

# BSc APPLIED COMPUTING PROJECT

## AJAX VS. JAVA APPLETS:

### A COMPARISON OF TWO ARCHITECTURES FOR BUILDING QUALITY WEB APPLICATIONS

MARC WICKENS

10450406

MAY 2007

## DECLARATION

“I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of BSc Applied Computing being studied at Thames Valley University.

I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised another’s work”.

## Abstract

This document explores the differences between developing web based applications using the Ajax (Asynchronous JavaScript and XML) architecture and the Java Applet architecture. In particular it compares how each style lends its self to developing what can be considered a good quality piece of software. The quality properties portability, scalability, reliability and usability are each put to the test with a combination of secondary data and first hand experiments.

# 1 TABLE OF CONTENTS

<b>2</b>	Introduction .....	8
2.1	Background .....	8
2.1.1	Ajax (asynchronous JavaScript and xml) .....	8
2.1.2	Java Applets .....	9
2.1.3	Quality Attributes .....	9
<b>3</b>	Literature Review .....	10
3.1.1	Java Applets .....	10
3.1.2	Ajax .....	11
3.2	Evaluation of Literature Review .....	15
<b>4</b>	Methodology .....	16
4.1	Overview .....	16
4.2	What Will be measured, and how? .....	17
4.2.1	Portability .....	17
4.2.2	Reliability .....	20
4.2.3	Measuring Usability .....	22
4.2.4	Scalability .....	24
4.3	Development Methodology .....	25
4.3.1	Spiral .....	25
4.3.2	Extreme Programming .....	25
4.4	Evaluation of Methodology .....	26
4.4.1	Why measure these quality properties? .....	26
4.4.2	Are the measurements appropriate? .....	26
4.5	Design .....	27
4.6	Overview .....	27

4.6.1	Application 1: Simple XML tag reading.....	28
4.6.2	Application 2: Putting the XML nodes into a TreeView .....	29
4.6.3	Application 3: The Stress Test Application .....	30
<b>5</b>	<b>Development .....</b>	<b>31</b>
5.1	Overview.....	31
5.1.1	List of Applications:.....	31
5.1.2	Application 1 (Page 36).....	31
5.1.3	Application 2 (Page 38).....	31
5.1.4	Application 3 (Page 42).....	31
5.2	First Ajax Application .....	32
5.3	First Java Applet .....	34
5.4	Application 1: The Ajax Version .....	35
5.5	Application 1: The Java Applet Version.....	36
5.6	Application 2: The Ajax Version .....	37
5.7	Application 2: The Java Applet Version .....	39
5.8	Application 3 (Ajax Version) .....	41
5.9	Application 3 (Java Applet Version).....	42
<b>6</b>	<b>Experiments &amp; Research Results .....</b>	<b>43</b>
6.1	Browser Statistics.....	43
6.1.1	Who is using what browser .....	43
6.2	Code Portability Research .....	44
6.2.1	Ajax.....	44
6.2.2	Java Applets .....	44
6.2.3	Testing The Applications for Cross Browser Compatibility .....	45
6.3	Testing the Speed of Execution .....	54

6.4	Modem Download Times .....	57
6.5	Code Complexity .....	58
<b>7</b>	<b>Analysis of Results .....</b>	<b>61</b>
7.1	Browser Statistics.....	62
7.2	Averages .....	62
7.2.1	Conclusion.....	62
7.3	Code Portability – Analysis .....	64
7.3.1	Extra Nodes in Mozilla Firefox and Internet Explorer:.....	66
7.3.2	Conclusion.....	67
7.4	Testing the Speed of Execution – Analysis.....	68
7.4.1	Ajax Application 3.....	68
7.4.2	Java Applet Application 3 .....	70
7.4.3	Conclusion.....	70
7.5	Modem Download Times – Analysis .....	72
7.5.1	Conclusion.....	72
7.6	Code Complexity – Analysis.....	73
7.6.1	Conclusion.....	73
<b>8</b>	<b>Overall Conclusion.....</b>	<b>74</b>
<b>9</b>	<b>Bibliography .....</b>	<b>76</b>
<b>10</b>	<b>APPENDICES .....</b>	<b>79</b>
10.1	XML Document used in tests .....	79
10.2	Ajax Application 1 .....	80
10.3	Ajax Application 2.....	82
10.4	Ajax Application 3.....	84
10.5	Java Applet Application 1 .....	86

10.6	Java Application 2.....	90
10.7	Java Application 3.....	93
10.8	JSP Page .....	97
10.9	Email From Sun .....	98

## Table of Figures

Figure 3–1.....	20
3–2:A tree View component in the collapsed view .....	29
Figure 3–3: a tree view component in the expanded view. ....	29
4–1: The Ajax learning exercise in Internet Explorer 7 .....	32
4–2: The Ajax learning exercise in Firefox 2.....	32
4–3: The Java Applet learning exercise in Firefox.....	34
4–4: The Java Applet learning exercise in Internet Explorer 7 .....	34
4–5: Ajax Application 1 in Internet Explorer (IE) .....	35
4–6: Ajax Application 1 in Firefox .....	35
4–7: Java Applet Application 1 in Firefox .....	36
4–8Java Applet Application 1 in IE.....	36
4–9: Ajax Application 2 in IE.....	37
4–10: : Ajax Application 2 in Firefox.....	37
Figure 4–11: Java Applet Application 2 in IE.....	39
Figure 4–12: : Java Applet Application 2 in Firefox .....	39
Figure 5–1: Testing Ajax application 1 in IE, all states. ....	46
Figure 5–2: Testing Ajax application 1 in IE, northern states. ....	46
Figure 5–3: Testing Ajax application 1 in Firefox, all states. ....	47
Figure 5–4: : Testing Ajax application 1 in Firefox, northern states. ....	47
Figure 5–5: Testing Java Applet Application 1 in IE, all states.....	48
Figure 5–6: Testing Java Applet Application 1 in IE, northern states.....	48
Figure 5–7: Testing Java Applet Application 1 in Firefox, all states.....	49

Figure 5—8: Testing Java Applet Application 1 in Firefox, northern states .....	49
Figure 5—9: Testing Ajax application 2 in internet explorer .....	50
Figure 5—10: Testing Ajax application 2 in Firefox, showing all nodes. ....	51
Figure 5—11: Testing Java Applet Application 2 in IE, showing all nodes. ....	52
Figure 5—12: Testing Java Applet Application 2 in Firefox, showing all nodes. ....	53
Figure 6—1: Fixed: Application 2 in IE .....	64
Figure 6—2: Firefox showing the value of mNode.childNodes.length .....	65
Figure 6—3: IE showing the value of mNode.childNodes.length .....	65
Figure 6—4: Firefox caused the whole computer to slow down. ....	69

## Acknowledgements

It should be acknowledged that all of the staff at TVU have provided a lot of support and inspiration throughout this project. In particular, my supervisor Andrew Barnes who made himself available for discussions often, and was always quick to respond when I asked many, many question through email - often just for reassurance. I would also like to acknowledge the help Graham Wilmott has provided as a previous student, his advice on such topics as layout and writing style has been very useful.

## 2 INTRODUCTION

This project aims to determine which of the two software architectures, Ajax and Java Applets are better suited to developing a web application than can be considered good quality.

The project will compare two techniques for developing “desktop class” web-based applications. Desktop-class will be defined as “application that resembles a desktop application by providing a responsive user interface, with common controls such as menus, toolbars and buttons”

A responsive user interface can be defined as “an interface where the results of an event triggered by the user are displayed in a timely manor, if they cannot be displayed in a timely manor then the user is given an indication and is made aware of that fact”

Many web applications are comprised of multiple tiers, commonly a client and a server. The scope of this project will be to look at client side development.

### 2.1 BACKGROUND

#### 2.1.1 AJAX (ASYNCHRONOUS JAVASCRIPT AND XML)

The web has long been a place of static HTML files connected through the use of hyperlinks. Web pages could be generated dynamically using server side programs, however in order to see any change the user would be required to load a new page.

**Ajax** allows new information to be loaded without reloading the page - a development that has lead to many new rich web applications being created. Many Ajax applications use Cascading Style Sheets (CSS) and JavaScript to recreate the common interface components found in many desktop applications such as menus, tree lists and tabs.

### 2.1.2 JAVA APPLETS

A Java Applet is a small Java program designed to run within the browser. Applets are able to make use of the built in features of Java providing end-users with an application that looks much like a native<sup>1</sup> application that runs directly from within their web browser. The only requirement is that the user has the Java plug-in installed for their web browser. The Java Applet is a Jar file stored on a web server, embedded into a web page using Hyper Text Mark-up Language<sup>2</sup> (HTML). Java applets run inside a sandbox with no access to the user's hard disk or any web server other than the one it is loaded from, this can be overcome by having the applet digitally signed by a trusted third party. JavaScript can also make use of the functions provided within a Java Applet.

### 2.1.3 QUALITY ATTRIBUTES

Measuring software is not a straightforward task. From the users' perspective a good piece of software will be easy to use and fail infrequently. From a developer's perspective a good piece of software will be easy to maintain. This project will take the developers perspective and look specifically at which of the two architectures lends itself to creating a quality desktop-class web-based application. The literature review will look at which quality attributes are recognised as applying to web applications.

---

<sup>1</sup> An application that looks native will have the same look and feel as the operating system.

<sup>2</sup> HTML is the language used to display information on a web page

## 3 LITERATURE REVIEW

### 3.1.1 JAVA APPLETS

According to its creators, a Java Applet is *“a special kind of Java program that a browser enabled with Java technology can download from the internet and run”* (Sun Microsystems Inc, 2004). One advantage of Java Applets is that they act just like desktop applications, only executed within the web browser. *“They allow for full user interface design, whereas HTML has a limited number of user interface components”* (Potts, 1998). This is not without drawbacks, *“Applets incur a download penalty which servlets<sup>3</sup> do not. Their size has to be kept to a minimum in order to ensure the download time is minimised, this is not the case with servlets. In tests the applet was slower than the servlet.”* One of the reasons for carrying out this project was to find out why Java Applets are not a lot more common, (Asleson & Schutta, 2005) consider that one reason why Java Applets have not gained popularity is because *“developers have to make sure the client has the proper version of Java installed”* although (Cadenhead & Lemay, 2004) disagree *“Java remains an effective choice for Web-based programming”*. (Burdy, Requet, & Lanet, 2003). Both of the previous statements are opinion, the “anti-applet” one is from a book about Ajax and the “pro-applet” opinion is from a book about Java.

---

<sup>3</sup> A servlet is a Java application that runs on a web server.

### 3.1.2 AJAX

Ajax is an acronym that stands for Asynchronous JavaScript And XML.

*“In essence, HTTP<sup>4</sup> requests can be made and responses received, completely in the background and without the user experiencing any visual interruptions” (McLellan, 2005). Asynchronous means that a request can be sent to the server from the client and client can continue working until a response is received.*

#### 3.1.2.1 JAVASCRIPT

---

JavaScript, an implementation of ECMAScript is seen to have some problems, and is often viewed as incomplete, *“The ECMAScript Programming Language went from prototype to world wide acceptance in a remarkably short time. This led to premature standardization, locking some mistakes and inconveniences into the language specification.” (Crockford, 2007)*

Yet applications written in JavaScript have never been more common, (Garrett, 2005) points out that *“Google is making a huge investment in developing the Ajax approach. All of the major products Google has introduced over the last year — Orkut, Gmail, the latest beta version of Google Groups, Google Suggest, and Google Maps — are Ajax applications.”*

There are a number of factors that conspire against JavaScript, *“even today the same script may behave differently across browsers”, “clients are free to turn off JavaScript”, “the difficulty of developing JavaScript” (Asleson & Schutta, 2005) . On the other hand, (Hillier, 1996) states “JavaScript is simple to use, lightweight, and dynamic. Developers can easily embed code functionality for interactive applications inside a web page”.*

Ajax Applications rely on the Document Object Model (DOM). The Document Object Model is a *“fully object-orientated representation of the page that can be modified with a scripting language such as JavaScript” (Asleson & Schutta, 2005)*

The most important aspect of JavaScript in the context of Ajax is the XMLHttpRequest object. JavaScript is used to create the XMLHttpRequest object,

---

<sup>4</sup> HTTP stands for Hyper Text Transfer Protocol. It is the protocol used to access documents stored on a web server.

and also to parse the XML received from it. (Ballard, 2006) Introduces the XMLHttpRequest object as an *“object to allow JavaScript to formulate HTTP requests and submit them to the server. Such calls can be made asynchronously in the background, allowing you to continue using the page without interruption of a browser refresh and the loading of a new or revised page.”* However different web browsers have different implementations of the XMLHttpRequest object, which can contribute to the problems expressed by (Asleson & Schutta, 2005).

### 3.1.2.2 XML

---

XML stands for eXtensible Mark-up Language. It is a language that *“allows data to be stored in a structured, hierarchal manner”* (Ballard, 2006). Most Ajax applications send a request to a server running an application written in a server side language such as PHP(Personal Home Page), JSP (Java Server Pages) or ASP (Microsoft Active Server Pages), it is this program that sends back the appropriate XML, which could even be a static XML file.

### 3.1.2.3 QUALITY

---

(Sommerville, 2004) lists a total of 15 quality attributes that can be applied to software, and measured, of which some are portability, reliability, usability and maintainability. (Offutt, 2002) Has a similar list that is more specific to web applications, including scalability, reliability, security and usability. It is surprising that (Offutt, 2002) does not mention portability for web applications, his assumption being at the front-end of the web application will be HTML and therefore must work in any web browser. If the client is to have any sort of interactivity then it must use more than plain old HTML.

#### 3.1.2.3.1 Portability

Portability is *“the ability of an implementation to be transferred from one environment to another”* (SEI, 2207). The essence of a web application is that it should run when the user visits regardless of their web browser of operating system.

(Sommerville, 2004) links good portability to the number of lines of code, suggesting that in order to port software, code will need to be changed. Web applications

however cannot be ported in this manner; they must be able to run in different environments without the need to edit code

#### 3.1.2.3.2 Scalability

Scalability is *“How well a solution to a given issue will work when the size of the issue increases”* (GMM, 2007).

Much of the literature on measuring scalability refers to the back-end system, with research focusing on back-end scalability web server’s ability to deal with many simultaneous requests (Guitart, Beltran, Carrera, & Ayguadé, 2005). (Offutt, 2002) *“As Web sites grow, small software weaknesses that had no initial noticeable effects can lead to failures (reliability problems), usability problems, and security breaches.”* So it is clear that scalability is a core property of quality.

#### 3.1.2.4 RELIABILITY

---

Reliability is *“the ability of a system or component to perform its required functions under stated conditions for a specified period of time”*. (Hammer, Rosenberg, & Shaw, 1998). (Sommerville, 2004) states *“The overall reliability of a program, therefore mostly depends on the number of inputs causing erroneous outputs during normal use of the system”*. Reliability is reduced as code becomes more complex, and so reliability can be measured by counting the number of lines of code (Sommerville, 2004) however, (Hammer et al, 1998) states *“While there are many different types of complexity measurements, the one used by the SATC<sup>5</sup> is logical Cyclomatic complexity<sup>6</sup>”* this reasoning for this is because coding styles may impact on the number of lines of code, for example a programming language such as Java will allow developers many options, even down to whether or not to place braces on new lines.

---

<sup>5</sup> Software Assurance Technology Centre

<sup>6</sup> See page 17 for a definition of cyclomatic complexity

#### 3.1.2.4.1 Usability

Usability is “*a quality attribute that assesses how easy user interfaces are to use*”

(Nielsen, 2003). There are 5 main points, Learnability, Efficiency, Memorability, Errors, Satisfaction (Nielsen, 2003). (Sommerville, 2004) points to “familiarity” as being a key principle of good interface design. (Fried, 2007) found download times to be a key drawback when interviewing the team behind Hotmail (a popular web-based email client) who found that “*It was particularly slow over dial-up connections, still used by roughly a third of Hotmail users*”.

## 3.2 EVALUATION OF LITERATURE REVIEW

Much of the research was conducted with the aim of learning about the two architectures and how they can be compared. I was unable to find any previous comparisons, which made me think that either a comparison is not relevant or nobody has got round to making one yet. There are a lot of informal comparisons between Java Applets and Ajax on the internet, but these are all based on opinion and people's own prejudices with no facts to backup any conclusions.

When researching the properties of quality I picked 4 properties that applied to web applications. With the intention of measuring these properties, I carried out further research. Portability is often measured with the assumption that each line will be ported. Portability will be defined here to mean software that once written, should not need to be rewritten for it to work across different platforms and so a custom mythology will be required to measure it. I also discovered that much of the research into scalability looks at how a server copes with increased load; this project will be focusing on the client side aspects of scalability.

## 4 METHODOLOGY

### 4.1 OVERVIEW

Three applications will be created that will allow these qualities properties to be measured:

- Portability
- Scalability
- Reliability
- Usability

The methodology will measure the these quality properties by running tests on 3 applications each coded using Ajax and Java Applet techniques, so a total of 6 applications.

## 4.2 WHAT WILL BE MEASURED, AND HOW?

### 4.2.1 PORTABILITY

It has been decided that a custom methodology is suited to measuring portability because it is not being measured in the traditional sense. Portability will be measured by testing how well the application runs across different web browsers.

(Sommerville, 2004) suggests counting the number of lines of code to determine portability, implying that the software would be manually ported from one environment to the other. The nature of the web means that web applications are expected to be written once to work on multiple platforms, and so portability cannot be measured by simply counting line numbers. The custom methodology will use two experiments, detailed below. The portability of each of architectures will be determined by a combination of research and experimentation

#### 4.2.1.1 BROWSER STATISTICS

---

Research will be carried out to find out what percentage of users use a web browser that supports either Ajax or Java Applets architecture. It can be determined that if *for example*, it is discovered not many users are running web browsers support Ajax then Ajax does not lend itself to having good portability because it simply won't work across a variety of platforms. It should be noted that that is an example and not the conclusion of this experiment.

#### 4.2.1.2 STRESS TEST

---

Measurements will be obtained firsthand by executing the same Ajax and Java Applet applications in different web browsers and measuring the difference in time taken. A shorter time would indicate that the architecture is more portable because its performance is not as unequal across different platforms.

### 4.2.1.3 IMPLEMENTATION

---

#### 4.2.1.3.1 Browser statistics

Statistics on web browser usage will be gathered and combined to give an overall figure on the percentage of people using Ajax enabled web browsers.

Research conducted on the internet for web browser usage statistics has discovered three web sites offering such statistics, **TheCounter.com**, **Upsdell.com** and **W3Schools.com**. Statistics will be gathered from all three sources, the statistics must all be from the same time period (April 2007) and the same units (percentage of users) and will be averaged to give one set of figures. The results will be analyzed by looking at whether the most used web browsers support Ajax, a percentage of users currently using an Ajax enabled web browser will be the result. TheCounter.com also provides statistics on what percentage of users have Java enabled. A conclusion will be drawn about which architecture has the greatest “reach” and therefore can be considered to be the most portable.

#### 4.2.1.3.2 Stress Test

To test how long it takes for architecture to perform a processing task, a routine will be written that performs a mathematical function and outputs the results to a text box. The time taken to complete the task will give an indication as to which of the platforms is more portable, in the sense that it executes in a timely manor on many platforms. While the traditional view of portability may be that is simply must execute, this project is looking specifically at the quality aspect.

#### 4.2.1.3.3 Processing Task

The Processing task will involve an integer being incremented from 1 to 500 inside a FOR loop. On each iteration the square root of the integer will be calculated and the result will be added to the text box, previous results are kept in the text box with a line break character separating them (so it can be proven that the square root of each number was indeed found).

#### 4.2.1.3.4 Measuring the time

The experiment consists of executing the code inside a Java Applet, and an Ajax application in the two most popular web browsers (determined in experiment 1).

A server will host a JSP page that prints the time to the server console. The application being tested will make a call to the page before the processing task, and after the processing task. The two times can be compared to get the length of time it took.

The architecture that has the greatest difference in times between web browsers will be determined to be less portable, because the performance of applications will vary the most across different platforms.

### **The Equipment**

The experiment will use two computers, A the client – and B the server hosting the JSP page. The server is on a local area network, so it is noted that network latency may add to the times here but the circumstances will be the same for both sets of tests. To avoid random network activity or background processes on either the server or the client interfering with the results, the tests will be carried out 4 times and an average taken. It is also noted that the server's CPU may take some time to create the new date object and print it. Since this remains equal to all experiments, comparison will still be relevant.

#### **Client:**

Celeron 2.93Ghz

1280Mb RAM

Microsoft Windows XP SP2 with JRE1.5

#### **Server:**

Celeron 2.93Ghz

1024Mb RAM

Fedora Core 5 running Tomcat 5

## 4.2.2 RELIABILITY

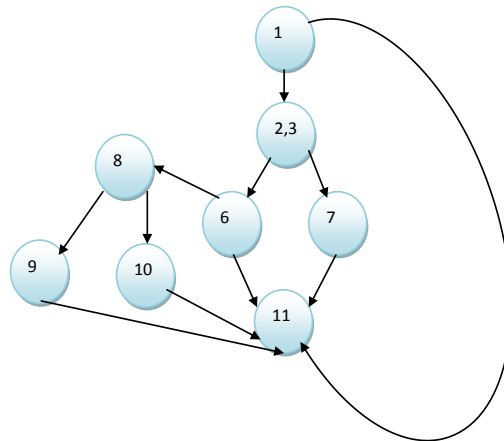
From the literature review it can be determined that the more lines of code, the greater the chances of a bug occurring. Counting lines of code however is not always a good measurement of complexity, since individual programming styles and different syntax of different languages may produce results which are not comparable. The methodology suggested by, (Hammer et al , 1998) will be used to determine the complexity of the code. (Sommerville, 2004) states that when complexity increases, reliability decreases.

### 4.2.2.1 MEASING RELIABILITY

The methodology will be to measure the cyclomatic complexity of the code. “Cyclomatic Complexity is a software metric that provides a quantitative measure of the logical complexity of a program. The value computed for cyclomatic complexity defines the number of independent paths in the program” (Pressman, 2005). Cyclomatic complexity is calculated using the following formula

#### 4.2.2.1.1 Cyclomatic Complexity

$$V(G) = E - N + 2$$



**Figure 4–1**

A flow diagram showing different execution paths.

E is the number of arrows (edges) and N is the number of nodes (numbered circles). Therefore the cyclomatic complexity number (CCN) in this example is 5. For more on cyclomatic complexity; see (Pressman, 2005, pp. 426-429).

The (CCN) can also be calculated using an automated C.A.S.E. tool, such as CCCC<sup>7</sup>. CCCC will calculate the CCN for Java and C++ source code files. Research has indicated that no such tools exist for measuring JavaScript, and so it will be calculated manually.

#### 4.2.2.2 WHY THIS METHODOLOGY?

---

The aim of this experiment is to find out which architecture is best suited to create a reliable application. By implementing two functionally identical applications using both architectures and comparing the complexity this can be determined. From the initial research it is known that JavaScript and Java are syntactically similar, and so the appropriateness of the methodology will be covered in the conclusion.

---

<sup>7</sup> Web site: <http://cccc.sourceforge.net/>

### 4.2.3 MEASURING USABILITY

#### 4.2.3.1 OVERVIEW

---

(Sommerville, 2004) points to “familiarity” as being a key principle of good interface design and since the scope usability is very large, here the assumption will be made that use of familiar graphical user interface components will contribute to good usability, enhancing learnability, memorability and satisfaction. It is beyond the scope of this project to debate whether familiarity does indeed lead to good usability.

The test will look at the range of interface widgets readily available to developers, and take one example for Ajax and one for Applet and implement them. The comparison will compare the number of the steps involved in implementing the component. The aim of the test is not to see how usable each of the architectures is, but to see how easy it is for the developer to implement and interface which can be considered easy to use.

#### 4.2.3.2 56K DOWNLOAD TIME

---

The literature survey found that according to its creators, around 33% of Hotmail users use modems. Both implementations of Application 1 will be accessed from a computer connected to the Internet via a 56K modem. The Tamper plug-in for Firefox will be used to measure how long each takes. To avoid random internet speed variations, 4 measurements will be taken for each and then averaged.

#### 4.2.3.3 IMPLEMENTING AN INTERFACE COMPONENT

---

Application 1 downloads parses an XML file, displaying the contents in a text box. The task will be to create a second application known as “Application 2” that downloads the XML file and puts its contents into a Tree List control. A Tree List is a popular user interface control for view hieratical data, and is therefore well suited to this type of data. The root of the tree indicates the top level. The user clicks on the root, to show a list of branches. The user can then click a branch and the list of nodes will be displayed. Nodes can be mixed in with branches and the theoretical depth of a tree is unlimited. The modification will attempt to display a root called “states” followed by four branches “north” “south” “west” and “east” that will each display the corresponding “state” entities from the XML file beneath them. The tree view control was chosen because it is found often in desktop applications. The Apple Macintosh Finder application uses Tree View, so does Microsoft Windows Explorer and Microsoft Outlook.

A TreeView control will be implemented for each of the architectures, specifically for the Java Applet the Swing component JTree will be used and for the Ajax application it will be the Yahoo TreeView control. The number of steps needed to paint the control on the screen without any customisation will be compared and a conclusion drawn as to which was easier to implement. For simplicity, the TreeView will be left in its default position during the first iteration, with the option to specify where it appears on the screen at a later date.

This is a custom methodology and is a way of trying to measure a very abstract concept of something being “easy to implement”. The reason for wanting to measure this is because if the architecture makes developing interfaces with good usability properties easy in itself, then it can be concluded that that architecture it better suited to created quality usable interfaces.

## 4.2.4 SCALABILITY

### 4.2.4.1 OVERVIEW

---

The method used by (Guitart, Beltran, Carrera, & Ayguadé, 2005) is not applicable to a client. This project will look at the client side, the “stress test” aspects of previous research (Guitart, Beltran, Carrera, & Ayguadé, 2005) will be taken and applied to the client.

The scenario proposed would be a situation where the client must carry out a large processing task. Such tasks are common in many applications, for example a web based email application may be required to sort a list of emails by date. The question is, while both technologies may perform well when the user has 30 emails in their inbox, would it if they had 400?

To test the scenario above; the results from the stress test application used to measure the difference in processing time between web browsers will be analysed.

The results from the Portability experiment will be examined, this time rather than looking for a difference in the time the processing took depending on which browser is used; the two development techniques Ajax and Java Applets will be compared. The user interface should ideally remain responsive while the progressing occurs, this is usually implemented by using two threads and so support for multithreading will also be determined.

By finding out which of the two architectures performs faster with large processing tasks, and if they support multithreading a conclusion can be made as which has the properties of scalability.

## 4.3 DEVELOPMENT METHODOLOGY

Both Java Applets and Ajax are will require new skills to be developed along the way. It has therefore been decided that a Spiral (Boehm, 1988) methodology with some elements of Extreme Programming (Beck, 2000) with be followed.

### 4.3.1 SPIRAL

Introduced in 1988, the spiral methodology suggests an iterative approach to software development. By excluding the feasibility study and concentrating on development, as new skills are developed they can be applied to each iteration.

### 4.3.2 EXTREME PROGRAMMING

Developed by (Beck, 2000) Extreme Programming is an iterative approach to software development, with short cycles and an “*evolutionary design process that lasts as long as the system lasts*”. This fits in well with developing software that requires new skills to be learnt “on the job”. XP also recommends software developers do not work after hours; “*No one can put in 60 hours a week for many weeks and still be fresh and creative*” (Beck, 2000, p. 60).

## 4.4 EVALUATION OF METHODOLOGY

### 4.4.1 WHY MEASURE THESE QUALITY PROPERTIES?

I have decided on these 4 quality properties, mainly because (Offutt, 2002) and (Sommerville, 2004) explicitly state them as important.

I have decided not to attempt to measure other quality properties such as security because this would require a separate in-depth study into the JavaScript implementations of the commonly used web browsers, as well as the Java Virtual Machine and would also depend on the operating system.

(Nichols & Peterson, 2007) Have developed a framework for securing web applications, it was however their conclusion that any code that blocks un-validated input and injection flaws should be implemented on the server, not exclusively the client. In conclusion I have decided to investigate what I deem the most relevant properties of quality, with the exception of security which I feel with the time and resources I would have not have been able complete a detailed study.

### 4.4.2 ARE THE MEASUREMENTS APPROPRIATE?

The methodology is based on that recommended by (Sommerville, 2004) and (Guitart, Beltran, Carrera, & Ayguadé, 2005) but with some modification because this project is testing the client and not the server. It uses a combination of secondary data that has already been published, such as web browser usage statistics, and primary data collected in the stress test. By applying these two approaches, I am confident that a solid conclusion can be drawn as to which architecture is best suited to developing quality web applications.

## 4.5 DESIGN

### 4.6 OVERVIEW

Three applications will be created; each will be implemented using Ajax techniques and using a Java Applet.

This section proposes three applications. How they are used in experiments will be explained in the in the “what will be measured” section. How the applications were built will be detailed in the development section.

Since these applications are not intended to ever be used in a real world environment and their only purpose is to test theories and concepts for this project, it was not deemed necessary to carry out a feasibility study, specify user requirements or go through a detailed design process with UML diagrams. The testing stage will be the experiments.

#### 4.6.1 APPLICATION 1: SIMPLE XML TAG READING

Application 1 will retrieve an XML file from the server. The XML file contains the states of America in the format that follows:

```
<states> <north> <state> Iowa </state> </north> </states>
```

The application contains two buttons, “show north” and “show all”. The first button displays all the northern states inside a text box, the second button shows all states.

The purpose of this application is so demonstrate a call to the server that updates the screen without a page refresh, and to demonstrate manipulating an XML file using DOM, both common requirements of a web based application.

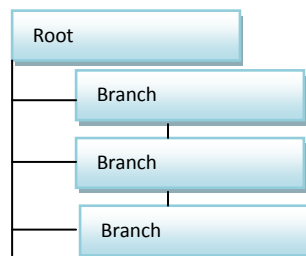
It should be noted that this application is being used in some tests because it was created by following a tutorial in (Asleson & Schutta, 2005).

#### 4.6.2 APPLICATION 2: PUTTING THE XML NODES INTO A TREEVIEW

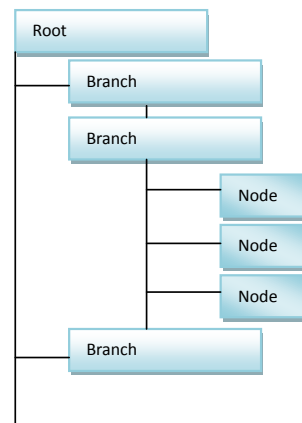
Application 2 builds on the code of Application 1. All nodes are placed inside a 3 level TreeView control. At the top level is States, followed by North, South, East and West, which contain the states beneath them. The Java Applet version uses the Swing JTree component (Sun Microsystems, 2004), while the Ajax version uses The Yahoo TreeView control that is distributed as part of their user interface library.<sup>8</sup>

The purpose of this application was to implement an interface component that is commonly found in desktop applications.

An example of a TreeView:



**4-2: A tree View component in the collapsed view**



**Figure 4-3: a tree view component in the expanded view.**

---

<sup>8</sup> Yahoo UI library web site: <http://developer.yahoo.com/yui/>

#### 4.6.3 APPLICATION 3: THE STRESS TEST APPLICATION

This application is used to test how long it takes to perform a large processing task.

The application increments integer from 0 to 1 inside a “FOR” loop.

During each iteration the square root of the integer is found and stored in a text box.

The purpose of this application is to simulate a large repetitive processing task.

## 5 DEVELOPMENT

### 5.1 OVERVIEW

This section details the development of the 6 applications and the two learning simple learning exercises. Screenshots are provided and anything noteworthy is discussed along with the relevant source code. The actual proof that the applications run, together with the experiment data is provided in the Experiments & Research Results section on page 43.

#### 5.1.1 LIST OF APPLICATIONS:

#### 5.1.2 APPLICATION 1 (PAGE 35)

The solitary requirement for this application was that it use connects to the server to retrieve the XML file and then carries out some business logic. The business logic is allowing the user to filter what is displayed. It will be used in the test on download time.

#### 5.1.3 APPLICATION 2 (PAGE 37)

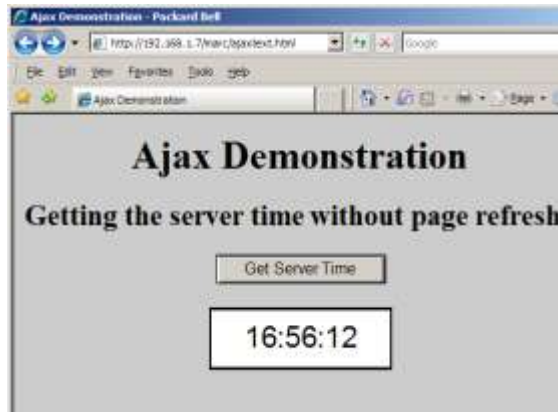
The requirement for this application is that it must retrieve an XML file and display it in a TreeView component. It will be used in the usability and reliability tests.

#### 5.1.4 APPLICATION 3 (PAGE 41)

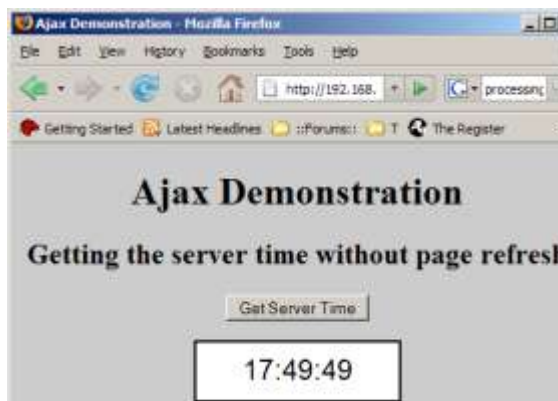
The application is used to simulate a large processing task, and will be used in the portability and scalability tests.

## 5.2 FIRST AJAX APPLICATION

To learn how to use Ajax a sample application<sup>9</sup> was created consisting of two files, the HTML file with JavaScript embedded inside and a PHP file that outputs the time to a very simple XML file.



5—1: The Ajax learning exercise in Internet Explorer 7



5—2: The Ajax learning exercise in Firefox 2

The web page uses an Ajax technique to connect to the server when the user presses the button. This involves creating a new XMLHttpRequest object, one for Internet Explorer using a built in ActiveX object and another native browser object for Mozilla Firefox. It is because of JavaScript's dynamic typing, a variable can change type throughout the lifecycle of the application, this means code can be written that assigns either the Internet Explorer XMLHttpRequest or the Mozilla Firefox version to the same variable, based on what browser is in use. There are many methods of

---

<sup>9</sup> The source code was developed by following a tutorial (Ballard, 2006, p. 114)

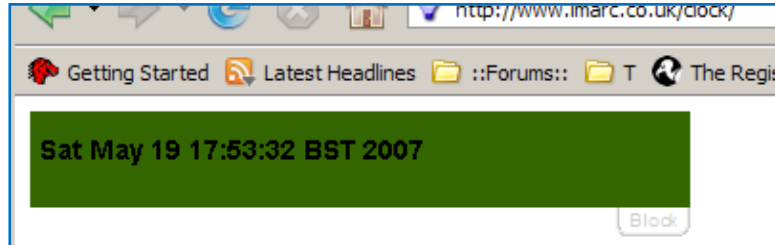
doing this, (Asleson & Schutta, 2005) suggest testing to see if the **window.activex** object exists and if it does then create the ActiveX object, if it doesn't exist then create the native object for Mozilla Firefox/IE7. In contrast (Ballard, 2006) suggests using TRY to create a new native XMLHttpRequest, and then creating an ActiveX object inside the catch statement on the assumption that it will simply throw an exception in Internet Explorer. In addition the version of the code by (Ballard, 2006) works

The following code returns an XMLHttpRequest object for either browser:

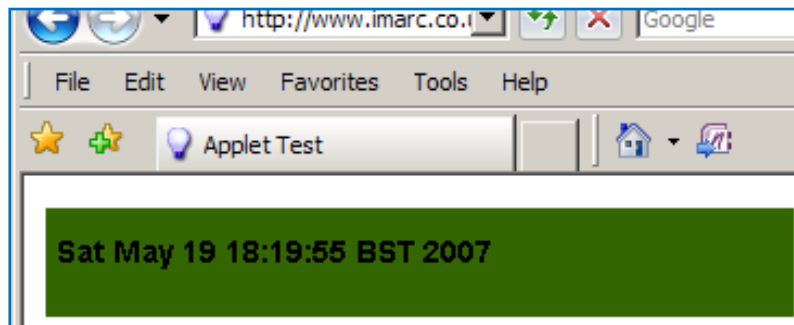
```
1. function getXMLHttpRequest() {
2.   try {
3.     req = new XMLHttpRequest();
4.   } catch(err1) {
5.     try {
6.       req = new ActiveXObject("Msxml2.XMLHTTP");
7.     } catch (err2) {
8.       try {
9.         req = new ActiveXObject("Microsoft.XMLHTTP");
10.      } catch (err3) {
11.        req = false;
12.      }
13.    }
14.  }
15.  return req;
16. }
```

## 5.3 FIRST JAVA APPLET

To learn how to program a Java Applet, this simple clock was created by following a tutorial<sup>10</sup>. Unlike the Ajax clock, this applet gets the time from the local machine.



5–3: The Java Applet learning exercise in Firefox



5–4: The Java Applet learning exercise in Internet Explorer 7

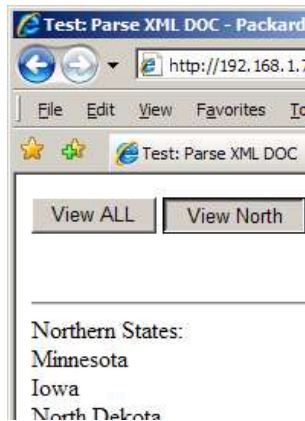
Unlike its Ajax counterpart, Java Applets are developed separately from the web page. They are a java class that extends the `javax.swing.JApplet`, which contains a number of methods that can be overridden. The `paint()` method is overridden here, this method is called when the applet window needs to be displayed or redisplayed.

---

<sup>10</sup>Tutorial followed: (Cadenhead & Lemay, 2004, p. 397)

## 5.4 APPLICATION 1: THE AJAX VERSION

This application uses the XMLHttpRequest object to request and XML file stored on the web server. Using the JavaScript to access the DOM, particularly the **getElementsByTagName** method, the application is able to in effect, filter the list of states, so it will show all; northern states or all states on the list. The application is based on a tutorial (Asleson & Schutta, 2005).



5—5: Ajax Application 1 in Internet Explorer (IE)



5—6: Ajax Application 1 in Firefox

The following code show how the DOM is accessed, here in this code xmlHttp is an instance of the XMLHttpRequest object and responseXML contains the XML file.

```

1. function listNorthStates() {
2.     var xmlDoc = xmlHttp.responseXML;
3.     var northNode =
   xmlDoc.getElementsByTagName("north")[0];
4.     var out = "Northern States";
5.     var northStates =
   northNode.getElementsByTagName("state");
6.
7.     outputList("Northern States:", northStates);
8. }

```

## 5.5 APPLICATION 1: THE JAVA APPLET VERSION

To develop this application knowledge about Java's DOM implementation was required (Sun Microsystems, 2007). Whereas the Ajax version of this application was coded by following a tutorial in a book, this was coded from scratch. The application obtains the XML document from the web server by creating a new URL object and placing the stream into a new Document object. The *Document* can then be parsed using Java's DOM API.



**5—7: Java Applet Application 1 in Firefox**



**5—8 Java Applet Application 1 in IE**

```

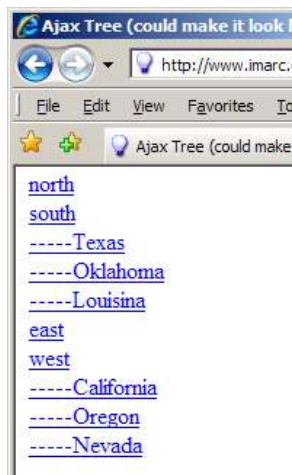
1. NodeList nodeNorth = doc.getElementsByTagName("north");
2.   NodeList statesList =
   nodeNorth.item(0).getChildNodes();
3.   for (int s = 0; s < statesList.getLength(); s++)
4.   {
5.       if (statesList.item(s).getNodeType() == 1)
6.       {
7.           out = out + "\n" +
           statesList.item(s).getChildNodes().item(0).getNodeValue(); } }
   }

```

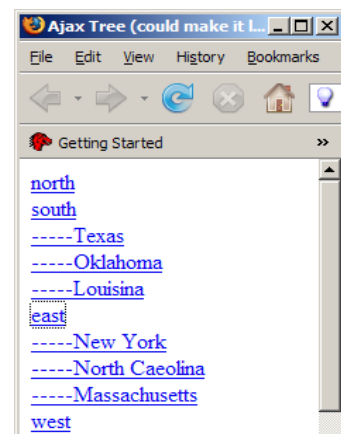
The code performs the same operation as the Ajax version of the application. A “NodeList” is assigned the values of all the nodes with the tag name “north”. A loop is then executed that prints the value. It is necessary to check if the node is the node type of “1” because if its not, an exception is thrown.

## 5.6 APPLICATION 2: THE AJAX VERSION

This is the second iteration of Application 1. This application places the contents of the XML file into a TreeView, which is provided by Yahoo. A number of different TreeView widgets are available on the web, and they can of course be written in JavaScript and CSS from scratch. It was decided to use Yahoo's TreeView control because it was free to use, and the online documentation and tutorials was very helpful.



5—9: Ajax Application 2 in IE



5—10: : Ajax Application 2 in Firefox

```

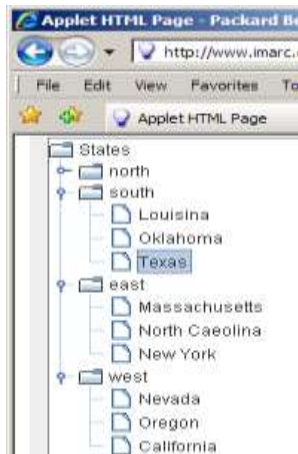
1. function listAllStates()
2. {
3.     var tree;
4.     var xmlDoc = xmlHttp.responseXML;
5.     var mNode =
6.     xmlDoc.getElementsByTagName("states")[0];
7.     var out = "Northern States";
8.     tree = new YAHOO.widget.TreeView("treeDiv1");
9.     var root = tree.getRoot();
10.
11.     for (i=0;i<=(mNode.childNodes.length - 2);i++)
12.     {
13.         if(mNode.childNodes[i].nodeType == 1)
14.         {
15.             var tagNm = mNode.childNodes[i].nodeName;
16.             // alert(tagNm);
17. var northStates = Node.getElementsByTagName(tagNm)[0];
18.             var bStates =
19. northStates.getElementsByTagName("state");
20. var tmpNode = new YAHOO.widget.TextNode(tagNm, root, false);
21. for (j=0;j<=(bStates.length - 1);j++)
22.         {
23. var tmpNode2 = new YAHOO.widget.TextNode("-----" +
24. bStates[j].childNodes[0].nodeValue, tmpNode, false);
25.         }
26.     }
27.     tree.draw(); }

```

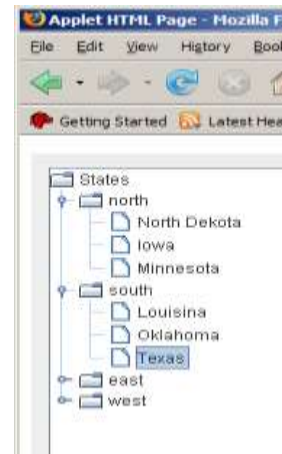
The code on the previous page creates a new TreeView object with the name of the HTML DIV tag that is to be replaced by the tree. This DIV tag can be positioned with CSS. The code then loops round each of the top level nodes, and gets their names, creating a tree node for each with the using nodeName property, and then looping round all of the nodes beneath these top level nodes, adding the children's value using the nodeValue property.

## 5.7 APPLICATION 2: THE JAVA APPLET VERSION

The Java Applet version implements the Swing JTree control. This choice was made due to Swing being distributed with all Java installations, and its main alternative did not provide a TreeView control built in.



**Figure 5—11: Java Applet  
Application 2 in IE**



**Figure 5—12 : Java Applet  
Application 2 in Firefox**

```

1.         NodeList nodeStates2 = nodeStates.item(0).getChildNodes();
2.         root = new DefaultMutableTreeNode("States");
3.         DefaultMutableTreeNode branches = null;
4.         for (int s = 0; s < nodeStates2.getLength(); s++)
5.         {
6.
7.             if(nodeStates2.item(s).getNodeName() == 1)
8.             {
9.                 Node nodeStateMain = nodeStates2.item(s);
10.
11.                 out = out + "\n " + nodeStateMain.getNodeName();
12.
13.                 NodeList nodeNorth =
14. doc.getElementsByTagName(nodeStateMain.getNodeName());
15.
16.                 NodeList statesList = nodeNorth.item(0).getChildNodes()
17. ;
18.                 DefaultMutableTreeNode nodeTop = new
19. DefaultMutableTreeNode(nodeStateMain.getNodeName());
20.                 root.add(nodeTop);
21.                 for (int a = 0; a < statesList.getLength(); a++)
22.                 {
23.                     if(statesList.item(a).getNodeName() == 1)
24.                     {
25.                         out = out + "\n " +
26. statesList.item(a).getChildNodes().item(0).getNodeValue();
27.
28.                         DefaultMutableTreeNode node = new
29. DefaultMutableTreeNode(statesList.item(a).getChildNodes().item(0).getNodeVa
30. lue());
31.
32.                         nodeTop.insert(node, 0) ;
33.                     }
34.                 }
35.             }
36.         }

```

The JTree unlike the Yahoo Tree requires the data model be a separate set of objects. Note that unlike JavaScript, Java uses methods to access the DOM while JavaScript uses properties.



## 5.9 APPLICATION 3 (JAVA APPLET VERSION)

The Java applet contains two methods one called `doIt()` which requests the page, then runs the `calcsqr()` method. It was discovered during testing that the Applet cached the page and did not retrieve it the second time, so the current time in milliseconds is appended to the URL to make it unique. The Ajax version was modified to do the same, as not to make the tests unfair.

```
1. private void doIt() throws MalformedURLException, IOException,
   ParserConfigurationException, SAXException
2. {
3.     Date a = new Date();
4.     URL url = new
   java.net.URL("http://192.168.1.7:8080/timeLog/index.jsp?hello="
   + a.getTime());
5.
6.         calcsqr();
7.         URLConnection urlconn2 = url.openConnection();
8.
9.
10. }
11.
12.
13. private void calcsqr()
14. {
15.     int i;;
16.     for (i = 1; i <= 500; i++)
17.     {
18.
19.         Double o= Math.sqrt(i);
20.         textArea1.setText(textArea1.getText() + "\n" +
   o.toString() + " " + i) ;
21.     }
22.
23. }
```

## 6 EXPERIMENTS & RESEARCH RESULTS

### 6.1 BROWSER STATISTICS

Statistics were gathered from three sources, which all date this information as being gathered during April 2007.

#### 6.1.1 WHO IS USING WHAT BROWSER

TheCounter.com		upsdell.com		W3Schools.com	
<b>MSIE7</b>	13.00%	MSIE7	26.00%	MSIE7	19.20%
<b>MSIE6</b>	56.00%	MSIE6	46.00%	MSIE6	37.30%
<b>Mozilla</b>	12.00%	Mozilla	23.00%	Mozilla	45.90%
<b>Firefox</b>		Firefox		Firefox	
<b>Safari</b>	3.00%	Safari		Safari	1.70%
<b>Opera</b>	1.00%	Opera	12.00%	Opera	1.70%

#### Java Statistics

TheCounter.com also provides statistics on the percentage of users with a Java enabled web browser:

Tue May 1 00:01:02 2007 - Tue May 8 08:58:00 2007 (Seven day period)

Java enabled: 17599388 (94%)

Java disabled: 163586 (0%)

Java unknown: 943415 (5%)

Internet Explorer 6 - 7, FireFox 1 – 2 and Opera 9 fully support the XMLHttpRequest object, but some require their own implementation.

## 6.2 CODE PORTABILITY RESEARCH

### 6.2.1 AJAX

As discovered when coding the first Ajax clock example, it is necessary to create a one XMLHttpRequest object for Internet Explorer 6 and another for Mozilla Firefox and Internet Explorer 7.

(Garrett, 2005) states the two major compatibility issues with Ajax are “differences in JavaScript implementations across browsers and providing alternate means of accessing the applications with older browsers that don’t fully support modern Ajax features. In both cases, additional development effort is required.”. Several Ajax toolkits exist that make the process of creating asynchronous connection to the web server transparent to the developer. The Yahoo User Interface Library for example, contains a Connection Manager<sup>11</sup> component.

### 6.2.2 JAVA APPLETS

Java Applets do not require this separate logic for different web browsers; however some Java code may not work on systems with an old version of the Java Runtime Environment installed. (Sun Microsystems Inc, 2004) suggest in their tutorials writing code to the Java 1.4 standard to ensure it will run on every system. Both Internet Explorer and Mozilla Firefox support the <APPLET> tag for embedding applets.

---

<sup>11</sup> This can be found online at <http://developer.yahoo.com/yui/connection/>

### 6.2.3 TESTING THE APPLICATIONS FOR CROSS BROWSER COMPATIBILITY

Software should be tested in a wide number of operating environments. These tests will ensure that Application 1 and Application 2 both work in Internet Explorer 6 and Mozilla Firefox; the two browsers used the most according the statistics gathered in the previous experiment. It should be noted that the corresponding XML file is printed along with all source code in the appendix.

### 6.2.3.1 AJAX APPLICATION 1 IN INTERNET EXPLORER

---

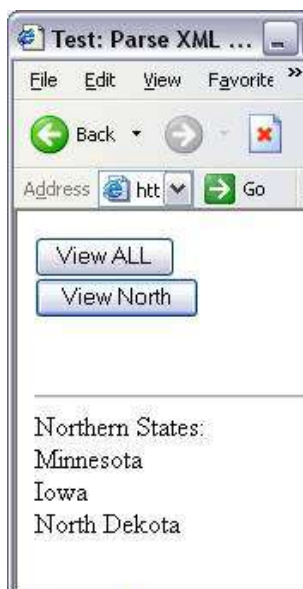
The “View ALL” button is pressed; it should show all the states.



**Figure 6—1: Testing Ajax application 1 in IE, all states.**

The test succeeded.

Now the “View North” button is pressed, only northern states should be shown.



**Figure 6—2: Testing Ajax application 1 in IE, northern states.**

The test succeeded.

### 6.2.3.2 AJAX APPLICATION 1 IN MOZILLA FIREFOX

---

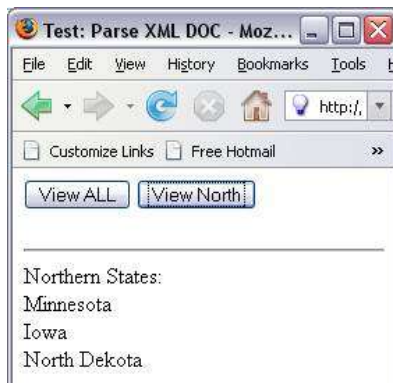
The “View ALL” button is pressed; it should show all the states.



**Figure 6—3: Testing Ajax application 1 in Firefox, all states.**

The test succeeded.

Now the “View North” button is pressed, only northern states should be shown.



**Figure 6—4: : Testing Ajax application 1 in Firefox, northern states.**

The test succeeded.

### 6.2.3.3 JAVA APPLET APPLICATION 1 IN INTERNET EXPLORER

The “Show All” button is pressed:



Figure 6—5: Testing Java Applet Application 1 in IE, all states

Test succeeded.

The show “Show northern” button is pressed:



Figure 6—6: Testing Java Applet Application 1 in IE, northern states

Test succeeded.

### 6.2.3.4 JAVA APPLLET APPLICATION 1 IN MOZILLA FIREFOX

---

The “Show All” button is pressed:



**Figure 6—7: Testing Java Applet Application 1 in Firefox, all states**

Test succeeded.

The show “Show northern” button is pressed:

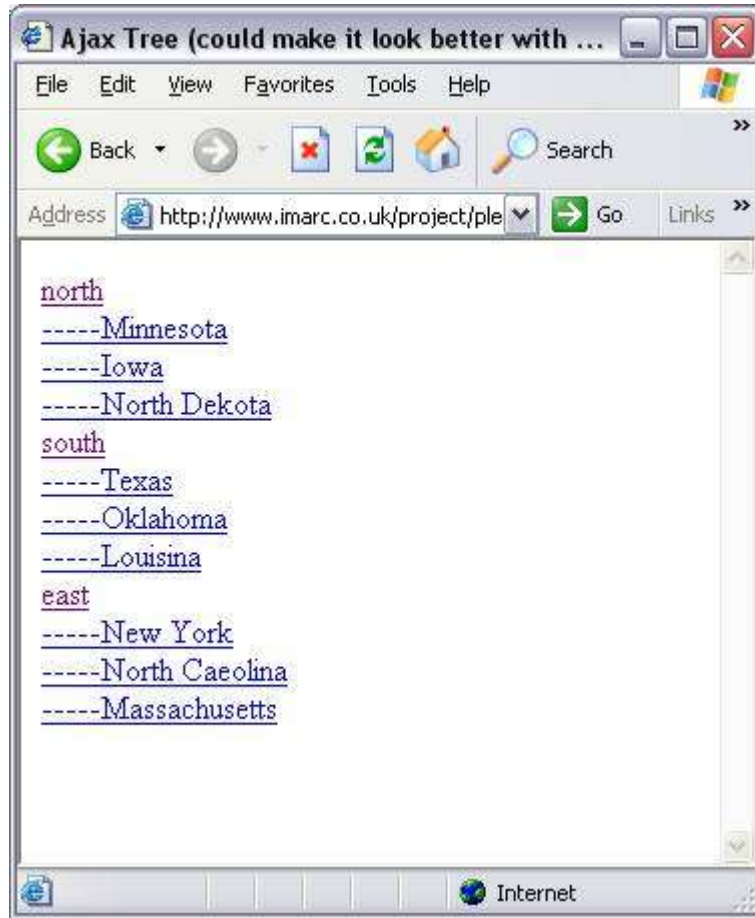


**Figure 6—8: Testing Java Applet Application 1 in Firefox, northern states**

Test succeeded.

### 6.2.3.5 AJAX APPLICATION 2 IN INTERNET EXPLORER

All states should be displayed in under their zone.



**Figure 6—9: Testing Ajax application 2 in internet explorer**

Test failed. The “West” node did not appear.

### 6.2.3.6 AJAX APPLICATION 2 IN MOZILLA FIREFOX

All states should be displayed in under their zone.

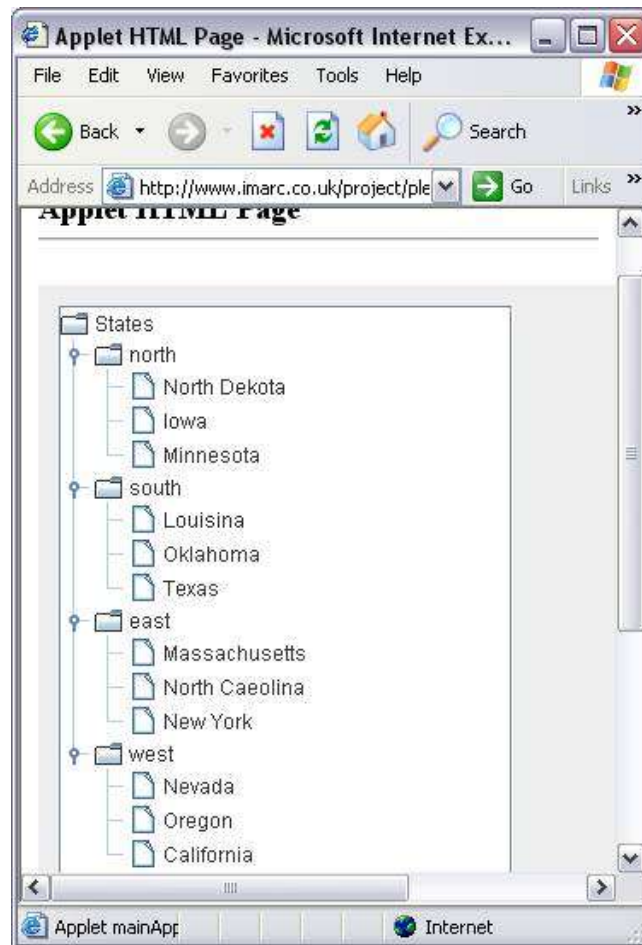


**Figure 6—10: Testing Ajax application 2 in Firefox, showing all nodes.**

Test succeeded.

### 6.2.3.7 JAVA APPLET APPLICATION 2 IN INTERNET EXPLORER

All states should be displayed in under their zone.

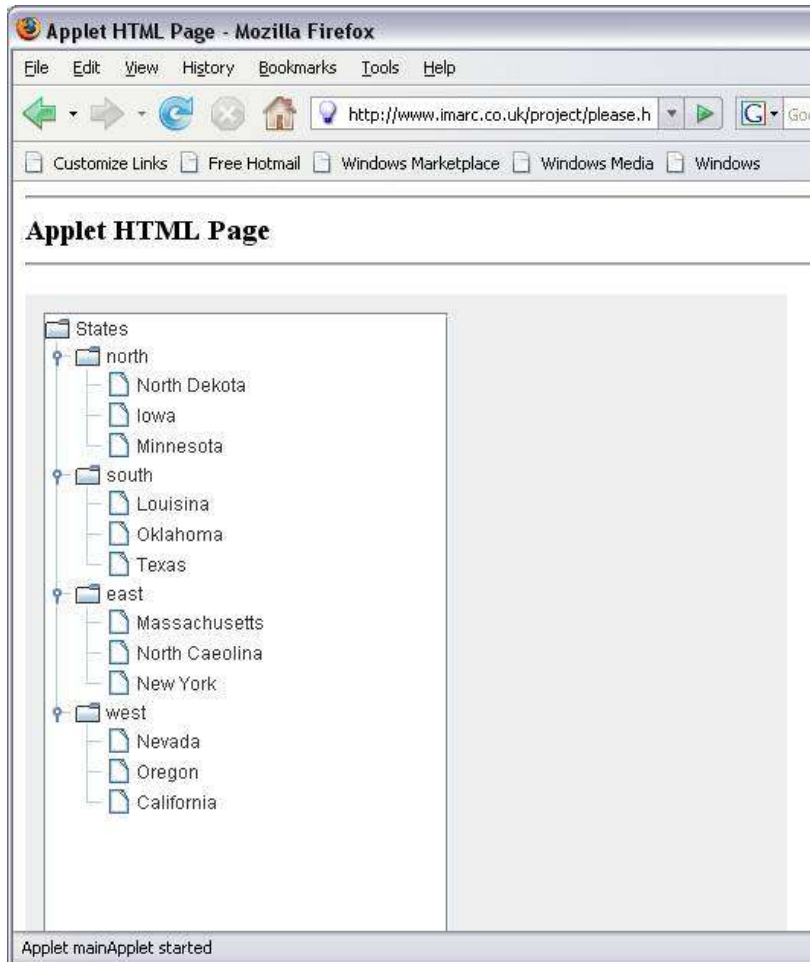


**Figure 6—11: Testing Java Applet Application 2 in IE, showing all nodes.**

Test succeeded. It should be noted that the states are displayed in the in the wrong order, which isn't a fail but it is not the expected result.

### 6.2.3.8 JAVA APPLICATION 2 IN MOZILLA FIREFOX

All states should be displayed in under their zone.



**Figure 6—12: Testing Java Applet Application 2 in Firefox, showing all nodes.**

Test succeeded. As with the Internet Explorer test the state nodes are in the wrong order.

## 6.3 TESTING THE SPEED OF EXECUTION

Application 3 increments an integer from 1 to 500, and places the square root of each of the increments into a text box.

### 6.3.1.1 STRESS TEST AJAX APPLICATION 3 IN INTERNET EXPLORER

Internet Explorer	Test 1	Test 2	Test 3	Test 4	Average
<b>Time 1*</b>	117880637586 2.00	117880644639 9.00	117880673769 3.00	117880685662 8.00	
<b>Time 2*</b>	117880638045 1.00	117880645073 7.00	117880674196 9.00	117880686099 1.00	
<b>Difference (ms)</b>	4589.00	4338.00	4276.00	4363.00	4391.50

Average time in seconds

$$\frac{4391.50}{1000} = 4.39$$

### 6.3.1.2 STRESS TEST AJAX APPLICATION 3 IN MOZILLA FIREFOX

Mozilla Firefox	Test 1	Test 2	Test 3	Test 4	Average
<b>Time 1*</b>	1178803499827.00	1178803882020.00	1178805720234.00	1178806005796.00	
<b>Time 2*</b>	1178803708065.00	1178804104965.00	1178805978318.00	1178806239212.00	
<b>Difference</b>	208238.00	222945.00	258084.00	233416.00	230670.75

Average time in seconds

$$\frac{230670.75}{1000} = 230.7$$

Difference in average time between Mozilla Firefox and Internet Explorer: Mozilla Firefox takes 226.31 seconds longer than Internet Explorer.

### 6.3.1.3 STRESS TEST JAVA APPLICATION 3 IN INTERNET EXPLORER

Internet Explorer	Test 1	Test 2	Test 3	Test 4	Average
Time 1*	1178802080068.00	1178802276117.00	1178802332028.00	1178802373964.00	
Time 2*	1178802085722.00	1178802281026.00	1178802334019.00	1178802377485.00	
Difference	5654.00	4909.00	1991.00	3521.00	4020

average time in seconds:  $\frac{4020}{1000} = 4.02$

### 6.3.1.4 STRESS TEST JAVA APPLICATION 3 IN MOZILLA FIREFOX

Mozilla Firefox	Test 1	Test 2	Test 3	Test 4	Average
Time 1*	1178802814681.00	1178803013567.00	1178803088828.00	1178803281026.00	
Time 2*	1178802819341.00	1178803017718.00	1178803092819.00	1178803283755.00	3.88
Difference	4660.00	4151.00	3991.00	2729.00	3880

Average time in seconds =  $\frac{3880}{1000} = 3.88$

\* These times are the number of milliseconds since January 1, 1970, 00:00:00 GMT.

## 6.4 MODEM DOWNLOAD TIMES

Application 1 was downloaded four times to a PC running Windows XP and Mozilla Firefox connected to the Internet using a 56k dial-up modem.

### 6.4.1.1 RESULTS

---

Java Applet Test No.	Time Measured (minutes, seconds, milliseconds)
1	1:35.580
2	0:52.054
3	00:43.685
4	00:36:01

Average 56.85 seconds.

Ajax Test No.	Time Measured (milliseconds)
1	797
2	766
3	813
4	7078

Average: 2.36 seconds.

## 6.5 CODE COMPLEXITY

The code complexity is measured for Application 2, the cyclomatic complexity of the entire application and the number of programming steps required to use the TreeView control.

A programming step is defined as a method call that is required to make the TreeView work, this includes the code needed to populate the TreeView and the code needed to position it on the screen. Other code required to access the DOM will not be measured.

### 6.5.1.1 JAVA APPLET TREE COMPLEXITY

Application's cyclomatic complexity is: **4**

Extract from the code that builds the Tree, with programming steps numbered:

```

1.      try
2.      {
3.          NodeList nodeStates = doc.getElementsByTagName("states");
4.          NodeList nodeStates2 = nodeStates.item(0).getChildNodes();
5.          root = new DefaultMutableTreeNode("States");           1
6.          DefaultMutableTreeNode branches = null;               2
7.          for (int s = 0; s < nodeStates2.getLength(); s++)
8.          {
9.
10.             if(nodeStates2.item(s).getNodeName() == " ")
11.             {
12.                 Node nodeStateMain = nodeStates2.item(s);
13.
14.
15.                 NodeList nodeNorth =
16.                 doc.getElementsByTagName(nodeStateMain.getNodeName());
17.
18.                 NodeList statesList = nodeNorth.item(0).getChildNodes()
19.                 ;
20.                 DefaultMutableTreeNode nodeTop = new
21.                 DefaultMutableTreeNode(nodeStateMain.getNodeName());           3
22.                 root.add(nodeTop);                                           4
23.                 for (int a = 0; a < statesList.getLength(); a++)
24.                 {
25.                     if(statesList.item(a).getNodeName() == " ")
26.                     {
27.                         DefaultMutableTreeNode node = new
28.                         DefaultMutableTreeNode(statesList.item(a).getChildNodes().item(0).getNodeName());
29.                         nodeTop.insert(node, 0) ; }}}           5

```

The code above builds a data model which is sent to the JTree.

The following code creates a JScrollPane object, which is needed to allow the JTree to scroll if the nodes inside it fill up too much space.

```
jScrollPane1 = new javax.swing.JScrollPane();      6
tree = new javax.swing.JTree(treebuild());        7
jScrollPane1.setViewportView(tree);                  8
getContentPane().add(jScrollPane1);              9
```

In all, 9 steps are required in total to populate and draw the TreeView. It should be noted that getContentPane() is a method inherited from the JApplet class.

### 6.5.1.2 AJAX TREE COMPLEXITY

---

The cyclomatic complexity of the application is 7

```
1. var tree; 1
2. var xmlDoc = xmlHttp.responseXML;
3. var mNode = xmlDoc.getElementsByTagName("states")[0];
4. tree = new YAHOO.widget.TreeView("treeDiv1"); 2
5. var root = tree.getRoot(); 3
6. for (i=0;i<=(mNode.childNodes.length - 2);i++)
7.     {
8.     if(mNode.childNodes[i].nodeType == 1) {
9.     var tagNm = mNode.childNodes[i].nodeName;
10.    var northStates = mNode.getElementsByTagName(tagNm)[0];
11.    var bStates = northStates.getElementsByTagName("state");
12.    var tmpNode = new YAHOO.widget.TextNode(tagNm, root, false); 4
13.    for (j=0;j<=(bStates.length - 1);j++) {
14.    var tmpNode2 = new YAHOO.widget.TextNode("-----" +
        bStates[j].childNodes[0].nodeValue, 5 tmpNode, false);
15.    } } }
16.     tree.draw(); } 6
```

Step 2 creates a new `TreeView`, and sends the parameter “treeDiv1”. This is the name of the `DIV` tag which is to contain the `TreeView` component. A `DIV` tag is a `HTML` container, and can be positioned using `Cascading Style Sheets (CSS)`.

```
1. <div id="treeDiv1"></div> 7
```

## 7 ANALYSIS OF RESULTS

This section will provide a summary of the results with some analysis. It is organised into a section for each experiment.

## 7.1 BROWSER STATISTICS

### 7.2 AVERAGES

<u>Web Browser</u>	<u>Usage</u>
<b>Microsoft Internet Explorer 6</b>	<b>46.43%</b>
<b>Mozilla Firefox</b>	26.97%
<b>Microsoft Internet Explorer 7</b>	19.40%
<b>Opera</b>	4.90%
<b>Apple Safari</b>	2.35%
<b>Microsoft Internet Explorer Combined</b>	65.83%
<b>Microsoft Internet Explorer and Mozilla Firefox</b>	92.8%
<b>Total:</b>	99.27%
<b>Other web browsers</b>	0.73%

Java is supported by 94% of web browsers.

#### 7.2.1 CONCLUSION

The first question I asked myself was “are these statistics reliable?”

The “TheCounter.com” statistics would appear to be the most reliable; this is because the data it provides is based on the web server log files of many the different web sites it provides hit counters for. Upsdell.com generates its data from visits to 6 different domains and then averages them. The data provided by W3Schools is gathered from their own access logs which could explain why their Mozilla Firefox usage is so high.

For Windows users (that is, the vast majority of users) Mozilla Firefox, unlike Internet Explorer, requires the user to go and download it - they need to make an active decision not to use Internet Explorer. Since W3Schools is a web site about web development, the type of visitor that might visit W3Schools is likely to be someone with a reasonable degree of knowledge about alternative web browsers, and the self-

confidence to install one. The web design periodical “.net” quotes the W3School figure in their most recent publication (Oliver, 2007) a sign that the W3Schools statistics are taken seriously by the industry. The statistics do not differentiate between older versions of Mozilla Firefox and Opera. The latest version of opera supports Ajax; however it can not be determined from these statistics what version is being used. In answer to my question, are the statistics reliable? I do not believe these statistics are representative of the population as a whole..It would be wrong to say “26% of web users are using Mozilla Firefox”. The averages I have computed may over estimate Mozilla Firefox usage because of the factors mentioned above, but from a developer’s perspective – which ever source you look at, Mozilla Firefox still has a large share of the market. The Java statistics show that the overwhelming majority of users have Java installed and configured with their web browser. I would have liked to have found more than one source to back this up. I would also have liked to have discovered what versions are in use, as this information would allow developers to make an informed decision as to which version to develop for, with most currently choosing Java 1.4 they are unable to use the some new functionality that exists in later versions. I did email Sun Microsystems to see if they had any data that could assist,

My general conclusion is that Microsoft Internet Explorer and Mozilla Firefox are the prominent web browsers, and when developing applications, developers would be wise to test in both. All of the web browsers listed in the table above support Ajax. Other web browsers that may or may not support Ajax are not used much at all. Java Applets are supported by the majority of web browsers, but Ajax has the greater reach.

## 7.3 CODE PORTABILITY - ANALYSIS

The Java Applets 1 and 2 both ran without any problems in Internet Explorer and Mozilla Firefox.

The Ajax application 1 ran in both browsers as expected. Ajax Application 2 ran in Mozilla Firefox as expected, but did not run in Internet Explorer as it should, with the last node not displaying. Further investigation found the cause of this problem was a loop's exit criteria.

```
2. for (i=0;i<=(mNode.childNodes.length - 2);i++)
```

The code was tested in Mozilla Firefox and it the assumption was made that it would also work in Internet Explorer. By changing **-2** to **-1** the code now works in Internet Explorer<sup>12</sup>, but why is this?

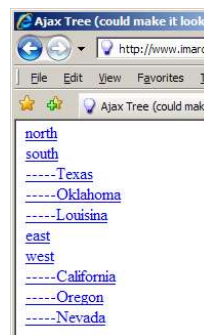


Figure 7—1: Fixed: Application 2 in IE

The value of **mNode.childNodes.length** should contain the number of child nodes in **mNode**. In this script, **mNode** is a list of all the child nodes of the node with the name “states”. There should be 4 such nodes.

---

<sup>12</sup> The screenshots here show Internet Explorer 7. These problems occurred in both Internet Explorer 6 and 7.

However when an alert statement that prints the value of **mNode.childNodes.length** is inserted before the loop, the results vary in both the browsers.

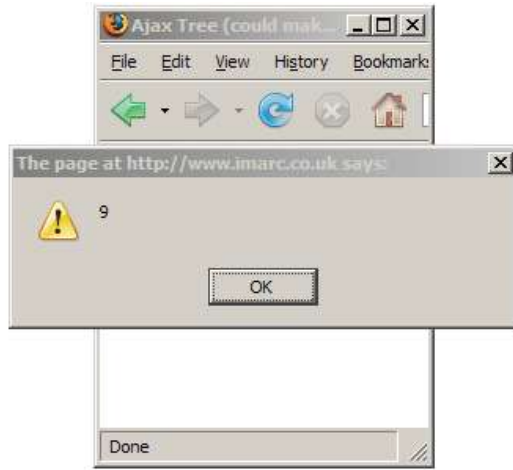


Figure 7—2: Firefox showing the value of mNode.childNodes.length

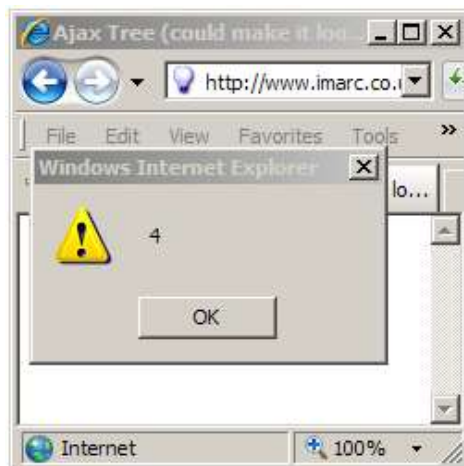
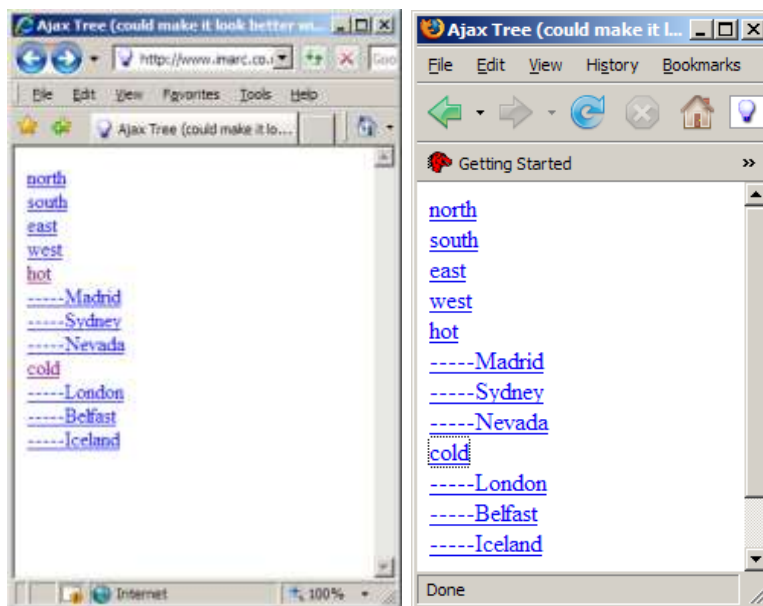


Figure 7—3: IE showing the value of mNode.childNodes.length

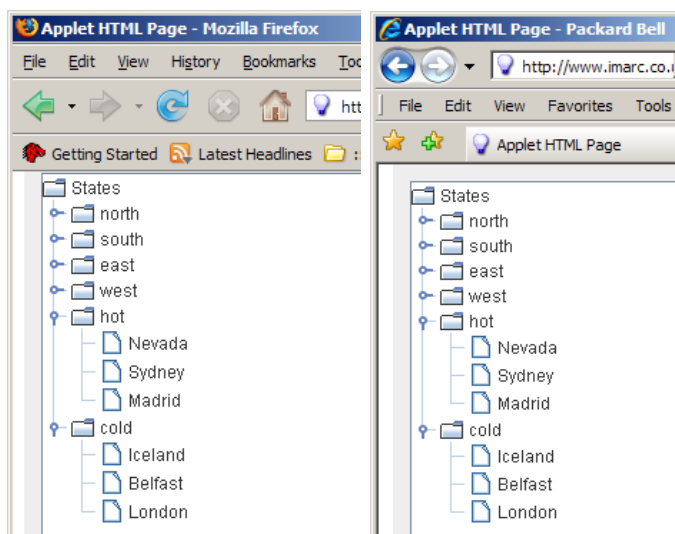
### 7.3.1 EXTRA NODES IN MOZILLA FIREFOX AND INTERNET EXPLORER:

With the code now working in both browsers, what would happen if the XML document contained more than 4 nodes? To test this, 2 more were added, called “hot” and “cold”.

The application displays all nodes, as expected. Removing the -1 altogether and looping for value of **mNode.childNodes.length** throws an error and no tree appears in both browsers.



To ensure the Java Applet also worked with the new XML file, it was also tested again in Internet Explorer and Mozilla Firefox.



### 7.3.2 CONCLUSION

Are there any weaknesses in my experiments? The experiment did only test a small amount of code; it did not test different web browser's implementation of JavaScript when using JavaScript to access the method inside a Java Applet. Many Java Applets require this functionality, and it is an area that needs to be tested. I would also have liked to have tested the Java Applet on a number of systems with different browsers and Java Runtime Environments, however this was not possible.

I have found that Java Applets behave the same way no matter which browser they are displayed in. JavaScript's behaviour (and therefore Ajax applications) varies significantly between the two most popular browsers, from a simple application that prints a TreeView component there I came across complications. The initial reason why the JavaScript tree application didn't work was down to a logic error on my part, using -2 to exit the loop instead of -1. However these logic errors lead me to the discovery that each browser was seeing a different number of children in the node list, a problem I was unable to get to the root cause of.

My initial research looked at the different implementations of XMLHttpRequest, however I found that many companies like Yahoo provide scripts provide pre-built scripts to overcome this, I also found that once I had written my own code that get round this problem, I didn't have to worry about it – it could easily be imported, or copied and pasted into all my scripts. So the XMLHttpRequest problem is not a big as I first thought, especially since the new version of Internet Explorer will be able to use the native XMLHttpRequest object. I am surprised at the problems other problems encountered and should time allow for it, I would like to find out why in the future.

In light of my findings, I would conclude that Java Applets are superior to Ajax applications in terms of cross browser capability. However as I have noted, more research needs to be conducted.

## 7.4 TESTING THE SPEED OF EXECUTION - ANALYSIS

### 7.4.1 AJAX APPLICATION 3

This tested the JavaScript part of Ajax and found Mozilla Firefox was on average took 226.31 seconds longer than Internet Explorer to complete the same task. The amount of time may reflect the specification on the computer, so a ratio is a more precise way of expressing the speed difference. On average Mozilla Firefox was 52 times slower than Internet Explorer.

This is a very significant finding; further research found that by removing the code that appends the result to the text box (below) the page loads instantly.

```
3. = Math.sqrt(i);
4. document.getElementById("text1").value
5. document.getElementById("text1").value + "\n " + " " +
6. o.toString();
```

It was thought that it could be the act of reading from the text box slowing down the operation, so the code was modified to use a String variable instead of a text box, in each iteration the text box's value would simply be set to equal this new String variable.

```
1. = Math.sqrt(i);
2. var total = total + "\n " + o.toString();
3. document.getElementById("text1").value = total;
```

Although no formal measurements were taken, it can be determined that this code did not make any noticeable difference, using rough methods it was discovered that the Mozilla Firefox browser took over 2 minutes 30 seconds to load the page, and for the duration of this time it effected the entire workstation by using 99% of the CPU time.

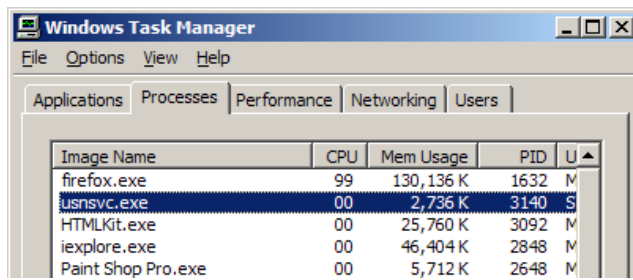


Image Name	CPU	Mem Usage	PID	U
firefox.exe	99	130,136 K	1632	M
usnsvc.exe	00	2,736 K	3140	S
HTMLKit.exe	00	25,760 K	3092	M
iexplore.exe	00	46,404 K	2848	M
Paint Shop Pro.exe	00	5,712 K	2648	M

**Figure 7—4: Firefox caused the whole computer to slow down.**

The code was changed again, this time with the String still being updated inside every iteration of the loop, but without it being displayed in the text box. The page loaded almost instantly. It can be concluded from this that the problem Mozilla Firefox has it to do with updating a text box, and not the mathematical functions.

### 7.4.2 JAVA APPLET APPLICATION 3

The difference in times between Java Applets across browsers was on average 0.14 seconds, and is not considered to be a problem. On average, the Ajax application took 113.58 seconds longer – mainly down to the Mozilla Firefox result. Most people use Internet Explorer, the difference between the average time Java Applets took to carry out the processing task and the time the Ajax application took in Internet Explorer is 0.44 seconds. So if the statistics gathered on web browser usage are correct, then the majority of people using the web will not find any noticeable difference between execution speeds of a Java applet and JavaScript. Mozilla Firefox has an issue with updating a text box.

### 7.4.3 CONCLUSION

Are there any weaknesses in my experiments? if there was more time, a wider variety of algorithms could be tested. The one I used was an attempt to combine a simple mathematical function with some visual output, a common occurrence within many desktop-class applications. The actual method for recording time by calling a JSP page I believe provided very accurate results. These results can be compared with each other – but not with external results since my test conditions were equal for all tests, the times may vary with external results, but I would hope that the ratio between results would remain. I would like to see if this were true in the future.

The execution speeds of JavaScript vary greatly between Internet Explorer and Mozilla Firefox, but this is only because Mozilla Firefox takes a very long time when performing repeated updates to a text box. I believe this is likely to be because Mozilla Firefox uses its own controls built in XML User Interface Language (XUL), not only allowing the interface to be customised with “skins” but allowing it to run work on different operating systems, such as Linux. Internet Explorer uses native Windows controls. It should be emphasised that this conclusion speculates that the cause is down to XUL, and further research would be needed to substitute this claim. The beta version of Firefox 3, codenamed “Gran Paradiso Alpha 1” has the same problem.

My recommendation is to suggest that Ajax applications keep to a strict “thin client” model, by this I mean they simply display data and do not carry out any processing within the client. However the problem with Firefox is not to do with its JavaScript’s processing of data, but with its ability to update a text box. A way to get around this would be to build the HTML for the new text box on the server, and just insert into the web page, or write the code so that a loop exit criteria cannot ever exceed a high number such as 100, if it does then an error would be thrown and the loop exited. This would avoid the user’s web browser becoming unresponsive for a long time, but may result in incomplete data being displayed. Further research is needed to understand which, if any of these suggestions would be more efficient.

I conclude that Ajax applications are better suited to a thin client model, while Java Applets can work in both thick client and thin client models.

## 7.5 MODEM DOWNLOAD TIMES - ANALYSIS

The Java Applet version of Application 1 took on average 56.85 seconds to download.

The Ajax version of Application took on average 2.36 seconds. This means that the Java Applet took roughly 24 times longer to download than the Ajax application.

Java Applets cannot access libraries stored on the local machine, even user interface libraries and so they must be distributed within the Applet's JAR file. Without compression (which JAR files support) including the org.jdesktop.layout package in the Applet added 192Kb to the total size of the JAR, which was a total of 219Kb, however it was only 144KB when compressed, but without the user interface package it was only 80Kb compressed. The Ajax page on the other hand was 2Kb.

### 7.5.1 CONCLUSION

As discovered, many users still use a modem – and so the time it takes for the page to load on a modem is relevant. As the findings show, Java Applets take a lot longer than an Ajax page that provides the same functionality. This is a very simple application, the differences in file size and therefore download time would be considerable for any application with a lot of functionality. The Java Runtime Environment does do a good job of caching the Applet, so the user won't have to download the Applet on each page view; however I speculate that waiting nearly a minute will put most users off. I conclude that Ajax is better suited to web applications that require a fast download time.

## 7.6 CODE COMPLEXITY - ANALYSIS

The Java Applet's JTree component took more steps to implement than the Yahoo TreeView component. Only two extra steps were required, however in the context of a web application – it took longer for me to learn the new skills required to use Swing. With the Ajax application, it was just a case of placing a DIV tag – which could be positioned anywhere with Cascading Style Sheets. A Swing component such as JTree requires many more steps than the Yahoo component TreeView if it is to be anywhere other than the default position of the top left corner of the screen.

### 7.6.1 CONCLUSION

I would have liked to have spent more time with different components, and looked further into how easy it is to make the components appear on the screen relative to other components. From what I discovered while learning about the two Tree components, the Swing JTree would have been the most complicated – but this is speculation and would require further research for it to be substantiated. I conclude that the Ajax tree required less steps to implement, and would require less steps to position on the screen.

## 8 OVERALL CONCLUSION

The results analysis will allow me to conclude which of the two architectures, Java Applets or Ajax is better suited to developing applications that meet specified quality criteria.

### 8.1.1.1 PORTABILITY

---

Java Applets came out on top. Code written in a Java Applet worked across different web browsers, with very little difference in performance and absolutely no difference in functionality. JavaScript did suffer from different implementations between Firefox and Internet Explorer, and did not perform well in the stress test in Firefox. My recommendation would be for all web browsers to use the same JavaScript interpreter. For the most part, all of the main web browsers use the same JRE plug-in, and the same Flash plug-in. For JavaScript to be different across browsers works to its disadvantage.

### 8.1.1.2 SCALABILITY

---

Java Applets were more suited to the “thick client” style., whereas Ajax was suited to “thin client”. Ajax applications are not suited to carrying out large processing tasks in the client, primarily because of JavaScript’s lack of support for multithreading, however Ajax applications can be written so that processing is carried out on the server, so I don’t see this as a major disadvantage – it simply requires a different approach to the application design that many developers of desktop-class applications are used to. It does seem wasteful that with current desktop computers becoming more powerful by the day, the industry should shift towards a web-based thin-client model. The debate over “thin vs. thick” client is beyond the scope of this project.

### 8.1.1.3 RELIABILITY

---

The Java Applets had a greater complexity, and needed more steps to create a TreeView component than the Ajax equivalent. However the differences in JavaScript implementations across web browsers mean a piece of JavaScript code is likely to be less reliable, even if it is less complex. There is a solution to this, and that is extra testing. A Java Applet that has for example, three paths of execution would need to be tested at least three times, ideally more. A piece of JavaScript with three paths of

execution should be tested a minimum of 6 times, each path should be tested in the two popular browsers. So to conclude, both Java Applets and Ajax applications can achieve the same degree of reliability, however JavaScript (and therefore Ajax) applications will require more testing.

#### 8.1.1.4 USABILITY

---

It was easier to create the user interface component in JavaScript than it was in Java and Swing, there was no need to add events for example, as they are built into the web browser. Many web developers will already know how to use CSS, but not how to use Swing – therefore making the component appear in the right place will take a lot less effort in JavaScript. On the other hand, the Swing JTree did look a lot more professional than the Yahoo TreeView, which could have been made to look more like the TreeView users expect to see with some CSS. Swing can also be developed by using a drag and drop IDE (integrated development environment) such as NetBeans; however knowledge of how the code works is still required. Overall I think the combination of JavaScript, CSS and Yahoo's readymade component is easier to implement than the Swing alternative. Ajax was also notably quicker to download when using a modem. As discovered in the research phase of this project, many users still use modems and so in terms of Usability – Ajax is the preferred.

As a final conclusion, if the guidelines I have laid out here are followed, that is Ajax applications follow a thin client model and more time is spent testing them - then Ajax is the preferred technology because of its greater reach and faster download times. It seems easier to “fix” the problems with Ajax than to fix the problems with Java Applets, increased complexity, large downloads and smaller install base. In the long-term I believe JavaScript needs multithreading capability, and a standard interpreter that all browsers use. With Sun announcing in recent weeks the launch of a new technology called JavaFX<sup>13</sup>, an alternative to Ajax and Flash – JavaScript needs to adapt to compete.

---

<sup>13</sup> See <http://java.sun.com/javafx/>

## 9 BIBLIOGRAPHY

Asleson, R., & Schutta, N. T. (2005). *Foundations of AJAX*. New York: APress.

Ballard, P. (2006). *Sams Teach Yourself AJAX in 10 Minutes*. Indianapolis: Sams Publishing.

Beck, K. (2000). *Extreme Programming Explained*. New Jersey: Addison-Wesley.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, 61-72.

Burdy, L., Requet, A., & Lanet, J.-L. (2003). Approach, Java Applet Correctness: a Developer-Oriented. *FME 2003: Formal Methods, LNCS 2805*, (pp. 422-439). Springer-Verlag, Pisa.

Cadenhead, R., & Lemay, L. (2004). *Sams Teach Yourself Java 2 in 21 Days*. Indianapolis: Sams Publishing.

Crockford, D. (2007, April 21st). *Recommendations for Modifications to the ECMAScript Language Specification*. Retrieved May 4th, 2007, from [crockford.com](http://www.crockford.com/javascript/recommend.html): <http://www.crockford.com/javascript/recommend.html>

Fried, I. (2007, May 7th). *Too Hotmail to Handle?* Retrieved May 2007, 8th, from CNET News.com: [http://news.com.com/2009-1038\\_3-6181300.html](http://news.com.com/2009-1038_3-6181300.html)

Garrett, J. J. (2005, February 18th). *Ajax: A New Approach to Web Applications*. Retrieved May 14th, 2007, from Adaptive Path: <http://www.adaptivepath.com/publications/essays/archives/000385.php>

GMM. (2007, May 1st). *Glossary of Web Terms*. Retrieved May 20th, 2007, from The Global Manufacturing Marketplace: [http://www.mfgquote.com/resources\\_web\\_terms\\_S.cfm](http://www.mfgquote.com/resources_web_terms_S.cfm)

Guitart, J., Beltran, V., Carrera, D., & Ayguadé, J. T. (2005). Characterizing Secure Dynamic Web Applications Scalability. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 1, 108.1.

- Hammer, T., Rosenberg, D. L., & Shaw, J. (1998). *Software Metrics and Reliability. 9th International Symposium, "BEST PAPER" Award.* Germany.
- Hillier, S. (1996). *Inside Microsoft Visual Basic: Scripting Edition.* Redmond: Microsoft Press.
- McLellan, D. (2005, February 9th). *Very Dynamic Web Interfaces.* Retrieved March 31st, 2007, from O'Reilly XML.com: <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>
- Nichols, E. A., & Peterson, G. (2007). A Metrics Framework to Drive Application Security Improvement . *IEEE Security and Privacy* , 5 (2), 88-91.
- Nielsen, J. (2003, August 25th). *Introduction to Usability.* Retrieved March 31st, 2007, from Use IT.
- Offutt, J. (2002). Quality attributes of Web software applications. *Software, IEEE* , 19 (2), 25-32.
- Oliver, D. (2007, June). IT Figures. *.Net* (163), p. 16.
- Paulson, L. D. (2005). Building Rich Web Applications with Ajax. *Computer* , 38 (10), 14-17.
- Potts, M. (1998). *Report Structure.* Retrieved May 4th, 2007, from <http://acet.rdg.ac.uk/~mab/Education/Past-students/postgrad/Melanie-Potts/>
- Pressman, R. S. (2005). *Software Engineering: A Practitioner's Approach.* New York: McGraw-Hill.
- Primode. (2007). *Information Security Glossary.* Retrieved April 26th, 2007, from Primode IT Security Hub: <http://www.primode.com/glossary.html>
- SEI. (2007). *SEI Open Systems Glossary.*
- Sommerville, I. (2004). *Software Engineering.* London: Pearson Addison Wesley.

Sun Microsystems. (2004, 09 12). *Constructing a User-Friendly JTree from a DOM*.

Retrieved April April, 2007, from The J2EE Tutorial:

<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPDOM6.html>

Sun Microsystems Inc. (2004, Jan 31). *Lesson: Applets*. Retrieved 1st May, 2007,

from The Java(tm) Tutorials:

<http://java.sun.com/docs/books/tutorial/deployment/applet/index.html>

Sun Microsystems. (2007, Feb 16rd). *Sun XML Web Site*. Retrieved March 28th,

2007, from Sun.com: <http://java.sun.com/xml/>

# 10 APPENDICES

## 10.1 XML DOCUMENT USED IN TESTS

```
<?xml version="1.0" encoding="UTF-8" ?>
= <states>
= <north>
  <state>Minnesota</state>
  <state>Iowa</state>
  <state>North Dekota</state>
</north>
= <south>
  <state>Texas</state>
  <state>Oklahoma</state>
  <state>Louisina</state>
</south>
= <east>
  <state>New York</state>
  <state>North Caeolina</state>
  <state>Massachusetts</state>
</east>
= <west>
  <state>California</state>
  <state>Oregon</state>
  <state>Nevada</state>
</west>
= <hot>
  <state>Madrid</state>
  <state>Sydney</state>
  <state>Nevada</state>
</hot>
= <cold>
  <state>London</state>
  <state>Belfast</state>
  <state>Iceland</state>
</cold>
</states>
```

## 10.2 AJAX APPLICATION 1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test: Parse XML DOC </title>

<script type="text/javascript">
function createXMLHttpRequest() {
if (window.ActiveXObject) {
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest) {
xmlHttp = new XMLHttpRequest();
}
}

var xmlHttp;
var requestType = "";

function startRequest(requested) {
requestType = requested;
createXMLHttpRequest();
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open("GET", "parseXML.xml", true);
xmlHttp.send(null);
}

function handleStateChange() {

if(xmlHttp.readyState == 4) {
if(xmlHttp.status == 200) {
// alert("ready");
if(requestType == "north") {

listNorthStates();
}
else if(requestType == "all") {
listAllStates();
}
}
}
}

function listNorthStates() {
var xmlDoc = xmlHttp.responseXML;
var northNode = xmlDoc.getElementsByTagName("north")[0];
var out = "Northern States";
var northStates = northNode.getElementsByTagName("state");

outputList("Northern States:", northStates);
}

function listAllStates()
{
var xmlDoc = xmlHttp.responseXML;
var allStates = xmlDoc.getElementsByTagName("state");
outputList("All States: ", allStates);
}

function outputList(title, states)
{
var out = title;
var currentState = null;

for(var i = 0; i < states.length; i++)
{
currentState = states[i];
out = out + "<br/> " + currentState.childNodes[0].nodeValue;
}
```

```
}
// alert(out);
document.getElementById('main').innerHTML = out;
}

</script>
</head>

<body>
<form action="#">
<input type="button" value="View ALL" onClick="startRequest('all');"/>

<input type="button" value="View North" onClick="startRequest('north');"/>
</form>

<br/> <hr/>
<div id="main" class="displaybox"></div>

</body>
</html>
```

## 10.3 AJAX APPLICATION 2

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Ajax Tree (could make it look better with CSS) </title>
<!-- Dependency source files -->
  <script src = "http://yui.yahooapis.com/2.2.2/build/yahoo/yahoo-min.js"
></script>
  <script src = "http://yui.yahooapis.com/2.2.2/build/event/event-min.js"
></script>

  <!-- TreeView source file -->
  <script src = "http://yui.yahooapis.com/2.2.2/build/treeview/treeview-min.js"
></script>

<script type="text/javascript">
function createXMLHttpRequest() {
if (window.ActiveXObject) {
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest) {
xmlHttp = new XMLHttpRequest();
}
}

var xmlHttp;
var requestType = "";

function startRequest() {
// requestType = requested;
createXMLHttpRequest();
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open("GET", "parseXML.xml", true);
xmlHttp.send(null);
}

function handleStateChange() {

if(xmlHttp.readyState == 4) {
if(xmlHttp.status == 200) {
// alert("ready");

listAllStates();

}
}
}

function listAllStates()
{
var tree;
var xmlDoc = xmlHttp.responseXML;
var mNode = xmlDoc.getElementsByTagName("states")[0];
var out = "Northern States";
tree = new YAHOO.widget.TreeView("treeDiv1");
var root = tree.getRoot();
// var northStates = mNode.getElementsByTagName("north")[0];

for (i=0;i<=(mNode.childNodes.length - 2);i++)
{
if(mNode.childNodes[i].nodeType == 1)
{
var tagNm = mNode.childNodes[i].nodeName;
// alert(tagNm);
var northStates =
mNode.getElementsByTagName(tagNm)[0];

```

```
var bStates =
northStates.getElementsByTagName("state");
// outputList("Northern States:", bStates);
var tmpNode = new YAHOO.widget.TextNode(tagNm,
root, false);

;
for (j=0;j<=(bStates.length - 1);j++)
{
//alert(bStates.length);

//alert(bStates[j].childNodes[0].nodeValue);
var tmpNode2 = new
YAHOO.widget.TextNode("-----" + bStates[j].childNodes[0].nodeValue, tmpNode,
false);
}
}

tree.draw();
}

</script>

</head>

<body onLoad="startRequest()">
<div id="treeDiv1"></div>

<br/>
<br/>

</body>
</html>
```

## 10.4 AJAX APPLICATION 3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Js TEst</title>

    <script type="text/javascript">

function createXMLHttpRequest() {
if (window.ActiveXObject) {
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest) {
xmlHttp = new XMLHttpRequest();
}
}

var xmlHttp;
var requestType = "";

function startRequest() {
var url = "index.jsp?" + new Date().getTime();

    // requestType = requested;
    createXMLHttpRequest();
    xmlHttp.onreadystatechange = handleStateChange;
    xmlHttp.open("GET", url, true);
    xmlHttp.send(null);
}

function handleStateChange() {
if(xmlHttp.readyState == 4) {
if(xmlHttp.status == 200) {
// alert("ready");

}
}
}

function calcsqr()
{

    startRequest();

    for (i = 1; i <= 500; i++)
    {

        o = Math.sqrt(i);
        // alert(i)
        document.getElementById("text1").value =
document.getElementById("text1").value + "\n " + " " + o.toString();

    }

    startRequest();
}
}
</script>

```

```
    }  
    </script>  
</head>  
<body onload="calcsqr();">  
  
    <form name="main">  
        <input type="button" id="num" name="num" value="go" onclick="calcsqr();" />  
<br/>  
        <textarea id="text1" name="text1" rows="20" cols="50">  
        </textarea>  
        <div id="output"> SQR </div>  
  
    </body>  
</html>
```

## 10.5 JAVA APPLET APPLICATION 1

```
import java.io.File;
import java.io.IOException;
import java.net.URLConnection;
import java.net.URL;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

/*
 * MarcApplet.java
 *
 * Created on 03 April 2007, 11:46
 */

/**
 *
 * @author Marc
 */
public class MarcApplet_1 extends java.applet.Applet
{
    static Document document;
    private String argv[] ;
    /** Initializes the applet MarcApplet */
    public void init()
    {
        try
        {
            java.awt.EventQueue.invokeLater(new Runnable()
            {
                public void run()
                {
                    initComponents();
                }
            });
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }

    /** This method is called from within the init() method to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents()
    {
        showStates = new java.awt.Button();
        button1 = new java.awt.Button();
        label1 = new java.awt.Label();
        textArea1 = new java.awt.TextArea();

        setBackground(new java.awt.Color(255, 255, 255));
        showStates.setLabel("Show All States");
        showStates.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                showStatesActionPerformed(evt);
            }
        });

        button1.setBackground(new java.awt.Color(192, 192, 192));
        button1.setLabel("Show Northern States");
    }
}
```

```

        button1.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                button1ActionPerformed(evt);
            }
        });

        label1.setFont(new java.awt.Font("Tahoma", 1, 24));
        label1.setText("Java Applet Example");

        textArea1.setBackground(new java.awt.Color(255, 255, 255));
        textArea1.setEditable(false);
        textArea1.setForeground(new java.awt.Color(0, 0, 153));

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(layout.createSequentialGroup()
                        .addContainerGap()
                        .add(textArea1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 493,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(layout.createSequentialGroup()
                        .addContainerGap()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING,
false)
                    .add(org.jdesktop.layout.GroupLayout.LEADING,
layout.createSequentialGroup()
                        .add(showStates,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .add(button1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(org.jdesktop.layout.GroupLayout.LEADING,
label1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(22, Short.MAX VALUE)
            );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                        .add(layout.createSequentialGroup()
                            .add(label1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
                                .add(showStates,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                .add(button1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                                    .add(58, 58, 58)
                                    .add(textArea1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
147, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                    .addContainerGap(82, Short.MAX VALUE)
                                );
                    );
        } // </editor-fold>

```

```

        private void button1ActionPerformed(java.awt.event.ActionEvent evt)
        {
// TODO add your handling code here:
            textAreal.setText("Northern States:" + getNorth());
        }

        private void showStatesActionPerformed(java.awt.event.ActionEvent evt)
        {
// TODO add your handling code here:
            textAreal.setText("All States:" + getAll() );
        }

// Variables declaration - do not modify
private java.awt.Button button1;
private java.awt.Label label1;
private java.awt.Button showStates;
private java.awt.TextArea textAreal;
// End of variables declaration

private static String getAll()
{
    String out = "";
    try
    {
        URL url = new URL("http://www.imarc.co.uk/project/parseXML.xml");
        URLConnection urlconn = url.openConnection();
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(urlconn.getInputStream());
        doc.getDocumentElement().normalize();
        NodeList nodeAll = doc.getElementsByTagName("state");

        for (int s = 0; s < nodeAll.getLength(); s++)
        {
            // System.out.println("All: " + s +
nodeAll.item(s).getNodeValue());
            Node myNode= nodeAll.item(s);
            Element eNode = (Element)myNode;
            NodeList eeNode = eNode.getChildNodes();
            // System.out.println("All: " + eeNode.item(0).getNodeValue());
            out = out + "\n" + eeNode.item(0).getNodeValue();
        }

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

    return out;
}

private static String getNorth()
{
    String out = "";
    URL url;
    try
    {
        url = new URL("http://www.imarc.co.uk/project/parseXML.xml");

        URLConnection urlconn = url.openConnection();
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(urlconn.getInputStream());
        doc.getDocumentElement().normalize();
        NodeList nodeNorth = doc.getElementsByTagName("north");

        NodeList statesList = nodeNorth.item(0).getChildNodes() ;
        for (int s = 0; s < statesList.getLength(); s++)

```

```
        {
            if (statesList.item(s).getNodeTypes() == 1)
            {
                out = out + "\n" +
statesList.item(s).getChildNodes().item(0).getNodeValue();
            }
        }

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

        return out;
    }
}
```

## 10.6 JAVA APPLICATION 2

```
import java.net.MalformedURLException;
import java.util.ArrayList;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import java.io.File;
import java.io.IOException;
import java.net.URLConnection;
import java.net.URL;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

/*
 * mainApplet.java
 *
 * Created on 11 May 2007, 13:12
 */

/**
 *
 * @author Marc
 */
public class mainApplet extends javax.swing.JApplet
{
    /** Initializes the applet mainApplet */
    public void init()
    {
        try
        {
            java.awt.EventQueue.invokeLater(new Runnable()
            {
                public void run()
                {
                    initComponents();
                }
            });
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }

    private DefaultMutableTreeNode treebuild()
    {
        DefaultMutableTreeNode root = null;
        try
        {
            String out = "";
            URL url = new URL("http://www.imarc.co.uk/project/parseXML.xml");
            URLConnection urlconn = url.openConnection();
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(urlconn.getInputStream());
            doc.getDocumentElement().normalize();
            NodeList nodeStates = doc.getElementsByTagName("states");
            NodeList nodeStates2 = nodeStates.item(0).getChildNodes();
            // ArrayList branches = new ArrayList();
            root = new DefaultMutableTreeNode("States");
            DefaultMutableTreeNode branches = null;
            for (int s = 0; s < nodeStates2.getLength(); s++)
            {
                if (nodeStates2.item(s).getNodeName().equals("state"))
                {
                    Node nodeStateMain = nodeStates2.item(s);
                    out = out + "\n " + nodeStateMain.getNodeName();
                }
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

```

        NodeList nodeNorth =
doc.getElementsByTagName(nodeStateMain.getNodeName());

        NodeList statesList = nodeNorth.item(0).getChildNodes() ;
        DefaultMutableTreeNode nodeTop = new
DefaultMutableTreeNode(nodeStateMain.getNodeName());
        root.add(nodeTop);
        for (int a = 0; a < statesList.getLength(); a++)
        {
            if(statesList.item(a).getNodeTypes() == 1)
            {
                out = out + "\n" +
statesList.item(a).getChildNodes().item(0).getNodeValue();

                DefaultMutableTreeNode node = new
DefaultMutableTreeNode(statesList.item(a).getChildNodes().item(0).getNodeValue(
));
                nodeTop.insert(node, 0) ;
            }
        }
    }
}

} catch (MalformedURLException ex)
{
    ex.printStackTrace();
} catch (IOException ex)
{
    ex.printStackTrace();
} catch (SAXException ex)
{
    ex.printStackTrace();
} catch (ParserConfigurationException ex)
{
    ex.printStackTrace();
}

return root;
}

private void initComponents()
{
    jScrollPane1 = new javax.swing.JScrollPane();

    tree = new javax.swing.JTree(treebuild());
    jButton1 = new javax.swing.JButton();

    jScrollPane1.setViewportView(tree);

    jButton1.setText("Load");
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            jButton1ActionPerformed(evt);
        }
    });
    getContentPane().add(jScrollPane1);

```

```
}  
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)  
{  
  
}  
  
// Variables declaration - do not modify  
private javax.swing.JButton jButton1;  
private javax.swing.JScrollPane jScrollPane1;  
private javax.swing.JTree tree;  
// End of variables declaration  
}
```

## 10.7 JAVA APPLICATION 3

```
import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URLConnection;
import java.net.URL;
import java.util.Date;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

/*
 * MarcApplet.java
 *
 * Created on 03 April 2007, 11:46
 */

/**
 *
 * @author Marc
 */
public class MarcApplet1 extends java.applet.Applet
{
    static Document document;
    private String argv[] ;
    /** Initializes the applet MarcApplet */
    public void init()
    {

        try
        {
            java.awt.EventQueue.invokeLaterAndWait(new Runnable()
            {
                public void run()
                {
                    initComponents();
                }
            });
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }

        try
        {
            doIt();
        } catch (MalformedURLException ex)
        {
            ex.printStackTrace();
        } catch (IOException ex)
        {
            ex.printStackTrace();
        } catch (ParserConfigurationException ex)
        {
            ex.printStackTrace();
        } catch (SAXException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

```

/** This method is called from within the init() method to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
// <editor-fold defaultstate="collapsed" desc=" Generated Code ">
private void initComponents()
{
    label1 = new java.awt.Label();
    textArea1 = new java.awt.TextArea();
    button1 = new java.awt.Button();

    setBackground(new java.awt.Color(255, 255, 255));
    label1.setFont(new java.awt.Font("Tahoma", 1, 24));
    label1.setText("Java Applet Example");

    button1.setBackground(new java.awt.Color(212, 208, 200));
    button1.setForeground(new java.awt.Color(0, 0, 0));
    button1.setLabel("Do Work!");
    button1.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            button1ActionPerformed(evt);
        }
    });

    org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(label1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
                        .add(textArea1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 330,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(button1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .addContainerGap(402, Short.MAX_VALUE))
                );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(layout.createSequentialGroup()
                    .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(7, 7, 7)
                        .add(button1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(textArea1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
440, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(24, Short.MAX_VALUE))
                    );
    }
}
// </editor-fold>

private void button1ActionPerformed(java.awt.event.ActionEvent evt)
{
    try
    {
        doIt();
    } catch (MalformedURLException ex)
    {
        ex.printStackTrace();
    }
}

```

```
        } catch (IOException ex)
        {
            ex.printStackTrace();
        } catch (ParserConfigurationException ex)
        {
            ex.printStackTrace();
        } catch (SAXException ex)
        {
            ex.printStackTrace();
        }
    }

    // Variables declaration - do not modify
    private java.awt.Button button1;
    private java.awt.Label label1;
    private java.awt.TextArea textArea1;
    // End of variables declaration

    private void doIt() throws MalformedURLException, IOException,
    ParserConfigurationException, SAXException
    {
        Date a = new Date();
        URL url = new java.net.URL("http://192.168.1.7:8080/timeLog/index.jsp?" +
a.getTime());
        URLConnection urlconn1 = url.openConnection();
        int j = urlconn1.getContentLength(); // request the JSP file.
        calcsqr();
        URLConnection urlconn2 = url.openConnection();
        int i = urlconn2.getContentLength();
        // get URL a second time
    }

    private void calcsqr()
    {
        int i;;
        for (i = 1; i <= 500; i++)
        {
            Double o= Math.sqrt(i);
            textArea1.setText(textArea1.getText() + "\n" + o.toString() + "
" + i) ;
        }
    }
}
```



## 10.8 JSP PAGE

```
<%@page import="java.util.Date"%>
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%
    Date a = new Date();
    long date = a.getTime();
    System.out.println(date);
%>

</body>
</html>
```

## 10.9 EMAIL FROM SUN

From: Cathy.Toft@Sun.COM [mailto:Cathy.Toft@Sun.COM]  
Sent: 09 May 2007 13:43  
To: Marc Wickens  
Subject: Re: JRE Installations

Hi Mark

Please find some information below.

Also find some reference points where you can find more info:

- \* [developers.sun.com](http://developers.sun.com)
- \* [java.net](http://java.net)
- \* [java.com](http://java.com)
- \* [java.sun.com/javase/downloads/index.jsp](http://java.sun.com/javase/downloads/index.jsp)

Kind regards

Java has been achieving milestones since Sun announced plans to open-source Java last year:

- 6 million Java developers worldwide, up 20%
- 5 billion Java enabled devices worldwide, up 36%
- Over 2 billion total Java enabled handsets, up 27%
- 180 operators deploying Java-based content/services
- 7 Million Java enabled set-top boxes, up 83%
- 4 Million Java-enabled Blu-ray devices
- Over 400 million Java Runtime Environment downloads

>  
>