/cal1.jsp?
date=current&time=7pm&description=meeting&submit
```

14 Appendix D

A tool called Badboy! was used to generate user session files. Users were asked to use it when browsing the application instead of using a browser.

Badboy! is an easy to use free tool for recording user interactions with a website. However, it is not designed to add a logging feature to a web application. It is rather used as a capture/replay tool. For the purpose of this project it can be used to produce user session files since the users are volunteers so the way the access the application can be controlled.

Figure D1 shows the interface for this tool.

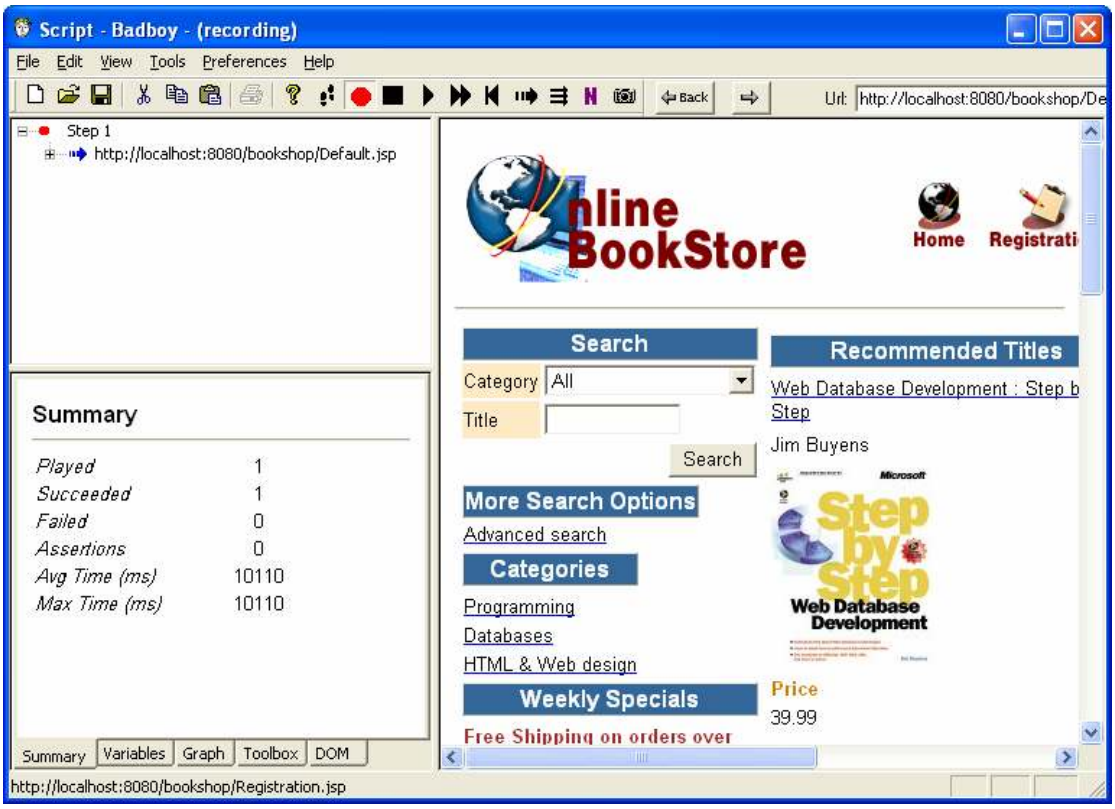

Figure D1: Badboy! interface (Recording Tool)

15 Appendix E

```
<Logfile>
 <Session>
  <SessionNum>1</SessionNum>
  <Step>http://localhost/bookshop/Default.jsp</Step>
  <Step>http://localhost/bookshop/Books.jsp?category_id=2&name=java</Step>
  <Step>http://localhost/bookshop/Login.jsp?ret_page=/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=nadia&Password=plath
      &FormAction=login&ret_page=/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/Registration.jsp</Step>
  <Step>http://localhost/bookshop/Registration.jsp?member_login=nadia&member_password
      =nadia&member_password2=nadia&first_name=nadia&last_name=saad&email=nadia@ya
      hoo.com&address=london&phone=999&card_type_id=1&card_number=22222222&FormN
      ame=Reg&FormAction=insert</Step>
  <Step>http://localhost/bookshop/Default.jsp</Step>
  <Step>http://localhost/bookshop/Books.jsp?category_id=3&name=html</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?category_id=3&name=html&item_id=20</S
      tep>
  <Step>http://localhost/bookshop/Login.jsp?querystring=category_id=3&name=html&pricemi
      n=&pricemax=&author=&item_id=20&&ret_page=/bookshop/BookDetail.jsp</Step>
  <Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=nadia&Password=nadia
      &FormAction=login&ret_page=/bookshop/BookDetail.jsp&querystring=category_id=3&na
      me=html&pricemin=&pricemax=&author=&item_id=20&</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?category_id=3&name=html&item_id=20</S
      tep>
  <Step>http://localhost/bookshop/BookDetail.jsp?rating=3&FormName=Rating&FormAction=
      update&item_id=20&rating_count=0&Trn_item_id=20&PK_item_id=20</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?item_id=20</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction
      =insert&item_id=20</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=1</Step>
  <Step>http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=2&FormName=Shopping
      CartRecord&FormAction=update&order_id=1&member_id=3&PK_order_id=1</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/MyInfo.jsp</Step>
  <Step>http://localhost/bookshop/MyInfo.jsp?member_password=nadia&name=nadia&last_n
      ame=saad&email=nadia@yahoo.com&address=london&phone=777&card_type_id=1&card
      _number=22222222&FormName=Form&FormAction=update&member_id=3&PK_member_
      id=3</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/Default.jsp</Step>
  <Step>http://localhost/bookshop/Books.jsp?category_id=2</Step>
  <Step>http://localhost/bookshop/AdvSearch.jsp</Step>
  <Step>http://localhost/bookshop/Books.jsp?name=java&author=mark</Step>
  <Step>http://localhost/bookshop/Books.jsp?name=java</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?name=java&item_id=14</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
    </Session>

  <Session>
  <SessionNum>2</SessionNum>
  <Step>http://localhost/bookshop/Default.jsp</Step>
  <Step>http://localhost/bookshop/Login.jsp</Step>
  <Step>http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=logout</Step>
  <Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=nadia&Password=nadia
      &FormAction=login</Step>
  <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  <Step>http://localhost/bookshop/Default.jsp</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?item_id=22</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?rating=2&FormName=Rating&FormAction=
      update&item_id=22&rating_count=0&Trn_item_id=22&PK_item_id=22</Step>
  <Step>http://localhost/bookshop/BookDetail.jsp?item_id=22</Step>
```

<Step>**http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction=insert&item_id=22**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=2**</Step>
<Step>**http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=4&FormName=ShoppingCartRecord&FormAction=update&order_id=2&member_id=3&PK_order_id=2**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp?member_password=nadia&name=nadia&last_name=saad&email=nadia@yahoo.com&address=london&phone=777&notes=test&card_type_id=1&card_number=22222222&FormName=Form&FormAction=update&member_id=3&PK_member_id=3**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/Books.jsp?category_id=2**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?category_id=2&item_id=10**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?rating=3&FormName=Rating&FormAction=update&item_id=10&rating_count=0&Trn_item_id=10&PK_item_id=10**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=10**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?quantity=3&FormName=Order&FormAction=insert&item_id=10**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp?member_password=nadia&name=nadia&last_name=saad&email=nadia@yahoo.com&address=london&phone=4343&notes=test&card_type_id=1&card_number=22222222&FormName=Form&FormAction=update&member_id=3&PK_member_id=3**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=10**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/AdvSearch.jsp**</Step>
<Step>**http://localhost/bookshop/Books.jsp?pricemin=3&pricemax=10**</Step>
<Step>**http://localhost/bookshop/AdvSearch.jsp**</Step>
<Step>**http://localhost/bookshop/Books.jsp?pricemin=5&pricemax=50**</Step>
<Step>**http://localhost/bookshop/Books.jsp?category_id=2&name=server**</Step>
<Step>**http://localhost/bookshop/Books.jsp?category_id=1&name=server**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?category_id=1&name=server&item_id=14**</Step>
</Session>

<Session>
<SessionNum>**3**</SessionNum>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp?FormName=Reg&FormAction=cancel**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=22**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?rating=4&FormName=Rating&FormAction=update&item_id=22&rating_count=1&Trn_item_id=22&PK_item_id=22**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=22**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=22**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=3**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp?FormName=Reg&FormAction=insert**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=22**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?quantity=5&FormName=Order&FormAction=insert&item_id=22**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>

```
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=4</Step>
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=0&FormName=Shopping
    CartRecord&FormAction=delete&order_id=4&member_id=3&PK_order_id=4</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=1</Step>
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=0&FormName=Shopping
    CartRecord&FormAction=update&order_id=1&member_id=3&PK_order_id=1</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=1</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/Registration.jsp</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=logout</Step>
<Step>http://localhost/bookshop/Registration.jsp</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=nadia&Password=nadia
    &FormAction=login</Step>
  </Session>

<Session>
<SessionNum>4</SessionNum>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=logout</Step>
<Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=nadia&Password=nadia
    &FormAction=login</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
<Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=1</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/BookDetail.jsp?item_id=22</Step>
<Step>http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction
    =insert&item_id=22</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
  </Session>

<Session>
<SessionNum>5</SessionNum>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/AdminMenu.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp?ret_page=/bookshop/AdminMenu.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp?FormName=Login&Login=admin&Password=admi
    n&FormAction=login&ret_page=/bookshop/AdminMenu.jsp</Step>
<Step>http://localhost/bookshop/AdminMenu.jsp</Step>
<Step>http://localhost/bookshop/CategoriesGrid.jsp</Step>
<Step>http://localhost/bookshop/CategoriesRecord.jsp?category_id=1</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/AdminMenu.jsp</Step>
<Step>http://localhost/bookshop/EditorialCatGrid.jsp</Step>
  </Session>

<Session>
<SessionNum>6</SessionNum>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
<Step>http://localhost/bookshop/Login.jsp</Step>
<Step>http://localhost/bookshop/Default.jsp</Step>
<Step>http://localhost/bookshop/Books.jsp?category_id=2</Step>
<Step>http://localhost/bookshop/BookDetail.jsp?category_id=2&item_id=10</Step>
<Step>http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction
    =insert&item_id=10</Step>
<Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
```

<Step>**http://localhost/bookshop/MyInfo.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp?member_password=nadia&name=nadia&last_name=saad&email=nadia@yahoo.com&address=london&phone=4343&notes=test&card_type_id=1&card_number=22222222&FormName=Form&FormAction=update&member_id=3&PK_member_id=3**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/AdvSearch.jsp**</Step>
<Step>**http://localhost/bookshop/Books.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=21**</Step>
</Session>

<Session>
<SessionNum>**7**</SessionNum>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=1**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction=insert&item_id=1**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Login.jsp**</Step>
<Step>**http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=logout**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?item_id=2**</Step>
<Step>**http://localhost/bookshop/Login.jsp?querystring=item_id=2&&ret_page=/bookshop/BookDetail.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp?member_login=n&member_password=n&member_password2=n&first_name=n&last_name=n&email=n&FormName=Reg&FormAction=insert**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/Books.jsp?category_id=2**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?category_id=2&item_id=4**</Step>
<Step>**http://localhost/bookshop/Login.jsp?querystring=category_id=2&name=&pricemin=&pricemax=&author=&item_id=4&&ret_page=/bookshop/BookDetail.jsp**</Step>
<Step>**http://localhost/bookshop/Login.jsp?FormName=Login&Login=n&Password=n&FormAction=login&ret_page=/bookshop/BookDetail.jsp&querystring=category_id=2&name=&pricemin=&pricemax=&author=&item_id=4&**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?category_id=2&item_id=4**</Step>
<Step>**http://localhost/bookshop/BookDetail.jsp?quantity=1&FormName=Order&FormAction=insert&item_id=4**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=8**</Step>
<Step>**http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=1&FormName=ShoppingCartRecord&FormAction=update&order_id=8&member_id=4&PK_order_id=8**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
</Session>
<Session>

<SessionNum>**8**</SessionNum>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/Login.jsp?ret_page=/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/Login.jsp?FormName=Login&Login=admin&Password=admin&FormAction=login&ret_page=/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/MyInfo.jsp**</Step>
<Step>**http://localhost/bookshop/ShoppingCart.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp**</Step>
<Step>**http://localhost/bookshop/Registration.jsp?member_login=new&member_password=new&member_password2=new&first_name=new&last_name=new&email=new@yahoo.com&FormName=Reg&FormAction=insert**</Step>
<Step>**http://localhost/bookshop/Default.jsp**</Step>
<Step>**http://localhost/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/EditorialCatGrid.jsp**</Step>
<Step>**http://localhost/bookshop/AdminMenu.jsp**</Step>
<Step>**http://localhost/bookshop/CardTypesGrid.jsp**</Step>

```
<Step>http://localhost/bookshop/CardTypesRecord.jsp?card_type_id=1</Step>
    </Session>
<Session>
 <SessionNum>9</SessionNum>
 <Step>http://localhost/bookshop/Default.jsp</Step>
 <Step>http://localhost/bookshop/BookDetail.jsp?item_id=1</Step>
 <Step>http://localhost/bookshop/BookDetail.jsp?quantity=0&FormName=Order&FormAction
     =insert&item_id=1</Step>
 <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
 <Step>http://localhost/bookshop/ShoppingCartRecord.jsp?order_id=9</Step>
 <Step>http://localhost/bookshop/ShoppingCartRecord.jsp?quantity=0&FormName=Shopping
     CartRecord&FormAction=delete&order_id=9&member_id=1&PK_order_id=9</Step>
 <Step>http://localhost/bookshop/ShoppingCart.jsp</Step>
 <Step>http://localhost/bookshop/Registration.jsp</Step>
 <Step>http://localhost/bookshop/Default.jsp</Step>
 <Step>http://localhost/bookshop/Books.jsp?category_id=4</Step>
 <Step>http://localhost/bookshop/Login.jsp</Step>
 <Step>http://localhost/bookshop/AdminMenu.jsp</Step>
 <Step>http://localhost/bookshop/Login.jsp</Step>
 <Step>http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=logout</Step>
 <Step>http://localhost/bookshop/AdminMenu.jsp</Step>
 <Step>http://localhost/bookshop/Login.jsp?ret_page=/bookshop/AdminMenu.jsp</Step>
 <Step>http://localhost/bookshop/Login.jsp?FormName=Login&FormAction=login&ret_page=
     /bookshop/AdminMenu.jsp</Step>
   </Session>
   </Logfile>
```

16 Appendix F

16.1 System Directory Structure

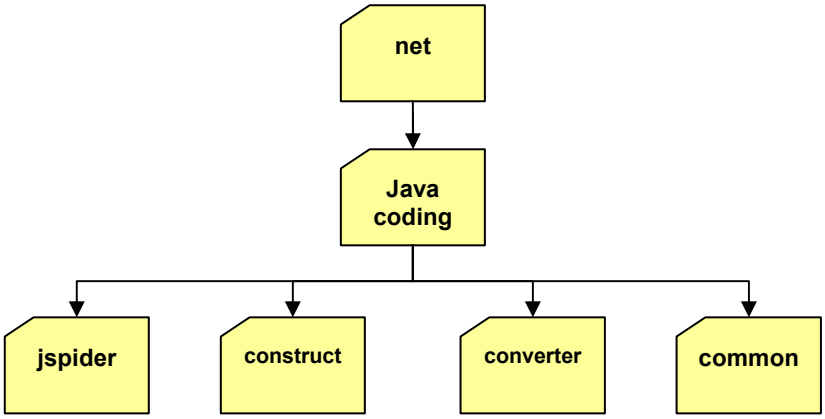

Figure B1: directory Structure of System

Each directory has all the files related to the component with the same name. The common directory contains all classes that are used by all components such as classes that access database tables.

16.2 Code Listings and Information

File Name	Lines of Code
TestCasesDB.java	96
DBI.java	91
FormFillingDB.java	101
ResourceDB.java	94
ResourceReferenceDB.java	216
ResourceReferenceRow.java	68
SessionsDB.java	272
TestCase.java	69
construct.java	79
URLExtractor.java	59
MatchingUtil.java	71
SequenceAdjuster.java	108
Session.java	69
TestCaseSeqConstructor.java	102
URLAdjuster.java	72
URLConstructor.java	63
uploadSessions.java	102
SessionPrinter.java	85
Converter.java	45
InterpreteHTMLTask.java	5 lines added
URLFinder.java	340 lines added
formatSessionFile.java	118
RegTestApplication.java	270