FAKULTÄT FÜR !INFORMATIK

Extending HtmlUnit for Test Automatisation of Web Applications using AJAX

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Andreas Langer
Matrikelnummer 0125651

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Grechenig
Mitwirkung: Dipl.-Ing. Mario Bernhart

Wien, 20.04.2009

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

DIPLOMA THESIS

Extending HtmlUnit for Test Automatisation of Web Applications using AJAX

Submitted for the academic degree of
Diplomingenieur
(Dipl.-Ing)

carried out at the
Institute of Computer Aided Automation
Research Group for Industrial Software

Vienna University of Technology

under the guidance of
Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Grechenig and
Dipl.-Ing. Mario Bernhart

by
Andreas Langer
Freyenthurmgasse 18/1/6
1140 Wien

Vienna, April 20, 2009

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am ---

Name

Acknowledgements

This work has been carried out at the research group for Industrial Software (www.inso.tuwien.ac.at), an informal group hosted by the Institute of Automation at the faculty of Informatics at the Vienna University of Technology.

I would like to express my sincere gratitude and appreciation to all those who made this diploma thesis possible. Special thanks are due to my advisors Mario Bernhart and Professor Thomas Grechenig and to the developers of HtmlUnit Daniel Gredler, Stan Silvert and Marc Guillemot.

I am deeply indepted to my parents for supporting and motivating me throughout the years of studying and stimulating my interest in technical science. I thank my sisters, friends and colleagues who made the course of my studies a marvellous time to remember.

Abstract

Software testing is a part of the software development process. It is a challenging task and testing Web-based applications may be even more difficult, due to the peculiarities of such applications. New technologies in internet applications require adaptations of most test frameworks. A very popular technology which is widely used from many well known internet services is called AJAX. AJAX stands for "Asynchronous JavaScript and XML" and is a combination of many known technologies, which allow asynchronous data communication between server and client. In contrast to the standard synchronous data communication, which means loading an entire web page after each request, AJAX allows dynamic refreshes of the web page and data exchange of parts of the page. Thus testing of such internet applications is different from testing of "normal" applications because there can be interactions with the server without the need to load a new page. Communication is done in the background and the current page stays the same except of minor changes.

This paper gives an overview of the basic principles of automated software tests, especially automated testing of web based applications. Furthermore it shows the influence of AJAX concerning web development and testing of web applications. The main issue of this paper is to extend HtmlUnit for test automation of web applications using AJAX. Based on the used concepts it is possible to extend further test frameworks.

HtmlUnit is a test framework for internet applications. It simulates a web browser and provides many functions to check the content of a web page. HtmlUnit doesn't have a graphical user interface but it is written in the programming language Java. Test cases are written using JUnit tests but like many other test frameworks HtmlUnit has some problems testing AJAX enabled internet applications.

keywords: test automation, AJAX, HtmlUnit, web testing

Kurzfassung

Software Testen ist ein fixer Bestandteil im Software Entwicklungsprozess. Automatisierte Tests können den Aufwand für das Testen der Software erheblich erleichtern. Speziell im Bereich der Internet Applikationen gibt es eine Reihe weiterer Aspekte die beim Testen beachtet werden müssen. Neue Technologien im Internet verlangen eine Anpassung bei vielen Testumgebungen. Die bekannteste Technologie die zurzeit sehr populär ist und von vielen Internet Diensten verwendet wird heißt AJAX. AJAX steht für „Asynchrounous JavaScript and XML“ und ist eine Kombination vieler bekannter Techniken, die zusammen asynchrone Kommunikation zwischen Client und Server erlaubt. Im Gegensatz zum konventionellen Ansatz, dass immer eine vollständige Seite geladen werden muss, ermöglicht AJAX den Datenaustausch von Teilen einer Webseite oder das dynamische Nachladen von Inhalten. Dadurch verändert sich das Testen von Internet Applikationen erheblich, da eine Interaktion mit dem Server nicht automatisch das Laden einer neuen Seite bedeutet sondern im Hintergrund abläuft und die aktuelle Seite, bis auf parzielle Veränderungen, dieselbe bleibt.
Diese Arbeit gibt eine Übersicht über das Automatisierte Testen von Software, im speziellen von Internet basierten Programmen. Des Weiteren werden der Einfluss von AJAX und die damit verbundenen Veränderungen analysiert. Der Kernpunkt dieser Arbeit ist das Erweitern von HtmlUnit, um damit auch Internet Applikationen, die AJAX verwenden, testen zu können. Aufbauend auf diesem Konzept ist es möglich, auch andere, ähnlich gebaute Test Werkzeuge zu erweitern.
HtmlUnit ist eine Testumgebung für Internet Applikationen. Es simuliert einen Browser und stellt viele Funktionen zum Überprüfen von Internet Seiten zur Verfügung. HtmlUnit hat keine Grafische Oberfläche sondern ist komplett in der Programmiersprache Java geschrieben. JUnit Tests bieten umfangreiche Möglichkeiten zum Testen von Applikationen, aber HtmlUnit hat, wie viele andere Testumgebungen, Probleme bei Internet Applikationen, die AJAX verwenden.

Stichwörter: Testautomatisierung, AJAX, HtmlUnit, Testen von Internet Applikationen

Contents

List of Figures

List of Tables

1 Introduction

Software Testing is a fixed component of software development. This includes creating automated regression tests. Specifically for Web applications, there are test environments like HtmlUnit that can imitate a web browser, but they only work after the classical principle of “request and wait for the response”. New technologies like AJAX (Asynchronous JavaScript and XML) and Flash extend the classical model and enable asynchronous data transmissions and updating only sections of a web page. These technologies require a new approach in software testing, since the response to a request only arrives delayed and meanwhile further actions could have been launched.

1.1 Problem Description

Asynchronous data communication in web applications increase the complexity of the application and hence the sources of error. Responses to requests may arrive too late because the user has already left the page or has done some other action and the response has no more relevance. For this reason it is especially necessary to run extensive tests in this area, and a part of these tests will be automated tests. The main problem here is the registration of asynchronous requests, the interception of responses, the assignment of the correct response to the request and finally the impact on the web browser

1.2 Motivation

The motivation of this work is to enable HtmlUnit, a very easy to use web application testing tool, handle applications that use AJAX techniques.

1.3 Goals and Objectives

The aim of this paper is the creation of a concept how to write automated tests for applications using asynchronous data transfers. Furthermore the existing test tool HtmlUnit is evaluated and extended to make it compatible to the new technologies.

1.4 Organization of this Thesis

After the general introduction in chapter 1 the next chapter reviews the technologies that are needed to understand this thesis. This includes basics about automated software tests, testing of web applications, description of all parts of AJAX especially the XMLHttpRequest object, description of the web testing tool HtmlUnit and a short comparison of HtmlUnit with JSFUnit, a web testing tool especially designed to test web applications using Java Server Faces (JSF). Chapter three to six assume the knowledge of the foundations.

Chapter 3 presents some criteria for the enhancements in HtmlUnit, presents different approaches and compares them to the existing one.

Chapter 4 describes the implementation of the enhanced HtmlUnit.
Chapter 5 specifies three web applications and the implementation of test cases using the old and new features of HtmlUnit.
Chapter 6 validates the test results and compares the different technologies.
Chapter 7 summarizes the key findings and presents the key benefits of the new implementation.
The appendix contains the source code of the new and the altered classes of HtmlUnit and the testcases used in chapter 5 and 6.

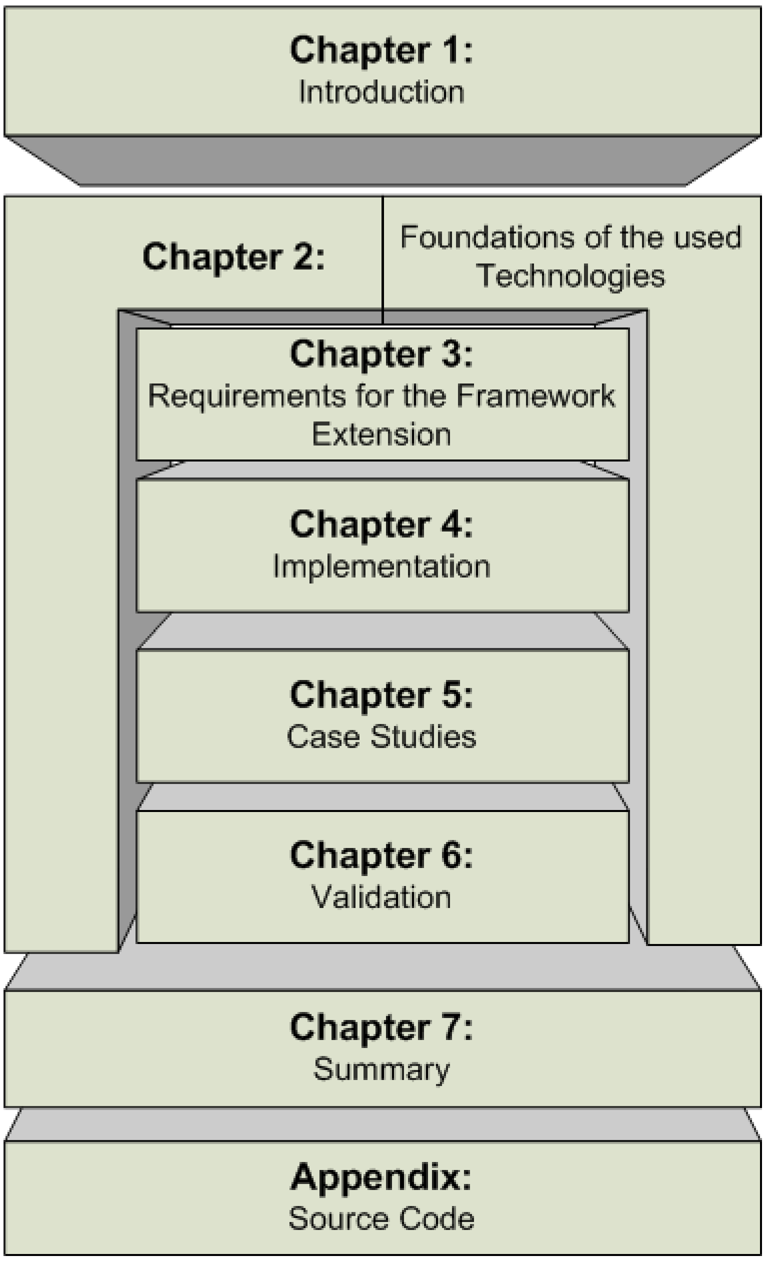


Figure 1.1 Organisation of this Thesis

2 Foundations

This chapter describes the technologies that are needed to understand the extension of HtmlUnit. It is divided into three sections.
The first one explains the basics of software test automation. It provides some information on how to write effective automated software tests and outlines some difficulties of automated tests compared to manual tests.
The second section describes how web applictions work and what to take account of when writing tests for web applications. It also explains the technologies used by AJAX enabled web applications.
The third section introduces the open source tools HtmlUnit and JSFUnit.

2.1 Software Test Automation

Software testing [6] is a crucial part of the software development process. Automating software testing can reduce the effort of adequate testing. Automated tests can be run within minutes while doing these tests manually would take several hours. That is why companies can save a lot of money using automated tests or at least produce better quality software more quickly.
A system has a very big number of possible tests. Only a few of them really can be implemented because the tester only has limited resources so he has to choose the right tests so that most of the defects of the software can be found. There are four criteria for software tests that describe the quality of software tests:

1. the defect detection effectiveness
2. it should be exemplary which means it should cover more than one thing
3. the effort to write, perform, analyze and debug the test case
4. the maintenance effort of the test case when the software changes

These four criteria have to be balanced one against another. Complex test cases that cover many issues and are very likely to detect defects normally need more effort to be written and maintained while Simple test cases are very easy to maintain and implement but they only cover special case and aren’t very likely to detect defects.

2.1.1 Objectives and non-objectives of automated software tests

Many managers expect software test automation to be a silver bullet that it will help to save money and to produce better quality software in less time compared to manual testing. Unfortunately this isn’t the whole truth. There are many falsely expected benefits of test automation [7]:

- All tests will be automated:

This isn’t desirable. According to the four criteria for software tests it has to be analyzed whether the effort to implement the tests justifies automating the test case.

- There will be immediate pay back from automation:

Usually the opposite happens. Building automated software tests takes more time than doing these tests manually and the company has to invest in test ware software licenses, training, additional hardware and other things, so first the company has to pay more money but automatic software tests will pay back later, after they are used several times after creation for build tests or every time the software changes or is improved.

- There is zero ramp up time:

It takes some time to automate tests. The right tools have to be selected and installed. The testers may need some training to use these tools and it takes more time to implement an automated test than to run it once manually.

- There is one tool for test automation that fits perfectly:

There are many commercial or free automation tools that fit some organization or project well but normally different projects require different tools. It depends on the technology used in the project to choose the right tool and the next project may require a totally different tool or toolset.

- Test automation provides automatic defect reporting without human interaction:

The test report always has to be reviewed by the tester. It could contain error cascades or other false detection of errors. Reviewing reports can require a lot of effort.

- The use of capture/replay tools can speed up creation of automated software tests:

On the first sight it seems that the fastest way to produce automated tests is to use a capture tool to record the manual test. This recording can be replayed by the tool. However capture/replay tools only work in the rare case when the product is so stable that there will be hardly any changes in the product. It is very difficult to maintain a captured test case and in most cases the tester just has to recapture the whole case. It is also possible that the replayed tests fail because of some changes in the database. Just imagine the captured test case deletes two items from a list and adds only one item. Each time the test is replayed by the tool the list will decrease by one and finally there will be an error because there are no items in the list that can be deleted. Some capture/replay tools even fail when the resolution of the monitor is changed or because of system messages that pop up. To sum up capture/replay is not test automation. There are also other forms of automated test case creation [23] [26], but they suffer the same problems.

2.1.2 The promise of test automation:

- Existing tests can be run on a new version of a program. This is the most obvious task of automated tests and it is also the field of application of test automation where the investment pays back.
- Tests can be run more often in less time.
- Special tests like load test can be performed. Load tests would need a few hundred testers but when using automated tests only a computer with enough re-

sources is needed. These automated tests can be replayed while manual tests with a few hundred testers would always differ when trying to replay them.

- The resources can be used better. Boring and repeating tests can be automated and performed by the test tool, while the human testers can put their effort into designing better test cases. Tests can be performed over night while the machines normally run idle.
- Consistency and repeatability of tests. Automated tests are always performed the same way with the same inputs. They can be executed on different hardware and software configurations which gives a consistency of cross-platform quality.
- Reuse of tests. Automated tests will be reused many times so it is worth spending time to make sure that they are reliable.
- Earlier time to market. Once the automated tests have been written, the testing elapsed time can be shortened when building a new version.

2.1.3 Automated Comparison

Test verification [13] is the process of checking whether the software has produced the right outcome or not. This is done by comparing the actual outcome to the expected outcome. While some tests only need one single comparison other more complex tests may need several. It is very important to do the right decisions which information to compare because it has a significant impact both the effectiveness and efficiency of the tests. When performing tests manually the tester simply has to look at what the software is producing and decides whether the output is correct or not based on an understanding of what the software should produce. When writing automated tests these decisions based on the understanding of the software have to be transformed into adequate comparisons.

Automated comparison tools are programs that detect differences between two sets of data, usually the test outcome data and the predicted outcome data. It depends on the particular comparator to do comparisons as accurate as possible. Simple comparators can only compare strings or numbers while others have extensive capabilities for comparing several different data formats like graphical formats or databases. Good comparators can highlight the differences, and provide facilities to help you browse the differences. However a comparator can’t tell whether a test has passed because it only compares data. Nevertheless we use it that way. It is nearly impossible to insure that a software behaves correct in every detail because it would be too much effort to compare every detail. We implement just the most important comparisons and assume that the test case has passed when all of them have passed.

General comparison guidelines

- Comparisons should be kept simple to avoid false failures or missed differences.

- Many small comparisons are easier to understand and maintain than one big comparison.
- All comparisons should be documented. The script should contain a description of what the comparison does and what it doesn’t do.
- Comparisons should be efficient. The tester has to keep in mind that some comparisons could take a long time and therefore he has to use efficient algorithms.
- Comparisons of bitmaps should be avoided. Hardly any tool exists that is able to reliably compare pictures with minimum differences.
- The comparisons should be selected to obtain a good balance between robust and sensitive tests. The precise description of robust and sensitive tests can be found in chapter 2.1.6: Sensitive tests versus robust tests.

2.1.3.1 Dynamic comparison

Dynamic comparison is performed while the test case is running. It is the most often used form of comparison and very similar to the way a tester would check things in manual tests. Every output can be compared to the expected values during runtime even when they get overwritten later. Even output which isn’t visible like the attributes of elements of a graphical user interface can be taken and compared to expected values.
However it makes more effort to write test scripts with many dynamic comparisons because they have to be embedded into the script which makes them more complex and increases the likeliness of errors in the test script. The maintenance costs also rise because of more script debugging. Trivial changes to the screen output can result in unimportant differences found by the comparator and accordingly to false error detections.

2.1.3.2 Post-execution comparison

Post-execution comparison is performed after the test case has run. It normally compares outputs other than those that have been sent to the screen like files that have been created or new entries in the database. Most test execution tools do not include this method so additional tools have to be used in order to perform post-execution comparison. That’s why using this method seems more difficult to implement than using dynamic comparisons. Post-execution comparison helps to keep the test script short and minor differences in the output of a program do not lead to false error detections.
Post-execution comparison can either be active or passive. When we only use the resulting document or the database after some fields were modified we talk about passive comparison. For performing active comparison we have to save the output of the program during test execution and perform comparisons afterwards. This method has a big advantage compared to dynamic comparisons. The whole test case has already run when the comparison starts and the tester can divide the comparisons into many groups and sub groups and execute the subgroups only when there was no error in the first group.

However when using active comparison the test script won't get simpler compared to the script using dynamic comparisons because the dynamic comparisons are only replaced by capture instructions. It is possible to use additional comparison tools and the comparison can be done offline. The comparison can be invoked directly by the test tool after the test execution or the tester has to invoke the post-execution comparison tool because they often can't be integrated well into the test tool.

2.1.3.3 Simple comparison

Simple comparison looks for identical matches. It is very easy to implement such comparisons so fewer mistakes are likely to be made when writing the test cases and it is easier to understand for new team members. Thus maintaining such comparisons is also easier then maintaining the above ones. When simple comparison is sufficient for a test case it should be used.

2.1.3.4 Complex comparison

Complex comparisons allow us to compare actual and expected outcomes with known differences. Common examples of where complex comparison is required are:

- dates and times
- unique identity numbers which are created every time
- different output orders
- different text formats
- different values within a range

The easiest common way to use complex comparisons is to just ignore these fields. Another way is to use some sort of pattern matching using regular expressions or to extract the fields that have to be compared to an expected value.

2.1.4 Comparing different types of outcome

2.1.4.1 Disk based outcomes

Text files

Text files are the easiest type of data to compare and nearly all comparators provide functions to scan them. Good comparators provide tools that can highlight differences and allow the tester to browse the differences.

XML files

XML files are a special type of text files. They can be parsed using comparators for normal test files but most testing tools provide additional tools for parsing xml files and building up an XML tree.

Non textual forms of data
These files usually need a special comparator which can read and parse the file and extract data structures.

Databases and binary files
These files also need a special comparator or a driver that can extract information from these files which can be processed using the standard text tools.

2.1.4.2 Screen based outcomes

Character based applications
Since the screen is already an array of characters it can be addressed using the row and column number.

GUI based applications
GUI applications offer a wide range of output styles. They use items like buttons, menu bars, text fields, bitmaps, checkboxes and others. There are different approaches to extract data from GUI based applications. The easiest way is to use a comparator that can handle that special application. Each application is written in a certain programming language and there are test tools for each language that are able to deal with graphical user interfaces.

Graphical images
Graphical images are an array of dots called pixels. It is very difficult to check and compare images but a few test tools allow to compare them and to specify a tolerance with the aim to ignore differences that aren't really noticeable. Many capture/replay tools offer bitmap comparison but they only work correctly when the tester uses a specified screen resolution and colour depth. Comparing images is a tough job and is hardly used in automated software tests. There are many projects in universities and companies that try to deal with image recognition.

2.1.4.3 Other types of outcome

Multimedia applications
Multimedia applications produce a test outcome that isn't readily comparable like sounds or videos. It is possible to use special tools that can analyze videos but testing tools normally do not provide such capabilities

Communicating applications
These are applications that have outcomes involving communication with other applications or hardware devices and normally also require special testing tools.

2.1.5 Comparison filters

Comparison filters are performed on both an expected outcome and the corresponding outcome to translate the original data into a simpler form so that simpler comparator tools can be used. A filter removes legitimate differences from the outcome like the header information of a web response. The filter can be applied only to the program outcome when the expected outcome has already the right format. A filter offers some advantages but also a few disadvantages:

- It enables the use of text manipulating tools that are widely available instead of complex and expensive comparison tools.
- It can be reused in many situations.
- It enables simple comparison tools performing complex comparisons.
- It simplifies debugging by dividing complex comparisons into many easy comparisons.
- It generally requires programming skills to create filters and they need to be tested itself before they can be used with confidence.
- A filter simplifies the output by ignoring fields and might cause the miss of defects.

2.1.6 Sensitive tests versus robust tests

We need to decide what information needs to be compared but it is hard to decide how much of the output should be compared and how often it should be compared to expected values. Tests that do a lot of comparisons are called sensitive tests and are more likely to fail compared to robust tests that only compare the most necessary values. On the other hand robust tests are more likely to miss defects. The tester has to find a compromise solution between these two methods. Each of them has some advantages in certain scopes:

- **Susceptibility to changes:**

Sensitive tests are more difficult to maintain because it is very likely that they have to be changed when the program changes.

- **Implementation effort:**

Sensitive tests cause more effort to be implemented because they contain more comparisons than robust tests.

- **Miss defects:**

Robust tests are more likely to miss defects because they include fewer comparisons and ignore a lot of the output.

- **Failure analysis effort:**

It is easier to analyze failures in sensitive tests because they provide more feedback and can find mistakes earlier than robust tests because the robust tests ignored the according field.

- **Storage space:**

Sensitive tests need more storage space because they compare more data than robust tests.

- **Redundant errors:**

When using sensitive tests it is more likely that many of them fill fail for the same reason, even when there is only a minor change.

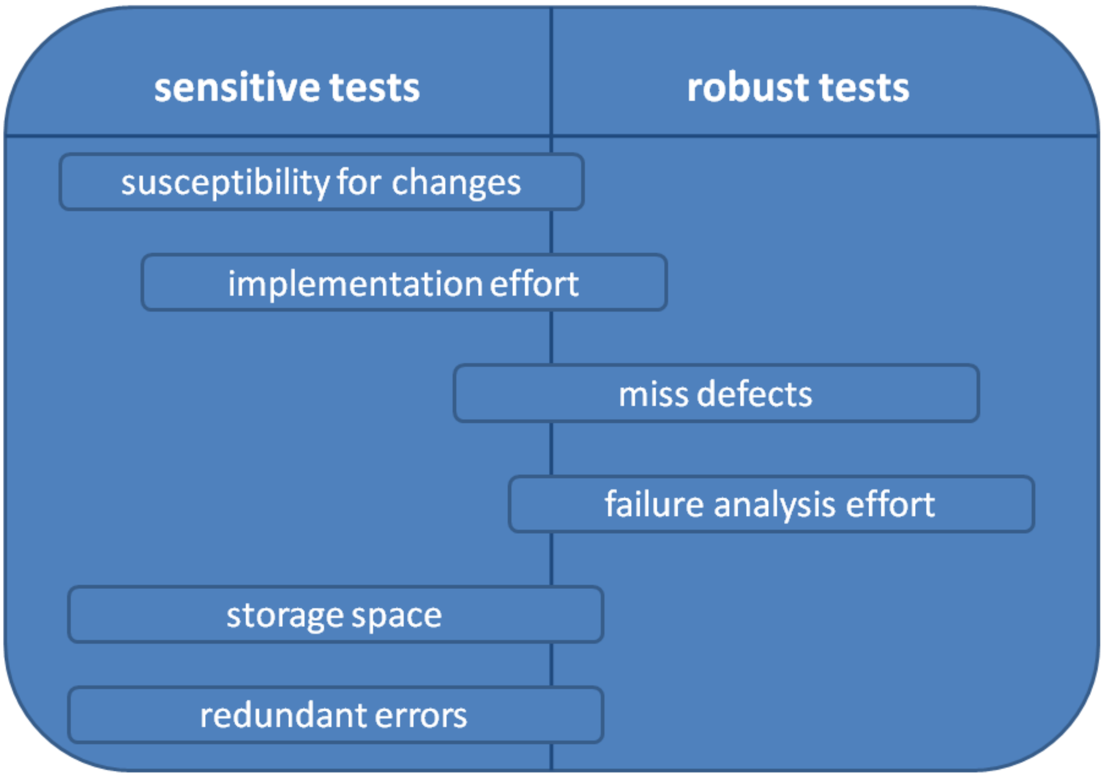


Figure 2.1 sensitive tests vs robust tests

2.1.7 Scripting techniques

Test scripts are a necessary part of test automation. It is very important to write maintainable scripts that can be reused. Writing scripts is very similar to programming software applications. The tester has to spend time testing the tests, debugging and fixing them. Changes in the application usually forces some tests to be changed. Recording a manual test case normally results in a long linear script containing all instructions. Recording many test cases produces many long scripts, which are difficult to handle because they often contain same pieces of code like a login function. In order to make these scripts maintainable, they have to be divided into many small pieces. The best idea is to code scripts from the scratch when starting to write automated tests. The tester should write small scripts that each perform a specific task. A test case uses many of these scripts and additional code. The idea is to avoid duplicate scripts [5]. When the one part of an application changes the tester normally just has to change the script using this part of the application and all test cases using this script will work again. When you just implement one test case after the other without making use of this technique you will have to change the code in each test case using the modified part which may last quite a long time. When test automation begins you will just have a fistful of scripts and it won't be difficult to keep track of them but when the number of scripts increases you need a way to organize them.

There are several characteristics for good scripts:

- The size of a script should be small
- Each script should have a clear, single purpose
- Each script should contain a specific documentation
- Many scripts should be reused be different test cases
- It should be easy to see and understand the structure of scripts to be able to make changes
- Scripts should be easy to maintain. Changes in the software should only affect a few scripts.

2.1.7.1 Linear scripts

A linear script is the result of recording a manual test case step by step. It contains all keystrokes and maybe additional dynamic comparisons or capture instructions. Each test case has its own script and is completely independent from the others.

Advantages of linear scripts:

- The tester can just sit down and write the test without any upfront work.
- It provides a production log of what was actually done.
- The tester doesn't have to be a sophisticated programmer. Capture/replay tools can help to produce these scripts.
- It is easy to build tests for demonstrations of the application and the test tool
- The test cases are independent from each other.

Disadvantages of linear scripts:

- Every test case has to be built from scratch, except when using copy and paste.
- The test input and all comparisons are hard coded inside the script
- There is no reuse or sharing of the scripts except when using copy and paste.
- These scripts are vulnerable to software changes.
- These scripts are very expensive to maintain because after a software change all scripts using the changed part of the software need to be rewritten or changed.

2.1.7.2 Structured scripts

Structured scripts are very similar to linear scripts but they additionally contain special instructions to control the execution of the script. There are three basic control structures:

- sequence control structure

The sequence is just a linear list of instructions which are performed one after the other.

- selection control structure

The selection provides the script the ability to make decisions. The best known selection structure is the "if" statement.

- iteration control structure

The iteration control structure allows the script to repeat some instructions a specific number of times or until a certain condition is met. This control structure is also known as "loop".
Furthermore structured scripts can call other scripts.

The advantages and disadvantages of structured scripts are very similar to the ones from linear scripts except:

- Structured scripts are more robust because they can check for specific things which would cause the linear scripts to fail.
- The tester requires at least basic programming skills.

2.1.7.3 Shared scripts

Shared scripts are shared or used by more than one test case. The idea is to write many small scripts with a clear, single purpose each. When a test case needs the functionality of a script it simply calls the script. It makes sense to use shared scripts for rapidly changing software because they do not require long and expensive maintenance. Shared scripts can be created from linear scripts by dividing them into several parts. Hard coded strings have to be replaced by variables. The tester needs programming skills to create shared scripts and also needs to organize and document them. When using shared scripts the number of these scripts can increase very fast. Furthermore they support the use of control structures like the sequence, selection and iteration control structure.
Shared scripts can either be application-specific or application-independent. Application-independent scripts can be used for many software applications, so they may be useful for a long term and they are worth putting additional effort into. Typical examples for application-independent scripts are logging, input retrieval, results storage, error recovery, checking and comparison scripts.
Application-specific scripts cover individual screen or window routines and the navigation in the application.

Advantages of shared scripts:

- Similar tests take less effort to implement because the shared scripts of other test cases can be used.
- The maintenance costs are lower than for linear or structured scripts
- Test code duplications are eliminated.
- It is worth investing more time into creating shared scripts especially when they get used from hundreds of test cases.

Disadvantages of shared scripts:

- The tester has to keep track of a large number of scripts and has to find the right shared scripts when implementing the test cases.
- The tester needs programming skills.
- They require more time for planning.

2.1.7.4 Data-driven scripts

Data-driven scripts [8] store the input data in a separate file. The test script reads the data from the file when the test is executed. The data file can be exchanged in order to run the test with different input values. Normally the data file contains many sets of data for the same test case and the test case is executed once for each data set. The scripts have to be created by a tester with good programming skills because they contain additional instructions to read the data from the file. They only contain variables and no hard coded input values. The data file can be created by anyone without any programming skills. Using data-driven scripts enables the tester to implement many more tests with minimal effort. The data-driven approach can be combined with the techniques used for shared scripts or structured scripts.

Advantages of data-driven scripts:

- It is easy to create additional tests with different input values.
- The writer of the data files doesn’t have to be a programmer.
- One test script is used for many tests and only the script has to be maintained when the application changes.

Disadvantages of data-driven scripts:

- The scripts have to be written by someone with a technical background.
- The set-up of data-driven tests takes more effort than using the script techniques mentioned before.
- It is only useful for applications with many tests.

2.1.7.5 Keyword-driven scripts

Keyword-driven scripts are data-driven scripts with some extensions. While data-driven scripts can be divided into a logic part, the control script, and a data part, the data file, which have to be synchronized, keyword-driven scripts allow the tester to put some control information into the data file. The tester should be able to specify automated test cases without having to specify all execution detail. The data file describes what the script should test but not how it should be tested. Keywords are used to identify the function that should be tested followed by some input data and expected outcomes. Of course the tester has to provide supporting scripts that can be referenced by the key-words.

The control script just reads the extended data script and calls the function referenced by the keyword using the additional information provided by the data file. The data file specifies the order of the test execution. The advantage of this approach is that the test cases can be developed separately from the test scripts without using a programming language. However first the keywords have to be specified and a good programmer has to implement the supporting scripts.

Advantages of keyword-driven tests:

- The number of supporting scripts is independent from the number of test cases because it is only dependent on the size and the functionality of the software application that has to be tested.
- The test cases are written in a separate file and are independent from the testing tool.
- No programming skills are needed to write the test cases.
- It is easy to create additional tests with different input values.
- The writer of the data files doesn't have to be a programmer.
- One test script is used for many tests and only the script has to be maintained when the application changes.

Disadvantages of keyword-driven tests:

- The scripts have to be written by someone with a technical background.
- The initial set-up can take a lot of effort
- It is only useful for applications with many tests.

2.1.7.6 Summary of scripting techniques

	Linear	Structured	Shared	Data-driven	Keyword-driven
structured or unstructured	unstructured	structured	structured	structured	structured
script contains	constants	constants	constants and variables	variables	variables and keywords
script approach intelligence	none	ifs and loops	ifs and loops	ifs, loops and data reading	ifs, loops, data reading and keyword interpreter
programming skills	none	low	medium	good/none	good/none
maintenance costs	very high	high	low	low	low
test set-up effort	very low	low	medium	high	very high
effort of writing additional tests	high	high	low	very low	very low
reusability of the scripts	none	low	high	high	very high
estimated ratio (scripts:test cases)	1:1	1:1	1:5	1:7	1:10

Table 2.1 Summary of scripting techniques

Table 2.1 summarizes all scripting techniques and gives a brief overview of the advantages and disadvantages of each technique. It also contains the estimated ratio between scripts and test cases of each technique which can be very helpful for finding the right technique for the particular project.

2.1.8 Test maintenance

There are some attributes that affect the maintainability of tests.

The number of test cases

The more tests there are the more tests will have to be maintained when the software changes. This effort can be reduced when using an adequate scripting technique. Normally the effort for maintenance grows indirectly proportional to the number of test cases.

The quantity of test data

Using lots of test data increases the effort of maintenance when data structures or types used in the software application are changed.

The test data format

Some software applications use special data formats. Test cases need generators or other tools to generate these data structures and these tools also have to be adopted when the software application changes. The text format data is the most flexible data format and should be used whenever it is possible.

The time to run a test case

Test cases which take a long time to execute usually perform many comparisons and other things. The chance for an error increases when the test case takes longer and a lot of tests aren’t executed when the error occurs in the middle of the test. This means the tester has to debug the test case and rerun it at least once. When possible the test cases should be as short as possible and only test a single particular function of the software application.

The debug-ability of test cases

When a test case fails it is important to find the reason why it fails. Therefore additional information has to be added to the test protocol. A tester should always keep in mind what information may be needed to understand a test failure and add some debugging information.

Independences between tests

Some test cases use the output of a former test case as input. It goes without saying that when the former test case fails the following test cases that try to use its output will also fail. When a test case has to be changed all other tests that use its output also have to be changed.

The naming convention

When the quantity of people in the software testing team exceeds the number of two it is strongly recommended to adopt some naming conventions. Otherwise the testers will need more time to find the right scripts or even write duplicate scripts which leads in higher maintenance efforts and costs.

Test complexity

Complex software tests are harder to understand and to maintain. They should be avoided when possible. Whenever you want to automate a complex test you should first compare the effort for writing and maintaining it to the effort likely to be saved later. Sometimes it is more reasonable to perform complex tests as manual tests.

Test documentation

In every software project it is very important to have a good documentation. Test automation is also a project and needs to be documented to save time and money when maintaining the tests.

2.1.9 Metrics

In software testing many things can be measured [10] to determine whether a choice that has been done was a good choice or whether the testing process in general is effective. These measurements help to evaluate choices and find the right decisions, monitor changes in the testing process, compare the software to other software projects and even predict problems.

Here is a list of things that can be measured in software projects and especially in software testing projects:

- lines of code
- number of classes, methods, functions
- size of compiled code
- time for writing and executing the tests
- number of developers
- number of tests
- code coverage
- number of defects found while testing
- number of defects found after testing
- total number of known defects
- number of defects fixed
- number of scripts
- number of automated tests
- costs of licenses, effort

Using only one of these metrics doesn’t make much sense because for example a big number of defects found can either mean that the testing was thorough and only a few bugs remain or that the software simply has a lot of bugs. But when taking and combining these metrics, some important information can be calculated.

Defect Detection Percentage (DDP)

$$DDP = \frac{defects\ found\ by\ testing\ before\ \mathrm{release}}{total\ known\ defects}$$

DDP specifies the percentage of defects that were found by testing compared to the total number of known defects. The total number of known defects normally increases as time goes on so the DDP gets down. The end value isn’t known before the software product is retired but as the number of new defects found gets lower as time goes on, it is possible to estimate the end value. When the development of a software application is finished and the product gets economically used all known defects so far have been found by testing but it is ridiculous to assume that the DDP is 100%. At this point it is important to estimate a DDP based upon the defects found so far and maybe the code coverage of the test cases. It is possible to compare this estimated DDP to DDPs of other software projects, which used a different test process, and to find improvements.

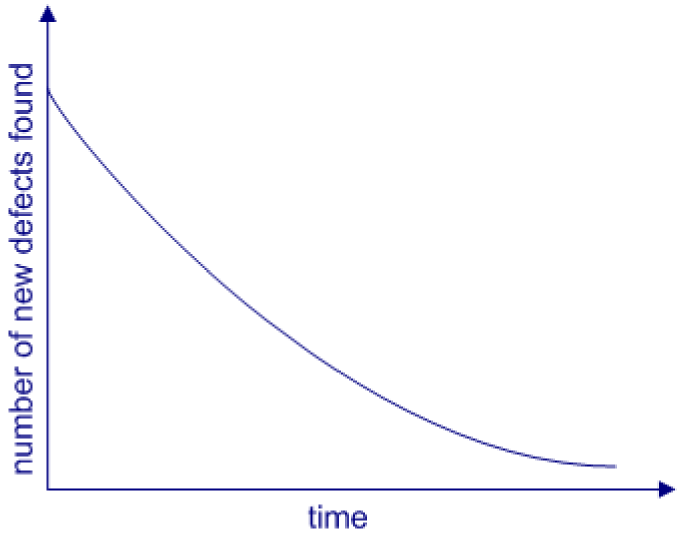


Figure 2.2 new defects found

After a short period the DDP becomes more precise and allows drawing a conclusion whether the new test process has been an advancement compared to the old one.

Defect Fix Percentage (DFP)

$$DDP = \frac{defects\ \ fixed\ before\ \mathrm{release}}{total\ known\ defects}$$

DFP is very similar to DDP. Not all defects that are found get fixed before the release and DFP only counts the fixed ones. DFP can never be higher than DDP but both values can be equal when all found defects are fixed. DFP is more important for software quality than DDP because in having a high DFP means that most defects were fixed while having a high DDP only shows that a lot of defects were found but it doesn't ensure that they were corrected.

However a high DFP and DDP need not automatically mean that the software testing was successful. There are some pitfalls when using these techniques:

- When the testing is really poor and no defects are found or the software application is really without defects the denominator of the division could be 0 which would cause an error. It is very unlikely that this will ever happen, but this in that case DFP and DDP should get the value 1.

- When testing after the release is very poor the DDP and DFP might keep at a higher level than they should.
- A small amount of reported errors can also be a result of unhappy users who do not like that software.
- The exact final result will never be reached because in a complex application there are always at least minor bugs that will never be found.
- The first results of these measures can’t be obtained before the application is re-leased.
- DFP and DDP are dependent on the number of people using the software and the amount of testing. Poor testing and a small number of users lead to high DDPs.

2.2 State of the Technology

2.2.1 Automated web testing

The testing of Web-based applications [3] [17] [18] has much in common with the test-ing of desktop systems: You need to test the usual functionality, configuration, and compatibility, as well as performing all the standard test types. But Web application testing is more difficult because complexities are multiplied by all the distributed sys-tem components that interact with the application. When we see an error in a Web envi-ronment, it's often difficult to pinpoint where the error occurs, and, because the behavior we see or the error message we receive may be the result of errors happening on differ-ent parts of the Web system, the error may be difficult to reproduce.

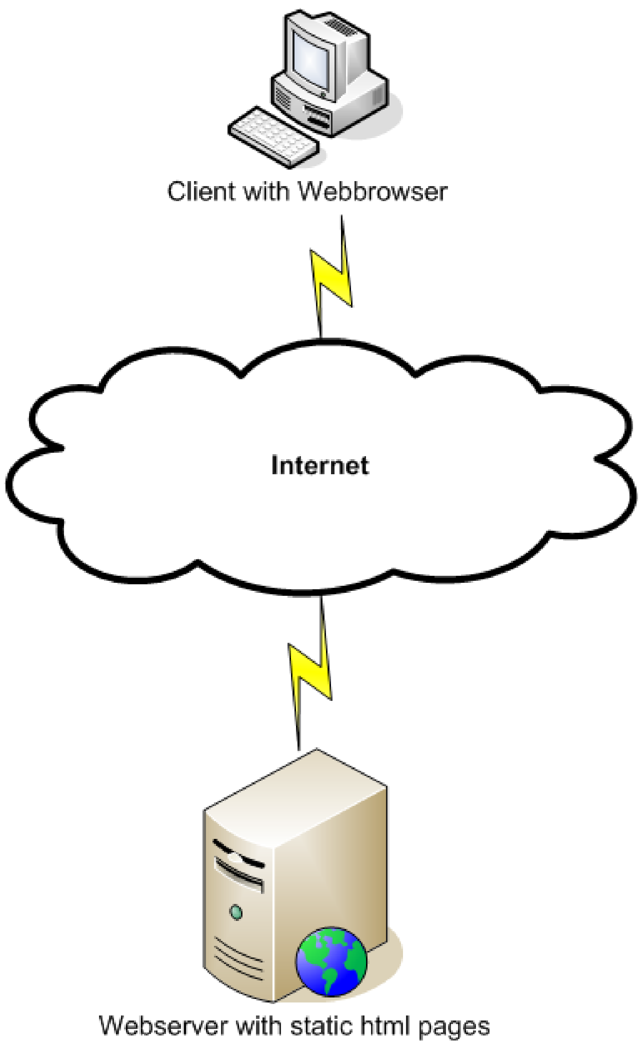


Figure 2.3 simple web environment

Figure 2.3 shows a client running a web browser gathering static html pages from a single server over the internet. When an error occurs there are different components that could have caused the error:

Client:

- the configuration of the browser
- the operating system of the client

Internet:

- routing problems

Server:

- internal error
- page not found

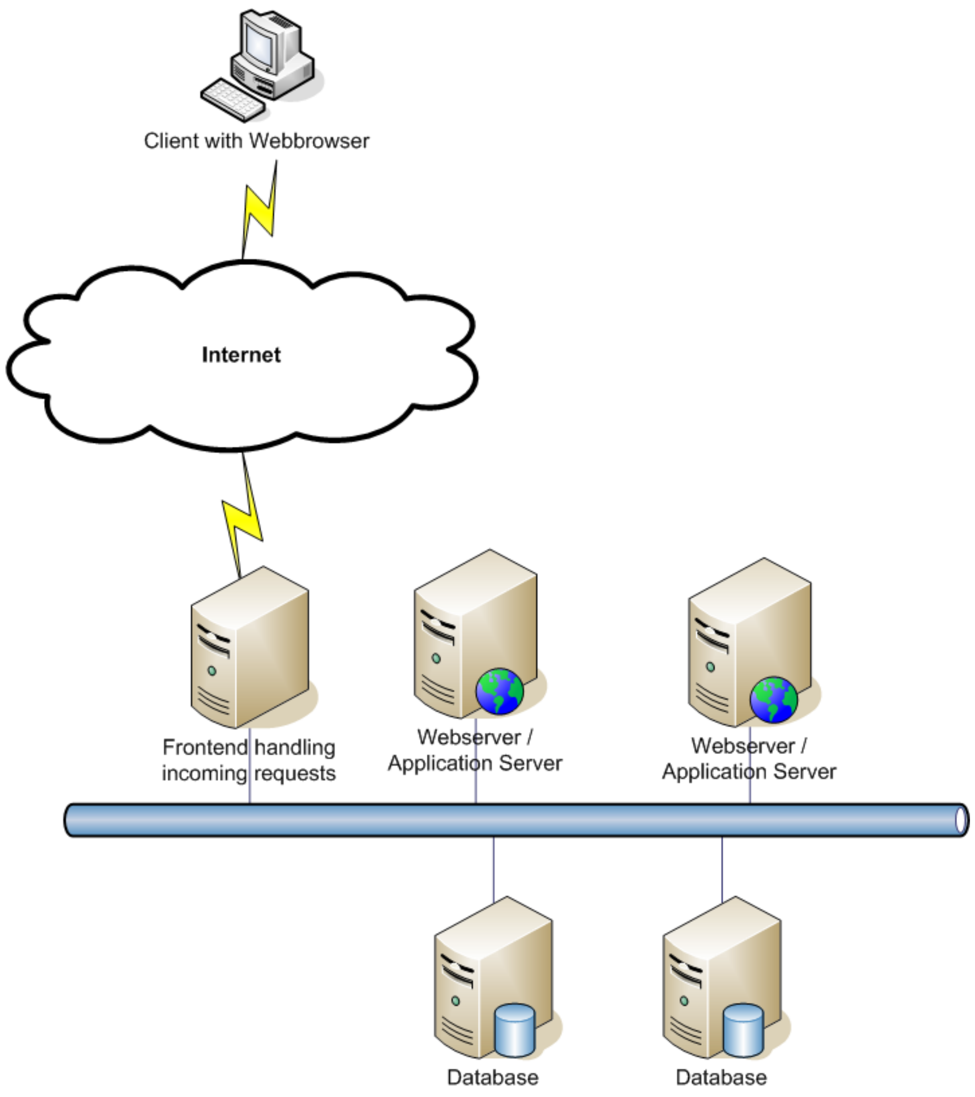

Client with Webbrowser
Internet
Frontend handling incoming requests
Webserver / Application Server
Webserver / Application Server
Database
Database

Figure 2.4 complex web environment

Figure 2.4 shows a more complex web configuration. There are several redundant web servers and database servers and an additional server for handling incoming requests. When an error occurs the reason could be one of the reasons mentioned before or defects in the server network or one of the servers.
When testing web applications it is very important to distinguish between [22]:

- Server programs with state-independent behavior
- Server programs with state-dependent behavior

Web applications are software projects with the following characteristics [25]:

- Web applications have short delivery times.
- Web applications are subject to a tremendous pressure for change.
- Turn over of Web application developers is high.
- Complexity and criticality of Web applications are increasing.
- User needs evolve quickly.

Here are five fundamental considerations of web application testing: [4]

1. **Is it an error or a symptom?**

Without diagnosing the environment, we can’t be certain what causes a symptom to appear. If one of the environment-specific variables from either the client side or the server side is removed or altered, we might not be able to reproduce the problem.

2. **Is the error environment-dependent?**

To reproduce an environment-dependent error we have to perfectly replicate both the exact sequence of activities and the environment conditions (operating system, browser version, add-on components, database server, Web server, third-party components, server/client resources, network bandwidth and traffic, etc.) in which the application operates

Environment independent errors, on the other hand, are relatively easier to reproduce it’s not necessary to replicate the operating environment. With environment-independent errors, all that need be replicated is the steps that reveal the error. More commonly, we refer to environment-independent errors as functionality-specific errors.

3. **Is it a coding error or a configuration problem?**

Errors may be resolved with code fixes or system reconfiguration. Don’t jump too quickly to the conclusion that it’s a bug!

4. **Which layer really causes the problem?**

Errors in Web systems are often difficult to consistently reproduce because of the many variables introduced by the distributed nature of client/server architecture. There are at least three usual suspects in a Web environment: The client, the server, and the network. Both the client and the server carry configuration and compatibility issues that are similar to PC environments, where all components are in one box. Issues multiply within client/server systems, however, because there may be many clients and servers connected on a network.

The network offers another set of variables. The network affects the Web application in several ways, including timing-related issues (race conditions, performance, time-outs, etc.) due to bandwidth and latency, potential configuration and compatibility issues due to hardware devices such as gateways and routers, and side effects related to security implementations.

5. **Static and dynamic operating environments are different.**

In general, there are two classes of operating environments. Each with its own unique testing implications:

Static Environments (i.e., configuration and compatibility errors) in which incompatibility issues may exist regardless of variable conditions such as processing speed and available memory.
Dynamic Environments (i.e., resource and time-related errors) in which otherwise compatible components may exhibit errors due to memory-related errors and latency conditions.

2.2.1.1 Testing the application

There are two different options to write test cases for a web application [20] [21]:

- **Program-Based Testing**

The web application is splitted into its technical parts and each part gets tested seperately. Afterwards some tests have to be written using interactions of these parts.

- **User-Session-Based Testing**

A user session is a sequence of user requests, which can also be seen as http requests.

Basically both forms should be used and combined as described in [24].

2.2.1.2 Client side testing [3] [15]

- **HTML Coding**

The HTML of a web page should be standard HTML or XHTML as specified by the World Wide Web Consortium (W3C). The W3C provides online tools that can check and validate web pages. Browser specific extensions shouldn’t be used because they may cause severe side effects on other browsers and they also fail the validation. There are also offline tools that are able to validate web pages and that can be used for automated software tests.

- **Design testing**

Design includes the layout and also graphical elements of the site. The design can hardly be tested by automated tests. It is also dependant on the rendering engine of the browser. One further aspect of design is the ability to print out information from the page. Many web pages provide extra links that render a more printer friendly page.

- **Usability testing**

Usability is tightly coupled with the design of a web page. Usability can only be tested by humans. There are many options to test the usability but I only want to present the usability testing with prototypes to give a basic idea how to do such tests. Prototypes can be non functional html pages or just made of paper. The test supervisor has to specify some scenarios and observe the test candidates performing these scenarios on the prototype. Together with a closing interview these usability tests can give some great feedback by identifying inconsistencies.

- **Performance testing**

Performance testing [11] can be divided into two groups:

1. **Qualitative:**

Many generated web pages include lots of tags which raises the file size and accordingly also the internet traffic. Web browsers need more time to render big pages. However this issue becomes unimportant nowadays because the CPU speed of modern computers is far beyond the needed speed for rendering internet pages and people use broadband internet connections instead of modems. Automatic software tests can measure the time that is needed to receive and render a page.

2. **Quantitative:**

Part of server side testing.

- **Client-side script testing**

Many internet pages use client-side scripts like VBScript or JavaScript. When such scripts cause an error usually a message pop up appears and prints out an error message.

There are three types of script problems:

1. **compilation errors:**

They occur as soon as a web page is loaded.

2. **runtime errors:**

They occur as soon as some action is performed which executes the bad script.

3. **logic errors:**

When a user enters an unexpected value like a string instead of a number the function causes a logical error.

Script testing can be done in automated software tests, as long as the test framework supports the scripts.

2.2.1.3 Server side testing

- **Performance testing**

Quantitative:

Many clients connect to one server. This can be done by many people testing concurrently one web application or it can be done by automated software tests, which should be preferred because automated tests can be repeated more easily with the same configurations while coordinating a great many of tester is nearly impossible.

- **Black box testing**

The client performs several tests. It sends requests to the server and inspects the results. Black box tests are the basic tests when writing automated tests for web applications.

- **Reliability / stability testing**

Reliability or stability can be quantified in the uptime of the server under certain conditions. First a reasonable number of users, the system should be able to handle, has to be specified and then some kind of stress test has to be executed. The uptime of a system normally is measured in percentage of uptime over a year. An uptime of 99.9% allows a downtime of less than 9 hours per year. It is important to run these tests over a long period of time under real world conditions but it is hard to simulate real world conditions in a lab and it is unrealistic to be able to perform a stability test even one month long before release. Therefore this kind of testing has to be mapped to a short period automated stress test which is performed in a lab. However the final results can't be obtained until the system has been released and has run a long time.

- **Scalability testing**

Scalability testing occurs on a complete system rather than on a single server. Not all applications scale as expected. The behavior of the system before and after scaling is tested by automated tests.
There are two ways that a system can scale:

1. **scale up:**

This means that changes in the current machine are done like upgrading the CPU or adding more memory. This type of scaling doesn't add any redundancy to the system.

2. **scale out:**

New machines are added to offload the processing. This type of scaling provides some redundancy but on the other hand it can cause additional failures because it is difficult to share the state across different machines.

- **Security testing**

Web applications can be accessed by a large number or users, mostly by anyone who is connected to the internet. That's why security testing is a very important part in web testing. Security vulnerabilities can have two origins, they can be caused by architectural problems concerning the servers on which the application is deployed or they can be caused by coding problems. There are special security scanners that are able to scan the system for known bugs like nessus [47] but they do not cover all architectural issues. Vulnerabilities caused by coding problems can be found by testing the application concentrating on the parts that are relevant for security. Most security holes are caused by typical coding mistakes and in web applications there are several additional kinds of security holes:

- unchecked input data
- SQL Injections

- Cross-Site Scripting
- Denial of Service
- Encryption related issues

tests	manual	automatic
HTML coding	no	yes
design	yes	no
usability	yes	no
performance - qualitative	yes	yes
performance - quantitative	no	yes
script	yes	yes
black box	yes	yes
reliability	no	partly
scalability	no	yes
security	yes	yes

Table 2.2 server side testing - summary

2.2.2 AJAX (Asynchronous JavaScript and XML)

AJAX [48] stands for "asynchronous JavaScript and XML" and is a collection of web development techniques used to create interactive web applications. With AJAX the web application can retrieve data from the server asynchrony in the background while the display of the existing page isn't interfered.

The main technologies used in AJAX are [2] [19]:

- XHTML and CSS for presentation of the data
- Document Object Model for dynamically modifying the web page
- JavaScript for interacting with all other techniques
- XML and XSLT for data exchange with the server
- XMLHttpRequest object for asynchronous communication

However it has been noted that some of these technologies can be replaced by others. It isn't necessary to use XML for the data exchange, so many applications use plain text, preformatted HTML or JavaScript Object Notation as an alternative format. Instead of JavaScript developers can use VBScript.

AJAX modifies the communication between client and server significantly. Without AJAX after each request to the server, a whole new web page is sent as response, or when using frames the whole content of at least one frame is sent. The user can’t work on the page until the new site has been received. On the other side AJAX requests are processed in the background and the response from the server is added to the page dynamically. No new page is loaded and the user can keep working on the page and trigger more AJAX requests while everything is processed in the background.

AJAX has many advantages compared to the classic web applications using just request/response but it also causes some problems [40] :
Advantages:

- AJAX reduces the amount of data that is transferred between server and client it doesn’t send the whole page but only the necessary parts that change.
- AJAX is supported by all popular browsers without the need to install a plug-in
- Using AJAX makes the user interface more interactive. It is possible to develop web based office applications that behave very similar to locally installed ones or to check input data of a form while the user still fills it and mark mistakes.
- There are no license costs for AJAX
- AJAX uses no new technologies so there is no need to learn additional programming languages for web developers.

Disadvantages:

- The back button of the browser doesn’t work with AJAX because no new page has been loaded.
- It is difficult or impossible to bookmark a certain state of a web page when using AJAX.
- JavaScript has to be enabled which could lead to other security problems.
- Search engines normally don’t execute JavaScript code so they can’t index the parts of a web application that can only be reached using AJAX.
- AJAX complicates web applications and raises the chance for errors.
- Many web testing tools can’t handle AJAX and therefore can’t be used for testing AJAX enabled web applications.

2.2.2.1 XMLHttpRequest

The most important part of AJAX is the XMLHttpRequest [39] which is responsible for the communication and which collects the response data from the server. Instead of AJAX request we can also use the term XMLHttpRequest.

2.2.2.1.1 *Methods*

abort()

Cancels the current request.

getAllResponseHeaders()

Returns the complete set of HTTP headers as a string.

getResponseHeader(headerName)

Returns the value of the specified HTTP header.

open(method, URL, async, userName, password)

Specifies the method, URL, and other optional attributes of a request.

- The method parameter can have a value of GET, POST, HEAD, PUT, DELETE, or a variety of other HTTP methods listed in the W3C specification.
- The URL parameter may be either a relative or complete URL.
- The async parameter specifies whether the request should be handled asynchronously or not – true means that script processing carries on after the send() method, without waiting for a response, and false means that the script waits for a response before continuing script processing.

send(content)

Sends the request. content can be a string or reference to a document.

setRequestHeader(label, value)

Adds a label/value pair to the HTTP header to be sent.

2.2.2.1.2 *Properties*

onreadystatechange

Specifies a reference to an event handler for an event that fires at every state change

readyState

Returns the state of the object as follows:

- 0 = unsent – open() has not yet been called.
- 1 = open – send() has not yet been called.
- 2 = headers received – send() has been called, headers and status are available.
- 3 = loading – Downloading, responseText holds partial data (although this functionality is not available in IE)
- 4 = done – Finished.

responseText

Returns the response as a string.

responseXML

Returns the response as XML. This property returns an XML document object, which can be examined and parsed using W3C DOM node tree methods and properties.

status

Returns the HTTP status code as a number (e.g. 404 for "Not Found" and 200 for "OK").

statusText

Returns the status as a string (e.g. "Not Found" or "OK").

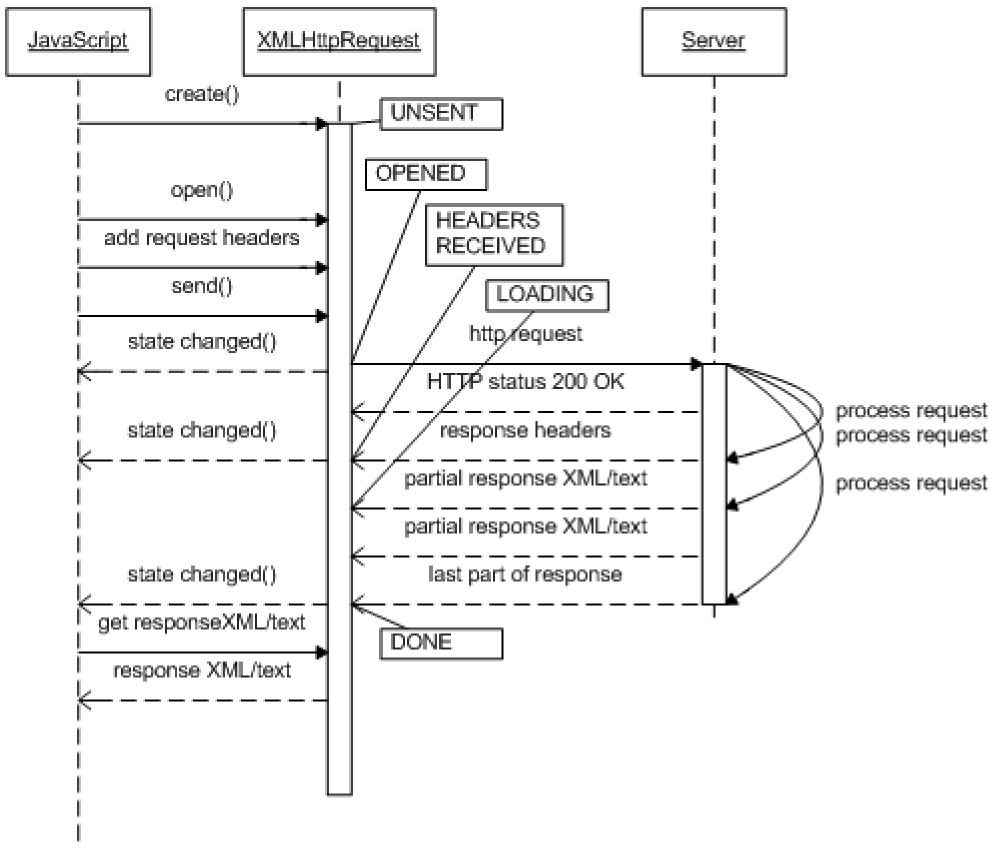


Figure 2.5 ready states of the XMLHttpRequest object

The XMLHttpRequest has to be in one of the following five states:

- Unsent or 0
- Opened or 1
- Headers received or 2
- Loading or 3
- Done or 4

When an XMLHttpRequest object is created the initial state is “unsent”. When the open() method is called the state is changed to “opened”. In the open method the user can specify whether the request should be executed synchrony or asynchrony. Synchrony requests will block the current thread, mostly the main thread, until the full response has been received while asynchrony requests are executed in the background in an own thread so the user can continue working with the application and initiate further re-

quests. In this state the request headers can be set using the setRequestHeader method and finally the send() method is called to send the request to the server.
Once all response headers have been received the status is changed to “headers received” and later when the first byte of the response entity body has been received or when there is no response entity body the status is switched to “loading”.
Finally when the request has successfully completed the status is switched to “done”.
After each status switch a readystatechange-event is dispatched which notifies the associated function which is responsible for processing the received information. Most applications just wait till the “done” status has been fired and process the whole data at once, which is adequate for small data amounts, but especially for large amounts of data it is advisable to show the user some kind of progress bar or even present the data which has already been received.
The response entity body can be read with the method responseText or responseXML. ResponseXML can only be called when the status is “done” because otherwise the resulting fragment xml document wouldn’t be a well-formed xml document while responseText can also be called when the state is “loading”. As the response entity body doesn’t need to be an xml document the responseXML method may also return a null string.

2.2.2.2 AJAX Example

Here is an example code from Wikipedia [44]

```
/*
This is the JavaScript file for the AJAX Suggest Tutorial

You may use this code in your own projects as long as this
copyright is left in place. All code is provided AS-IS.
This code is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

 For the rest of the code visit http://www.DynamicAJAX.com

 Copyright 2006 Ryan Smith / 345 Technical / 345 Group.

*/
//Gets the browser specific XmlHttpRequest Object
function getXmlHttpRequestObject() {
 if (window.XMLHttpRequest) {
 return new XMLHttpRequest();
 } else if(window.ActiveXObject) {
 return new ActiveXObject("Microsoft.XMLHTTP");
 } else {
 alert("Ihr Browser ist schon etwas aelter.\nEventuell sollten Sie
     einmal ueber ein Update nachdenken...");
 }
}

//Our XmlHttpRequest object to get the auto suggest
var searchReq = getXmlHttpRequestObject();
var searchLang = "de";

//Called from keyup on the search textbox.
//Starts the AJAX request.
function searchSuggest(lang) {
 searchLang = lang;
 if (searchReq.readyState == 4 || searchReq.readyState == 0) {
 var str = escape(document.getElementById('txtSearch').value);
 searchReq.open("GET", 'suggest.php?lang=' + searchLang +
     '&search=' + str, true);
 searchReq.onreadystatechange = handleSearchSuggest;
 searchReq.send(null);
 }
}

//Called when the AJAX response is returned.
function handleSearchSuggest() {
 if (searchReq.readyState == 4) {
 var ss = document.getElementById('search_suggest')
 document.getElementById('search_suggest').style.visibility =
     "visible";
 ss.innerHTML = '';
 //ss.innerHTML += '<a href=""
     onclick="document.getElementById(\'search_suggest\').style.vi
     sibility = \'visible\'; return false;">[x]</a>';
 var str = searchReq.responseText.split("\n");
 if (str.length > 1) {
 var entry = str[0].split("\t");
 ss.innerHTML += '<div class="suggest_link">Treffer f&uuml;r "' +
     entry[0] + '"</div>';
 var searchString = entry[0];

 for(i=1; i < str.length - 1; i++) {
 var entry = str[i].split("\t");
 //alert (entry);
 //Build our element string. This is cleaner using the DOM, but
 //IE doesn't support dynamically added attributes.
 var suggest = '<div onmouseover="javascript:suggestOver(this);"
';
 suggest += 'onmouseout="javascript:suggestOut(this);" ';
 //suggest += 'onclick="javascript:setSearch(this.innerHTML);" ';
 suggest += 'class="suggest_link"><a href="/go?l='+ searchLang
     +'&q=' + entry[0] + '">';

 if (document.getElementById('txtSearch').value == entry[0]) {
 suggest += '<b>' + entry[0] + '</b>';
 } else {
 suggest += entry[0];
 }

 suggest += '</a> ';
 //suggest += entry[1];
 suggest += '</div>';
 // suggest += '</a> ' + entry[1] + '</div>';
 ss.innerHTML += suggest;
 }
 ss.innerHTML += '<hr noshade size=1 style="background-
     color:#ffffff;">';
 }

80   if (searchString) {
81
82   ss.innerHTML += '<div class="suggest_link">"' + searchString + '"
         in Wikipedia suchen mit</div>';
83
84   var suggest = '<div onmouseover="javascript:suggestOver(this);"
';
85   suggest += 'onmouseout="javascript:suggestOut(this);" ';
86   suggest += 'class="suggest_link"><img src="/favicon.ico"
         width="16" height="16" title="Suchen mit Wikipedia"
         border="0" >&nbsp;&nbsp;<a href="/go?l=' + searchLang +
         '&e=wikipedia&s=search&q=' + searchString +
         '">wikipedia.org</a></div>';
87   ss.innerHTML += suggest;
88
89   if (searchLang == "de") {
90
91   var suggest = '<div onmouseover="javascript:suggestOver(this);"
';
92   suggest += 'onmouseout="javascript:suggestOut(this);" ';
93   suggest += 'class="suggest_link"><img src="/img/t-online.ico"
         width="16" height="16" title="Suchen mit T-Online" border="0"
         />&nbsp;&nbsp;<a href="/go?l=' + searchLang + '&q=' +
         searchString + '&e=t-online&s=search">t-online.de</a></div>';
94   ss.innerHTML += suggest;
95
96   var suggest = '<div onmouseover="javascript:suggestOver(this);"
';
97   suggest += 'onmouseout="javascript:suggestOut(this);" ';
98   suggest += 'class="suggest_link"><img src="/img/web.de.ico"
         width="16" height="16" title="Suchen mit web.de" border="0"
         />&nbsp;&nbsp;<a href="/go?l=' + searchLang + '&q=' +
         searchString + '&e=web.de&s=search">web.de</a></div>';
99   ss.innerHTML += suggest;
100
101   var suggest = '<div onmouseover="javascript:suggestOver(this);"
';
102   suggest += 'onmouseout="javascript:suggestOut(this);" ';
103   suggest += 'class="suggest_link"><img src="/img/exalead.ico"
         width="16" height="16" title="Suchen mit exalead" border="0"
         />&nbsp;&nbsp;<a href="/go?l=' + searchLang + '&q=' +
         searchString + '&e=exalead&s=search">exalead.de</a></div>';
104   ss.innerHTML += suggest;
105   }
106
107   }
108
109
110
111   }
112  }
```

Table 2.3 Wikipedia search suggest

This script suggests Wikipedia articles matching the user input in the search field.
The XMLHttpRequest object is created by the function getXmlHttpRequestObject() on line 16 and stored in the variable searchReq on line 27. As there is only one request object, Wikipedia can handle only one AJAX request at once.

After each key press from the user the function searchSuggest on line 32 is called. On line 34 the script checks whether the ready state of searchReq is “done” or “unsent” because any other state would indicate that there is already a running search and the current search would be cancelled. This means that the suggest search won’t be really executed after each key press as long as the response of the pending search request hasn’t been received.

If the searchReq object is ready for a new search request the open method is called on line 36 where the http method, the URL with the search string and language parameters and a Boolean variable indicating to process the request asynchrony are passed to the object. Further request headers aren’t needed so the next step is to specify the function handleSearchRequest() to be called after each state change. The function itself is specified on line 43. Now the request is sent with “null” as content because the search string is embedded in the URL.

The function handleSearchRequest() only processes the received data when the ready state has been set to “done” (line 44). All other state changes are ignored.

From line 45 to 47 the prepared element on the webpage which presents the search results is set visible and old search results are deleted.

On line 49 the search result is read using the method responseText() and afterwards the results get surrounded by some html tags and presented on the page.

2.3 Selected Open Source Testing Frameworks

2.3.1 HtmlUnit

HtmlUnit [41] is a web browser written in java which allows manipulating web pages. It was designed for test automation of web applications. It doesn’t have a graphical user interface so the user can’t observe the program while executing the tests. HtmlUnit isn’t a test framework itself but it is intended to use HtmlUnit within a test framework like JUnit or TestNG.

The key features of HtmlUnit are

- Support for the HTTP and HTTPS protocols.
- Support for cookies
- Ability to specify whether failing responses from the server should throw exceptions or should be returned as pages of the appropriate type
- Support for submit methods POST and GET (as well as HEAD, DELETE, ...)
- Ability to customize the request headers being sent to the server
- Support for HTML responses
- Wrapper for HTML pages that provides easy access to all information contained inside them

- Support for submitting forms
- Support for clicking links
- Support for walking the DOM model of the HTML document
- Proxy server support
- Support for basic and NTLM authentication
- Good support for JavaScript

This means that HtmlUnit supports all “normal” web pages, but there is no support for further plug-ins like Adobe Flash or Microsoft Silverlight.
HtmlUnit is not a stand-alone application, but it depends on many other projects:

- JUnit: is used as test framework
- Mozilla: the Rhino JavaScript engine for java has been adopted for HtmlUnit
- Apache: provides the http-client and many utility classes
- Xerces: is used as XML parser
- Many others

The central class in HtmlUnit is the WebClient. It can be instantiated with arguments specifying a proxy server and a specific browser version. Supported browsers are Internet Explorer, Netscape and Mozilla, but this list can be easily extended. The default browser is Internet Explorer 6.0.
The web-client further contains:

- WebConnection
- JavaScriptEngine
- CredentialProvider
- Some JavaScript handlers
- Some internet settings

The web-client provides many methods to let the user interact like in a real browser. When the user wants to load a page he just has to call the method getpage(). HtmlUnit provides for each html-tag an equivalent class which offers all setter methods, getter methods and interactions the tag offers. The user has to cast the return values into the right class explicitly. Elements of a web page can be found and extracted by searching for them using their ids or names or by getting a collection of all elements of a certain type and taking the one in the right position.

Here is a simple test example taken from the HtmlUnit homepage:

```
public void testHomePage() throws Exception {
    final WebClient webClient = new WebClient();
    final         HtmlPage         page          =          (HtmlPage)
              webClient.getPage("http://htmlunit.sourceforge.net");
    assertEquals("HtmlUnit      -      Welcome      to      HtmlUnit",
page.getTitleText());
}
```

Table 2.4 HtmlUnit sample test

First a default web-client is created. This means that this web-client behaves like Internet Explorer 6.0.
The method getPage() is called with the URL of the desired web page as argument and the result is casted to an HtmlPage object.
At the end there is a simple assertion using a JUnit method whether the title of the page is the expected one.

HtmlUnit doesn’t have a graphical user interface which makes it very difficult or impossible to observe the tests. It isn’t possible to make assertions about the layout of a web page, only the document object model and the data can be verified. On the other hand HtmlUnit consumes hardly computer resources. It is possible to run many instances of this program in parallel and with just a little effort it is possible to use the test-cases written in HtmlUnit for performance tests, which is also very important for internet applications. Many other test frameworks for web pages provide the user a graphical user interface and just record the user interaction. They record the coordinates of the mouse pointer and just do a replay when they execute the tests. This seems very easy and useful at first sight because it is very fast and easy to record such tests and the person who records the tests only needs little knowledge about the structure of the web page or the application as far as he gets a sheet with the instructions of the test cases he has to implement. In HtmlUnit or other test frameworks without a graphical user interface it takes more time to implement these test cases and the person implementing the test cases has to study the html code of the web page and find the right ids and names of the html elements he needs to access. On the other hand these test cases are more likely to work after changes in the application because HtmlUnit doesn’t care about the position of text-fields or command-buttons, it only needs to know their names or ids to address them and to execute the test case the way they have been implemented. Test frameworks that just replay the recorded coordinates and clicks will fail as soon as the field they want to access moves by a few pixels.
As automatic tests are normally done for incremental testing for continuous builds of applications it is very important that they work as long as possible or even forever after they were implemented because automatic tests should support software testers by taking away some tasks from the testers but as soon as they have to be corrected or even rewritten by the testers too often there would be no benefit by investing in automatic tests.
The conclusion of this paragraph is that it is more important to implement robust automatic tests which may be quite a lot of work and consume some time but will work for a long time instead of using “fast” techniques that are very likely to fail soon because of further development and changes in the design of the internet application. First it seems

that it isn’t too comfortable to use HtmlUnit but after implementing the first few test-cases you will become familiar with the program and you will value the program.

HtmlUnit is an open source project which is released under the Apache 2 license [49] and there are many developers and contributors fixing bugs, extending the applicability of HtmlUnit and answering technical questions which are asked in the mailing list.

HtmlUnit offers handlers for the following protocols:

- JavaScript
- About (only about:blank is supported yet)
- File
- Data (only in Netscape mode)
- All other prefixes are handled like web-requests (http, https)

Requests are stored in objects of the type RequestSettings and before sending the request a WebResponse object is created to store the response data. The web-connection object is called to handle the request and receiving the response. The web-connection uses the web-client from the apache project for low level communications.
HtmlUnit first checks the status code of the response and either throws an exception because something went wrong, resends the request to another URL because the location of the requested page has changed or processes the response and presents the new page to the user. Redirection is handled without notifying the web-client like in real web browsers and the result is always a web page with an error code or the requested page. Everything is done in a single thread so it is quite easy for the user to find the right position in the test case for making the assertions.

2.3.1.1 HtmlUnit and AJAX

Processing AJAX requests is a bit more complicated. First the user activates a java-script function and therefore the java-script engine has to be activated. Then it depends on the script how the program proceeds. The function could just be a local JavaScript function which calculates the result of a calculation and writes the result into a field but the function could also create an XMLHttpRequest object and send an AJAX request to the server. These requests are handled in their own thread so the main thread with the web-client can continue. This is the point where software testing becomes tricky because it is impossible to forecast how much time it will take till the AJAX request is finished. Assertions can only be done in the main thread but there is no notification from HtmlUnit that the child thread has finished so the tester doesn’t know when to check whether the result is as expected. It is even possible that the main thread of a test case finishes before the child thread of the asynchronous request has finished and possible errors in the child case will never show up.

There is a workaround [50] to do a timeout for a few seconds in the main thread and do the assertions afterwards, but this is still no assurance that the AJAX thread has been finished till then and the workaround also prolongates the execution time of the tests because these timeouts stop the main thread longer than necessary.
Another workaround is very similar to the one above. Its idea is to stop the main thread only for a short time and to check whether there exist certain elements that should be created by the AJAX call. If these elements don't exist it can be assumed that the call hasn't finished yet and the main thread again stops for a short time and then rechecks for these elements until they finally appear. This should indicate that the AJAX call has finished and all wanted assertions can be done now.
This workaround solves the main problems from the first one:

- The main thread stops exactly as long as necessary
- Execution time is faster

On the other hand the tester has to find appropriate indications in his program that reliably show the end of the call. It depends on the particular java-script function whether it is easy to find or not. Most AJAX calls create elements to present their results but when a certain function is called several times with different arguments these elements might stay visible and only the content of these elements is changed, so it is no use to wait for these elements to appear because they already exist.
The third workaround pursues the same strategy but it uses a more technical approach. The idea is to resynchronize the asynchronous AJAX calls. HtmlUnit has already implemented this solution. There is a class called NicelyResynchronizeAjaxController [50] which extends the class AjaxController. AjaxController is just a dummy class which is activated by default but doesn't change anything in the operation sequence. NicelyResynchronizeAjaxController can be initialized by the tester and passed to the central web-client to replace the default controller. Just before creating the child process for the AJAX call the new controller checks whether the request was performed from the main thread and just executes the call in the main thread instead of creating it's own one. This means that the main thread can not proceed till the AJAX call has finished. This technique has the big advantage that the user need not check for an indicator, which signals the end of the AJAX call, to appear because the main thread won't proceed the execution till the call has finished. Technically HtmlUnit tries to use the open method of the XMLHttpRequest object with the parameter "synchronous" instead of "asynchronous".
However the implementation of resynchronizing AJAX calls in HtmlUnit has some limitations. The AJAX controller isn't called until an XMLHttpRequest object is created and ready for being transmitted. The controller checks whether the request still takes place in the main thread and synchronizes the request but it is possible and very common to embed the AJAX request in the JavaScript function setTimeout() or setInterval() which causes the browser to create a new thread for the embedded JavaScript. In this case the built in AJAX controller of HtmlUnit won't be able to synchronize the request

because it is already running in its own thread and can't be controlled by the main thread that easily.

- setTimeout(FunctionX(), timeout)
  This method waits a certain amount of milliseconds specified by "timeout" before the function "FunctionX()" is called. This method is often used by auto complete forms which use AJAX to suggest some matching according to the users input, for example in Wikipedia. Without that method typing the letters "xmlhtt" would cause an AJAX request for each letter, but with a small timeout of half a second the request will only be executed when the user makes a short break and a lot of unnecessary internet traffic will be saved. While filling the form with letters, each new letter aborts the before AJAX request and starts a new one until nothing is changed in the form for 500 milliseconds.
- setInterval(FunctionY(), timeout)
  This method calls the function "FunctionY()" again and again every milliseconds specified by "timeout" and returns an identifier for the interval-method which is needed to stop it by the method stopInterval(identifier).

These three workarounds can solve some issues concerning software testing of web pages with asynchronous calls but they have a big disadvantage in common. They all try to turn the multithreaded application into a single-threaded application with different approaches and it is true that they allow the tester to do assertions at calls that couldn't be done before but on the other hand they change the whole application flow. When a user triggers many different AJAX calls in a web page which take different times to be finished it is very likely that some fast calls are finished before the long time processing calls, even when they were triggered afterwards. The web-page of the internet user behaves different from the web page of the test program. It is possible that the automatic tests just work fine because the asynchronous calls were synchronized but when a user wants to work on the page with an ordinary web browser the application fails.

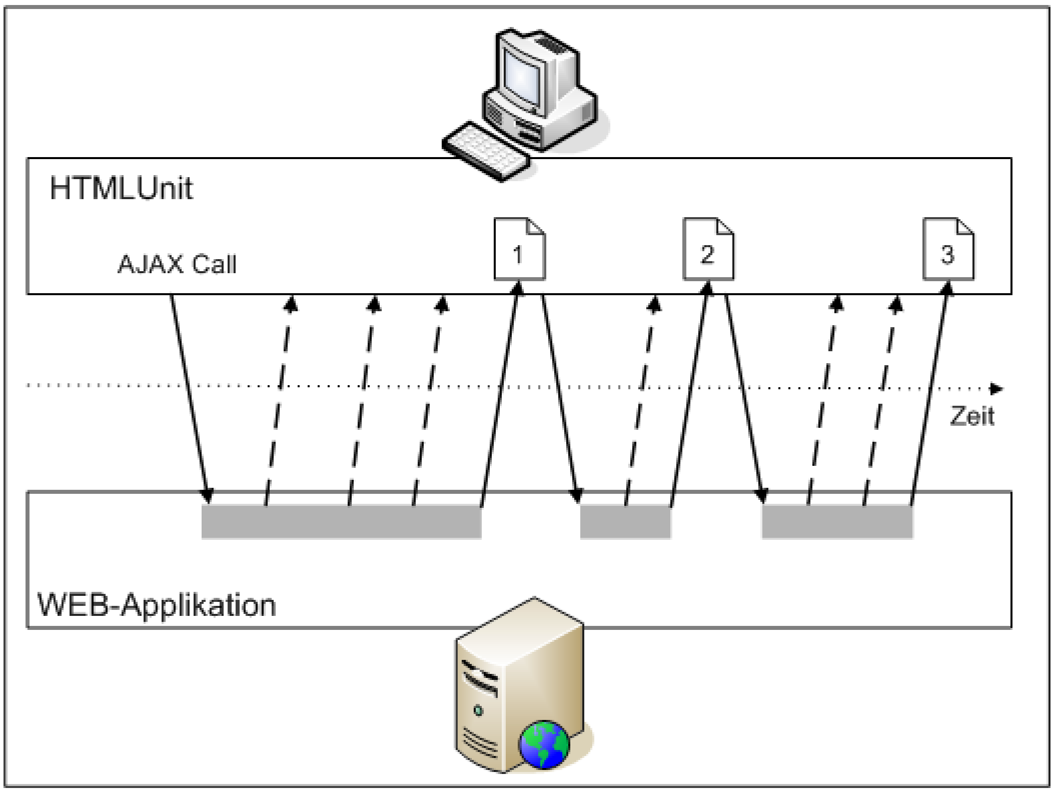


Figure 2.6 resynchronizing the AJAX calls

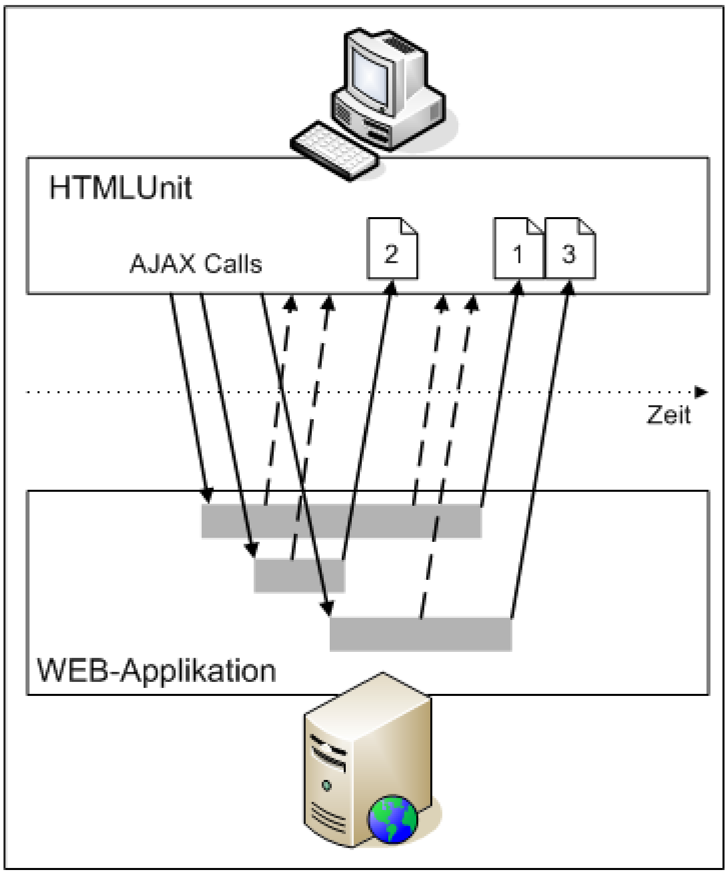


Figure 2.7 normal flow of AJAX requests

The main aim of this paper is to find a way to implement test cases for web applications using AJAX without changing the application flow and to find a way for resynchronizing all AJAX calls, even for those that are embedded in setInterval() or setTimeout() methods.

2.3.1.2 Techniques supported by HtmlUnit:

Testing techniques:

- HTML coding

HtmlUnit tests automatically whether the HTML page is well formed and provides additional assertion methods.

- Scripts

The scripting engine creates error and warning messages automatically.

- Performance – qualitative

The execution time can be measured using the standard Java tools.

- Performance – quantitative

HtmlUnit consumes hardly any resources so it is possible to run many instances simultaneously and perform load and stress tests.

- Black box

HtmlUnit is basically used to write JUnit tests.

- Reliability

Stress tests can be done to simulate long time running issues.

- Scalability

The performance tests to measure scalability can be written using HtmlUnit.

- Security

Security vulnerabilities affecting the source code of the application can be tested with black box tests.

Scripting techniques:

- Linear scripts:

HtmlUnit doesn’t support capture/replay of test cases but it is possible to write test cases instruction by instruction just like working with a common web browser.

- Structured scripts:

HtmlUnit tests are written in Java so they support many kinds of selection control structures and iteration control structures.

- Shared scripts:

Java is an object-oriented programming language so all necessary techniques are supported.

- Data-driven scripts:

See shared scripts.

- Keyword-driven scripts:

See shared scripts.

2.3.2 JSFUnit

JSFUnit [1][16] is a testing framework for JSF (Java Server Faces) applications. It is designed to allow complete integration testing and unit testing of JSF applications using a simplified API. JSFUnit tests run inside the container, which provides the developer full access to managed beans, the FacesContext, EL Expressions, and the internal JSF component tree. At the same time, you also have access to parsed HTML output of each client request.
It supports the Sun and MyFaces JSF implementations and also the Java Server Pages technology.

JSFUnit offers many ways to test JSF applications:

- Static analysis

JSFUnit can check attributes, methods and interfaces of managed beans, configuration issues and JSF tags.

- Performance analysis

It provides tools to measure the time it takes to complete for a task or a subtask

- Black box tests

It is possible to perform black bock tests from a client-centric view. JSFUnit is used with additional tools like HtmlUnit. A tester can mix up JSFUnit and HtmlUnit statements as long as the main page is initialized by JSFUnit.

- White box tests

Server based classes can be tested without running the whole application. Therefore it needs mock classes which however can change the behaviour of the whole application

- Grey Box Tests

Grey box tests are a combination of black box and white box tests. It is possible to test the client side html and the server side state at the same time.

2.3.2.1 JSFUnit and AJAX

JSF offers a rich component library for easy integration of AJAX capabilities into the applications [42][43]. This component is called RichFaces and is based on the open source framework Ajax4Jjsf. JSFUnit has its own RichFacesClient that offers to invoke

AJAX functions directly on the server. Contrary to HtmlUnit JSFUnit doesn’t directly use an XMLHttpRequest object but it allows watching all changes inside the bean.
JSFUnit won’t be discussed in this paper because it is specialized in testing only JSF application while we look for a client based test tool that can be used with any AJAX enabled technology. On the other hand it is important to know that there exist many tools for nearly every internet technology that can be used to directly test the server classes.

3 Requirements for the Framework Extension

This chapter specifies the critera for the new controller that is able to handle AJAX requests. Thereafter the already existing NicelyResynchronizingAjaxController and two new solutions are described and checked whether they satisfy all criteria.

3.1 Criteria for a satisfying controller

I worked out requirements for an AJAX-controller together with some developers of the HtmlUnit developer team.
Irrespective of the implementation the ideal AJAX-controller should satisfy following criteria:

1. It should work with all AJAX requests, independent from other JavaScript functions in which the request is embedded
2. It should be able to retrieve the status of the request any time from the test-case
3. The AJAX request should run in its own thread
4. Every XMLHttpRequest object should have its own controller
5. It should be possible to view the payload that is about to be send
6. It should be possible to modify the payload that is about to be send
7. It should be possible to view the response before it is handled by the JavaScript engine
8. It should be possible to modify the response before it is handled by the JavaScript engine
9. It should be possible to add assertions to the controller
10. It should be possible to get a callback when the request has finished

According the design of HtmlUnit there are three different ideas for implementations that handle AJAX requests:

1. NicelyResynchronizingAjaxController
2. AjaxHandler
3. AjaxFilterWebConnection

The first one is already implemented in HtmlUnit but suffers from poor support for the criteria while the other two have to be newly developed.

3.2 Proposals for solution

The first solution is just a review of the already built in AJAX controller, while the other two solutions are based on suggestions from some developers of the HtmlUnit project. I called the first new solution AjaxHandler to avoid confusions with the built in controller. The last solution is called AjaxFilterWebConnection according to the naming conventions of HtmlUnit. HtmlUnit normally uses the class HttpWebConnection which will be replaced by the enhanced AjaxFilterWebConnection.

3.2.1 NicelyResynchronizingAjaxController

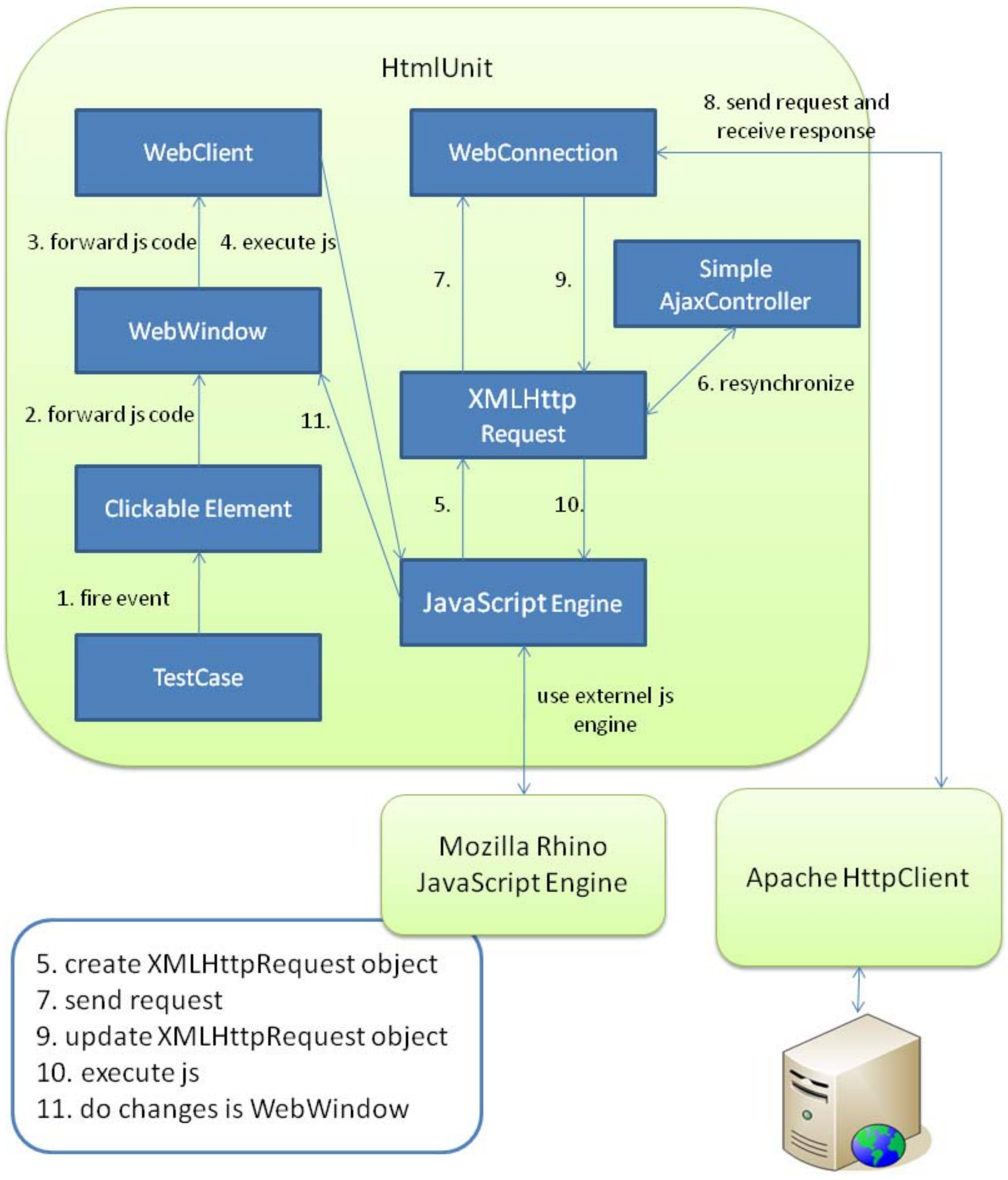


Figure 3.1 processing AJAX requests in HtmlUnit

The central class of HtmlUnit is the WebClient. It creates a default web-connection, stores all windows, pages and html-elements and provides a JavaScript engine and several parsers and listeners. When a test-case fires an event that releases an AJAX request the corresponding JavaScript code is passed to the WebClient which invokes the JavaScript engine.

The JavaScript engine uses an external engine from the Mozilla project to decode and execute the JavaScript. As soon as an XMLHttpRequest object has to be created the real

AJAX execution starts. HtmlUnit offers a simple AJAX controller called NicelyResynchronizingAjaxController which prevents the AJAX request to create its own thread. However there is no way to influence the controller and it also has some problems when the AJAX code is embedded in another JavaScript function. The XMLHttpRequest object sends the AJAX request to the web-server using the web-connection from the WebClient and the HttpClient from the apache project and receives the response. The response data is passed to the JavaScript engine which executes the corresponding function which changes the content of the web-window.
However the test-case is neither informed about the status of the AJAX communication during the whole procedure nor about the successful or unsuccessful finish of it. At least when resynchronizing of AJAX requests is enabled and possible the test-case pauses the execution till the AJAX request has finished but it is desirable to have a more reliable option to deal with such requests.

The criteria for a satisfying controller are based upon the NicelyResynchronizingAjaxController to improve handling of AJAX requests so this controller hardly fulfills any of them:

1. It doesn’t work with all AJAX requests. When the JavaScript engine creates a new thread before instancing the XMLHttpRequest object the controller can’t handle it
2. The status of the request can’t be retrieved
3. The AJAX request runs in the main thread unless the controller isn’t able to handle it
4. There is only one controller for the whole program
5. The payload can’t be viewed
6. The payload can’t be modified
7. The response can’t be viewed
8. The response can’t be modified
9. No assertions can be added
10. There is no callback when the request has finished

3.2.2 AjaxHandler

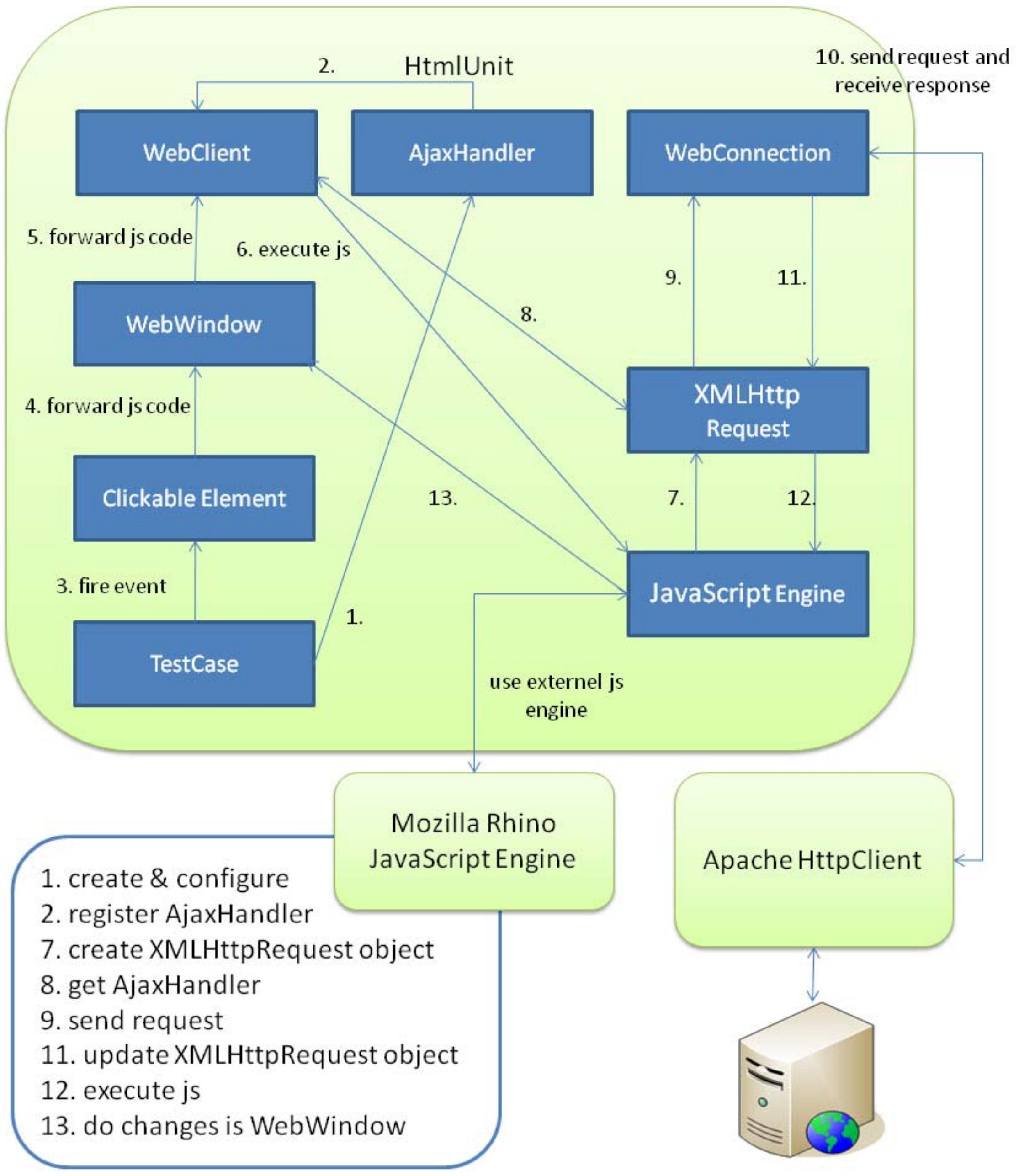


Figure 3.2 processing AJAX requests with AjaxHandler

In Figure 3.2 the simple AJAX controller is replaced by a completely different AJAX-handler. The handler satisfies all criteria mentioned above.

It is created and configured from within a test-case, so the tester just has to keep a reference to the object to be able to retrieve all data from the XMLHttpRequest object which of course first has to be connected to the handler. The tester has no direct access to the XMLHttpRequest object so he can't assign the handler to it. The solution to this problem is to invert the assign process. The XMLHttpRequest should assign to the handler as soon as it gets created but therefore it needs the handler to be registered at a place where it can easily be found. The best place is the WebClient object because there is exactly one of it in every browser instance and it already stores all html elements, listeners and parsers. So before raising the event the AJAX-handler has to be registered at the WebClient and the XMLHttpRequest class has to be modified to automatically search for the handler, register it and automatically notify the handler whenever something changes inside the class.

Only two classes have to be modified and two classes have to be created to implement the AJAX handler:

New classes:

1. **AjaxHandler**

This class has to be newly developed and needs a reference to the XMLHttpRequest object and many set and get methods for manipulating that object. In addition it needs methods that are called from the XMLHttpRequest object whenever a state change appears and that can be overwritten by the tester to add assertions directly into the handler.

2. **AjaxError**

This class stores all error messages and warnings that occurred while processing the AJAX calls. It is created automatically and called either by the AjaxHandler class or the XMLHttpRequest class. It doesn’t need any configuration thus the tester doesn’t need to worry about it but it is advisable to check its content after each test case.

Classes that have to be modified:

1. **WebClient**

The WebClient class needs additional methods to store and retrieve the handlers and errors.

2. **XMLHttpRequest**

Many methods have to be adopted. It has to look for an AJAX-handler when it gets created and it also has to notify the handler whenever its state changes

The AjaxHandler satisfies all criteria:

1. It works with all AJAX requests, because each request creates an XMLHttpRequest object which of course has to be adopted to automatically register the handler.
2. The handler is created by the tester and the tester can retrieve the status of the request any time as long as he keeps a reference to the handler object.
3. The handler doesn’t influence the workflow so the AJAX request runs in its own thread.

4. Each XMLHttpRequest object gets its own AJAX-handler as long as the tester creates and registers the handlers.

The XMLHttpRequest object always notifies the handler when something changes so it is possible to:

5. view the payload
6. modify the payload
7. view the result
8. modify the result
9. add assertions concerning payload, result or state
10. get callbacks.

3.2.3 AjaxFilterWebConnection

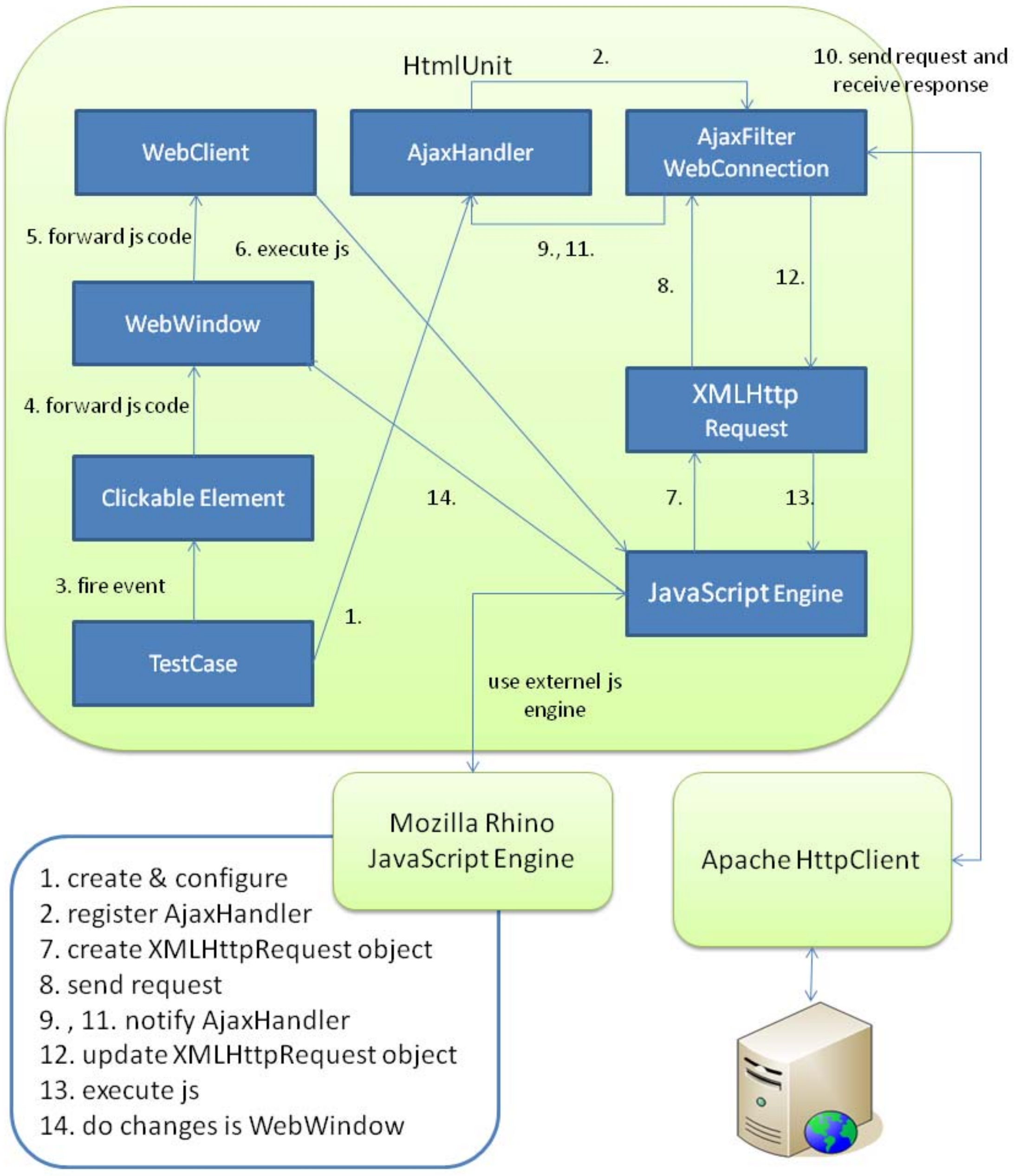


Figure 3.3 processing AJAX requests with AjaxFilter

The idea of this approach is that all communication with the web server is done using the web connection of HtmlUnit. The web connection has to be extended with a filter that filters out all AJAX traffic and reports it to a handler. As all traffic is handled by the web connection all packages have to be parsed and analyzed to distinguish normal http traffic from AJAX traffic which is a hard job to do. The handler has to unpack the payload and offer nearly the some methods as the XMLHttpRequest object with additional methods for easier manipulation of the payload.

Only one class has to be created and one has to be modified to implement the filter:
AjaxFilterWebConnection
The basic WebConnection class has to be extended. The payload of all packages has to be parsed before it gets transmitted to the server and

The AjaxFilterWebConnection satisfies nearly all criteria:

1. It works with all AJAX requests, independent from other JavaScript functions in which the request is embedded
2. It isn’t able to retrieve the state of the request any time from the test-case because the state is stored in the XMLHttpRequest object and the AjaxFilterWebConnection doesn’t keep references to these objects.
3. The AJAX request runs in its own thread
4. Every XMLHttpRequest object has its own controller
5. It is possible to view the payload that is about to be send but the request has to be recognized as an AJAX request
6. It is be possible to modify the payload that is about to be send
7. It is possible to view the response before it is handled by the JavaScript engine
8. It is possible to modify the response before it is handled by the JavaScript engine
9. It is possible to add assertions to the controller
10. It is possible to get a callback when the request has finished

3.3 Summary

#	Criteria	NicelyResynchro-nizingAjaxController	AjaxHandler	AjaxFilterWebConnection
1	works with all AJAX requests	no	yes	yes
2	retrieve status	no	yes	no
3	AJAX runs in its own thread	no	yes	yes
4	one con-troller for each ob-ject	no	yes	yes
5	view payl-oad	no	yes	partly
6	modify payload before sending	no	yes	partly
7	view re-sponse	no	yes	partly
8	modify response before further processing	no	yes	partly
9	add asser-tions	no	yes	partly
10	send call-back when finished	no	yes	partly
	ease of use	easy	easy - medium	medium - hard

Table 3.1 summary of the three solutions

The AjaxHandler seems to be the best solution for handling AJAX requests. It is nearly as simple to use as the NicelyResynchronizingAjaxController but offers a lot more setting options. The AjaxFilterWebConnection approach is the hardest to implement and doesn’t meet all criteria.

4 Implementation

This chapter shows the implementation of the already existing NicelyResynchronizingAjaxController and the implementation of the new AjaxHandler. The whole source-code is provided in the appendix.

4.1 AjaxController

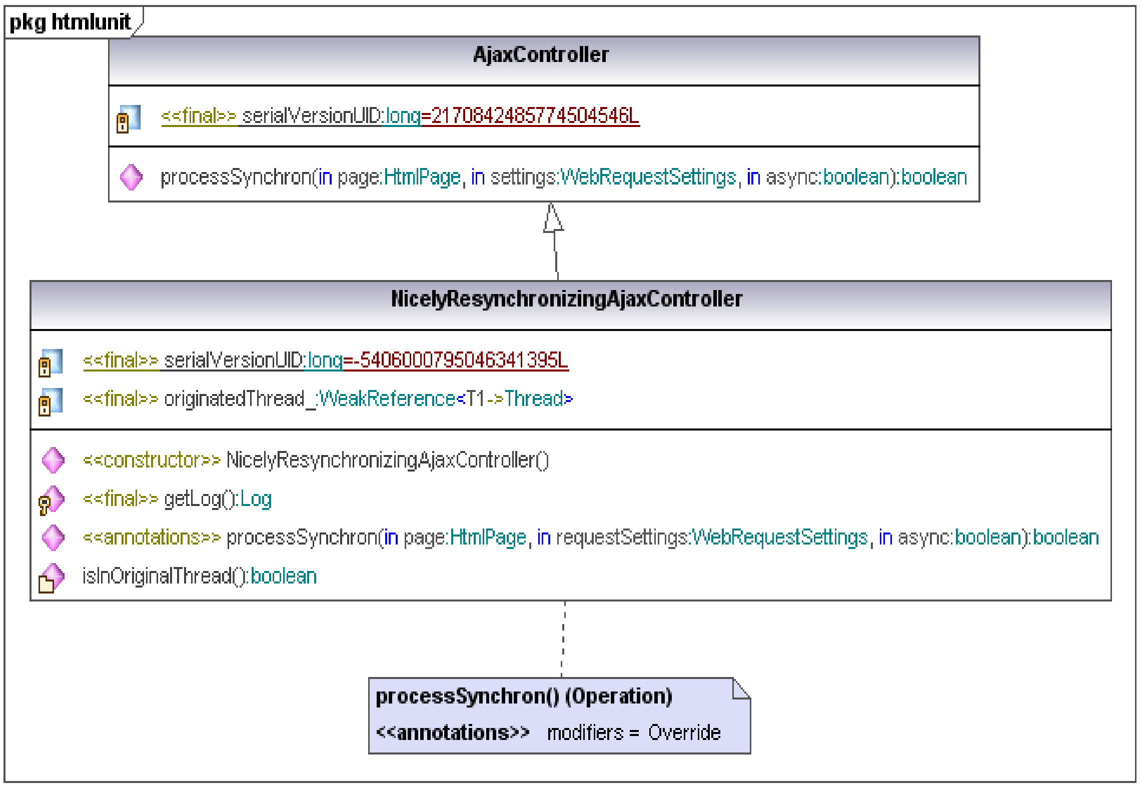

pkg htmlunit
AjaxController
<<final>> serialVersionUID:long=2170842485774504546L
processSynchron(in page:HtmlPage, in settings:WebRequestSettings, in async:boolean):boolean
NicelyResynchronizingAjaxController
<<final>> serialVersionUID:long=-5406000795046341395L
<<final>> originatedThread_:WeakReference<T1->Thread>
<<constructor>> NicelyResynchronizingAjaxController()
<<final>> getLog():Log
<<annotations>> processSynchron(in page:HtmlPage, in requestSettings:WebRequestSettings, in async:boolean):boolean
isInOriginalThread():boolean
processSynchron() (Operation)
<<annotations>> modifiers = Override

Figure 4.1 AjaxController

Every WebClient has an integrated AjaxController. It provides one dummy method “processSynchron” which always returns “false” independent of the arguments passed to that method. It is called from the XMLHttpRequest object just before sending the request to the server to determine whether to try to resynchronize the request.
The standard controller is created automatically but the tester can exchange it with the NicelyResynchronizingAjaxController which extends the basic controller. It keeps a weak reference to the thread where the test case is running in. The method “processSynchron” compares the stored thread to the thread of the calling object and returns “true” when they are the same to signal that resynchronization is possible.
There is no possibility for further adjustments of the controller. As soon as it gets activated the controller is used for every XMLHttpRequest object but it the tester can replace it with the default dummy controller any time to deactivate it.

4.2 *AjaxHandler Implementations*

4.2.1 AjaxHandler

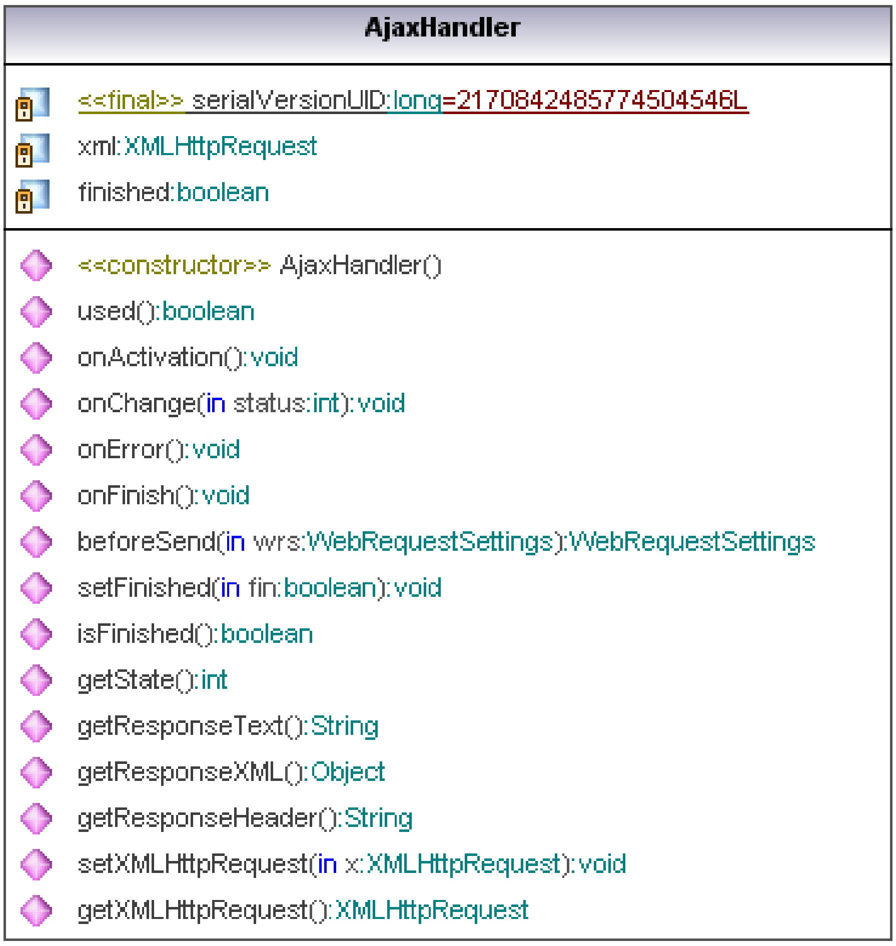

AjaxHandler
<<final>> serialVersionUID:long=2170842485774504546L
xml:XMLHttpRequest
finished:boolean
<<constructor>> AjaxHandler()
used():boolean
onActivation():void
onChange(in status:int):void
onError():void
onFinish():void
beforeSend(in wrs:WebRequestSettings):WebRequestSettings
setFinished(in fin:boolean):void
isFinished():boolean
getState():int
getResponseText():String
getResponseXML():Object
getResponseHeader():String
setXMLHttpRequest(in x:XMLHttpRequest):void
getXMLHttpRequest():XMLHttpRequest

Figure 4.2 AjaxHandler

The AjaxHandler is created by the tester and registered in the WebClient. An XMLHttpRequest object gets the handler from the web-client and registers itself using “setXMLHttpRequest()”. There methods of the AjaxHandler can be divided into three groups:
Methods for receiving information from the associated XMLHttpRequest(XHR) object

- getState(): returns the current state of the XHR. Possible values are zero to four representing the states unsent, opened, headers-received, loading and done.
- getResponseText(): returns the content of the response as a string object
- getResponseXML(): returns an XML document containing the parsed content of the response
- getResponseHeader(): returns the header information of the response
- getXMLHttpRequest(): returns the whole XHR object

Methods for showing the internal status

- used(): shows whether the handler is already associated with an XHR object

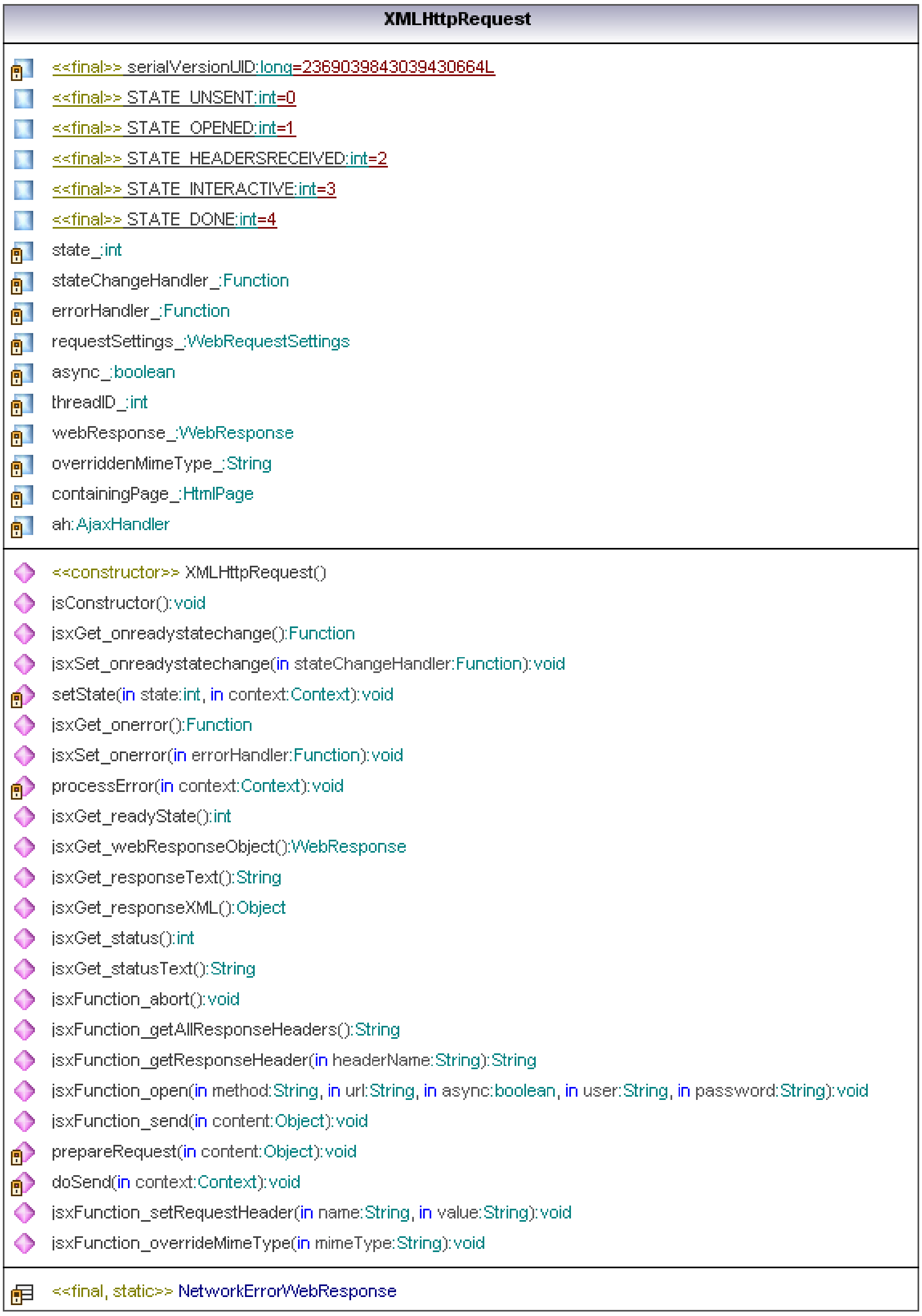
- isFinished(): shows whether the AJAX request has finished

Methods for manipulating the AJAX request

These methods are called from the XHR object and should be overwritten by the tester to insert assertions and instructions.

- onActivation(): is called when an XHR object registes at the handler
- onChange(state): is called every time the state of the XHR object changes
- onError(): is called when an error occurs
- onFinish(): is called when the AJAX request has finished
- beforeSend(): is called just before the request is sent to the server. This method allows manipulations to the request settings.
- setFinished() is used by the XHR object to reset the finished flag

4.2.2 XMLHttpRequest



```
XMLHttpRequest

<<final>> serialVersionUID:long=2369039843039430664L
<<final>> STATE_UNSENT:int=0
<<final>> STATE_OPENED:int=1
<<final>> STATE_HEADERSRECEIVED:int=2
<<final>> STATE_INTERACTIVE:int=3
<<final>> STATE_DONE:int=4
state_:int
stateChangeHandler_:Function
errorHandler_:Function
requestSettings_:WebRequestSettings
async_:boolean
threadID_:int
webResponse_:WebResponse
overriddenMimeType_:String
containingPage_:HtmlPage
ah:AjaxHandler

<<constructor>> XMLHttpRequest()
jsConstructor():void
jsxGet_onreadystatechange():Function
jsxSet_onreadystatechange(in stateChangeHandler:Function):void
setState(in state:int, in context:Context):void
jsxGet_onerror():Function
jsxSet_onerror(in errorHandler:Function):void
processError(in context:Context):void
jsxGet_readyState():int
jsxGet_webResponseObject():WebResponse
jsxGet_responseText():String
jsxGet_responseXML():Object
jsxGet_status():int
jsxGet_statusText():String
jsxFunction_abort():void
jsxFunction_getAllResponseHeaders():String
jsxFunction_getResponseHeader(in headerName:String):String
jsxFunction_open(in method:String, in url:String, in async:boolean, in user:String, in password:String):void
jsxFunction_send(in content:Object):void
prepareRequest(in content:Object):void
doSend(in context:Context):void
jsxFunction_setRequestHeader(in name:String, in value:String):void
jsxFunction_overrideMimeType(in mimeType:String):void

<<final, static>> NetworkErrorWebResponse
```

Figure 4.3 XMLHttpRequest

The XHR object is responsible for the main part of an AJAX request. Some of its methods have to be extended to collaborate with the handler and it also needs a reference to the handler. Each AJAX request has its own XHR object but it is also possible to reuse an XHR object or to abort a running AJAX call.
When the XHR object is created firstly there is no information about the task it has to perform. The method jsxFunction_open passes all request settings to the object.

```
public void jsxFunction_open(final String method, final String url,
                       final boolean async,
                       final String user, final String password) {
  // (URL + Method + User + Password) become a WebRequestSettings instance.
  containingPage_ = (HtmlPage) getWindow().getWebWindow().getEnclosedPage();
  try {
    final URL fullUrl = containingPage_.getFullyQualifiedUrl(url);
    final WebRequestSettings settings = new WebRequestSettings(fullUrl);
    settings.setCharset("UTF-8");
    settings.addAdditionalHeader("Referer",
        containingPage_.getWebResponse().getUrl().
                                 toExternalForm());
    final HttpMethod submitMethod = HttpMethod.valueOf(method.toUpperCase());
    settings.setHttpMethod(submitMethod);
    if (user != null) {
      final DefaultCredentialsProvider dcp = new DefaultCredentialsProvider();
      dcp.addCredentials(user, password);
      settings.setCredentialsProvider(dcp);
    }
    requestSettings_ = settings;
  }
  catch (final MalformedURLException e) {
    getLog().error(
        "Unable to initialize XMLHttpRequest using malformed URL '" +
        url + "'.");
    return;
  }
  // Async stays a boolean.
  async_ = async;
  // Change the state!
  setState(STATE_LOADING, null);
}
```

Table 4.1 open method from XMLHttpRequest

4.2.3 WebClient

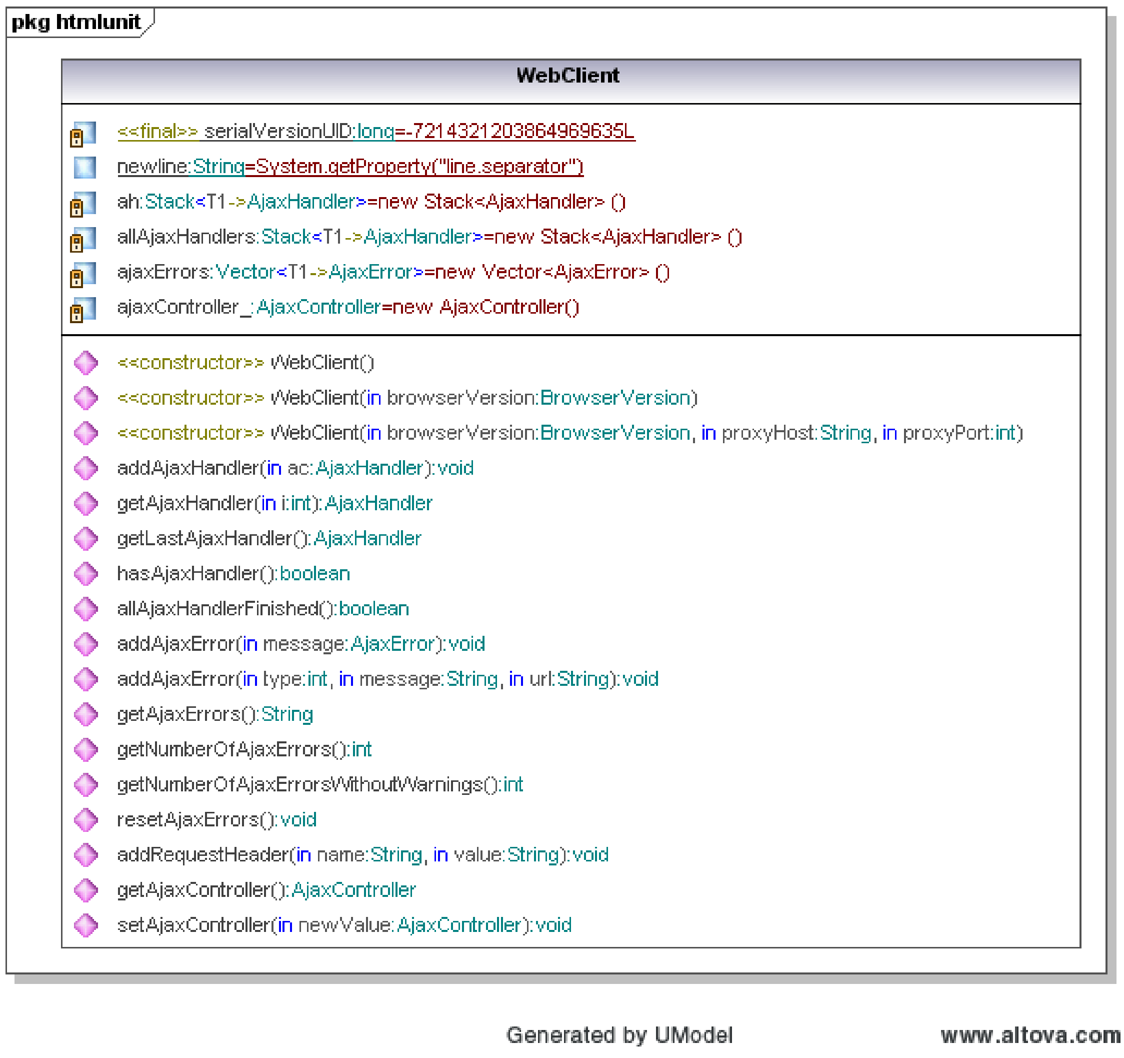

pkg htmlunit
WebClient
<<final>> serialVersionUID:long=-7214321203864969635L
newline:String=System.getProperty("line.separator")
ah:Stack<T1->AjaxHandler>=new Stack<AjaxHandler> ()
allAjaxHandlers:Stack<T1->AjaxHandler>=new Stack<AjaxHandler> ()
ajaxErrors:Vector<T1->AjaxError>=new Vector<AjaxError> ()
ajaxController_:AjaxController=new AjaxController()
<<constructor>> WebClient()
<<constructor>> WebClient(in browserVersion:BrowserVersion)
<<constructor>> WebClient(in browserVersion:BrowserVersion, in proxyHost:String, in proxyPort:int)
addAjaxHandler(in ac:AjaxHandler):void
getAjaxHandler(in i:int):AjaxHandler
getLastAjaxHandler():AjaxHandler
hasAjaxHandler():boolean
allAjaxHandlerFinished():boolean
addAjaxError(in message:AjaxError):void
addAjaxError(in type:int, in message:String, in url:String):void
getAjaxErrors():String
getNumberOfAjaxErrors():int
getNumberOfAjaxErrorsWithoutWarnings():int
resetAjaxErrors():void
addRequestHeader(in name:String, in value:String):void
getAjaxController():AjaxController
setAjaxController(in newValue:AjaxController):void
Generated by UModel
www.altova.com

Figure 4.4 WebClient

Figure 4.4 shows only the additional fields and methods of the class WebClient that were added for integrating the AjaxHandler. The WebClient basically stores information and references about the handlers and acts as a hub connecting the XMLHttpRequest, AjaxHandler, AjaxError and test objects. The methods can be assigned to a specific class.

Methods that can be assigned to the testcase class:

- **addAjaxHandler()**

The tester adds an AjaxHandler object to the system using this method. The web client stores a reference in a stack object called ah temporary until an XMLHttpRequest object fetches the handler, and a second reference is stored permanently in another stack called allAjaxHandlers.

- **allAjaxHandlersFinished()**

The tester calls this method to assert that all requests have finished just before ending the test.

- **getNumberOfAjaxErrors()**

This method is used to retrieve the number of errors, exceptions and warnings that occured inside the AjaxHandler during execution.
getNumberOfAjaxErrorsWithoutWarnings()
This method retrieves only the number of errors and exceptions but ignores the warnings.

- **getAjaxErrors()**

Returns the string representation of all errors, exceptions and warnings that occurred during execution. The message contains the type of error, the description of the error and the request url of the XMLHttpRequest object that caused that error message.
resetAjaxErrors()
Deletes all error messages that have been stored.

Methods that can be assigned to the XMLHttpRequest:

- **hasAjaxHandler()**

The XHR object checks whether there is an unused handler using this method before it trys to fetch one.

- **getAjaxHandler()**

The XHR object fetches a handler using this method. The handler is removed from the stack of the available handlers when calling this method.

- **addAjaxError()**

Stores an AjaxError object containing the error type, error message and the request url.

Methods that can be assigned to the AjaxHandler

- **addAjaxError()**

Stores an AjaxError object containing the error type, error message and the request url.

4.2.4 AjaxError

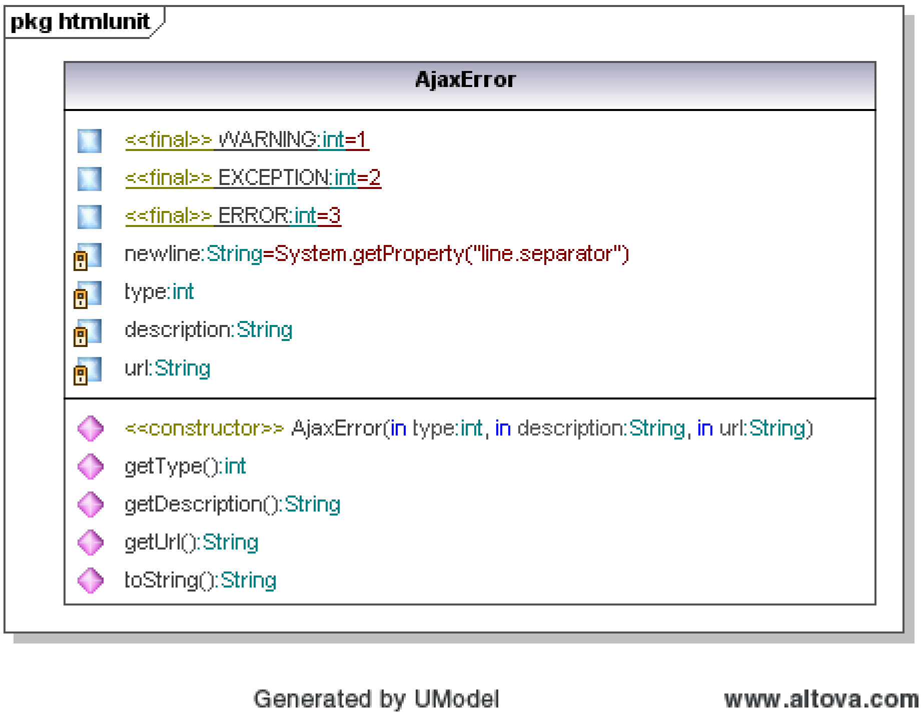


Figure 4.5 AjaxError

The AjaxError class is a helper class storing error information. It saves the error type, the error message and the request URL of the affected XHR object.
There are three types of errors:

1. **Warning** messages occur when an XHR object gets interrupted by an AJAX request while processing another AJAX request. The first request has to be aborted without waiting for the final response and the new one is started. These error messages are created by the XHR object. The description contains besides the interruption message the URL of the old AJAX request.
2. **Error** messages are created by an AjaxHandler object. When assertions inside the handler fail the error is caught by the handler and an AjaxError object is created including the error message and stored at the web client. It is important to catch the error because otherwise the AJAX thread would fail without informing the main thread of the test case and the test case might signal that it passed although an error occurred.
3. **Exception** messages are also created by an AjaxHandler object. Whenever an exception is thrown inside the handler it gets caught by the handler and instead of the exception an AjaxError object is passed to the web client.

The toString() method returns a string representation of the error using the following format:

```
AJAX $TYPE: $MESSAGE \n
URL: $URL
```

It is very important for the tester to check whether errors occurred when using AjaxHandler before the test case ends and to read the error messages or at least write them into a log file.

4.3 AjaxFilterWebConnection

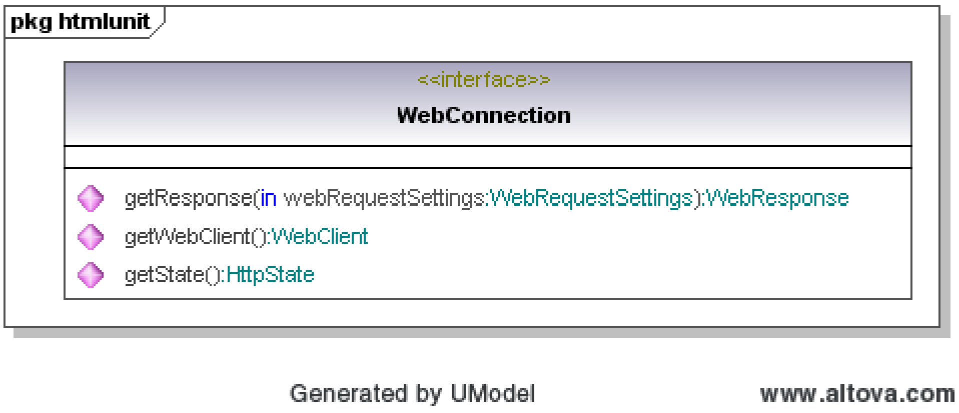


Figure 4.6 AjaxFilterWebConnection

HtmlUnit offers a basic interface for a web-connection. The most important method of that interface is the getResponse() method which is used for all communication between HtmlUnit and the web server. HtmlUnit has two different implementations of that interface:

MockWebConnection

This class isn’t really a web connection. It just stores responses mapped to certain URLs and returns the adequate response to the request URL whenever it receives a request.

HttpWebConnection

This class is used by default. It uses the httpclient from the apache project to send and receive data. This class is also the source for the AjaxFilterWebConnection. It offers additional methods for creating the header information which is also needed for the AjaxFilter. There Filter has to be added to the method getResponse which takes a WebRequestSettings object, delivers it to the server and returns a WebResponse object. The filter has to intercept the workflow on several positions.

```
public WebResponse getResponse(final WebRequestSettings webRequestSettings) throws
    IOException {

  final URL url = webRequestSettings.getUrl();

  final HttpClient httpClient = getHttpClient();

  final HttpMethodBase httpMethod = makeHttpMethod(webRequestSettings);
  try {
    final HostConfiguration hostConfiguration = getHostConfiguration(
        webRequestSettings);
    final long startTime = System.currentTimeMillis();
    final int responseCode = httpClient.executeMethod(hostConfiguration,
14         httpMethod);
15     final long endTime = System.currentTimeMillis();
16     return makeWebResponse(responseCode, httpMethod, webRequestSettings,
17                             endTime - startTime,
18                             webRequestSettings.getCharset());
19   }
20   catch (final HttpException e) {
21     if (url.getPath().length() == 0) {
22       final StringBuilder buffer = new StringBuilder();
23       buffer.append(url.getProtocol());
24       buffer.append("://");
25       buffer.append(url.getHost());
26       buffer.append("/");
27       if (url.getQuery() != null) {
28         buffer.append(url.getQuery());
29       }
30       //TODO: There might be a bug here since the original encoding type
is lost.
31       final WebRequestSettings newRequest = new WebRequestSettings(new
URL(
32           buffer.toString()));
33       newRequest.setHttpMethod(webRequestSettings.getHttpMethod());
34
newRequest.setRequestParameters(webRequestSettings.getRequestParameters());
35
newRequest.setAdditionalHeaders(webRequestSettings.getAdditionalHeaders());
36       return getResponse(newRequest);
37     }
38     e.printStackTrace();
39     throw new RuntimeException("HTTP Error: " + e.getMessage());
40   }
41   finally {
42     onResponseGenerated(httpMethod);
43   }
44 }
```

Table 4.2 getResponse method from WebConnection

First a handler has to be registered at the web-client.
line 03: The AjaxFilter has to get a reference to the handler and pass the request setting object to the handler to allow it to check or manipulate the settings.
line 16: The response must not be returned immediately. First it has to be stored in a temporary object inside of the method. Then the handler has to be informed that the state has changed and the response object has to be passed to the handler so it can validate and manipulate the response. Adjusting these two lines should be enough to meet the demands for the tester.
Unfortunately there are some major problems when trying to implement the AjaxFilterWebConnection.
The httpclient from the apache project has some bugs. It is possible that it throws an exception just because a slash is missing although other web browsers have no problems sending the same request. That’s why the method catches the exception on line 20 reforms the request settings and resends the new request by calling the getResponse method recursively.

It is impossible to distinguish between AJAX requests and normal web requests like page reload. The tester has to register the handler right before firing the AJAX request and unregister it afterwards because otherwise all other requests would also use the handler.
These issues make the AjaxFilter become stale when compared to the AjaxHandler which is quite easier to handle. That’s why the idea of the AjaxFilterWebConnection is rated useless and the development is stopped. The focus now lies upon the AjaxHandler.

5 Case Studies for Validation

This chapter contains three simple web applications using AJAX and simple test cases for them. Each test case is implemented in three different ways:

1. Using the old NicelyResynchronizingAjaxController.
2. Using the new AjaxHandler by simply resynchronizing the AJAX calls.
3. Using the new AjaxHandler with all newly available features.

5.1 Wikipedia

Wikipedia [44] is a free, multilingual encyclopaedia project. The start page contains a search field which uses AJAX techniques to suggest topics according to the user input.

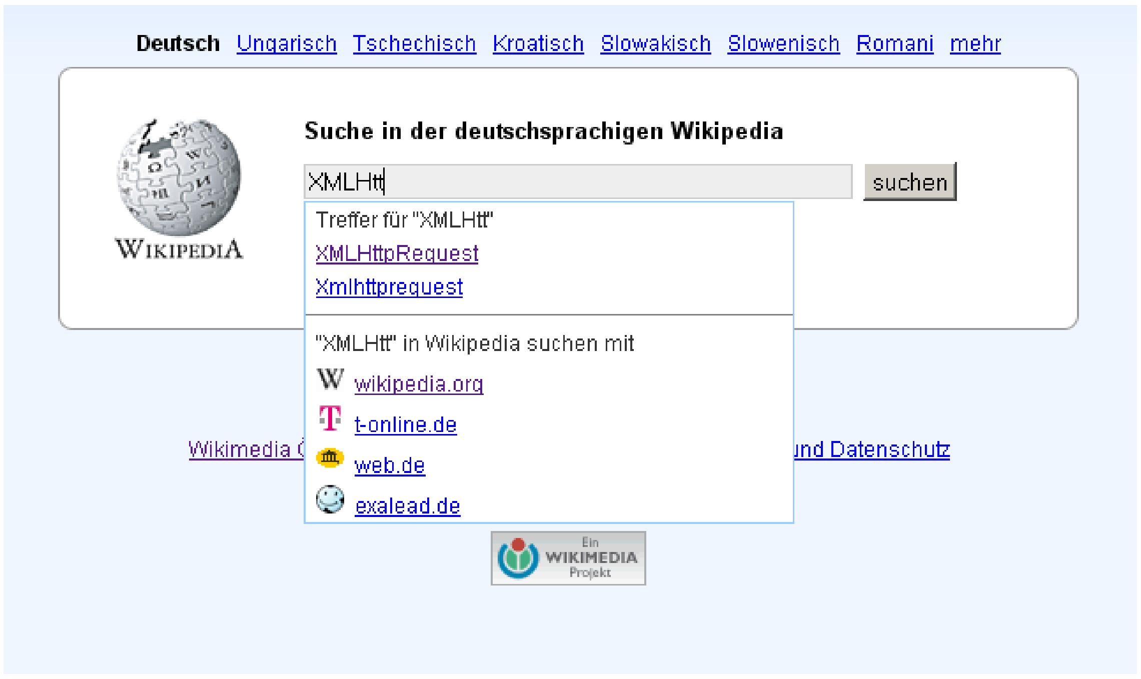


Figure 5.1 Wikipedia screenshot

After each keystroke an event is fired and a query is sent to the server which returns up to ten hits matching with the user input. These matches are presented on the webpage as web-links and can directly be used. When the user appends more characters a new AJAX event is fired and the matches on the web page are replaced by the new ones.

Testcases:

Id	Item	Input	Output	Depends
1	suggest	“XMLHtt”	List including “XMLHttpRequest”	none

Table 5.1 testcase for Wikipedia

5.1.1 AjaxController

```
1.   public void testCase_1_AjaxController() throws Exception {
2.     // create default firefox browser
3.     final WebClient webClient = new WebClient(BrowserVersion.
4.                                             FIREFOX_2);
5.     // initialize and register AjaxController
6.     AjaxController ac = new NicelyResynchronizingAjaxController();
7.     webClient.setAjaxController(ac);
8.     // connect to Wikipedia.at
9.     final HtmlPage page = (HtmlPage) webClient.getPage(
10.         "http://www.wikipedia.at");
11.     HtmlPage page2;
12.     WebAssert.assertTitleContains(page, "wikipedia.at");
13.     // parse the document for the HtmlInput field
14.     HtmlForm form = (HtmlForm) page.getForms().get(0);
15.     HtmlInput input = form.getInputByName("q");
16.     // write into the field and fire keyup event
17.     input.setValueAttribute("XMLHtt");
18.     input.keyup();
19.     // parse result and click on XMLHttpRequest Link
20.                 HtmlElement    search_suggest    =    (HtmlElement)
page.getElementById(
21.         "search_suggest");
22.     WebAssert.assertTextPresentInElement(page, "XMLHttpRequest",
23.                                         "search_suggest");
24.     List li = search_suggest.getByXPath("div/a");
25.     assertFalse(li.isEmpty());
26.     HtmlAnchor link = (HtmlAnchor) li.get(0);
27.     System.err.println(link.asXml());
28.     page2 = (HtmlPage) link.click();
29.     WebAssert.assertTitleContains(page2, "XMLHttpRequest");
30.   }
```

Table 5.2 testcase 1 - AjaxController

This table shows a default testcase using HtmlUnit with the already built in AjaxController. The controller is initialized and activated in the lines six and seven. In line 17 the word to search for is specified and in line 18 the AJAX request is fired. The built in AjaxConroller resynchronizes the AJAX request and continues processing the testcase when the response has been received. In the rest of the testcase some asserts are done and the link "XMLHttpRequest" is clicked.

5.1.2 AjaxHandler

```
1.   public void testCase_1_AjaxHandler() throws Exception {
2.     // create default firefox browser
3.     final WebClient webClient = new WebClient(BrowserVersion.
4.                                             FIREFOX_2);
5.     // initialize and register the default AjaxHandler
6.     AjaxHandler ah = new AjaxHandler();
7.     webClient.addAjaxHandler(ah);
8.     // connect to wikipedia.at
9.     final HtmlPage page = (HtmlPage) webClient.getPage(
10.         "http://www.wikipedia.at");
11.     HtmlPage page2;
12.     WebAssert.assertTitleContains(page, "wikipedia.at");
13.     // parse the document for the HtmlInput field
14.     HtmlForm form = (HtmlForm) page.getForms().get(0);
15.     HtmlInput input = form.getInputByName("q");
16.     // write into the field and fire keyup event
17.     input.setValueAttribute("XMLHtt");
18.     input.keyup();
19.     do {
20.       synchronized (this) {
21.         wait(600);
22.       }
23.     }
24.     while (!ah.isFinished());
25.     // parse result and click on XMLHttpRequest Link
26.                 HtmlElement    search_suggest    =    (HtmlElement)
page.getElementById(
27.         "search_suggest");
28.     System.err.println(page.asXml());
29.     WebAssert.assertTextPresentInElement(page, "XMLHttpRequest",
30.                                          "search_suggest");
31.     List li = search_suggest.getByXPath("div/a");
32.     assertFalse(li.isEmpty());
33.     HtmlAnchor link = (HtmlAnchor) li.get(0);
34.     System.err.println(link.asXml());
35.     page2 = (HtmlPage) link.click();
36.     WebAssert.assertTitleContains(page2, "XMLHttpRequest");
37.     assertEquals(0, webClient.getNumberOfAjaxErrors());
38.   }
```

Table 5.3 testcase 1 - AjaxHandler

The AjaxHandler works very similar to the AjaxController. The first difference is that no AjaxController is initialized but instead of it a default AjaxHandler is initialized and registered in the lines six and seven. The AjaxHandler doesn’t resynchronize the AJAX request so the test case needs some other instructions to wait with the assertions until the response of the AJAX request has arrived. In line nineteen to twenty-four a loop checks every 600 milliseconds whether the AJAX request has finished by asking the AjaxHandler. The AjaxHandler is automatically assigned to the AJAX request as soon as the request is fired. The rest of the testcase is exactly the same as the testcase using the AjaxController.

5.1.3 AjaxHandler - improved

```
public void testCase_1_AjaxHandlerImproved() throws Exception {
  // create default firefox browser
  final WebClient webClient = new WebClient(BrowserVersion.
                                          FIREFOX_2);
  // initialize and register the default AjaxHandler
  AjaxHandler ah = new AjaxHandler() {
    @Override
    public void onChange(int state) {
      if (state == 4) {
        assertTrue(getXMLHttpRequest().jsxGet_responseText().
                   indexOf(
                       "XMLHttpRequest") >= 0);
      }
    }
  };
  webClient.addAjaxHandler(ah);
  // connect to wikipedia.at
  final HtmlPage page = (HtmlPage) webClient.getPage(
      "http://www.wikipedia.at");
  HtmlPage page2;
  WebAssert.assertTitleContains(page, "wikipedia.at");
  // parse the document for the HtmlInput field
  HtmlForm form = (HtmlForm) page.getForms().get(0);
  HtmlInput input = form.getInputByName("q");
  // write into the field and fire keyup event
  input.setValueAttribute("XMLHtt");
  input.keyup();
  do {
    synchronized (this) {
      wait(600);
    }
  }
  while (!ah.isFinished());
  // parse result and click on XMLHttpRequest Link
  HtmlElement search_suggest = (HtmlElement) page.getElementById(
      "search_suggest");
  System.err.println(page.asXml());
  List li = search_suggest.getByXPath("div/a");
  assertFalse(li.isEmpty());
  HtmlAnchor link = (HtmlAnchor) li.get(0);
  page2 = (HtmlPage) link.click();
  WebAssert.assertTitleContains(page2, "XMLHttpRequest");
  assertTrue(webClient.allAjaxHandlerFinished());
  assertEquals(0, webClient.getNumberOfAjaxErrors());
}
```

Table 5.4 testcase 1 – AjaxHandler improved

In this testcase more features of the AjaxHandler are used. While the first testcase using the ajaxhandler just copied the behaviour of the AjaxController this one goes one step further. In the lines six to thirteen an AjaxHandler is initialized but the method onChange is overridden. While onChange() basically is an empty method in the default AjaxHandler this time it asserts that the text “XMLHttpRequest” is part of the response of the AJAX request. This assertion is executed when the state of the XMLHttpRequest is four which means that the request has finished. The rest of the testcase is the same as in the first one except of the last assertion which checks whether no errors have occurred inside the AjaxHandler threads. When an error or exception is raised in an other thread than the main thread it isn’t recognized by the main thread and the test might pass although some error occurred. That’s why these errors are stored in the WebClient and at the end of the testcase the tester has to assure that no error messages are stored.

5.2 AJAX Login System

The AJAX Login System [45] is a small application created by James Dam. It uses AJAX for validating the username and password from a login form without doing a page refresh.

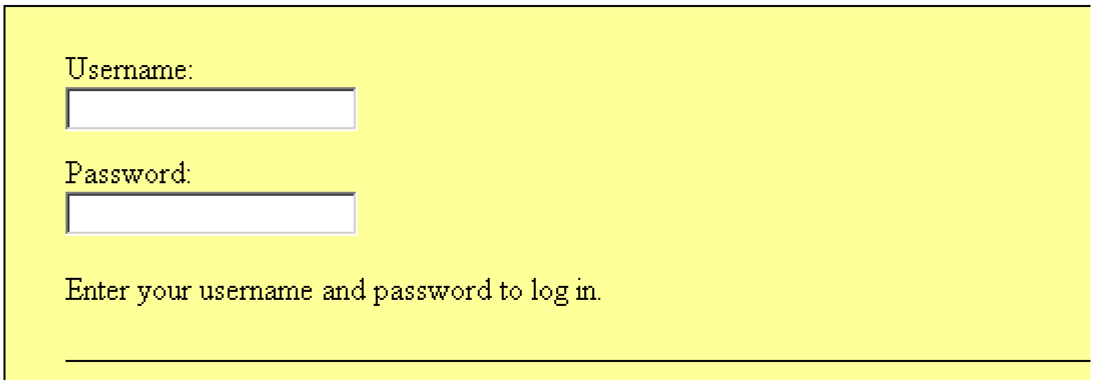


Figure 5.2 the main page of the AJAX login system

Actually the web page contains two different AJAX functions. The first one is fired when the username or the password field get the focus and a random seed is received from the server to encrypt the password when finished. The second AJAX function is activated when the username or the password field loose the focus and both fields contain non empty strings. The password is encrypted using the seed and sent together with the username to the server. The response from the server is either the real name for the given password and username combination or an error message saying that the user has entered an invalid combination.

Testcases

Id	Item	Input	Output	Depends

1	log-in	username = “user1”
		password = “pass1”
			John Doe logged in	

2	log-out	press “logout”	log-in screen	1

3	log-in	username = “user1”
		password = “wrongpass”
			invalid u/p combination	

4	log-in	username = “user” no password	no changes (no AJAX request)	
5	log-in	no username password = “pass1”	no changes (no AJAX request)	

Table 5.5: test cases for the AJAX login system

There are two possible results:
Login succeeded

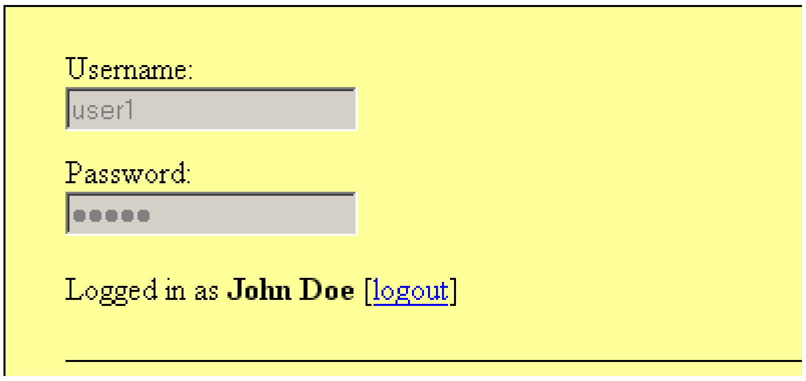

Username:
user1
Password:
●●●●●
Logged in as **John Doe** [logout]

Figure 5.3: login succeed - AJAX login system

Login failed

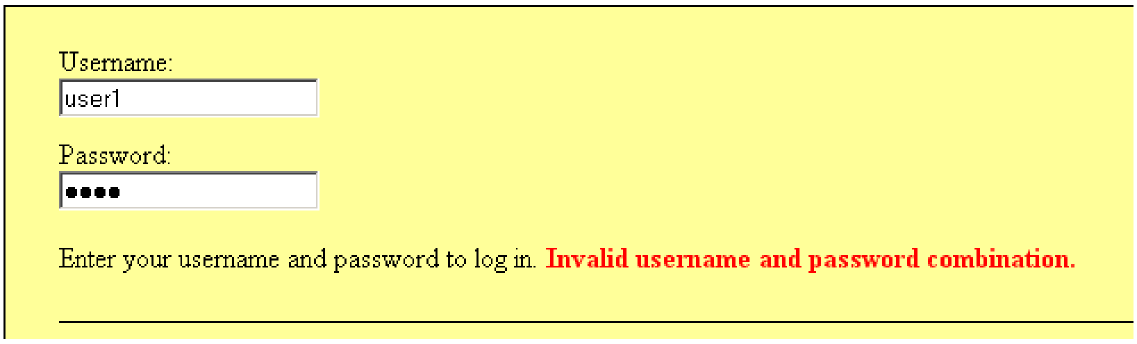

Username:
user1
Password:
●●●●
Enter your username and password to log in. **Invalid username and password combination.**

Figure 5.4: login failed - AJAX login system

5.2.1 AjaxController

```
1. public void testCase_1_AjaxController() throws Exception {
2.   final WebClient webClient = new WebClient(BrowserVersion.
3.                                              FIREFOX_2);
4.   //register NicelyResynchronizingAjaxController
5.   AjaxController ac = new NicelyResynchronizingAjaxController();
6.   webClient.setAjaxController(ac);
7.   //load Page
8.   final HtmlPage page = (HtmlPage) webClient.getPage(
9.       "http://www.jamesdam.com/ajax_login/login.html");
10.   WebAssert.assertTitleContains(page, "AJAX Login System");
11.   // get form and get username and password field
12.   HtmlForm form = (HtmlForm) page.getForms().get(0);
13.   HtmlInput user = form.getInputByName("username");
14.   HtmlInput pass = form.getInputByName("password");
15.   //set username
16.   user.focus();
17.   user.setValueAttribute("user1");
18.   user.blur();
19.   //set password
20.   pass.focus();
21.   pass.setValueAttribute("pass1");
22.   pass.blur();
23.   //do assertions
24.   WebAssert.assertTextPresentInElement(page, "Logged in as",
25.                                       "login");
26.   WebAssert.assertTextPresentInElement(page, "John Doe", "login");
27.   WebAssert.assertLinkPresentWithText(page, "logout");
28. }
```

Table 5.6: testcase 1 – AjaxController

The AjaxController just has to be activated. On line 5 the built in controller gets initialized and on line 6 it is passed to the web client where it replaces the default non-functional controller. There is nothing else the tester can do with the controller. The rest of the testcase is a standard HtmlUnit testcase.
The first 3 test cases are easy to implement and very similar to the first one but the last two exceed the capabilities of the built in AjaxController. The program shouldn’t start an AJAX request when either the username or the password field is empty. There is no possibility to check whether an AJAX call is performed. The tester can only take a snapshot of the page and compare it to the page after focusing and leaving one of these fields but the result can only prove that the content of the page hasn’t changed and not that no request was fired.

```
1. public void testCase_4_AjaxController() throws Exception {
2.   final WebClient webClient = new WebClient(BrowserVersion.
3.                                              FIREFOX_2);
4.   AjaxController ac = new NicelyResynchronizingAjaxController();
5.   webClient.setAjaxController(ac);
6.   final HtmlPage page = (HtmlPage) webClient.getPage(
7.       "http://www.jamesdam.com/ajax_login/login.html");
8.   WebAssert.assertTitleContains(page, "AJAX Login System");
9.   HtmlForm form = (HtmlForm) page.getForms().get(0);
10.   HtmlInput user = form.getInputByName("username");
11.   HtmlInput pass = form.getInputByName("password");
12.   user.focus();
13.   user.setValueAttribute("user1");
14.   user.blur();
15.   pass.focus();
16.   pass.setValueAttribute("");
17.   String savePage = page.asXml();
18.   pass.blur();
19.   assertEquals(savePage, page.asXml());
20. }
```

Table 5.7: testcase 4 – AjaxController

The testcase just saves the whole page in a string object on line 17 just before the password field looses the focus. Normally this action would activate an AJAX request but in this case the password field only contains an empty string so nothing happens. On line 19 the saved page is compared to the actual page. The 5th test case is very similar to this one.

5.2.2 AjaxHandler – legacy mode

```
public void testCase_1_AjaxHandler() throws Exception {
  final WebClient webClient = new WebClient(BrowserVersion.
                                            FIREFOX_2);
  //register AjaxHandler
  AjaxHandler ah = new AjaxHandler();
  webClient.addAjaxHandler(ah);
  //load Page
  final HtmlPage page = (HtmlPage) webClient.getPage(
      "http://www.jamesdam.com/ajax_login/login.html");
  WebAssert.assertTitleContains(page, "AJAX Login System");
  // get form and get username and password field
  HtmlForm form = (HtmlForm) page.getForms().get(0);
  HtmlInput user = form.getInputByName("username");
  HtmlInput pass = form.getInputByName("password");
  //set username
  user.focus();
  do {
    synchronized (this) {
      wait(100);
20.     }
21.   }
22.   while (!ah.isFinished());
23.   user.setValueAttribute("user1");
24.   user.blur();
25.   //set password
26.   pass.focus();
27.   pass.setValueAttribute("pass1");
28.   pass.blur();
29.   do {
30.     synchronized (this) {
31.       wait(100);
32.     }
33.   }
34.   while (!ah.isFinished());
35.   //do assertions
36.   WebAssert.assertTextPresentInElement(page, "Logged in as",
37.                                        "login");
38.   WebAssert.assertTextPresentInElement(page, "John Doe", "login");
39.   WebAssert.assertLinkPresentWithText(page, "logout");
40. }
```

Table 5.8 testcase 1 – AjaxHandler

This test case shows the use of AjaxHandler imitating the AjaxController. The Ajax-Controller resynchronizes the AJAX calls so the main thread has to wait until the request has finished before it can continue. The AjaxHandler on the opposite doesn't re-synchronize the AJAX calls. The main test thread would continue execution so the tester has to build in some kind of pause. This is done from the lines 17 to 22 and from the lines 29 to 34. The test case is stopped using a loop that waits for the AJAX handler to be finished.

Initializing and registering the AjaxHandler is very similar to the procedure using the AjaxController. The tester just has to use another method for registering. Normally every AJAX request has its own handler but in the AJAX login system implementation only one XMLHttpRequest object is created and used for both AJAX functions. When the username field gets the focus on line 16 the XHR object catches the AjaxHandler object from the web client and updates the handler every time a state change happens. When the response has been received the handler is notified that AJAX has finished and the main test thread can go on. On line 28 the second AJAX request is fired and the same XHR object is taken and just updated with the new request settings. The XHR object has already a handler so it just notifies that a new job has arrived and executes the request.
At first sight the AjaxHandler just seems to be more difficult to use compared with the AjaxController because the tester just has to write more code but the handler has a very big advantage compared to the controller. It really works with all AJAX calls while the

controller might not be able to resynchronize all calls. In this case the controller apparently had no problems.

5.2.3 AjaxHandler – improved

This time we try to use the full abilities of the AjaxHandler.

```
public void testCase_1_AjaxHandlerImproved() throws Exception {
  final WebClient webClient = new WebClient(BrowserVersion.
                                            FIREFOX_2);
  //create AjaxHandler
  AjaxHandler ah = new AjaxHandler() {
    @Override
    public void onChange(int state) {
      if (state == 4) {
        assertTrue(getXMLHttpRequest().jsxGet_responseText().
                    indexOf("John Doe") >= 0);
      }
    }
  };
  //load Page
  final HtmlPage page = (HtmlPage) webClient.getPage(
      "http://www.jamesdam.com/ajax_login/login.html");
  WebAssert.assertTitleContains(page, "AJAX Login System");
  // get form and get username and password field
  HtmlForm form = (HtmlForm) page.getForms().get(0);
  HtmlInput user = form.getInputByName("username");
  HtmlInput pass = form.getInputByName("password");
  //set username
  user.focus();
  user.setValueAttribute("user1");
  user.blur();
  //set password
  pass.focus();
  pass.setValueAttribute("pass1");
  webClient.addAjaxHandler(ah);
  pass.blur();
  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (!ah.isFinished());
  assertEquals(0, webClient.getNumberOfErrors());
}
```

Table 5.9 testcase 1 - AjaxHandler improved

This time the main assertions aren't done in the test case itself but they are integrated into the AjaxHandler. The method onChange() is replaced by our own method from the lines 6 to 13. When the state of the XHR object is four which means that the response has been received the content of the response can directly be viewed and it is easy to do assertions using the content. However the handler must not be registered at once because the first AJAX request which gets fired on line 23 would connect to that handler and cause an assertion error because instead of a username that request only returns a random seed. The handler has to be registered just before using firing the second AJAX request on line 30. At the end of the test case the test thread has to wait until the AJAX request has finished because otherwise it would end the test end possible errors in the AJAX thread won't be recognized. There are many possibilities to wait for all AJAX requests to be finished. Another idea is to use the allAjaxHandlerFinished() method from the WebClient class or to add a notify() to the onFinish() method in the AjaxHandler.
The last thing the tester should do is to check whether no errors have occurred. The test cases two and three can be solved the same way, just the assertions have to be adapted.

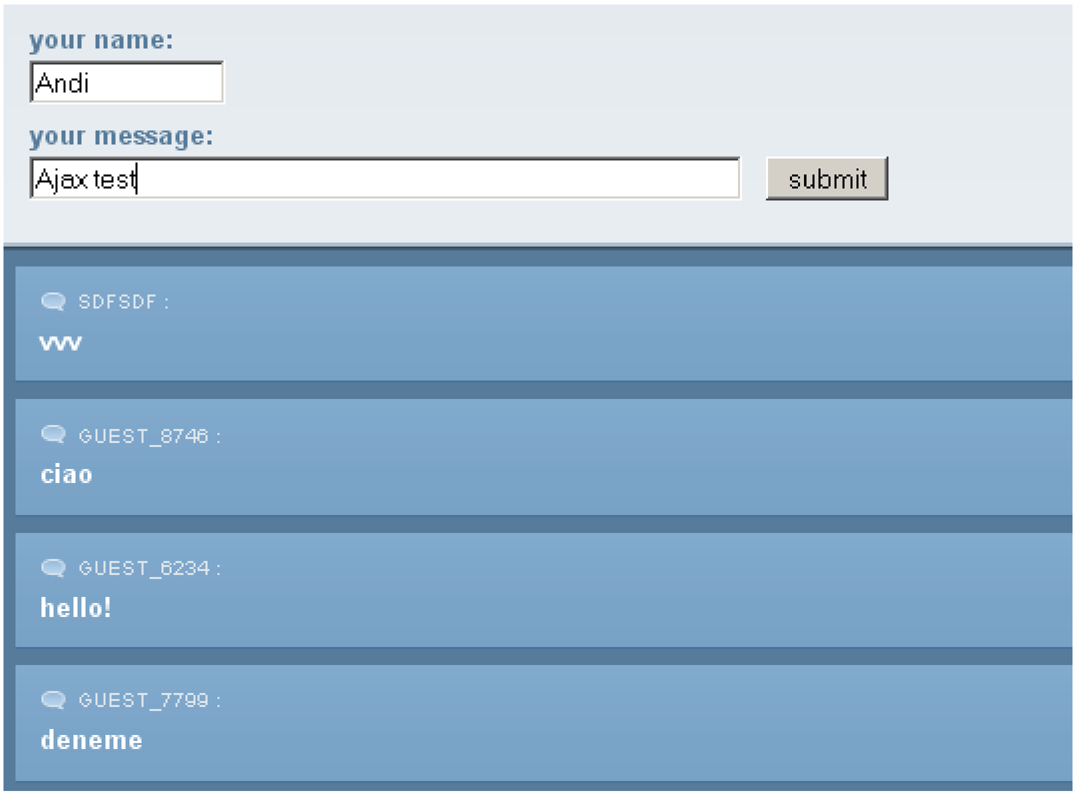
```
1. public void testCase_4_AjaxHandler() throws Exception {
2.   final WebClient webClient = new WebClient(BrowserVersion.
3.                                             FIREFOX_2);
4.   AjaxHandler ah = new AjaxHandler();
5.   final HtmlPage page = (HtmlPage) webClient.getPage(
6.       "http://www.jamesdam.com/ajax_login/login.html");
7.   WebAssert.assertTitleContains(page, "AJAX Login System");
8.   HtmlForm form = (HtmlForm) page.getForms().get(0);
9.   HtmlInput user = form.getInputByName("username");
10.   HtmlInput pass = form.getInputByName("password");
11.   user.focus();
12.   webClient.addAjaxHandler(ah);
13.   user.setValueAttribute("user1");
14.   user.blur();
15.   assertFalse(ah.used());
16.   pass.focus();
17.   pass.setValueAttribute("");
18.   pass.blur();
19.   assertFalse(ah.used());
20. }
```

Table 5.10 testcase 4 – AjaxHandler improved

The fourth test case is really easy to implement using the AjaxHandler. We just need to create a basic handler and register it after the first AJAX request has been fired. When the statement on line 18 fires an AJAX request the XHR object would use the handler. So the tester just has to check whether the handler is being used.

5.3 XHTML live Chat

The XHTML live Chat [46] is a web based chat system. Anyone can post messages that appear after a few seconds on all other web browsers where this web page is active. All data communication is implemented using AJAX.



SDFSDF :
vvv

GUEST_8746 :
ciao

GUEST_6234 :
hello!

GUEST_7799 :
deneme

Figure 5.5 XHTML live Chat

The chat system uses two XHR objects. One is responsible for sending the message when clicking on submit while the other one is executed automatically every four to five seconds to get the latest posts from the server and present them on the page.

Testcases

Id	Item	Input	Output
1	post & receive	Post message "AJAX Test" with user name "Andi"	The posted message
2	post & multiple receive	Start two web browsers. Post message "AJAX Test 2" with user name "Andi" from browser one.	The posted message is shown on both browsers

Table 5.11 testcases for the XHTML Chat System

5.3.1 AjaxController

```
public void testCase_1_AjaxController() throws Exception {
```

```
2.      //configure the browser
3.      final WebClient webClient = new WebClient(BrowserVersion.
4.                                                   FIREFOX_2);
5.      // create default ajax controller
6.      AjaxController ac = new NicelyResynchronizingAjaxController();
7.      webClient.setAjaxController(ac);
8.      AjaxHandler ah = new AjaxHandler();
9.      webClient.addAjaxHandler(ah);
10.      //connect to the page and find all necessary elements
11.      final HtmlPage page = (HtmlPage) webClient.getPage(
12.          "http://chat.plasticshore.com/");
13.      WebAssert.assertTitleContains(page, "live chat");
14.
15.      HtmlForm form = (HtmlForm) page.getFormByName("chatForm");
16.      HtmlInput name = form.getInputByName("name");
17.      HtmlInput input = form.getInputByName("chatbarText");
18.             HtmlSubmitInput   submit   =   (HtmlSubmitInput)
form.getInputByName(
19.          "submit");
20.      //set username and message and send everything
21.      String nameString = "con1";
22.      name.setValueAttribute(nameString);
23.      input.focus();
24.      String message = "hello!";
25.      input.setValueAttribute(message);
26.      submit.click();
27.      // workaround: the program has to wait some time until
28.      // the new messages have been received
29.      Thread.currentThread().sleep(6000);
30.      // parse the outputlist for the first element
31.                                          List       list        =
page.getByXPath("//ul[attribute::id='outputList']/li");
32.      assertFalse(list.isEmpty());
33.      HtmlListItem li = (HtmlListItem) list.get(0);
34.      HtmlUnorderedList ul = (HtmlUnorderedList) page.getByXPath(
35.          "//ul[attribute::id='outputList']").get(0);
36.      System.err.println(ul.asXml());
37.      assertTrue(ul.asXml().contains(message));
38.      assertTrue(ul.asXml().contains(nameString));
39.      assertTrue(li.asXml().contains(message));
40.      assertTrue(webClient.getNumberOfAjaxErrorsWithoutWarnings() ==
0);
41.   }
```

Table 5.12 testcase 1 - AjaxController

Again the default AjaxController is initialized and registered. In the lines twenty-two to twenty-six a message is sent and afterwards the web page is scanned for the just written message. It takes some time until the message really arrives on the web page because there is a JavaScript function which catches all new messages periodically and that's why the execution of this thread has to be paused some time. There is no possibility to get the exact amount of time the function will need so on line 29 the testcase stops for 6 seconds. Normally the web page shouldn't need more than one second to update but several tries showed that it can last up to four seconds so we take six seconds to be on the safe side. Without this stop the testcase would fail for sure.

5.3.2 AjaxHandler

```
1.   public void testCase_1_AjaxHandler() throws Exception {
2.     //configure the browser
3.     final WebClient webClient = new WebClient(BrowserVersion.
4.                                                 FIREFOX_2);
5.     // create two default ajax handler: one for receiving
6.     // and one for sending chet messages
7.     AjaxHandler ah2 = new AjaxHandler();
8.     webClient.addAjaxHandler(ah2);
9.     AjaxHandler ah = new AjaxHandler();
10.     webClient.addAjaxHandler(ah);
11.     //connect to the page and find all necessary elements
12.     final HtmlPage page = (HtmlPage) webClient.getPage(
13.         "http://chat.plasticshore.com/");
14.     WebAssert.assertTitleContains(page, "live chat");
15.     //force loading all current messages
16.     do {
17.       synchronized (this) {
18.         wait(100);
19.       }
20.     }
21.     while (ah.isFinished());
22.     // wait until the XHR for receiving finishes
23.     do {
24.       synchronized (this) {
25.         wait(100);
26.       }
27.     }
28.     while (!ah.isFinished());
29.     HtmlForm form = (HtmlForm) page.getFormByName("chatForm");
30.     HtmlInput name = form.getInputByName("name");
31.     HtmlInput input = form.getInputByName("chatbarText");
32.              HtmlSubmitInput   submit   =   (HtmlSubmitInput)
form.getInputByName(
33.         "submit");
34.     //set username and message and send everything
35.     String nameString = "qwe";
36.     name.setValueAttribute(nameString);
37.     input.focus();
38.     String message = "aghdf";
39.     input.setValueAttribute(message);
40.     submit.click();
41.     //wait until sending has finished
42.     do {
43.       synchronized (this) {
44.         wait(100);
45.       }
46.     }
47.     while (!ah2.isFinished());
48.     //wait until the next receiving thread starts
49.
50.     do {
51.       synchronized (this) {
52.         wait(100);
53.       }
54.     }
55.     while (ah.isFinished());
56.
57.     // wait until the XHR for receiving finishes
58.     do {
59.       synchronized (this) {
60.         wait(100);
61.       }
62.     }
63.     while (!ah.isFinished());
64.     // parse the outputlist for the first element
65.                                          List        list        =
page.getByXPath("//ul[attribute::id='outputList']/li");
66.     assertFalse(list.isEmpty());
67.     HtmlListItem li = (HtmlListItem) list.get(0);
68.     HtmlUnorderedList ul = (HtmlUnorderedList) page.getByXPath(
69.         "//ul[attribute::id='outputList']").get(0);
70.     assertTrue(ul.asXml().contains(message));
71.     assertTrue(ul.asXml().contains(nameString));
72.     assertTrue(li.asXml().contains(message));
73.     assertTrue(webClient.getNumberOfAjaxErrorsWithoutWarnings() ==
0);
74.   }
```

Table 5.13 testcase 1 - AjaxHandler

Using the AjaxHandler is a bit more complicated. First the tester has to investigate the program code. The chat application uses two XMLHttpRequest objects. One is called automatically every five seconds to receive the new messages while the other one is used for sending the message. That’s why two AjaxHandlers are needed. When loading the page the XHR object that is responsible for receiving new messages has to load the last 60 messages to fill the webpage and to show the user the actual topic. That’s why in the lines sixteen to twenty-five the testcase waits until the AJAX request starts and finishes. From the lines forty-two to sixty-three the testcase again waits until the message has been sent and the receiving has finished. Afterwards several assertions can be done.

The testcase pauses several times but it needs only exactly the time the web application needs to finish the AJAX requests so that the assertions can be done.

5.3.3 AjaxHandler - Improved

```
public void testCase_2_AjaxHandlerImroved() throws Exception {
  final String nameString = "me";
  final String message = "hurra!";
  //configure the browser
  final WebClient webClient_1 = new WebClient(BrowserVersion.
                                              FIREFOX_2);
  // create two default ajax handler: one for receiving
  // and one for sending chet messages
  AjaxHandler ah_1_s = new AjaxHandler();
  webClient_1.addAjaxHandler(ah_1_s);
  AjaxHandler ah_1_r = new AjaxHandler();
  webClient_1.addAjaxHandler(ah_1_r);
  AjaxHandler ah_1_r_2 = new AjaxHandler() {
    @Override
    public void onChange(int state) {
      if (state == 4) {
        assertTrue(getXMLHttpRequest().jsxGet_responseText().
                   indexOf(nameString) >= 0);
        assertTrue(getXMLHttpRequest().jsxGet_responseText().
                   indexOf(message) >= 0);
      }
    }
  };
  //the same with the second browser
  //configure the browser
  final WebClient webClient_2 = new WebClient(BrowserVersion.
                                              FIREFOX_2);
  // create two default ajax handler: one for receiving
  // and one for sending chet messages
  AjaxHandler ah_2_s = new AjaxHandler();
  webClient_2.addAjaxHandler(ah_2_s);
  AjaxHandler ah_2_r = new AjaxHandler();
  webClient_2.addAjaxHandler(ah_2_r);
  //connect to the page and find all necessary elements
  final HtmlPage page_1 = (HtmlPage) webClient_1.getPage(
      "http://chat.plasticshore.com/");
  WebAssert.assertTitleContains(page_1, "live chat");
  final HtmlPage page_2 = (HtmlPage) webClient_2.getPage(
      "http://chat.plasticshore.com/");
  WebAssert.assertTitleContains(page_2, "live chat");
  //force loading all current messages
  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (ah_1_r.isFinished());
  // wait until the XHR for receiving finishes
  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (!ah_1_r.isFinished());
  HtmlForm form_1 = (HtmlForm) page_1.getFormByName("chatForm");
  HtmlInput name = form_1.getInputByName("name");
  HtmlInput input = form_1.getInputByName("chatbarText");
  HtmlSubmitInput submit = (HtmlSubmitInput) form_1.getInputByName(
      "submit");
  //set username and message and send everything
  name.setValueAttribute(nameString);
  input.focus();
  input.setValueAttribute(message);
  submit.click();

  ah_1_r.removeHandler();
  webClient_1.addAjaxHandler(ah_1_r_2);
  //wait until sending has finished
  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (!ah_1_s.isFinished());
  //wait until the next receiving thread starts

  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (ah_1_r_2.isFinished());
  // wait until the XHR for receiving finishes
  do {
    synchronized (this) {
      wait(100);
    }
  }
  while (!ah_1_r_2.isFinished());
  // parse the outputlist for the first element
  List list_1 = page_1.getByXPath(
      "//ul[attribute::id='outputList']/li");
  assertFalse(list_1.isEmpty());
  HtmlListItem li_1 = (HtmlListItem) list_1.get(0);
  HtmlUnorderedList ul_1 = (HtmlUnorderedList) page_1.getByXPath(
      "//ul[attribute::id='outputList']").get(0);
  assertTrue(ul_1.asXml().contains(message));
  assertTrue(ul_1.asXml().contains(nameString));
  assertTrue(li_1.asXml().contains(message));
    assertTrue(webClient_1.getNumberOfAjaxErrorsWithoutWarnings() ==
                0);
    // the same with the second client
    int i1 = 0;
    do {
      synchronized (this) {
      wait(100);
      }
      i1++;
    }
    while (ah_2_r.isFinished());
    // wait until the XHR for receiving finishes
    int i2 = 0;
    do {
      synchronized (this) {
      wait(100);
      }
      i2++;
    }
    while (!ah_2_r.isFinished());
    List list_2 = page_2.getByXPath(
      "//ul[attribute::id='outputList']/li");
    System.err.println(i1 + " : " + i2);
    assertFalse(list_2.isEmpty());
    HtmlUnorderedList ul_2 = (HtmlUnorderedList) page_2.getByXPath(
      "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul_2.asXml().contains(message));
    assertTrue(ul_2.asXml().contains(nameString));
    assertTrue(webClient_2.getNumberOfAjaxErrorsWithoutWarnings() ==
                0);
  }
```

Table 5.14 testcase 2 - AjaxHandler improved

Using the improved AjaxHandler is very similar to the above testcase. There is one main difference. The XMLHttpRequest objects are used several times. When such an object is associated with an AjaxHandler that executes some assertions these assertions will be executed every time the XHR object is used. In this case the special AjaxHandler object parses the response from the receiver object whether it contains the just sent message. These assertions will fail every time except the one time our message really is sent. That’s why first a dummy AjaxHandler is initialized and gets associated with the XHR object that is responsible for receiving the new messages. In the lines sixty-six to sixty-seven the dummy AjaxHandler is removed and the special handler is registered so the XHR object takes the new handler for the next object. In this testcase a second browser is started and just assures that it really receives the message which was sent from the first browser.

6 Validation

This chapter validates the testcases from chapter 5 and explains why some test cases failed.

6.1 Wikipedia

In chapter 2.2.2.2 we already got to know Wikipedia and so we know that the AjaxController might have problems testing this application because it uses the JavaScript function “setTimeout”.

6.1.1 AjaxController

Test results:

Id	Item	Input	Output	pass/fail
1	suggest	“XMLHtt”	List including “XMLHttpRequest”	failed

Table 6.1 Wikipedia test result - AjaxController

Surprisingly the test fails when using the already built in AjaxController. For any reason the AjaxController couldn’t resynchronize the call. When adding a statement to wait some seconds after the AJAX request has been sent in the testcase it passes the test.

```
input.keyup();
Thread.currentThread.sleep(5000);   //new
// parse result and click on XMLHttpRequest Link
```

Table 6.2 additional code for the AjaxController

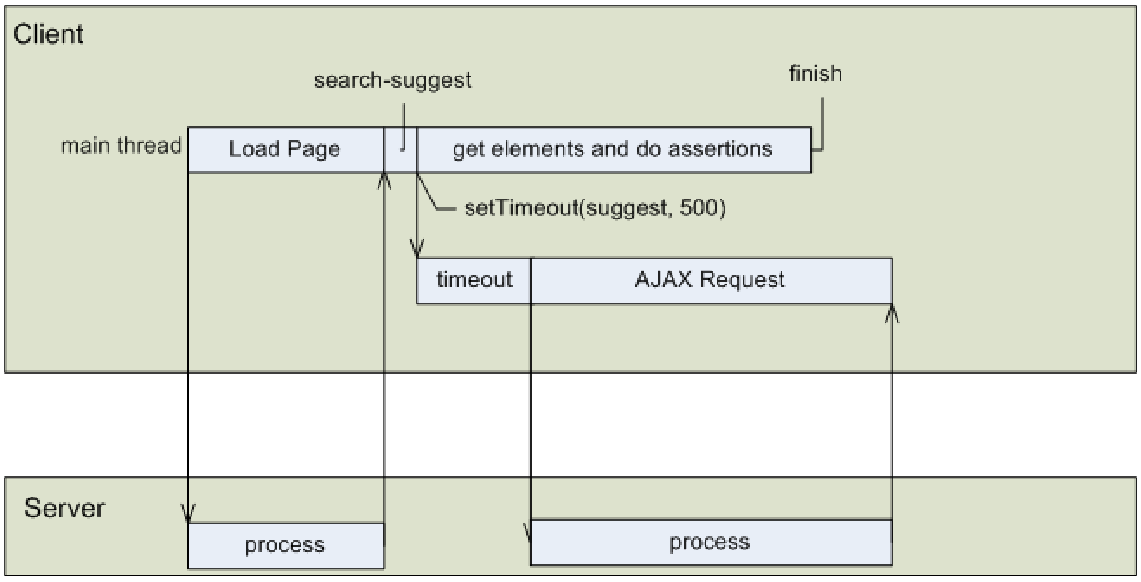


Figure 6.1 workflow of Wikipedia using AjaxController

Figure 6.1 shows why the testcase using the AjaxController failed. The AJAX request “search-suggest“ isn’t fired at once but the JavaScript function setTimeout is called which creates a new thread and executes the AJAX request after the given timeout. The AjaxController can’t resynchronize this request because it already runs in its own thread so the main thread continues parsing the elements and doing the assertions before the response is received. When implementing the workaround in the testcase the workflow is as follows:

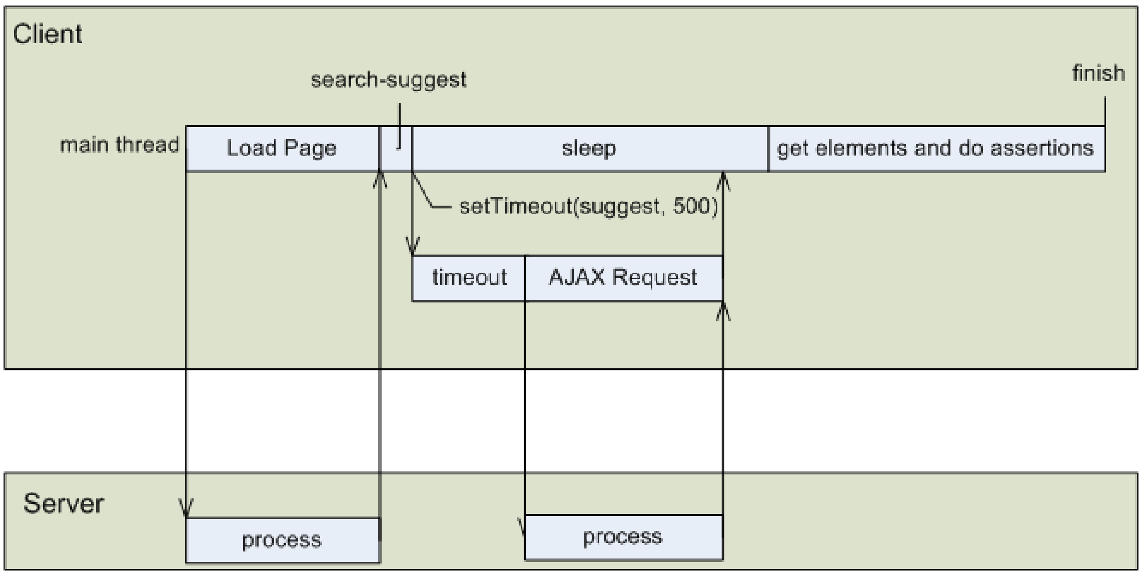


Figure 6.2 workflow of Wikipedia using AjaxController and a sleep statement

The time for pausing the main thread has to be long enough to be sure that the response has arrived. As communication over the internet can take some time in the peak time the break should last a few seconds although one second would be enough most of the times.

6.1.2 AjaxHandler

Test results:

Id	Item	Input	Output	pass/fail
1	suggest	“XMLHtt”	List including “XMLHttpRequest”	passed

Table 6.3 Wikipedia test result - AjaxHandler

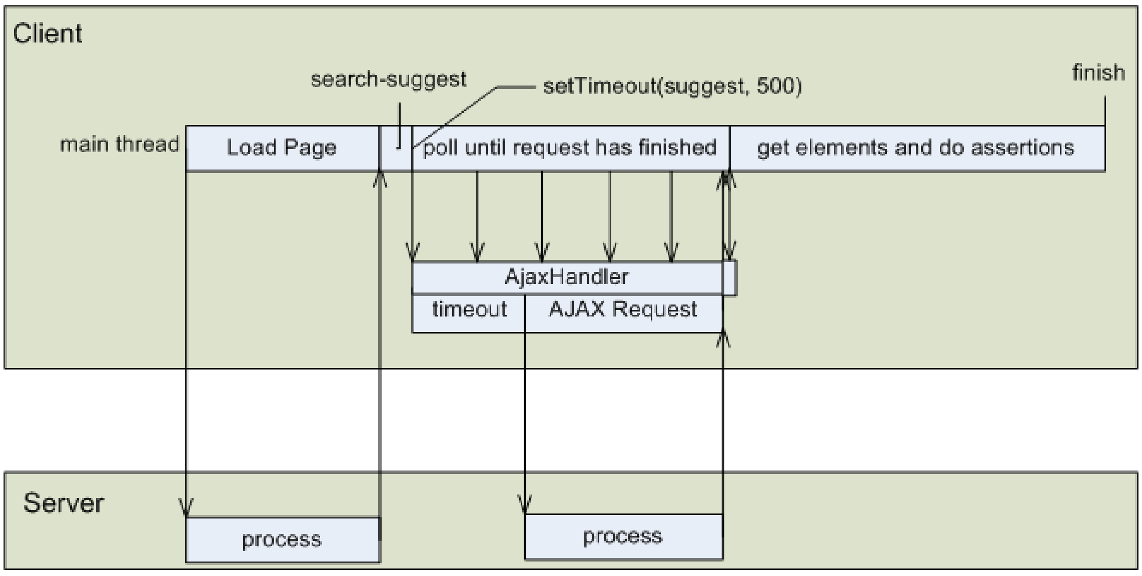


Figure 6.3 workflow of Wikipedia using AjaxHandler

Using the AjaxHandler handler solves the problem. The main thread polls the AjaxHandler as long as it signals that the response has finished. The main thread doesn’t continue exactly when the response arrives but the next time it polls for the status so it depends on the poll interval.

6.1.3 AjaxHandler - improved

Test results:

Id	Item	Input	Output	pass/fail
1	suggest	“XMLHtt”	List including “XMLHttpRequest”	passed

Table 6.4 Wikipedia test result - AjaxHandler improved

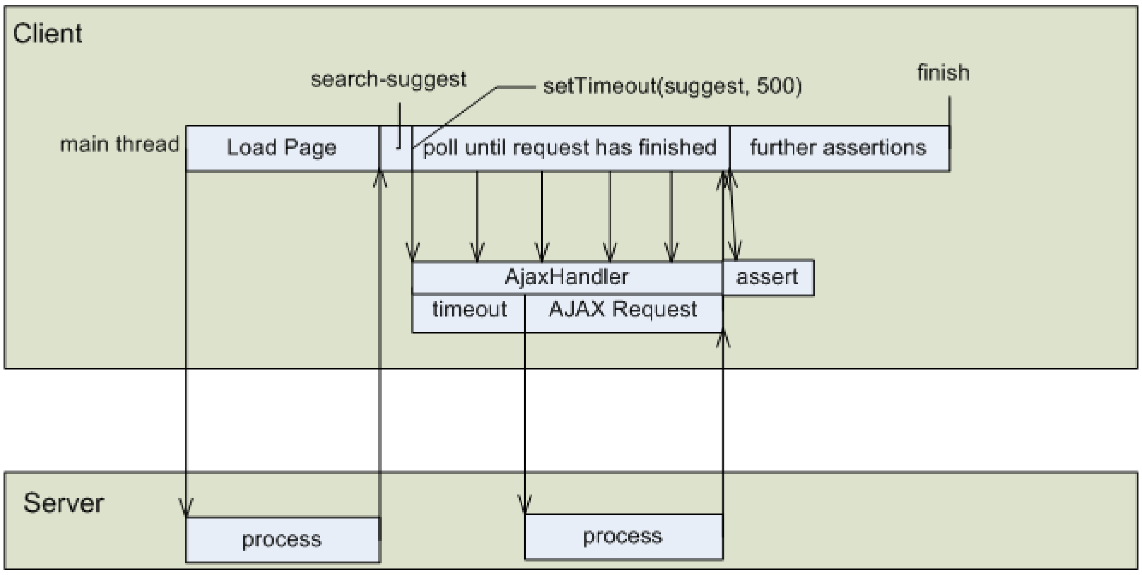


Figure 6.4 workflow of Wikipedia using the improved AjaxHandler

The improved AjaxHandler works very similar to the first one. The only difference is that the handler itself contains the assertions and executes them at the time when the response arrives. Further assertions are done in the main thread.

6.1.4 Summary

The AjaxController is the easiest way to deal with AJAX requests but it doesn't satisfy the needs. In this case it failed a test just because it wasn't able to cope with the request which was embedded in another JavaScript function. The AjaxHandler on the other side passed the test. The code of their testcases is a bit longer than the code of the AjaxController testcase but on the other hand it assures that the testcase is executed exactly how the tester intended.

6.2 AJAX Login System

The AJAX Login System doesn't use JavaScript code, which creates new threads, exept the XmlHttpRequest object, so all testcases should pass.

6.2.1 AjaxController

Test results:

Id	Item	Input	Output	pass/fail

1	log-in	username = “user1”
		password = “pass1”
			John Doe logged in	passed

2	log-out	press “logout”	log-in screen	passed

3	log-in	username = “user1”
		password = “wrongpass”
			invalid u/p combination	passed

4	log-in	username = “user” no password	no changes (no AJAX request)	passed*
5	log-in	no username password = “pass1”	no changes (no AJAX request)	passed*

Table 6.5: AJAX Login System test results - AjaxController

* page didn’t change

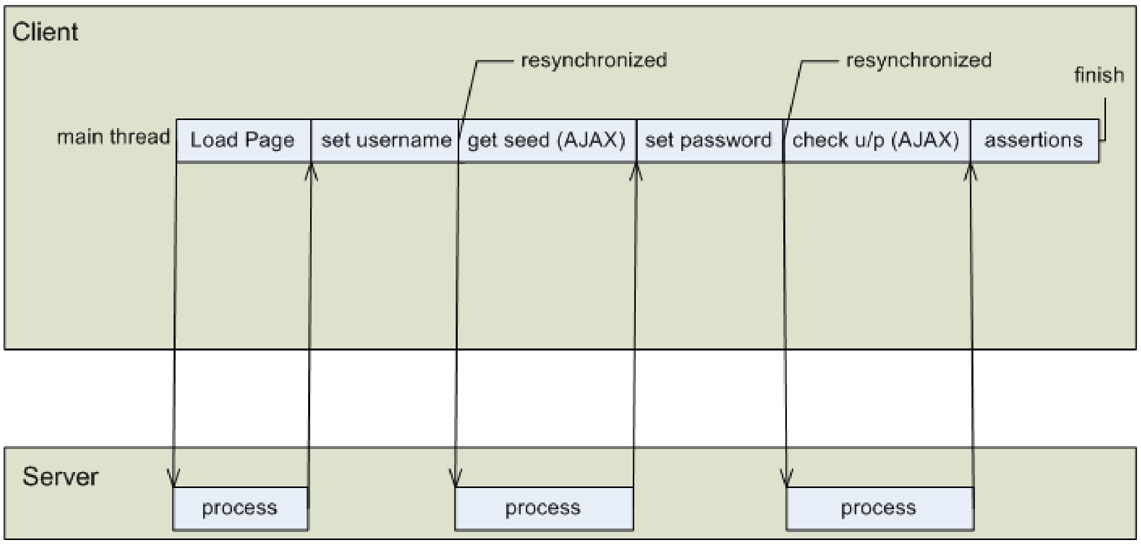


Figure 6.5 workflow of AJAX Login System using AjaxController

In this web application everything seems to work fine when using the AjaxController. In opposite to the Wikipedia test case this time the controller succeeds in resynchronizing the calls. Everything is executed in the main thread and all assertions pass because all responses have already arrived when executing them.
The testcases four and five also pass but there is no certainty that really no AJAX request has been sent. We can only prove that the web page hasn’t changed but it is also possible that there has been an AJAX request which just hasn’t changed anything on the page.

6.2.2 AjaxHandler

Test results:

Id	Item	Input	Output	pass/fail

1	log-in	username = “user1”
		password = “pass1”
			John Doe logged in	passed

2	log-out	press “logout”	log-in screen	passed

3	log-in	username = “user1”
		password = “wrongpass”
			invalid u/p combination	passed

4	log-in	username = “user” no password	no changes (no AJAX request)	passed*
5	log-in	no username password = “pass1”	no changes (no AJAX request)	passed*

Table 6.6 AJAX Login System test results – AjaxHandler

* page didn’t change

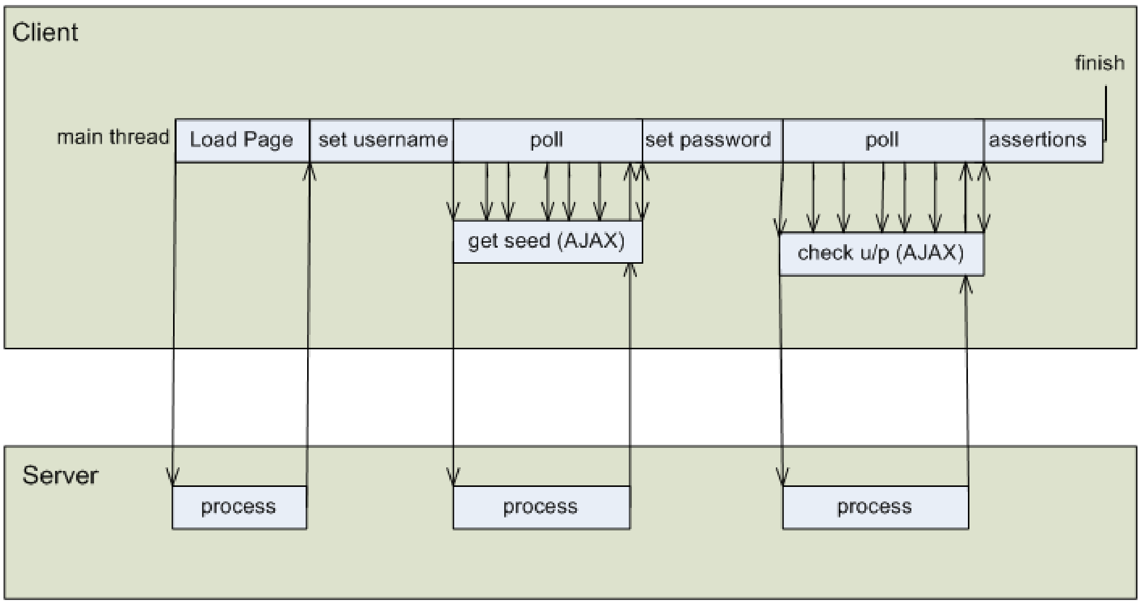


Figure 6.6 workflow of AJAX Login System using AjaxHandler

In this case the AjaxHandler is just used to copy the functionality of the AjaxController. This is again done by polling for the response. This time the execution time of the AjaxHandler is a bit longer than the execution time of the AjaxController but the difference amounts only a few milliseconds and can be neglected.

6.2.3 AjaxHandler – improved

Test results:

Id	Item	Input	Output	pass/fail

1	log-in	username = “user1”
		password = “pass1”
			John Doe logged in	failed

2	log-out	press “logout”	log-in screen	failed

3	log-in	username = “user1”
		password = “wrongpass”
			invalid u/p combination	failed

4	log-in	username = “user” no password	no changes (no AJAX request)	passed
5	log-in	no username password = “pass1”	no changes (no AJAX request)	passed

Table 6.7 AJAX Login System test results – AjaxHandler improved

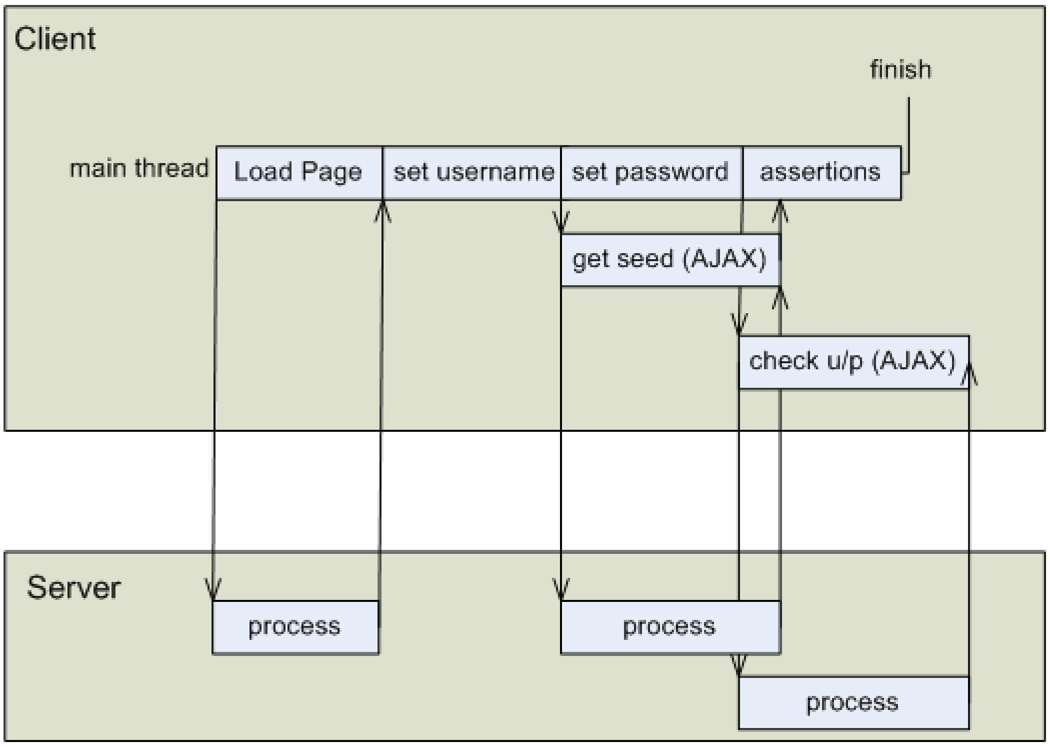


Figure 6.7 workflow of AJAX Login System using the improved AjaxHandler

In this test case we try to implement everything without synchronizing or polling. The first three testcases fail and when looking at Figure 6.7 it is quite clear why. The assertions are done before all responses have arrived so at least at the end of the testcase we need to poll for the response of the check. When doing this the first three testcases still don’t work.

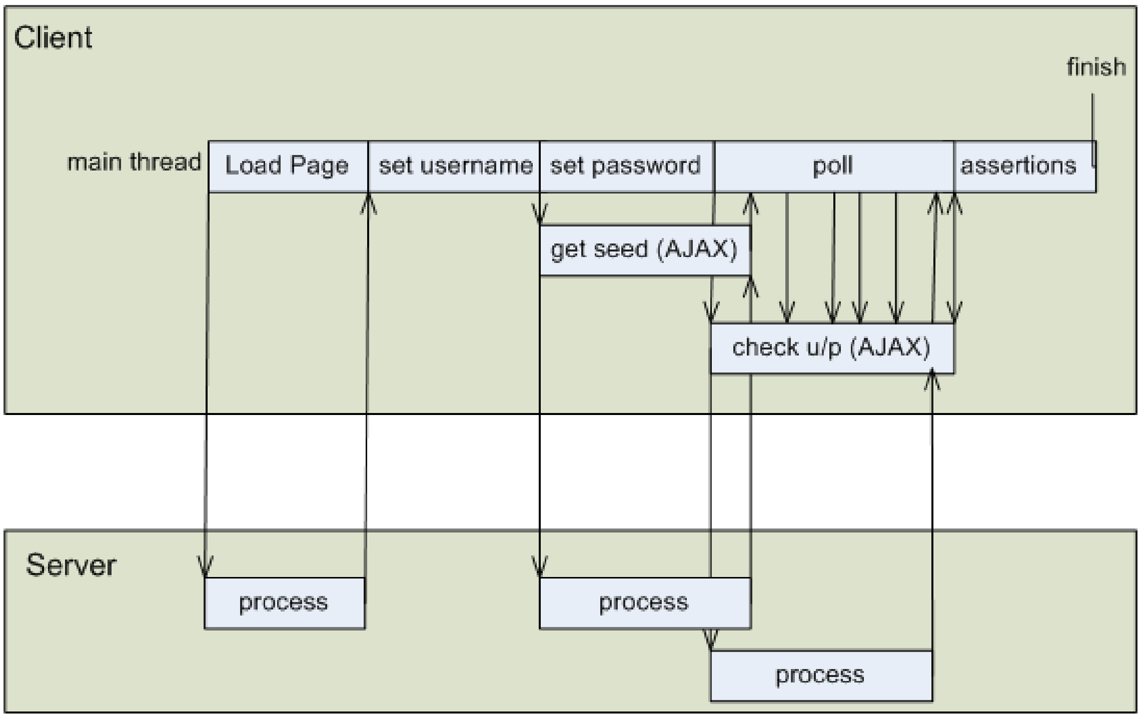


Figure 6.8 corrected workflow of AJAX Login System using the improved AjaxHandler

An assertion error occurs inside the AjaxHandler. The response from the server isn't "OK; John Doe" but "Unknown error (hacking attempt)". So why do we get an error message when using our new AjaxHandler while the old AjaxController works fine? The answer is obvious. This time we didn't resynchronize the AJAX calls so the failure has something to do with the multithreaded execution. The best way to find the reason is to take a look at the error messages. Two messages have been stored:

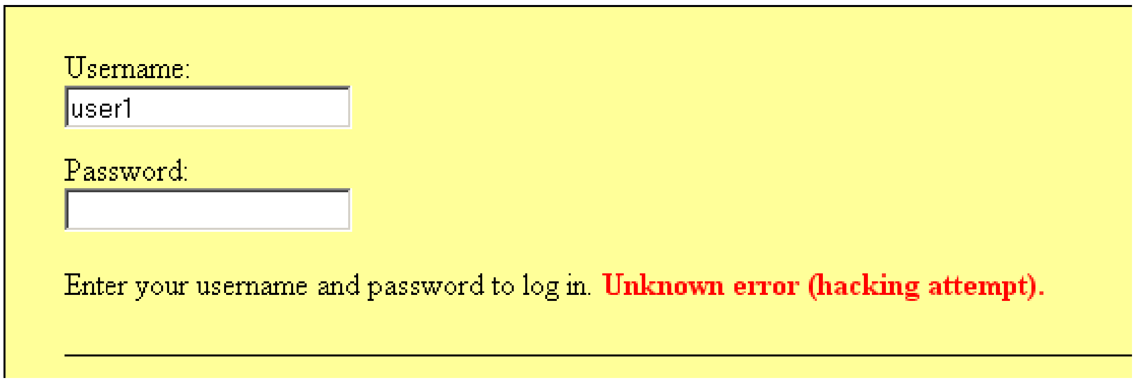
```
AJAX WARNING: ajax request http://www.jamesdam.com/ajax_login/
login.php?task=getseed was interrupted!
URL: new url: login.php?task=checklogin&username=user1
&id=0&hash=2764bc1c2acd79685d987880344f1261
AJAX ERROR: null
URL: http://www.jamesdam.com/ajax_login/
login.php?task=checklogin&username=user1&id=0&
hash=2764bc1c2acd79685d987880344f1261
```

Table 6.8 error messages from AJAX Login System

The first one is a warning that the first AJAX request which tried to get a random seed was interrupted by another AJAX request which tries to authenticate the user.

The second message is an assertion error when analyzing the response from the second AJAX request.

The problem occurred because the authentication has been started before receiving the random seed which is needed for encrypting the password. Indeed the failure can be reproduced on the web page. When the user types in his username and password really fast the web page returns the following message:



Username:

user1

Password:

Enter your username and password to log in. Unknown error (hacking attempt).

Figure 6.9 error message from the AJAX Login System

This error shows that simplifying the workflow can prevent the program to find errors. The testcases four and five are really easy to implement using the AjaxHandler. The tester just has to assure that the AjaxHandlers stay inactive.

6.2.4 Summary

This testcases demonstrate further weaknesses of the AjaxController. Firstly it isn't possible to implement all testcases and secondly it isn't possible to find the software error in this web application. Using the new AjaxHandler eliminates all weaknesses of the AjaxController so far.

6.3 XHTML Live Chat

The XHTML Live Chat also uses the JavaScript function "setTimeout" but this time we already added the necessary sleep command when using the AjaxController so all test cases should pass.

6.3.1 AjaxController

Id	Item	Input	Output	pass/fail
1	post & receive	Post message "AJAX Test" with user name "Andi"	The posted message	passed
2	post & multiple receive	Start two web browsers. Post message "AJAX Test 2" with user name "Andi" from browser one.	The posted message is shown on both browsers	passed

Table 6.9 XHTML Live Chat test results - AjaxController

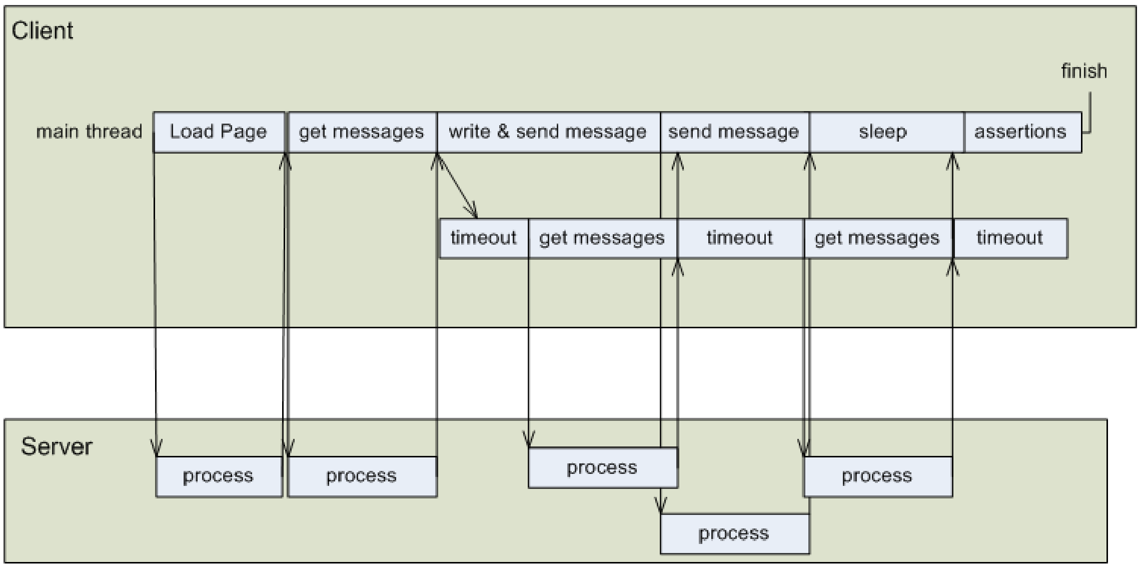


Figure 6.10 workflow of the XHTML Live Chat using AjaxController

The AjaxController isn't able to resynchronize periodically executed AJAX request so the main thread has to sleep some time to wait for the responses. The first AJAX request for receiving the messages can be resynchronized but after this one the program uses the JavaScript function setTimeout(). The other AJAX request which is responsible for sending the message can always be resynchronized.

6.3.2 AjaxHandler

Id	Item	Input	Output	pass/fail
1	post & receive	Post message "AJAX Test" with user name "Andi"	The posted message	passed
2	post & multiple receive	Start two web browsers. Post message "AJAX Test 2" with user name "Andi" from browser one.	The posted message is shown on both browsers	passed

Table 6.10 XHTML Live Chat test results - AjaxHandler

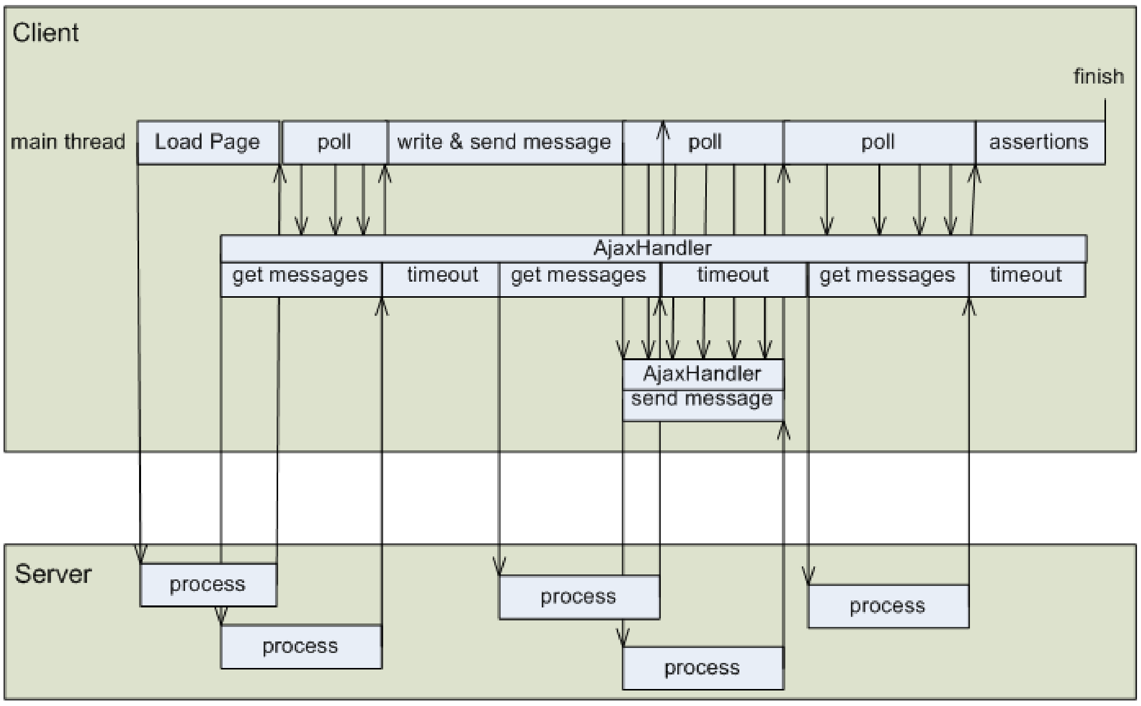


Figure 6.11 workflow of the XHTML Live Chat using AjaxHandler

The main thread synchronizes three times with the handlers. The first time it waits until all initial messages have been received. The second time it observes the sending process and afterwards waits for the next process that catches the just sent message. All test cases pass.

6.3.3 AjaxHandler – improved

Id	Item	Input	Output	pass/fail
1	post & receive	Post message “AJAX Test” with user name “Andi”	The posted message	passed
2	post & multiple receive	Start two web browsers. Post message “AJAX Test 2” with user name “Andi” from browser one.	The posted message is shown on both browsers	passed

Table 6.11 XHTML Live Chat test results - AjaxHandler improved

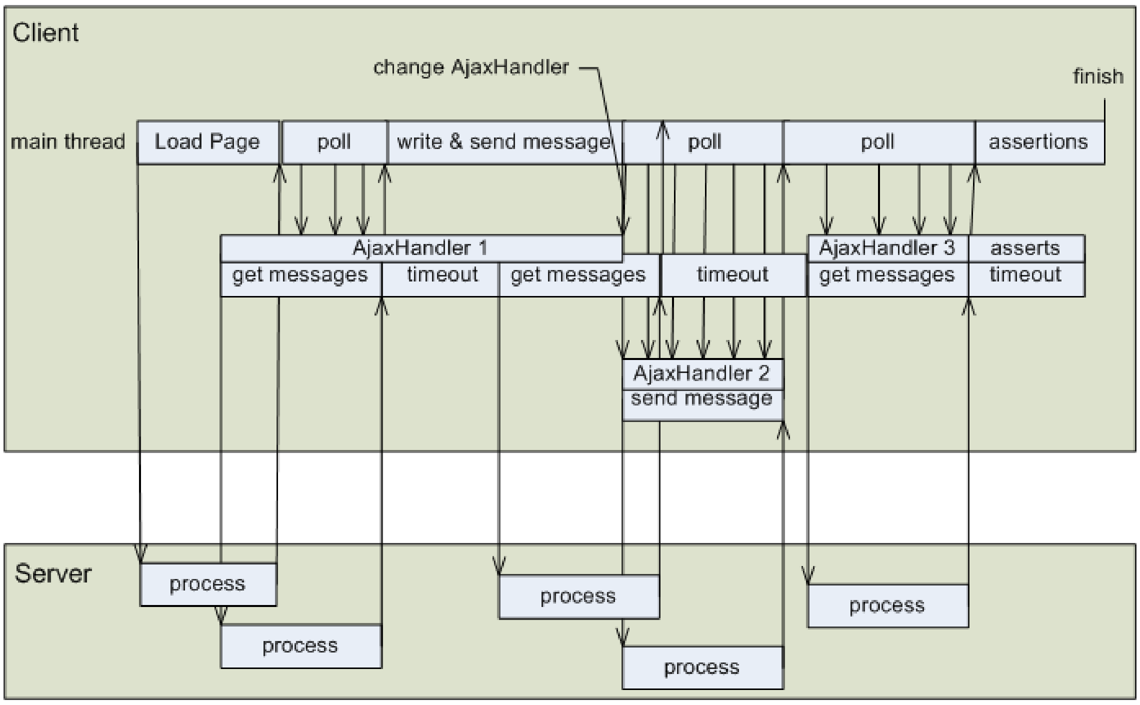


Figure 6.12 workflow of the XHTML Live Chat using the improved AjaxHandler

The improved AjaxHandler can’t be associated with the XMLHttpRequest object from the beginning because this would cause assertion errors the first 2 times it is used. That’s why first a default AjaxHandler is initialized and gets replaced by the improved one just after the “send message” button is pressed.

6.4 Summary

These use cases demonstrate many advantages of the AjaxHandler compared to the AjaxController. Of course these use cases weren’t chosen by random by they cover a very big scope of using AJAX. The web applications tested are very small applications because it is easier to explain issues when the complexity stays low but even big web applications using AJAX like Google docs use the same techniques and just have a lot more local JavaScript code.
The following techniques were used:

- embedded AJAX requests in JavaScript code
- periodically executed AJAX requests
- AJAX requests using data from former AJAX requests
- static XMLHttpRequest objects
- dynamic XMLHttpRequest objects

The AjaxContoller has been developed by the HtmlUnit team to add at least some AJAX functionality to the test suite but the use cases showed up several weaknesses:

- some tests failed although the web application worked correct
- some test passed although there was a software fault in the application

- tests normally need longer execution time compared to manual tests or tests using the AjaxHandler because of many sleep statements
- some tests can’t be implemented using just HtmlUnit with the AjaxController

In one case the execution time of the AjaxController was faster than the execution time of the AjaxHandler that just copied its functionality but that was only a matter of a few milliseconds and after all this testcase passed although it should have failed because of an application error.

The AjaxHandler can be used in different ways. The first one is to copy the functionality of the AjaxController which means to poll till the response arrives and which is only another form of resynchronizing the calls. Although these calls are executed in their own thread it is technically the same as using the AjaxController except of one thing: the AjaxHandler always works.

The second way to use the AjaxHandler is to override some methods of the class and to add assertions or create debug information of the tests. This is the preferred way because it doesn’t resynchronize the AJAX calls.

7 Conclusion and Future Work

The aim of this paper was to extend HtmlUnit to be able to test internet applications that use AJAX which was achived by the AjaxHandler. The solution sounds quite simple: the AjaxHandler provides the tester direct access to the XMLHttpRequest object which is part of all AJAX requests and enables the tester to use this object to get additional information for comparisions or to do some synchronization between the threads. The validation showed that using just simple tricks like waiting a certain amount of time or resynchronizing AJAX calls isn't enough to handle AJAX applications and only web testing frameworks that provide access to the XMLHttpRequest object or at least provide some information about this object, are able to handle all internet applications using AJAX. However HtmlUnit is a stand alone web browser and, as in all other web browsers, there can be compatibility issues with some web pages especially when they use non standard conform javascript code. The tester needs to have basic knowledge about the application he is testing to be able to use the AjaxHandler correctly, because he has to know which functions use AJAX and how many XmlHttpRequest objects are used by the web application.

The old AjaxContoller has been developed by the HtmlUnit team to add at least some AJAX functionality to the test suite but the case studies showed up several weaknesses, which were corrected by the new AjaxHandler:

- some tests failed although the web application worked correct
- some tests passed although there was a software fault in the application
- tests normally need longer execution time compared to manual tests or tests using the AjaxHandler because of many sleep statements
- some tests can't be implemented using just HtmlUnit with the AjaxController

The AjaxHandler satisfies all criteria specified in chapter 3.1. It makes the implementation of test cases a bit more complex but this isn't a surprise because web applications using AJAX are simply more complex than applications that do not use this technique.

The extension presented in this work hasn't been integrated into the HtmlUnit project yet, but there will be some discussion with the project team about what to do with the extension. The best thing would be a complete integration of the extension while in the worst case the extension would stay in a separate, poorly maintained package. Another solution would be to integrate a modified version, which doesn't modify the WebClient that much, into the project.

This work presents only a very basic AjaxHandler which is rather an interface than a class because most methods are empty and have to be overwritten by the tester. Future versions could extend this class and provide a collection of methods that can automatically extract the XML payload or provide a callback interface to get rid of polling for the result.

Bibliography

[1] Alexander Jesse, “JSFUnit Introduction”, Jazoon 08 International Conference on Java Technology, June 23-26, 2008, Zurich

[2] Alessandro Marchetto, Filippo Ricca, Paolo Tonella, "A case study-based comparison of web testing techniques applied to AJAX web applications", International Journal on Software Tools for Technology Transfer, Volume 10, Issue 6, Pages 477-492, 2008

[3] Di Lucca, G.A., "Testing Web-based applications: the state of the art and future trends", Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, Volume 2, Pages: 65 - 69, July 2005

[4] Hung Nguyen, "Testing Web-based Applications", STQE Magazine Issue: May/Jun 2000 (Vol. 2 Issue 3)

[5] Whittaker, J.A., "What is software testing? And why is it so hard?", Software, IEEE, Volume 17, Issue 1, Pages:70 - 79, Jan/Feb 2000

[6] Fewster Mark, Graham Dorothy, “Software Test Automation: Effective use of test execution tools”, ACM Press Addison-Wesley, June 1999

[7] Douglas Hoffman, “Cost Benefit Analysis of Test Automation”, Software Quality Methods, STARW ‘99, 1999

[8] Kaner, C. “Pitfalls and strategies in automated testing”, Computer Magazine, Vol. 30, April 1997

[9] Kung, D.C.; Chien-Hung Liu; Pei Hsia “ An Object-Oriented Web Test Model for Testing Web Applications”, First Asia-Pacific Conference on Quality Software, Proceedings, 2000, Pages: 111-120

[10] G.A. Di Lucca, A.R. Fasolino, F. Faralli, U. De Carlini, "Testing Web applications", International Conference on Software Maintenance, 2002 Proceedings, 2002, Pages: 310 - 319

[11] Weyuker, E.J. “Can we measure software testing effectiveness”, First International Software Metrics Symposium, Proceedings, 21-22 May 1993, Pages: 100 – 107

[12] Ali Mesbah, Arie van Deursen, “ Migrating Multi-page Web Applications to Single-page AJAX Interfaces”, CSMR Proceedings of the 11th European Conference on Software Maintenance and Reengineering, 2007, Pages: 181-190

[13] Elaine J. Weyuker, Stewart N. Weiss, Dick Hamlet, “Comparison of program testing strategies”, International Symposium on Software Testing and Analysis, 1991, Pages: 1-10

[14] Smith, K., “Simplifying Ajax-style Web development”, Computer Magazine, Vol. 39, Issue 5, May 2006, Pages: 98 – 101

[15] Filippo Ricca, Paolo Tonella, “Analysis and testing of Web applications”, Proceedings of the 23rd International Conference on Software Engineering, 2001, Pages: 25 – 34

[16] Stan Silvert, "Testing JSF Applications", JBoss World, Orlando 2008

[17] Ji-Tzay Yang, Jiun-Long Huang, Feng-Jian Wang, William C. Chu, "An Object-Oriented Architecture Supporting Web Application Testing", 23rd International Computer Software and Applications Conference, 1999, Page: 122

[18] Lei Xu, Baowen Xu, Zhenqiang Chen, Jixiang Jiang, Huowang Chen, "Regression testing for Web applications based on slicing", Computer Software and Applications Conference, COMPSAC 2003, Proceedings, 27th Annual International, 3-6 Nov. 2003 Pages: 652 – 656

[19] Ali Mesbah, Arie van Deursen, "An Architectural Style for Ajax", WICSA, Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture, 2007, Page 9

[20] Sreedevi Sampath , Sara Sprenkle , Emily Gibson , Lori Pollock , Amie Souter Greenwald, "Applying Concept Analysis to User-Session-Based Testing of Web Applications", IEEE Transactions on Software Engineering, v.33 n.10, p.643-658, October 2007

[21] Mohammadreza Mollahoseini Ardakani , Mohammad Morovvati, "A multi-agent system apporach for user-session-based testing of web applications", Proceedings of the 7th WSEAS international conference on Distance learning and web engineering, p.326-331, September 15-17, 2007, Beijing, China

[22] Filippo Ricca , Paolo Tonella, "Web application quality: supporting maintenance and testing", Information modeling for internet applications, Idea Group Publishing, Hershey, PA, 2003

[23] Jessica Sant , Amie Souter , Lloyd Greenwald, "An exploration of statistical models for automated test case generation", ACM SIGSOFT Software Engineering Notes, v.30 n.4, July 2005

[24] Sreedevi Sampath , Sara Sprenkle , Emily Gibson , Lori Pollock, "Integrating customized test requirements with traditional requirements in web application testing", Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications, p.23-32, July 17-17, 2006, Portland, Maine

[25] Filippo Ricca , Paolo Tonella, "Testing Processes of Web Applications", Annals of Software Engineering, v.14 n.1-4, p.93-114, December 2002

[26] William G. J. Halfond , Alessandro Orso, "Improving test case generation for web applications using automated interface discovery", Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, September 03-07, 2007, Dubrovnik, Croatia

[27] Yuetang Deng , Phyllis Frankl , Jiong Wang, "Testing web database applications", ACM SIGSOFT Software Engineering Notes, v.29 n.5, September 2004

[28] Sebastian Elbaum , Srikanth Karre , Gregg Rothermel, "Improving web application testing with user session data", Proceedings of the 25th International Conference on Software Engineering, May 03-10, 2003, Portland, Oregon

[29] Gary Wassermann , Dachuan Yu , Ajay Chander , Dinakar Dhurjati , Hiroshi Inamura , Zhendong Su, Dynamic test input generation for web applications, Proceedings of the 2008 international symposium on Software testing and analysis, July 20-24, 2008, Seattle, WA, USA

[30] Carlo Bellettini , Alessandro Marchetto , Andrea Trentini, “TestUml: user-metrics driven web applications testing”, Proceedings of the 2005 ACM symposium on Applied computing, March 13-17, 2005, Santa Fe, New Mexico

[31] Shay Artzi , Adam Kiezun , Julian Dolby , Frank Tip , Danny Dig , Amit Paradkar , Michael D. Ernst, “Finding bugs in dynamic web applications”, Proceedings of the 2008 international symposium on Software testing and analysis, July 20-24, 2008, Seattle, WA, USA

[32] C. Stringfellow , Z. Kurunthottical, “Analysis of the effectiveness of students' test data”, Journal of Computing Sciences in Colleges, v.21 n.4, p.223-229, April 2006

[33] Jianhua Hao , Emilia Mendes, “Usage-based statistical testing of web applications”, Proceedings of the 6th international conference on Web engineering, July 11-14, 2006, Palo Alto, California, USA

[34] Larson, J.: Testing ajax applications with selenium. InfoQ magazine (2006)

[35] Lei Xu, Baowen Xu, Jixiang Jiang, “Testing web applications focusing on their specialties”, ACM SIGSOFT Software Engineering Notes, Vol 30 , Issue 1, January 2005

[36] Rudolf Ramler, Edgar Weippl, Mario Winterer, Wieland Schwinger, Josef Altmann, "A quality-driven approach to web testing", Proceedings of ICWE’02 Conference, Santa Fe, Argentinia, September 2002, pages 81-95

[37] Lihua Ran , Curtis Dyreson , Anneliese Andrews , Renée Bryce , Christopher Mallery, Building test cases and oracles to automate the testing of web database applications, Information and Software Technology, v.51 n.2, p.460-477, February, 2009

[38] Paolo Tonella , Filippo Ricca, A 2-Layer Model for the White-Box Testing of Web Applications, Proceedings of the Web Site Evolution, Sixth IEEE International Workshop on (WSE'04), p.11-19, September 11-11, 2004

Websites

[39] The XMLHttpRequest object: http://www.w3.org/TR/XMLHttpRequest/ (10.03.2009), 2008

[40] Brett McLaughlin: Ajax meistern, Teil 1: http://www.oreilly.de/artikel/ajax1/index.html (10.03.2009), 2005

[41] HtmlUnit: http://htmlunit.sourceforge.net/ (10.03.2009), 2008

[42] JSFUnit: http://www.jboss.org/jsfunit/ (10.03.2009), 2009

[43] RichFaces Demo: http://jsfunit.demo.jboss.com/jboss-jsfunit-examples-richfaces/ (10.03.2009), 2008

[44] Wikipedia: http://www.wikipedia.at/ (10.03.2009), 2006

[45] AJAX Login System: http://www.jamesdam.com/ajax_login/login.html (10.03.2009), 2005

[46] XHTML Live Chat: http://chat.plasticshore.com/ (10.03.2009), 2008

[47] nessus – the network vulnerability scanner: http://www.nessus.org/nessus/

[48] AJAX: http://en.wikipedia.org/wiki/AJAX (10.03.2009), 2009

[49] Apache 2 License: http://www.apache.org/licenses/LICENSE-2.0.html (10.03.2009), 2004

[50] HtmlUnit blog: http://mguillem.wordpress.com/category/htmlunit/ (10.03.2009), 2009

Appendix

7.1 New Classes

7.1.1 AjaxError

```java
package com.gargoylesoftware.htmlunit;

/**
 * <p>Description: model for storing ajax error messages</p>
 *
 * <p>Copyright: Copyright (c) 2008</p>
 *
 * <p>Organisation: TU Wien</p>
 *
 * @author Andreas Langer
 * @author Mario Bernhart
 * @author Thomas Grechenig
 * @version 1.0
 */
public class AjaxError {
  public static final int WARNING = 1;
  public static final int EXCEPTION = 2;
  public static final int ERROR = 3;
  private String newline = System.getProperty("line.separator");
  private int type;
  private String description;
  private String url;
  /**
   *
   * @param type int type of the ajax error; possible values: AjaxError.WARNING,
   * AjaxError.EXCEPTION, AjaxError.ERROR
   * @param description String description of the error
   * @param url String request url of the ajax request
   */
  public AjaxError(int type, String description, String url) {
    this.type = type;
    this.description = description;
    this.url = url;
  }

  /**
   * returns the type of the error
   * @return int
   */
  public int getType() {
    return type;
  }

  /**
   * returns the description of the error
   * @return String
   */
  public String getDescription() {
    return description;
  }

  /**
   * returns the url of the ajax request
   * @return String
   */
  public String getUrl() {
    return url;
  }

  /**
   * builds and returns a string containing all information about the error
   * @return String
   */
  public String toString() {
    switch (type) {
      case WARNING:
        return "AJAX WARNING: " + description + newline +
            "URL: " +
            url;
      case EXCEPTION:
        return "AJAX EXCEPTION: " + description + newline +
            "URL: " +
            url;
      case ERROR:
        return "AJAX ERROR: " + description + newline + "URL: " +
            url;
      default:
        return "INTERNAL ERROR! " + description + newline +
            "URL: " +
            url;
    }
  }
}
```

7.1.2 AjaxHandler

```
package com.gargoylesoftware.htmlunit;

import java.io.Serializable;
import com.gargoylesoftware.htmlunit.javascript.host.
    XMLHttpRequest;

/**
 *
 * <p>Headline: AjaxHandler</p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2008</p>
 *
 * <p>Organisation: TU Wien</p>
 *
 * @author Andreas Langer
 * @author Mario Bernhart
 * @author Thomas Grechenig
 * @version 1.0
 */
public class AjaxHandler
```

```
    implements Serializable {

  private static final long serialVersionUID =
      2170842485774504546L;
  public static final int ON_ACTIVATION = 1;
  public static final int ON_CHANGE = 2;
  public static final int ON_FINISH = 3;
  public static final int ON_ERROR = 4;
  private XMLHttpRequest xml;
  private boolean finished;
  private WebClient webClient;

  public AjaxHandler() {
    finished = true;
  }

  public boolean used() {
    return (xml != null);
  }

  public void callFunction(int type, int status) {
    try {
      switch (type) {
        case ON_ACTIVATION:
          onActivation();
          break;
        case ON_CHANGE:
          onChange(status);
          break;
        case ON_ERROR:
          onError();
          break;
        case ON_FINISH:
          onFinish();
          break;
      }
    }
    catch (junit.framework.AssertionFailedError ex) {
      String url = xml.getRequestSettings().getUrl().
          toExternalForm() +
          "  " + xml.getRequestSettings().getRequestBody();
      ex.printStackTrace();
      webClient.addAjaxError(AjaxError.ERROR, ex.getMessage(),
                             url);
    }
    catch (Exception ex) {
      String url = xml.getRequestSettings().getUrl().
          toExternalForm() +
          "  " + xml.getRequestSettings().getRequestBody();
      ex.printStackTrace();
      webClient.addAjaxError(AjaxError.EXCEPTION, ex.getMessage(),
                             url);
    }
  }

  public WebRequestSettings callBeforeSend(WebRequestSettings
                                           wrs) {
    try {
      return beforeSend(wrs);
    }
```

```
   catch (junit.framework.AssertionFailedError ex) {
     String url = xml.getRequestSettings().getUrl().
         toExternalForm() +
         "  " + xml.getRequestSettings().getRequestBody();
     ex.printStackTrace();
     webClient.addAjaxError(AjaxError.ERROR, ex.getMessage(),
                            url);
   }
   catch (Exception ex) {
     String url = xml.getRequestSettings().getUrl().
         toExternalForm() +
         "  " + xml.getRequestSettings().getRequestBody();
     ex.printStackTrace();
     webClient.addAjaxError(AjaxError.EXCEPTION, ex.getMessage(),
                            url);
   }
   return wrs;
  }

  public void onActivation() {
  }

  public void onChange(int status) {
    System.err.println("!!!!!!!!!!!!!!!CHANGE!!!!!!!!!!!!: " +
                       status);
  }

  public void onError() {
  }

  public void onFinish() {
  }

  public WebRequestSettings beforeSend(WebRequestSettings wrs) {
    System.err.println(wrs.getUrl().toExternalForm() + "  " +
                       wrs.getRequestBody());
    return wrs;
  }

  public void setFinished(boolean fin) {
    finished = fin;
    if (fin)
      callFunction(ON_FINISH, 0);
  }

  public boolean isFinished() {
    return finished;
  }

  public int getState() {
    return xml.jsxGet_readyState();
  }

  public String getResponseText() {
    System.err.println(xml.jsxGet_responseText());
    return xml.jsxGet_responseText();
  }

  public Object getResponseXML() {
    return xml.jsxGet_responseXML();
  }

  public String getResponseHeader() {
    return xml.jsxFunction_getAllResponseHeaders();
  }

  public void setXMLHttpRequest(XMLHttpRequest x) {
    xml = x;
    callFunction(ON_ACTIVATION, 0);
  }

  public XMLHttpRequest getXMLHttpRequest() {
    return xml;
  }

  public void removeHandler() {
    if (xml != null) {
      xml.removeAjaxHandler();
      xml = null;
    }
    if (!finished) {
      String url = xml.getRequestSettings().getUrl().
          toExternalForm() +
          "  " + xml.getRequestSettings().getRequestBody();
      webClient.addAjaxError(AjaxError.WARNING,
                             "removing unfinished ajax handler!",
                             url);
      finished = true;
    }
  }

  public void setWebClient(WebClient wc) {
    webClient = wc;
  }
}
```

7.2 *Modified Classes*

7.2.1 XMLHttpRequest

```
/*
 * Copyright (c) 2002-2008 Gargoyle Software Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing
```

```
permissions and
 * limitations under the License.
 */
package com.gargoylesoftware.htmlunit.javascript.host;

import java.io.*;
import java.net.*;
import java.util.*;

import org.apache.commons.httpclient.*;
import org.apache.commons.lang.*;
import org.mozilla.javascript.*;
import com.gargoylesoftware.htmlunit.*;
import com.gargoylesoftware.htmlunit.HttpMethod;
import com.gargoylesoftware.htmlunit.html.*;
import com.gargoylesoftware.htmlunit.javascript.*;
import com.gargoylesoftware.htmlunit.util.*;
import com.gargoylesoftware.htmlunit.xml.*;

/**
 * A JavaScript object for a XMLHttpRequest.
 *
 * @version $Revision: 3129 $
 * @author Daniel Gredler
 * @author Marc Guillemot
 * @author Ahmed Ashour
 * @author Stuart Begg
 * @author Andreas Langer
 * @author Mario Bernhart
 * @author Thomas Grechenig
 * @see <a href="http://developer.apple.com/internet/webcontent/xmlhttpreq.html">Safari documentation</a>
 */
public class XMLHttpRequest
    extends SimpleScriptable {

  private static final long serialVersionUID =
      2369039843039430664L;

  /** The object has been created, but not initialized (the open() method has not been called). */
  public static final int STATE_UNINITIALIZED = 0;
  /** The object has been created, but the send() method has not been called. */
  public static final int STATE_LOADING = 1;
  /** The send() method has been called, but the status and headers are not yet available. */
  public static final int STATE_LOADED = 2;
  /** Some data has been received. */
  public static final int STATE_INTERACTIVE = 3;
```

```java
  /** All the data has been received; the complete data is available in responseBody and responseText. */
  public static final int STATE_COMPLETED = 4;

  private int state_;
  private Function stateChangeHandler_;
  private Function errorHandler_;
  private WebRequestSettings requestSettings_;
  private boolean async_;
  private int threadID_;
  private WebResponse webResponse_;
  private String overriddenMimeType_;
  private HtmlPage containingPage_;
  //private boolean enabledAjaxHandler;
  private AjaxHandler ah;

  /**
   * Creates a new instance. JavaScript objects must have a default constructor.
   */
  public XMLHttpRequest() {
    state_ = STATE_UNINITIALIZED;

  }

  public void removeAjaxHandler() {
    ah = null;
  }

  /**
   * JavaScript constructor.
   */
  public void jsConstructor() {
    // Empty.
  }

  /**
   * Returns the event handler that fires on every state change
   * @return the event handler that fires on every state change
   */
  public Function jsxGet_onreadystatechange() {

    return stateChangeHandler_;
  }

  /**
   * Sets the event handler that fires on every state change.
   * @param stateChangeHandler the event handler that fires on every state change
   */
  public void jsxSet_onreadystatechange(final Function
                                        stateChangeHandler) {
    stateChangeHandler_ = stateChangeHandler;
    if (state_ == STATE_LOADING) {
      setState(state_, null);
    }
  }

  /**
   * Sets the state as specified and invokes the state change handler if one has been set.
   * @param state the new state
   * @param context the context within which the state change handler is to be invoked;
   *                if <tt>null</tt>, the current thread's context is used.
   */
  private void setState(final int state, Context context) {
    state_ = state;
    /*if (ah != null){
      ah.onChange(state);
             }*/
    //Firefox doesn't trigger onreadystatechange handler for sync requests
    final boolean isIE = getBrowserVersion().isIE();
    if (stateChangeHandler_ != null && (isIE || async_)) {
      if (context == null) {
        context = Context.getCurrentContext();
      }
      final Scriptable scope = stateChangeHandler_.
          getParentScope();
      final JavaScriptEngine jsEngine = containingPage_.
          getWebClient().
          getJavaScriptEngine();

      final int nbExecutions;
      if (async_ && STATE_LOADING == state) {
        // quite strange but IE and Mozilla seem both to fire state loading twice
        // in async mode (at least with HTML of the unit tests)
        nbExecutions = 2;
      }
      else {
        nbExecutions = 1;
      }

      for (int i = 0; i < nbExecutions; i++) {
        getLog().debug(
            "Calling onreadystatechange handler for state " +
            state);
        jsEngine.callFunction(containingPage_,
                              stateChangeHandler_,
                              context,
                              this, scope,
                              ArrayUtils.EMPTY_OBJECT_ARRAY);
        getLog().debug("onreadystatechange handler: " +
                       context.
                           decompileFunction(stateChangeHandler_,
4));
        getLog().debug(
            "Calling onreadystatechange handler for state " +
            state +
            ". Done.");
      }
    }

    if (ah != null) {
      ah.callFunction(AjaxHandler.ON_CHANGE, state);
    }

  }

  /**
   * Returns the event handler that fires on error.
   * @return the event handler that fires on error
   */
  public Function jsxGet_onerror() {

    return errorHandler_;
  }

  /**
   * Sets the event handler that fires on error.
   * @param errorHandler the event handler that fires on error
   */
  public void jsxSet_onerror(final Function errorHandler) {
    errorHandler_ = errorHandler;
  }

  /**
   * Invokes the onerror handler if one has been set.
   * @param context the context within which the onerror
handler is to be invoked;
   *                if <tt>null</tt>, the current thread's
context is used.
   */
  private void processError(Context context) {
    if (ah != null) {
      ah.callFunction(AjaxHandler.ON_ERROR, 0);
    }

    if (errorHandler_ != null && !getBrowserVersion().isIE()) {
      if (context == null) {
        context = Context.getCurrentContext();
      }
      final Scriptable scope = errorHandler_.getParentScope();
      final JavaScriptEngine jsEngine = containingPage_.
          getWebClient().
          getJavaScriptEngine();

      getLog().debug("Calling onerror handler");
      jsEngine.callFunction(containingPage_, errorHandler_,
                            context, this,
                            scope, ArrayUtils.EMPTY_OBJECT_ARRAY);
      getLog().debug("onerror handler: " +
                         context.decompileFunction(errorHandler_, 4));
      getLog().debug("Calling onerror handler done.");
    }
  }

  /**
   * Returns the current state of the HTTP request. The possible values are:
   * <ul>
   *   <li>0 = uninitialized</li>
   *   <li>1 = loading</li>
   *   <li>2 = loaded</li>
   *   <li>3 = interactive</li>
   *   <li>4 = complete</li>
   * </ul>
   * @return the current state of the HTTP request
   */
  public int jsxGet_readyState() {
    return state_;
  }

  public WebRequestSettings getRequestSettings() {
    return requestSettings_;
  }

  /**
   * Returns the WebResponse object containing the data retrieved from the server.
   * @return the WebResponse object containing the data retrieved from the server
   */
  public WebResponse jsxGet_webResponseObject() {
    if (webResponse_ != null) {
      return webResponse_;
    }
    return null;
  }

  /**
   * Returns a string version of the data retrieved from the server.
   * @return a string version of the data retrieved from the server
   */
  public String jsxGet_responseText() {
    if (webResponse_ != null) {
      return webResponse_.getContentAsString();
    }
    getLog().debug("XMLHttpRequest.responseText was retrieved before the response was available.");
    return "";
  }

  /**
   * Returns a DOM-compatible document object version of the data retrieved from the server.
   * @return a DOM-compatible document object version of the data retrieved from the server
   */
  public Object jsxGet_responseXML() {
    if (webResponse_.getContentType().contains("xml")) {
      try {
        final XmlPage page = new XmlPage(webResponse_,
                                         getWindow().
                                         getWebWindow());
        final XMLDocument doc;
        if (getBrowserVersion().isIE()) {
          doc = ActiveXObject.buildXMLDocument(null);
        }
        else {
          doc = new XMLDocument();
          doc.setPrototype(getPrototype(doc.getClass()));
        }
        doc.setParentScope(getWindow());
        doc.setDomNode(page);
        return doc;
      }
      catch (final IOException e) {
        getLog().warn("Failed parsing XML document " +
                      webResponse_.getUrl() + ": " +
                      e.getMessage());
        return null;
      }
    }
    getLog().debug(
        "XMLHttpRequest.responseXML was called but the "
        "response is "+ webResponse_.getContentType());
    return null;
  }

  /**
   * Returns the numeric status returned by the server, such as 404 for "Not Found"
   * or 200 for "OK".
   * @return the numeric status returned by the server
   */
  public int jsxGet_status() {
    if (webResponse_ != null) {
      return webResponse_.getStatusCode();
    }
    getLog().error("XMLHttpRequest.status was retrieved before the response was available.");
    return 0;
  }

  /**
   * Returns the string message accompanying the status code, such as "Not Found" or "OK".
   * @return the string message accompanying the status code
   */
  public String jsxGet_statusText() {
    if (webResponse_ != null) {
      return webResponse_.getStatusMessage();
    }
    getLog().error("XMLHttpRequest.statusText was retrieved before the response was available.");
    return null;
  }

  /**
   * Cancels the current HTTP request.
   */
  public void jsxFunction_abort() {
    getWindow().getWebWindow().getThreadManager().stopThread(
        threadID_);
  }

  /**
   * Returns the labels and values of all the HTTP headers.
   * @return the labels and values of all the HTTP headers
   */
  public String jsxFunction_getAllResponseHeaders() {
    if (webResponse_ != null) {
      final StringBuilder buffer = new StringBuilder();
      for (final NameValuePair header :
           webResponse_.getResponseHeaders()) {
        buffer.append(header.getName()).append(": ").append(
            header.
            getValue()).append("\n");
      }
      return buffer.toString();
    }
    getLog().error("XMLHttpRequest.getAllResponseHeaders() was called before the response was available.");
    return null;
  }

  /**
   * Retrieves the value of an HTTP header from the response body.
   * @param headerName the (case-insensitive) name of the header to retrieve
   * @return the value of the specified HTTP header
   */
  public String jsxFunction_getResponseHeader(final String
                                              headerName) {
    if (webResponse_ != null) {
      return webResponse_.getResponseHeaderValue(headerName);
    }
    getLog().error("XMLHttpRequest.getResponseHeader() was called before the response was available.");
    return null;
  }

  /**
   * Assigns the destination URL, method and other optional attributes of a pending request.
   * @param method the method to use to send the request to the server (GET, POST, etc)
   * @param url the URL to send the request to
   * @param async Whether or not to send the request to the server asynchronously
   * @param user If authentication is needed for the specified URL, the username to use to authenticate
   * @param password If authentication is needed for the specified URL, the password to use to authenticate
   */
  public void jsxFunction_open(final String method,
                               final String url,
                               final boolean async,
                               final String user,
                               final String password) {
    WebClient wc = ( (HtmlPage) getWindow().getWebWindow().
                    getEnclosedPage()).
        getWebClient();
    if ( (wc.hasAjaxHandler()) && (ah == null)) {
      ah = wc.getLastAjaxHandler();
      ah.setXMLHttpRequest(this);
    }
    if (ah != null) {
      if (!ah.isFinished()) {
        wc.addAjaxError(AjaxError.WARNING,
                        "ajax request " +
                        requestSettings_.getUrl().toExternalForm() +
                             " was interrupted!", ("new url: " + url));
      }
      ah.setFinished(false);
    }
    // (URL + Method + User + Password) become a WebRequestSettings instance.
    containingPage_ = (HtmlPage) getWindow().getWebWindow().
        getEnclosedPage();
    try {
      final URL fullUrl = containingPage_.getFullyQualifiedUrl(
          url);
      final WebRequestSettings settings = new WebRequestSettings(
          fullUrl);
      settings.setCharset("UTF-8");
      settings.addAdditionalHeader("Referer",
                                   containingPage_.
                                   getWebResponse().
                                   getUrl().
                                   toExternalForm());
      final HttpMethod submitMethod = HttpMethod.valueOf(method.
          toUpperCase());
      settings.setHttpMethod(submitMethod);
      if (user != null) {
        final DefaultCredentialsProvider dcp = new
            DefaultCredentialsProvider();
        dcp.addCredentials(user, password);
        settings.setCredentialsProvider(dcp);
      }
      requestSettings_ = settings;
    }
    catch (final MalformedURLException e) {
      getLog().error(
            "Unable to initialize XMLHttpRequest using malformed URL '" +
          url + "'.");
      return;
    }
```

```
                /*WebClient wc = ( (HtmlPage) getWindow().getWebWindow().getEnclosedPage()).
        getWebClient();
             if ((wc.hasAjaxHandler())&&(ah == null)) {
      ah = wc.getLastAjaxHandler();
      ah.setXMLHttpRequest(this);
      ah.setFinished(false);
             }*/

    // Async stays a boolean.
    async_ = async;
    // Change the state!
    setState(STATE_LOADING, null);
  }

  /**
   * Sends the specified content to the server in an HTTP request and receives the response.
   * @param content the body of the message being sent with the request
   */
  public void jsxFunction_send(final Object content) {
    prepareRequest(content);
    if (ah != null) {
      requestSettings_ = ah.callBeforeSend(requestSettings_);
    }
    final AjaxController ajaxController = getWindow().
        getWebWindow().
        getWebClient().getAjaxController();
    final HtmlPage page = (HtmlPage) getWindow().getWebWindow().
        getEnclosedPage();
    final boolean synchron = ajaxController.processSynchron(page,
        requestSettings_, async_);
    if (synchron) {
      doSend(Context.getCurrentContext());
    }
    else {
        // Create and start a thread in which to execute the request.
      final Object startingScope = getWindow();

      final ContextAction action = new ContextAction() {
        public Object run(final Context cx) {
          cx.putThreadLocal(JavaScriptEngine.KEY_STARTING_SCOPE,
                            startingScope);
          doSend(cx);
          return null;
        }
      };
      final Runnable t = new Runnable() {
        public void run() {
          ContextFactory.getGlobal().call(action);
        }
      };

      getLog().debug(
            "Starting XMLHttpRequest thread for asynchronous request");
      threadID_ = getWindow().getWebWindow().getThreadManager().
          startThread(t, "XMLHttpRequest.send");
    }
  }

  /**
   * Prepares the WebRequestSettings that will be sent.
   * @param content the content to send
   */
  private void prepareRequest(final Object content) {
    if (HttpMethod.POST == requestSettings_.getHttpMethod()
        && content != null
        && !Context.getUndefinedValue().equals(content)) {
      final String body = Context.toString(content);
      if (body.length() > 0) {
        getLog().debug("Setting request body to: " + body);
        requestSettings_.setRequestBody(body);
      }
    }
  }

  /**
   * The real send job.
   * @param context the current context
   */
  private void doSend(final Context context) {
    final WebClient wc = getWindow().getWebWindow().getWebClient();
    try {
      setState(STATE_LOADED, context);
      final WebResponse webResponse = wc.loadWebResponse(
          requestSettings_);
      getLog().debug("Web response loaded successfully.");
      if (overriddenMimeType_ == null) {
        webResponse_ = webResponse;
      }
      else {
        webResponse_ = new WebResponseWrapper(webResponse) {
          @Override
          public String getContentType() {
            return overriddenMimeType_;
          }
        };
      }
      setState(STATE_INTERACTIVE, context);
      setState(STATE_COMPLETED, context);
      if (ah != null) {
        ah.setFinished(true);
      }
    }
    catch (final IOException e) {
      getLog().debug(
          "IOException: returning a network error response.");
      webResponse_ = new NetworkErrorWebResponse(
          requestSettings_);
      setState(STATE_COMPLETED, context);
      processError(context);
    }
  }

  /**
   * Sets the specified header to the specified value. The <tt>open</tt> method must be
   * called before this method, or an error will occur.
   * @param name the name of the header being set
   * @param value the value of the header being set
   */
  public void jsxFunction_setRequestHeader(final String name,
                                           final String value) {
    if (requestSettings_ != null) {
      requestSettings_.addAdditionalHeader(name, value);
    }
    else {
      throw Context.reportRuntimeError(
               "The open() method must be called before setRequestHeader().");
    }
  }

  /**
   * Override the mime type returned by the server (if any). This may be used, for example, to force a stream
   * to be treated and parsed as text/xml, even if the server does not report it as such.
   * This must be done before the send method is invoked.
   * @param mimeType the type used to override that returned by the server (if any)
   * @see <a href="http://xulplanet.com/references/objref/XMLHttpRequest.html#method_overrideMimeType">XUL Planet</a>
   */
  public void jsxFunction_overrideMimeType(final String mimeType) {
    overriddenMimeType_ = mimeType;
  }

  private static final class NetworkErrorWebResponse
      implements WebResponse {
    private final WebRequestSettings webRequestSettings_;

    private NetworkErrorWebResponse(final WebRequestSettings
                                    webRequestSettings) {
      webRequestSettings_ = webRequestSettings;
    }

    public int getStatusCode() {
      return 0;
    }

    public String getStatusMessage() {
      return "";
    }

    public String getContentType() {
      return "";
    }

    public String getContentAsString() {
      return "";
    }

    public InputStream getContentAsStream() throws IOException {
      return null;
    }

    public URL getUrl() {
      return webRequestSettings_.getUrl();
    }

    public HttpMethod getRequestMethod() {
      return webRequestSettings_.getHttpMethod();
    }

    public List<NameValuePair> getResponseHeaders() {
      return Collections.emptyList();
    }

    public String getResponseHeaderValue(final String headerName)
{
      return "";
    }

    public long getLoadTimeInMilliSeconds() {
      return 0;
    }

    public String getContentCharSet() {
      return "";
    }

    public byte[] getResponseBody() {
      return new byte[0];
    }

    public WebRequestSettings getRequestSettings() {
      return webRequestSettings_;
    }
  }
}
```

7.2.2 WebClient

```
/*
 * Copyright (c) 2002-2008 Gargoyle Software Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.gargoylesoftware.htmlunit;

import static com.gargoylesoftware.htmlunit.attachment.
    Attachment.
    isAttachment;

import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLStreamHandler;
import java.security.GeneralSecurityException;
import java.util.ArrayList;
import java.util.BitSet;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Stack;
import java.util.StringTokenizer;

import org.apache.commons.codec.DecoderException;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.NameValuePair;
import org.apache.commons.httpclient.URI;
import org.apache.commons.httpclient.URIException;
import org.apache.commons.httpclient.auth.CredentialsProvider;
import org.apache.commons.httpclient.cookie.CookiePolicy;
import org.apache.commons.httpclient.protocol.Protocol;
import org.apache.commons.httpclient.protocol.
    ProtocolSocketFactory;
import org.apache.commons.httpclient.util.URIUtil;
import org.apache.commons.io.FileUtils;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.gargoylesoftware.htmlunit.attachment.
    AttachmentHandler;
import com.gargoylesoftware.htmlunit.html.FrameWindow;
import com.gargoylesoftware.htmlunit.html.HTMLParser;
import com.gargoylesoftware.htmlunit.html.HTMLParserListener;
import com.gargoylesoftware.htmlunit.html.HtmlElement;
import com.gargoylesoftware.htmlunit.html.HtmlPage;
import com.gargoylesoftware.htmlunit.javascript.
    HtmlUnitContextFactory;
import com.gargoylesoftware.htmlunit.javascript.JavaScriptEngine;
import com.gargoylesoftware.htmlunit.javascript.host.Event;
import com.gargoylesoftware.htmlunit.javascript.host.HTMLElement;
import com.gargoylesoftware.htmlunit.javascript.host.Window;
import com.gargoylesoftware.htmlunit.protocol.data.
    DataUrlDecoder;
import com.gargoylesoftware.htmlunit.ssl.
    InsecureSSLProtocolSocketFactory;
import com.gargoylesoftware.htmlunit.util.UrlUtils;
import java.util.Iterator;
import java.util.Vector;

/**
 * The main starting point in HtmlUnit: this class simulates a web
browser.
 * <p>
 * A standard usage of HtmlUnit will start with using the {@link
#getPage(String)} method
 * (or {@link #getPage(URL)}) to load a first {@link Page}
 * and will continue with further processing on this page
depending on its type.
 * </p>
 * <b>Example:</b><br>
 * <br>
 * <code>
 * final WebClient webClient = new WebClient();<br/>
 * final {@link HtmlPage} startPage = ({@link HtmlPage}) webClient.getPage("http://htmlunit.sf.net");<br/>
 * assertEquals("HtmlUnit - Welcome to HtmlUnit", startPage.{@link HtmlPage#getTitleText() getTitleText}());
 * </code>
 *
 * @version $Revision: 3186 $
 * @author <a href="mailto:mbowler@GargoyleSoftware.com">Mike Bowler</a>
 * @author <a href="mailto:gudujarlson@sf.net">Mike J. Bresnahan</a>
 * @author Dominique Broeglin
 * @author Noboru Sinohara
 * @author <a href="mailto:chen_jun@users.sourceforge.net">Chen Jun</a>
 * @author David K. Taylor
 * @author <a href="mailto:cse@dynabean.de">Christian Sell</a>
 * @author <a href="mailto:bcurren@esomnie.com">Ben Curren</a>
 * @author Marc Guillemot
 * @author Chris Erskine
 * @author Daniel Gredler
 * @author Sergey Gorelkin
 * @author Hans Donner
 * @author Paul King
 * @author Ahmed Ashour
 * @author Bruce Chapman
 * @author Sudhan Moghe
 * @author Andreas Langer
 * @author Mario Bernhart
 * @author Thomas Grechenig
 */
public class WebClient
    implements Serializable {

  private static final long serialVersionUID = -
      7214321203864969635L;

  /**
   * HtmlUnit's cookie policy is to be browser-compatible. Code which requires access to
   * HtmlUnit's cookie policy should use this constant, rather than making assumptions
   * and using one of the HttpClient {@link CookiePolicy} constants directly.
   */
  public static final String HTMLUNIT_COOKIE_POLICY =
      CookiePolicy.
      BROWSER_COMPATIBILITY;

  private transient WebConnection webConnection_;
  private boolean printContentOnFailingStatusCode_ = true;
  private boolean throwExceptionOnFailingStatusCode_ = true;
  private CredentialsProvider credentialsProvider_ = new
      DefaultCredentialsProvider();
  private ProxyConfig proxyConfig_;
  private JavaScriptEngine scriptEngine_;
  private boolean javaScriptEnabled_ = true;
  private boolean cookiesEnabled_ = true;
  private boolean cssEnabled_ = true;
  private boolean popupBlockerEnabled_;
  private String homePage_;
  private final Map<String, String> requestHeaders_ =
      Collections.
      synchronizedMap(new HashMap<String, String> (89));
  private IncorrectnessListener incorrectnessListener_ = new
      IncorrectnessListenerImpl();

  public static String newline = System.getProperty(
      "line.separator");
  private Stack<AjaxHandler> ah = new Stack<AjaxHandler> ();
  private Stack<AjaxHandler> allAjaxHandlers = new Stack<
      AjaxHandler> ();
   private Vector<AjaxError> ajaxErrors = new Vector<AjaxError> ();

  /**
     * Like Firefox default value for network.http.redirection-limit
   */
  private static final int ALLOWED_REDIRECTIONS_SAME_URL = 20;

  private AlertHandler alertHandler_;
  private ConfirmHandler confirmHandler_;
  private PromptHandler promptHandler_;
  private StatusHandler statusHandler_;
  private AttachmentHandler attachmentHandler_;
  private AjaxController ajaxController_ = new AjaxController();

  private BrowserVersion browserVersion_;
  private boolean isRedirectEnabled_ = true;
  private PageCreator pageCreator_ = new DefaultPageCreator();

  private final Set<WebWindowListener> webWindowListeners_ = new
      HashSet<WebWindowListener> (5);
  private final List<WebWindow> webWindows_ = Collections.
      synchronizedList(new ArrayList<WebWindow> ());

  private WebWindow currentWindow_;
  private Stack<WebWindow> firstWindowStack_ = new Stack<
      WebWindow> ();
  private final Stack<WebWindow> windows_ = new Stack<WebWindow> ();
  private int timeout_;
  private HTMLParserListener htmlParserListener_;
  private OnbeforeunloadHandler onbeforeunloadHandler_;
  private Cache cache_ = new Cache();

  private static URLStreamHandler JavaScriptUrlStreamHandler_
      = new com.gargoylesoftware.htmlunit.protocol.javascript.
      Handler();
  private static URLStreamHandler AboutUrlStreamHandler_
      = new com.gargoylesoftware.htmlunit.protocol.about.Handler();
  private static URLStreamHandler DataUrlStreamHandler_
      = new com.gargoylesoftware.htmlunit.protocol.data.Handler();
```

```

  /**
   * URL for "about:blank".
   */
  public static final URL URL_ABOUT_BLANK;
  static {
    URL tmpUrl = null;
    try {
      tmpUrl = new URL(null, "about:blank",
                       AboutUrlStreamHandler_);
    }
    catch (final MalformedURLException e) {
      // impossible
      e.printStackTrace();
    }
    URL_ABOUT_BLANK = tmpUrl;
  }

  //singleton WebResponse for "about:blank"
  private static final WebResponse WEB_RESPONSE_FOR_ABOUT_BLANK = new
      StringWebResponse("", URL_ABOUT_BLANK);

  private ScriptPreProcessor scriptPreProcessor_;

  private Map<String, String> activeXObjectMap_ = Collections.
      emptyMap();
  private RefreshHandler refreshHandler_ = new
      ImmediateRefreshHandler();
  private boolean throwExceptionOnScriptError_ = true;

  /**
   * Creates a web client instance using the browser version returned by
   * {@link BrowserVersion#getDefault()}.
   */
  public WebClient() {
    this(BrowserVersion.getDefault());
  }

  /**
   * Creates a web client instance using the specified {@link BrowserVersion}.
   * @param browserVersion the browser version to simulate
   */
  public WebClient(final BrowserVersion browserVersion) {
    WebAssert.notNull("browserVersion", browserVersion);

    homePage_ = "http://www.gargoylesoftware.com/";
    browserVersion_ = browserVersion;
    proxyConfig_ = new ProxyConfig();
    try {
      scriptEngine_ = createJavaScriptEngineIfPossible(this);
    }
    catch (final NoClassDefFoundError e) {
      scriptEngine_ = null;
    }

    // The window must be constructed after the script engine.
    addWebWindowListener(new CurrentWindowTracker());
    currentWindow_ = new TopLevelWindow("", this);
    HtmlUnitContextFactory.putThreadLocal(browserVersion);
  }

  /**
   * Creates an instance that will use the specified {@link BrowserVersion} and proxy server.
   * @param browserVersion the browser version to simulate
   * @param proxyHost the server that will act as proxy
   * @param proxyPort the port to use on the proxy server
   */
  public WebClient(final BrowserVersion browserVersion,
                   final String proxyHost, final int proxyPort) {
    WebAssert.notNull("browserVersion", browserVersion);
    WebAssert.notNull("proxyHost", proxyHost);

    homePage_ = "http://www.gargoylesoftware.com/";
    browserVersion_ = browserVersion;
    proxyConfig_ = new ProxyConfig(proxyHost, proxyPort);
    try {
      scriptEngine_ = createJavaScriptEngineIfPossible(this);
    }
    catch (final NoClassDefFoundError e) {
      scriptEngine_ = null;
    }
    // The window must be constructed after the script engine.
    currentWindow_ = new TopLevelWindow("", this);
    HtmlUnitContextFactory.putThreadLocal(browserVersion);
  }

  /**
   * Adds an AjaxHandler that will be used by an XMLHttpRequest object
   * @param ac AjaxHandler: the AjaxHandler for the next XMLHttpRequest object
   */
  public void addAjaxHandler(AjaxHandler ac) {
    ac.setWebClient(this);
    this.ah.push(ac);
    allAjaxHandlers.push(ac);
  }

  /**
   *
   * @param i int
   * @return AjaxHandler
   */
  public AjaxHandler getAjaxHandler(int i) {
    return ah.get(i);
  }

  /**
   * returns an AjaxHandler for an XMLHttpRequest object
   * @return AjaxHandler: handler for an XMLHttpRequest object
   */
  public AjaxHandler getLastAjaxHandler() {
    return ah.pop();
  }

  /**
```

```
   * Checks whether there are any AjaxHandlers available
   * @return boolean
   */
  public boolean hasAjaxHandler() {
    if (!ah.isEmpty()) {
      AjaxHandler ah = this.ah.peek();
      if (!ah.used()) {
        return true;
      }
    }
    return false;
  }

  /**
   * Checks whether all Ajax requests have finished
   * @return boolean
   */
  public boolean allAjaxHandlerFinished() {
    for (Iterator<AjaxHandler> i = allAjaxHandlers.iterator();
         i.hasNext(); ) {
      AjaxHandler a = i.next();
      if (a.used() && ! (a.isFinished())) {
        return false;
      }
    }
    return true;
  }

  public void addAjaxError(AjaxError message) {
    ajaxErrors.add(message);
  }

  public void addAjaxError(int type, String message, String url)
{
    ajaxErrors.add(new AjaxError(type, message, url));
  }

  public String getAjaxErrors() {
    String ret = "";
    for (Iterator<AjaxError> i = ajaxErrors.iterator();
         i.hasNext(); ) {
      ret = ret + i.next().toString() + newline;
    }
    return ret;
  }

  public int getNumberOfAjaxErrors() {
    return ajaxErrors.size();
  }

  public int getNumberOfAjaxErrorsWithoutWarnings() {
    int ret = 0;
    for (Iterator<AjaxError> i = ajaxErrors.iterator();
         i.hasNext(); ) {
      if (i.next().getType() != AjaxError.WARNING) {
        ret++;
      }
    }
    return ret;
  }

  public void resetAjaxErrors() {
    ajaxErrors.removeAllElements();
  }
...

```

7.3 Test Classes

7.3.1 AJAXLoginTest

```
package com.gargoylesoftware.htmlunit.test;

import com.gargoylesoftware.htmlunit.*;
import com.gargoylesoftware.htmlunit.html.*;
import junit.framework.*;

public class AJAXLoginTest
    extends TestCase {
  public AJAXLoginTest() {
  }

  public void testCase_1_AjaxController() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    //register NicelyResynchronizingAjaxController
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    //load Page
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    // get form and get username and password field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    //set username
    user.focus();
    user.setValueAttribute("user1");
    user.blur();
    //set password
    pass.focus();
    pass.setValueAttribute("pass1");
    pass.blur();
    //do assertions
    WebAssert.assertTextPresentInElement(page, "Logged in as",
                                         "login");
    WebAssert.assertTextPresentInElement(page, "John Doe", "login");
    WebAssert.assertLinkPresentWithText(page, "logout");
  }

  public void testCase_1_AjaxHandler() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    //register AjaxHandler
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    //load Page
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    // get form and get username and password field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    //set username
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    //set password
    pass.focus();
    pass.setValueAttribute("pass1");
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    //do assertions
    WebAssert.assertTextPresentInElement(page, "Logged in as",
                                         "login");
    WebAssert.assertTextPresentInElement(page, "John Doe", "login");
    WebAssert.assertLinkPresentWithText(page, "logout");
  }

  public void testCase_1_AjaxHandlerImproved() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    //create AjaxHandler
    AjaxHandler ah = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf("John Doe") >= 0);
//System.err.println(getXMLHttpRequest().jsxGet_responseText());
        }
      }
    };
    AjaxHandler dummyAH = new AjaxHandler();
    webClient.addAjaxHandler(dummyAH);
    //load Page
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    // get form and get username and password field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    //set username
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!dummyAH.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    //set password
    pass.focus();
    pass.setValueAttribute("pass1");
    dummyAH.removeHandler();
    webClient.addAjaxHandler(ah);
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
  }

  public void testCase_2_AjaxController() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                                FIREFOX_2);
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();

    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("pass1");
    pass.blur();

    WebAssert.assertTextPresentInElement(page, "Logged in as",
                                             "login");
    WebAssert.assertTextPresentInElement(page, "John Doe", "login");
    WebAssert.assertLinkPresentWithText(page, "logout");

    HtmlAnchor logout = page.getFirstAnchorByText("logout");
    HtmlPage page2 = (HtmlPage) logout.click();

    WebAssert.assertTextNotPresent(page2, "Logged in as");
    WebAssert.assertTextNotPresent(page2, "John Doe");
    WebAssert.assertLinkNotPresentWithText(page2, "logout");
    WebAssert.assertTextPresent(page2,
        "Enter your username and password to log in");
  }

  public void testCase_2_AjaxHandler() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("pass1");
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    WebAssert.assertTextPresentInElement(page, "Logged in as",
                                         "login");
    WebAssert.assertTextPresentInElement(page, "John Doe", "login");
    WebAssert.assertLinkPresentWithText(page, "logout");

    HtmlAnchor logout = page.getFirstAnchorByText("logout");
    HtmlPage page2 = (HtmlPage) logout.click();

    WebAssert.assertTextNotPresent(page2, "Logged in as");
    WebAssert.assertTextNotPresent(page2, "John Doe");
    WebAssert.assertLinkNotPresentWithText(page2, "logout");
    WebAssert.assertTextPresent(page2,
                                "Enter your username and password to log in");
  }

  public void testCase_2_AjaxHandlerImproved() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxHandler ah = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(
                         "John Doe") >= 0);
        }
      }
    };
    AjaxHandler dummyAH = new AjaxHandler();
    webClient.addAjaxHandler(dummyAH);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!dummyAH.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("pass1");
    dummyAH.removeHandler();
    webClient.addAjaxHandler(ah);
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());

    WebAssert.assertLinkPresentWithText(page, "logout");
    ah.removeHandler();
    HtmlAnchor logout = page.getFirstAnchorByText("logout");
    HtmlPage page2 = (HtmlPage) logout.click();

    WebAssert.assertTextNotPresent(page2, "Logged in as");
    WebAssert.assertTextNotPresent(page2, "John Doe");
    WebAssert.assertLinkNotPresentWithText(page2, "logout");
    WebAssert.assertTextPresent(page2,
                             "Enter your username and password to log in");
  }

  public void testCase_3_AjaxController() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                                     FIREFOX_2);
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("wrongpass");
    pass.blur();
    WebAssert.assertTextPresentInElement(page,
        "Invalid username and password combination", "login");
    WebAssert.assertLinkNotPresentWithText(page, "logout");

  }

  public void testCase_3_AjaxHandler() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("wrongpass");
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    WebAssert.assertTextPresentInElement(page,
        "Invalid username and password combination", "login");
    WebAssert.assertLinkNotPresentWithText(page, "logout");
  }

  public void testCase_3_AjaxHandlerImproved() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxHandler ah = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(
                                  "Invalid username and password combination") >=
                     0);
        }
      }
    };
    AjaxHandler dummyAH = new AjaxHandler();
    webClient.addAjaxHandler(dummyAH);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("wrongpass");
    dummyAH.removeHandler();
    webClient.addAjaxHandler(ah);
    pass.blur();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    System.err.println("OOOOOOOOOOOOOOO " + webClient.getAjaxErrors() +
        "00000000000000");
    System.err.println(webClient.getNumberOfAjaxErrors());
    System.err.println(webClient.getNumberOfAjaxErrorsWithoutWarnings());
    WebAssert.assertLinkNotPresentWithText(page, "logout");
    assertEquals(0, webClient.getNumberOfAjaxErrors());
  }

  public void testCase_4_AjaxController() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                             FIREFOX_2);
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    user.setValueAttribute("user1");
    user.blur();
    pass.focus();
    pass.setValueAttribute("");
    String savePage = page.asXml();
    pass.blur();
    assertEquals(savePage, page.asXml());
  }

  public void testCase_4_AjaxHandler() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                             FIREFOX_2);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    ah.removeHandler();
    webClient.addAjaxHandler(ah);
    user.setValueAttribute("user1");
    user.blur();
    assertFalse(ah.used());
    pass.focus();
    pass.setValueAttribute("");
    pass.blur();
    assertFalse(ah.used());

  }

  public void testCase_4_AjaxHandlerImproved() throws Exception {
    testCase_4_AjaxHandler();
  }

  public void testCase_5_AjaxController() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    user.setValueAttribute("");
    user.blur();
    pass.focus();
    pass.setValueAttribute("pass1");
    String savePage = page.asXml();
    pass.blur();
    assertEquals(savePage, page.asXml());
  }

  public void testCase_5_AjaxHandler() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.jamesdam.com/ajax_login/login.html");
    WebAssert.assertTitleContains(page, "AJAX Login System");
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput user = form.getInputByName("username");
    HtmlInput pass = form.getInputByName("password");
    user.focus();
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    ah.removeHandler();
    webClient.addAjaxHandler(ah);
    user.setValueAttribute("");
    user.blur();
    assertFalse(ah.used());
    pass.focus();
    assertFalse(ah.used());
    pass.setValueAttribute("pass1");
    pass.blur();
    assertFalse(ah.used());
  }

  public void testCase_5_AjaxHandlerImproved() throws Exception {
    testCase_5_AjaxHandler();
  }

  public void main(String[] args) {
    try {
      TestSuite ts = new TestSuite();
      AJAXLoginTest at = new AJAXLoginTest();
      ts.addTest(at);
      TestResult tr = new TestResult();
      ts.run(tr);

    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

7.3.2 WikipediaTest

```
package com.gargoylesoftware.htmlunit.test;

import com.gargoylesoftware.htmlunit.*;
import com.gargoylesoftware.htmlunit.html.*;
import com.gargoylesoftware.htmlunit.javascript.host.MouseEvent;

import junit.framework.*;
import java.awt.event.KeyEvent;
import org.w3c.dom.Element;
import java.util.Iterator;
import java.util.List;

public class AJAXWikipediaTest
    extends TestCase {
  public AJAXWikipediaTest() {
  }

  public void testCase_1_AjaxController() throws Exception {
    // create default firefox browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    // initialize and register AjaxController
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    // connect to wikipedia.at
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.wikipedia.at");
    HtmlPage page2;
    WebAssert.assertTitleContains(page, "wikipedia.at");
    // parse the document for the HtmlInput field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput input = form.getInputByName("q");
    // write into the field and fire keyup event
    input.setValueAttribute("XMLHtt");
    input.keyup();
    // parse result and click on XMLHttpRequest Link
                HtmlElement   search_suggest   =   (HtmlElement)
page.getElementById(
        "search_suggest");
    WebAssert.assertTextPresentInElement(page, "XMLHttpRequest",
                                         "search_suggest");
    List li = search_suggest.getByXPath("div/a");
    assertFalse(li.isEmpty());
    HtmlAnchor link = (HtmlAnchor) li.get(0);
    System.err.println(link.asXml());
    page2 = (HtmlPage) link.click();
    WebAssert.assertTitleContains(page2, "XMLHttpRequest");
  }

  public void testCase_1_AjaxHandler() throws Exception {
    // create default firefox browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    // initialize and register the default AjaxHandler
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    // connect to wikipedia.at
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.wikipedia.at");
    HtmlPage page2;
    WebAssert.assertTitleContains(page, "wikipedia.at");
    // parse the document for the HtmlInput field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput input = form.getInputByName("q");
    // write into the field and fire keyup event
    input.setValueAttribute("XMLHtt");
    input.keyup();
    do {
      synchronized (this) {
        wait(600);
      }
    }
    while (!ah.isFinished());
    // parse result and click on XMLHttpRequest Link
                HtmlElement   search_suggest   =   (HtmlElement)
page.getElementById(
        "search_suggest");
    System.err.println(page.asXml());
    WebAssert.assertTextPresentInElement(page, "XMLHttpRequest",
                                         "search_suggest");
    List li = search_suggest.getByXPath("div/a");
    assertFalse(li.isEmpty());
    HtmlAnchor link = (HtmlAnchor) li.get(0);
    System.err.println(link.asXml());
    page2 = (HtmlPage) link.click();
    WebAssert.assertTitleContains(page2, "XMLHttpRequest");
    assertEquals(0, webClient.getNumberOfAjaxErrors());
  }

  public void testCase_1_AjaxHandlerImproved() throws Exception {
    // create default firefox browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    // initialize and register the default AjaxHandler
    AjaxHandler ah = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(
                         "XMLHttpRequest") >= 0);
        }
      }
    };
    webClient.addAjaxHandler(ah);
    // connect to wikipedia.at
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://www.wikipedia.at");
    HtmlPage page2;
    WebAssert.assertTitleContains(page, "wikipedia.at");
    // parse the document for the HtmlInput field
    HtmlForm form = (HtmlForm) page.getForms().get(0);
    HtmlInput input = form.getInputByName("q");
    // write into the field and fire keyup event
    input.setValueAttribute("XMLHtt");
    input.keyup();
    do {
      synchronized (this) {
        wait(600);
      }
    }
    while (!ah.isFinished());
    // parse result and click on XMLHttpRequest Link
    HtmlElement search_suggest = (HtmlElement) page.getElementById(
        "search_suggest");
    System.err.println(page.asXml());
    List li = search_suggest.getByXPath("div/a");
    assertFalse(li.isEmpty());
    HtmlAnchor link = (HtmlAnchor) li.get(0);
    page2 = (HtmlPage) link.click();
    WebAssert.assertTitleContains(page2, "XMLHttpRequest");
    assertTrue(webClient.allAjaxHandlerFinished());
    assertEquals(0, webClient.getNumberOfAjaxErrors());
  }

  public void main(String[] args) {
    try {
      AJAXWikipediaTest at = new AJAXWikipediaTest();
      //at.testCase_1_AjaxController();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

7.3.3 AjaxChatTest

```
package com.gargoylesoftware.htmlunit.test;

import com.gargoylesoftware.htmlunit.*;
import com.gargoylesoftware.htmlunit.html.*;
import com.gargoylesoftware.htmlunit.javascript.host.MouseEvent;

import junit.framework.*;
import java.awt.event.KeyEvent;
import java.util.List;

public class AJAXChatTest
    extends TestCase {
  public AJAXChatTest() {

  }

  public void testCase_1_AjaxController() throws Exception {
    //configure the browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    // create default ajax controller
    AjaxController ac = new NicelyResynchronizingAjaxController();
    webClient.setAjaxController(ac);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    //connect to the page and find all necessary elements
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page, "live chat");

    HtmlForm form = (HtmlForm) page.getFormByName("chatForm");
    HtmlInput name = form.getInputByName("name");
    HtmlInput input = form.getInputByName("chatbarText");
            HtmlSubmitInput   submit   =   (HtmlSubmitInput)
form.getInputByName(
        "submit");
    //set username and message and send everything
    String nameString = "con1";
    name.setValueAttribute(nameString);
    input.focus();
    String message = "hello!";
    input.setValueAttribute(message);
    submit.click();
    // workaround: the program has to wait some time until
    // the new messages have been received
    Thread.currentThread().sleep(6000);
    // parse the outputlist for the first element
    List list = page.getByXPath("//ul[attribute::id='outputList']/li");
    assertFalse(list.isEmpty());
    HtmlListItem li = (HtmlListItem) list.get(0);
    HtmlUnorderedList ul = (HtmlUnorderedList) page.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    System.err.println(ul.asXml());
    assertTrue(ul.asXml().contains(message));
    assertTrue(ul.asXml().contains(nameString));
    assertTrue(li.asXml().contains(message));
    assertTrue(webClient.getNumberOfAjaxErrorsWithoutWarnings() == 0);
  }

  public void testCase_1_AjaxHandler() throws Exception {
    //configure the browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                                FIREFOX_2);
    // create two default ajax handler: one for receiving
    // and one for sending chet messages
    AjaxHandler ah2 = new AjaxHandler();
    webClient.addAjaxHandler(ah2);
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);
    //connect to the page and find all necessary elements
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page, "live chat");
    //force loading all current messages
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah.isFinished());
    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    HtmlForm form = (HtmlForm) page.getFormByName("chatForm");
    HtmlInput name = form.getInputByName("name");
    HtmlInput input = form.getInputByName("chatbarText");
    HtmlSubmitInput submit = (HtmlSubmitInput) form.getInputByName(
        "submit");
    //set username and message and send everything
    String nameString = "qwe";
    name.setValueAttribute(nameString);
    input.focus();
    String message = "aghdf";
    input.setValueAttribute(message);
    submit.click();
    //wait until sending has finished
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah2.isFinished());
    //wait until the next receiving thread starts

    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah.isFinished());

    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    // parse the outputlist for the first element
    List list = page.getByXPath("//ul[attribute::id='outputList']/li");
    assertFalse(list.isEmpty());
    HtmlListItem li = (HtmlListItem) list.get(0);
    HtmlUnorderedList ul = (HtmlUnorderedList) page.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul.asXml().contains(message));
    assertTrue(ul.asXml().contains(nameString));
    assertTrue(li.asXml().contains(message));
    assertTrue(webClient.getNumberOfAjaxErrorsWithoutWarnings() == 0);
  }

  public void testCase_1_AjaxHandlerImproved() throws Exception {
    //configure the browser
    final WebClient webClient = new WebClient(BrowserVersion.
                                              FIREFOX_2);
    // create two ajax handler: one for receiving
    // and one for sending chet messages
    final String nameString = "ajaxtester";
    final String message = "hello world!";
    AjaxHandler ah2 = new AjaxHandler();
    webClient.addAjaxHandler(ah2);
    AjaxHandler ah3 = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(nameString) >= 0);
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(message) >= 0);
        }
      }
    };
    AjaxHandler ah = new AjaxHandler();
    webClient.addAjaxHandler(ah);

    //connect to the page and find all necessary elements
    final HtmlPage page = (HtmlPage) webClient.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page, "live chat");
    //force loading all current messages
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah.isFinished());

    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah.isFinished());
    HtmlForm form = (HtmlForm) page.getFormByName("chatForm");
    HtmlInput name = form.getInputByName("name");
    HtmlInput input = form.getInputByName("chatbarText");
    HtmlSubmitInput submit = (HtmlSubmitInput) form.getInputByName(
        "submit");
    //set username and message and send everything
    name.setValueAttribute(nameString);
    input.focus();
    input.setValueAttribute(message);
    submit.click();
    ah.removeHandler();
    webClient.addAjaxHandler(ah3);
    //wait until sending has finished
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah2.isFinished());
    //wait until the next receiving thread starts

    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah3.isFinished());

    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah3.isFinished());
    ah3.removeHandler();
    // parse the outputlist for the first element
    List list = page.getByXPath("//ul[attribute::id='outputList']/li");
    assertFalse(list.isEmpty());
    HtmlListItem li = (HtmlListItem) list.get(0);
    HtmlUnorderedList ul = (HtmlUnorderedList) page.getByXPath(
        "//ul[attribute::id='outputList']").get(0);

    assertTrue(ul.asXml().contains(message));
    assertTrue(ul.asXml().contains(nameString));
    assertTrue(li.asXml().contains(message));

    System.err.println(webClient.getAjaxErrors());
    assertTrue(webClient.getNumberOfAjaxErrorsWithoutWarnings() == 0);

  }

  public void testCase_2_AjaxController() throws Exception {
    //define the input strings
    String message = "hurra!";
    String nameString = "me";
    //configure the browser
    final WebClient webClient_1 = new WebClient(BrowserVersion.
                                                    FIREFOX_2);
    // create default ajax controller
    AjaxController ac1 = new NicelyResynchronizingAjaxController();
    webClient_1.setAjaxController(ac1);

    //the same with the second browser
    //configure the browser
    final WebClient webClient_2 = new WebClient(BrowserVersion.
                                                    FIREFOX_2);
    // create default ajax controller
    AjaxController ac2 = new NicelyResynchronizingAjaxController();
    webClient_2.setAjaxController(ac2);
    //connect to the page and find all necessary elements
    final HtmlPage page_1 = (HtmlPage) webClient_1.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page_1, "live chat");
    final HtmlPage page_2 = (HtmlPage) webClient_2.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page_2, "live chat");

    HtmlForm form_1 = (HtmlForm) page_1.getFormByName("chatForm");
    HtmlInput name = form_1.getInputByName("name");
    HtmlInput input = form_1.getInputByName("chatbarText");
    HtmlSubmitInput submit = (HtmlSubmitInput) form_1.getInputByName(
        "submit");
    //set username and message and send everything
    name.setValueAttribute(nameString);
    input.focus();
    input.setValueAttribute(message);
    submit.click();
    //workaround
    Thread.currentThread().sleep(6000);

    // parse the outputlist for the first element
    List list_1 = page_1.getByXPath(
        "//ul[attribute::id='outputList']/li");
    assertFalse(list_1.isEmpty());
    HtmlListItem li_1 = (HtmlListItem) list_1.get(0);
    HtmlUnorderedList ul_1 = (HtmlUnorderedList) page_1.getByXPath(
          "//ul[attribute::id='outputList']").get(0);
      assertTrue(ul_1.asXml().contains(message));
      assertTrue(ul_1.asXml().contains(nameString));
      assertTrue(li_1.asXml().contains(message));
      assertTrue(webClient_1.getNumberOfAjaxErrorsWithoutWarnings()
==
                 0);
      // the same with the second client
      List list_2 = page_2.getByXPath(
          "//ul[attribute::id='outputList']/li");
      assertFalse(list_2.isEmpty());
      HtmlUnorderedList ul_2 = (HtmlUnorderedList)
page_2.getByXPath(
          "//ul[attribute::id='outputList']").get(0);
      assertTrue(ul_2.asXml().contains(message));
      assertTrue(ul_2.asXml().contains(nameString));
    }

    public void testCase_2_AjaxHandler() throws Exception {
      //configure the browser
      final WebClient webClient_1 = new WebClient(BrowserVersion.
                                                  FIREFOX_2);
      // create two default ajax handler: one for receiving
      // and one for sending chet messages
      AjaxHandler ah_1_s = new AjaxHandler();
      webClient_1.addAjaxHandler(ah_1_s);
      AjaxHandler ah_1_r = new AjaxHandler();
      webClient_1.addAjaxHandler(ah_1_r);
      //the same with the second browser
      //configure the browser
      final WebClient webClient_2 = new WebClient(BrowserVersion.
                                                  FIREFOX_2);
      // create two default ajax handler: one for receiving
      // and one for sending chet messages
      AjaxHandler ah_2_s = new AjaxHandler();
      webClient_2.addAjaxHandler(ah_2_s);
      AjaxHandler ah_2_r = new AjaxHandler();
      webClient_2.addAjaxHandler(ah_2_r);
      //connect to the page and find all necessary elements
      final HtmlPage page_1 = (HtmlPage) webClient_1.getPage(
          "http://chat.plasticshore.com/");
      WebAssert.assertTitleContains(page_1, "live chat");
      final HtmlPage page_2 = (HtmlPage) webClient_2.getPage(
          "http://chat.plasticshore.com/");
      WebAssert.assertTitleContains(page_2, "live chat");
      //force loading all current messages
      do {
        synchronized (this) {
          wait(100);
        }
      }
      while (ah_1_r.isFinished());
      // wait until the XHR for receiving finishes
      do {
        synchronized (this) {
          wait(100);
        }
      }
      while (!ah_1_r.isFinished());
      HtmlForm form_1 = (HtmlForm)
```

```
page_1.getFormByName("chatForm");
    HtmlInput name = form_1.getInputByName("name");
    HtmlInput input = form_1.getInputByName("chatbarText");
    HtmlSubmitInput submit = (HtmlSubmitInput) form_1.getInputByName(
        "submit");
    //set username and message and send everything
    String nameString = "me";
    name.setValueAttribute(nameString);
    input.focus();
    String message = "hurra!";
    input.setValueAttribute(message);
    submit.click();
    //wait until sending has finished
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah_1_s.isFinished());
    //wait until the next receiving thread starts

    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah_1_r.isFinished());
    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah_1_r.isFinished());
    // parse the outputlist for the first element
    List list_1 = page_1.getByXPath(
        "//ul[attribute::id='outputList']/li");
    assertFalse(list_1.isEmpty());
    HtmlListItem li_1 = (HtmlListItem) list_1.get(0);
    HtmlUnorderedList ul_1 = (HtmlUnorderedList) page_1.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul_1.asXml().contains(message));
    assertTrue(ul_1.asXml().contains(nameString));
    assertTrue(li_1.asXml().contains(message));
    assertTrue(webClient_1.getNumberOfAjaxErrorsWithoutWarnings() ==
               0);
    // the same with the second client
    int i1 = 0;
    do {
      synchronized (this) {
        wait(100);
      }
      i1++;
    }
    while (ah_2_r.isFinished());
    // wait until the XHR for receiving finishes
    int i2 = 0;
    do {
      synchronized (this) {
        wait(100);
      }
      i2++;
    }
    while (!ah_2_r.isFinished());
    List list_2 = page_2.getByXPath(
        "//ul[attribute::id='outputList']/li");
    System.err.println(i1 + " : " + i2);
    assertFalse(list_2.isEmpty());
    HtmlUnorderedList ul_2 = (HtmlUnorderedList) page_2.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul_2.asXml().contains(message));
    assertTrue(ul_2.asXml().contains(nameString));
    assertTrue(webClient_2.getNumberOfAjaxErrorsWithoutWarnings() ==
               0);
  }

  public void testCase_2_AjaxHandlerImroved() throws Exception {
    final String nameString = "me";
    final String message = "hurra!";
    //configure the browser
    final WebClient webClient_1 = new WebClient(BrowserVersion.
                                                FIREFOX_2);
    // create two default ajax handler: one for receiving
    // and one for sending chet messages
    AjaxHandler ah_1_s = new AjaxHandler();
    webClient_1.addAjaxHandler(ah_1_s);
    AjaxHandler ah_1_r = new AjaxHandler();
    webClient_1.addAjaxHandler(ah_1_r);
    AjaxHandler ah_1_r_2 = new AjaxHandler() {
      @Override
      public void onChange(int state) {
        if (state == 4) {
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(nameString) >= 0);
          assertTrue(getXMLHttpRequest().jsxGet_responseText().
                     indexOf(message) >= 0);
        }
      }
    };
    //the same with the second browser
    //configure the browser
    final WebClient webClient_2 = new WebClient(BrowserVersion.
                                                FIREFOX_2);
    // create two default ajax handler: one for receiving
    // and one for sending chet messages
    AjaxHandler ah_2_s = new AjaxHandler();
    webClient_2.addAjaxHandler(ah_2_s);
    AjaxHandler ah_2_r = new AjaxHandler();
    webClient_2.addAjaxHandler(ah_2_r);
    //connect to the page and find all necessary elements
    final HtmlPage page_1 = (HtmlPage) webClient_1.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page_1, "live chat");
    final HtmlPage page_2 = (HtmlPage) webClient_2.getPage(
        "http://chat.plasticshore.com/");
    WebAssert.assertTitleContains(page_2, "live chat");
    //force loading all current messages
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah_1_r.isFinished());
    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah_1_r.isFinished());
    HtmlForm form_1 = (HtmlForm) page_1.getFormByName("chatForm");
    HtmlInput name = form_1.getInputByName("name");
    HtmlInput input = form_1.getInputByName("chatbarText");
    HtmlSubmitInput submit = (HtmlSubmitInput) form_1.getInputByName(
        "submit");
    //set username and message and send everything
    name.setValueAttribute(nameString);
    input.focus();
    input.setValueAttribute(message);
    submit.click();

    ah_1_r.removeHandler();
    webClient_1.addAjaxHandler(ah_1_r_2);
    //wait until sending has finished
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah_1_s.isFinished());
    //wait until the next receiving thread starts

    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (ah_1_r_2.isFinished());
    // wait until the XHR for receiving finishes
    do {
      synchronized (this) {
        wait(100);
      }
    }
    while (!ah_1_r_2.isFinished());
    // parse the outputlist for the first element
    List list_1 = page_1.getByXPath(
        "//ul[attribute::id='outputList']/li");
    assertFalse(list_1.isEmpty());
    HtmlListItem li_1 = (HtmlListItem) list_1.get(0);
    HtmlUnorderedList ul_1 = (HtmlUnorderedList) page_1.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul_1.asXml().contains(message));
    assertTrue(ul_1.asXml().contains(nameString));
    assertTrue(li_1.asXml().contains(message));
    assertTrue(webClient_1.getNumberOfAjaxErrorsWithoutWarnings() ==
               0);
    // the same with the second client
    int i1 = 0;
    do {
      synchronized (this) {
        wait(100);
      }
      i1++;
    }
    while (ah_2_r.isFinished());
    // wait until the XHR for receiving finishes
    int i2 = 0;
    do {
      synchronized (this) {
        wait(100);
      }
      i2++;
    }
    while (!ah_2_r.isFinished());
    List list_2 = page_2.getByXPath(
        "//ul[attribute::id='outputList']/li");
    System.err.println(i1 + " : " + i2);
    assertFalse(list_2.isEmpty());
    HtmlUnorderedList ul_2 = (HtmlUnorderedList) page_2.getByXPath(
        "//ul[attribute::id='outputList']").get(0);
    assertTrue(ul_2.asXml().contains(message));
    assertTrue(ul_2.asXml().contains(nameString));
    assertTrue(webClient_2.getNumberOfAjaxErrorsWithoutWarnings() ==
               0);
  }

}
```