Test-Driven Development of Ajax enabled web applications on the Java platform

by

Jørgen Andersen
Jan Roar Edvardsen

Thesis in partial fulfilment of the degree of Master in Technology in Information and Communication Technology

Agder University College
Faculty of Engineering and Science

Grimstad
Norway

May 2006

ABSTRACT

The introduction of new technologies is often based on a response to the obstacles their predecessors could not overcome. In the history of the World Wide Web, the last years has provided us with new technologies presenting new possibilities for web application development. Among these technologies we find a breed of new technologies labeled under the expression Rich Internet Applications.

Created to enhance the web with the power of traditional desktop applications the RIA technologies present the next generation of the Web, the Web 2.0. Among these technologies we find a new arrival, Ajax.

To aid developers in software development the usage of defined methodologies are guiding lights. In our thesis we have studied and introduced the Agile software methodology Test Driven Development.

In this thesis we will introduce the challenges connected to usage of TDD on Ajax enabled web development. We will also introduce the Ajax architecture, and the impact of Design Patterns to improve design.

We will also discuss Ajax and eventual standardization issues to prevent developers from compatibility and lock-in situations.

It is expected much of the next generation of web applications, keeping a close eye on the architecture and using TDD can help development and structure to applications. This can also be combined with design patterns and framework to help development even further.

PREFACE

This thesis is submitted in the partial fulfilment of the requirements for the degree Master of Science at Agder University College, faculty of Engineering and Science. The thesis has been completed at Agder University College under the supervision of associate professor Ole-Christoffer Granmo, and co-supervisor Asle Pedersen, InterMedium.

We wish to thank Ole-Christoffer Granmo and Asle Pedersen for the guidance and expertise shared with us throughout the project period.

We would also thank our families, our co-students and Agder University College.

Grimstad, May 2006

Jørgen Andersen and Jan Roar Edvardsen

CONTENTS

Abstract 2
Preface 3
Contents 4
List of figures 8
List of tables 9
Code examples 10
1 Introduction 11
  1.1 Thesis definition 11
  1.2 Delimitations 13
  1.3 Methodology 13
  1.4 Report outline 13
2 Background 15
  2.1 Web 2.0 15
  2.2 RIA (Rich Internet Applications) 16
    2.2.1 Introducing the term RIA 16
    2.2.2 A brief look at existing RIA technologies 18
      2.2.2.1 Macromedia Flash/Flex 18
      2.2.2.2 Java (Applet, Web Start) 20
      2.2.2.3 Ajax (Asynchronous JavaScript and XML) 22
  2.3 Introducing Ajax 23
    2.3.1 The Ajax architecture 23
    2.3.2 Ajax, under the hood 26
      2.3.2.1 JavaScript and DOM 27
      2.3.2.2 CSS (Cascading Style Sheets) 28
      2.3.2.3 XML and XMLHttpRequest 29
    2.3.3 The Ajax engine, frameworks and tool support 29
      2.3.3.1 Direct Web Remoting (DWR) 29
  2.4 Agile software development 30

2.4.1 Agile software development 30
2.4.2 Software testing 31
2.4.2.1 Unit testing 32
2.4.2.2 Integration testing 33
2.5 TDD, basic ingredients 34
2.5.1 TDD and design 37
2.5.1.1 Agile Draw basics 38
2.5.2 Unit testing, frameworks and mock objects 40
2.5.2.1 JUnit 41
2.5.2.2 JsUnit 42
2.5.2.3 Script.aculo.us 42
2.5.2.4 J3Unit 42
2.5.2.5 Mock objects 42
2.5.3 A decent portion of TDD knowledge 43
3 TDD and Ajax 44
3.1 Identifying the challenges 44
3.1.1 Choosing a unit testing framework for JavaScript 44
3.1.2 Testing asynchronous communication in JavaScript 49
3.1.2.1 The waitForAWhile solution 49
3.1.2.2 The mock object approach 52
3.1.3 Ajax and integration testing. 55
3.2 Summarizing the challenges 56
4 Ajax 57
4.1 Making a Ajax request 57
4.2 Processing the request and response. 58
4.3 Design Patterns 61
4.3.1 The Facade design pattern 62
4.3.2 The Model-View-Controller design pattern 66
4.3.2.1 The View 67
4.3.2.2 The Controller 67
4.3.2.3 The Model 68

4.4 Frameworks and Direct web remoting (DWR) 68
5 Web standards 70
5.1 standardizing the web 70
5.2 Why standardization 73
5.3 Standardizing Ajax 74
5.3.1 JavaScript 74
5.3.2 XML 74
5.3.3 CSS 75
5.3.4 DOM 75
5.3.5 XMLHttpRequest 75
6 Results 78
6.1 Results, TDD and Ajax challenges 78
6.1.1 The main challenges 78
6.1.2 A testing framework for JavaScript 78
6.1.3 Asynchronous request/response in the unit test phase 79
6.1.3.1 The waitForAWhile solution 79
6.1.3.2 The mock object approach 79
6.1.4 A test framework in the integration test phase 79
6.2 Results, Ajax architecture and design patterns 80
6.3 Results, Ajax and web STANDARDIZATION 80
7 Discussion and future work 81
7.1 Discussion, Ajax and TDD challenges 81
7.1.1 The main challenges 81
7.1.1.1 Selecting a test framework for JavaScript 81
7.1.1.2 The waitForAWhile solution 82
7.1.1.3 The mock object approach 82
7.1.1.4 A test framework for integration testing 83
7.2 Ajax architecture and Design patterns 83
7.3 Ajax and web standardization 84
7.4 Future work 84
8 Conclusion 86

Appendix A Glossary & abbreviations 87
Appendix B References 89

LIST OF FIGURES

Figure 1: RIA, Macromedia vision 18
Figure 2: The Flash/Flex architecture..20
Figure 3: Java Web Start/Applet architecture..21
Figure 4: Classic versus Ajax web application model...............................24
Figure 5: Classic web application model (synchronous) 25
Figure 6: Ajax web application model (asynchronous) 26
Figure 7: Ajax, the building blocks..27
Figure 8: Adaptive versus predictive methodologies 31
Figure 9: The development and test cycle 33
Figure 10: The TDD cycle 35
Figure 11: Agile Draw, basic components 39
Figure 12: Running JsUnit tests from the Eclipse platform.......................46
Figure 13 JsUnit test suite, browser screenshot......................................48
Figure 14: JsUnit test suite, Eclipse screenshot 48
Figure 15: XMLHttpRequest mock object for JavaScript 54
Figure 16: Detailed UML sequence diagram of the Ajax architecture. 61
Figure 17: Facade Pattern diagram..66
Figure 18: Diagram of how DWR can be put to use 69
Figure 19: The original WWW architecture diagram, 1990.......................71
Figure 20: Browser statistics April 2006 76

LIST OF TABLES

Table 1: The core principles of Agile Draw...38
Table 2: Points, description...39
Table 3: Connectors, description..39
Table 4: JUnit, assertion methods...41
Table 5: Mock object patterns 53
Table 6: TDD - Ajax challenges...56
Table 7: The possible values of the readyState property 59

CODE EXAMPLES

Code example 1: Initialize a XMLHttpRequest object 46

Code example 2: The testPage.html 47

Code example 3: The waitForAWhile.js script..49

Code example 4: Testing the asynchronous request 50

Code example 5: Waiting for the XMLHttpRequest response 52

Code example 6: JavaScript event handler example 57

Code example 7: The getZipData function...58

Code example 8: Java servlet example 59

Code example 9: XMLHttpRequest, callback handler 60

Code example 10: Creating a XMLHttpRequest object............................63

Code example 11: Seperation of View and the Controller........................68

1 INTRODUCTION

This thesis is based on an initial initiative from the software company InterMedium. The background was to consider a new technology Ajax, and its relation to the Java platform and TDD.

Based on the initial initiative, we have in collaboration with our supervisors defined a thesis definition presented in Section 1.1.

The main goals of this thesis are presented as questions in the background thesis, and will serve as the basis guidelines in our proposed solutions, results and discussions.

Ajax is a new technological approach that is able to create dynamic web applications with the strengths of traditional desktop applications. We will throughout this thesis introduce Ajax, its architecture, and looked into how Ajax enabled web development and TDD fits together.

The thesis will also discuss the usage of design patterns, and show how they can improve software design. It will also discuss possible standardization issues in elucidation of web standardization.

1.1 THESIS DEFINITION

The full title of this thesis is “Test-Driven development of Ajax enabled web applications on the Java platform - Challenges and solutions.” The background for this thesis is the following definition.

Traditionally, web applications have been "page oriented" with a series of request-responses between client (web browser) and server, usually resulting in whole page updates. Introduction of Rich Internet Applications (RIA) technologies such as Ajax has made it possible to create “frame oriented” web applications, updating parts of the user interface view. Test-Driven Development (TDD) is a technique based on the conformity of functional code to written tests. Ajax represents a new technological approach, and use of TDD can add structure, improve design and ensure correctness.

The main goal of this investigation is to look into TDD of Ajax enabled web applications on the Java platform. The following factors have been identified that affect the development process with TDD and Ajax which are of particular interest to this project:

- Testability: to follow the TDD paradigm the application and framework must be simple to test.
- Efficiency and quality: to efficiently develop applications of high quality proper tools are important.
- Best Practice: to avoid traps concerning new technology, adopting the right design patterns are important.
- Standardization: when dealing with new technologies standardization is important and lack of such might lead to problems with lock-in, compatibility and reuse of components.

The following questions arise:

1. What challenges does Ajax introduce to TDD?

2. Is there a need for Ajax adapted tool support to address these challenges?

3. What kind of architecture do Ajax applications have and which design-patterns are important to consider.

4. How does Ajax relate to current and future web/Java standards?

During this investigation a prototype using Ajax technology will be developed. The experience gained during this development, and literature studies will be used to answer the above questions. The prototype is to be developed with Java in Eclipse.

1.2 DELIMITATIONS

The prototype developed will not be introduced as a standalone application. To answer the questions from the thesis background we will develop only necessary code, and present this as examples throughout the report.

1.3 METHODOLOGY

To answer the questions asked in the thesis background we will go through a literature study of the involved technologies and elements identified from the literature study. We will also use our prototype development to gather experience to answer the background questions.

1.4 REPORT OUTLINE

Chapter 1

In this chapter we introduce the thesis and the background definition. We also present delimitations set, and what methodology we have used to answer the background definition.

Chapter 2

In this chapter we will introduce information gathered from our literature study. There will be information attached to Ajax, TDD and an introduction of all the elements we have identified during the thesis.

Chapter 3

In this chapter we will introduce the main challenges identified when using TDD to develop Ajax enabled web applications.

Chapter 4

In this chapter we will introduce a in-depth look at the Ajax architecture, and how usage of Design Patterns can help to improve the software design.

Chapter 5

In this chapter we investigate possible issues found when looking at Ajax and web standardization.

Chapter 6

In this Chapter we present the results achieved.

Chapter 7

In this chapter we discuss the results presented in Chapter 6, and introduce future work that can be done.

Chapter 8

In this Chapter we present the conclusion of the work we have done.

2 BACKGROUND

The technological implementation of the WWW has over the past decades revolutionized the way literature and media can be presented to the end users. Together with accessibility to affordable and high bandwidth, the amount of users has increased rapidly, and introduced a new world of possibilities for both ordinary people, companies and governments. The possibilities are only limited to creativity and technological barriers. As new technologies enter, they present both possibilities and challenges. As they grow old, new or evolved technologies is born, all driving the evolution of the web further.

In this chapter we will introduce a new breed of web technologies, Rich Internet Applications. We will also introduce the Ajax and TDD

2.1 WEB 2.0

The web has proven itself to be an excellent medium to present information on, and accessibility is probably one of the key factors. Everyone with access to an internet service provider, a computer and a web browser have access to billions of web pages. Only in Norway it is estimated that four out of ten households have broadband access, and six out of ten internet access. [1] In a statistical report by Eurostat it is estimated that one quarter of the European households and two-thirds of the enterprises has broadband internet access. [2] With such amounts of potential users with access to high-speed internet, the web community has reached a new maturity level.

The huge success of the web has turned it into an increasingly more important platform to distribute information, services and increasingly complex applications. Providing users with a web application instead of a desktop program simplifies distribution as one would only have to distribute the URL. The user would not require any special tools to use the application. They can continue to use the browser that they are already familiar with. Update becomes easier too since no installing would be required on the user client side, all that is needed is to replace files on the web server. Another tread in recent years has been the use of application service provider, ASP, applications, where the applications a company or organization would use are offered to users from servers. Often large parts of the application will be executed on the server while only the user interface is updated on the client side. ASP applications and web applications is in that aspect closely related. Connecting to servers from specialized clients is however not always with out it hassles and this is something one would avoid with web applications, pushing the drive for Web 2.0 even further.

2.2 RIA (RICH INTERNET APPLICATIONS)

And as a response, the industry has developed technologies to drive the evolution one step further. Some are now using the conception Web 2.0, the next generation of web technology. One thing is for sure, the generation shift will come, and in this process we believe a collection of new web technologies assembled in the conception RIA will be of great importance when standards are to be decided on.

2.2.1 Introducing the term RIA

Now what is RIA technology? In a report on RIA technologies, Tom Noda and Shawn Helwig describe the term like this: “Rich Internet Application capitalize on the strengths of both web and desktop applications.” [3, section 2] Approaching the possibilities of desktop applications, RIA technologies make it possible for developers to develop more advanced web applications. RIA technologies introduce new possibilities both in visual richness, and usability.

Traditional desktop applications have several advantages over web applications:

- Richer user experience (Audio, video, communications)
- No page reloading
- Support both online and offline
- Enable more complex applications (e.g. MS Outlook versus Web mail)

- More responsive and interactive

Introducing technologies that remove these limitations, RIA is the first steps towards the Web2.0.

The leading marketing intelligence bureau IDC points out in an article written by Joshua Duhl [4, page 1], that the market demands more complex web applications. They have provided a list that concludes that RIA technology:

- Provide a very viable technology capable of addressing a broad range of internet, intranet and corporate application needs without requiring wholesale replacement of existing Web application investments.
- Empower companies to create wholly new kinds of engaging, innovative user experiences and applications with features or capabilities that in most cases would be extremely difficult or impossible for a developer to create using traditional Web technologies.
- Deliver a variety of substantial business benefits including: highly qualified lead generation, increased sales, increased brand loyalty, longer stays on sites, more frequent repeat visits, reduced bandwidth costs, reduced support calls, and deepened customer relationships.
- Offer the potential for a fundamental shift in the experience of Internet applications, leading to applications that come closer to delivering on the promise of the Internet.

The IDC article was written in 2003 and show that several companies have both increased sales revenues and improved product user appeal considerably using the RIA technology Macromedia Flash compared to the use of traditional web technology.

Being the pioneer company, Macromedia presented the term RIA in 2002. Their vision was and is to combine the best of desktop software, the best of the web, and the best of communications in one term RIA. The original RIA figure is represented in Figure 1.

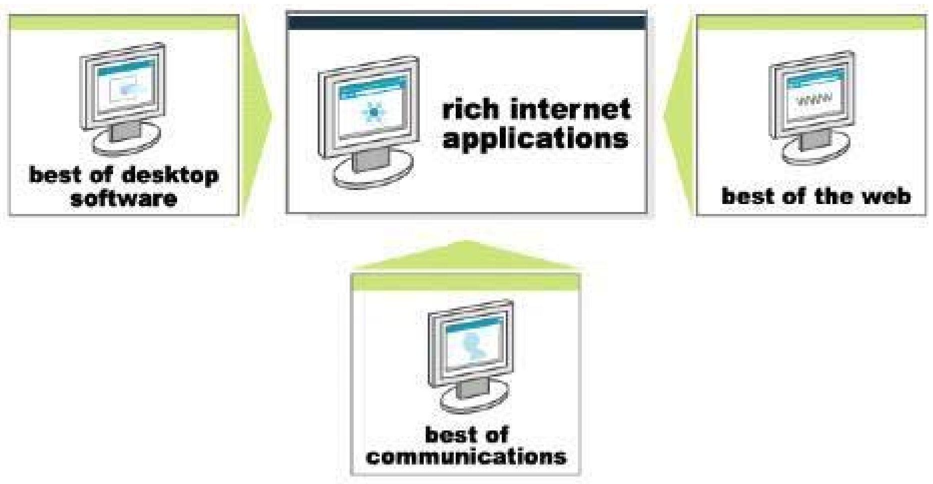


Figure 1: RIA, Macromedia vision

Using RIA technologies like Flash, Ajax or Java (Applet, Web Start) it is now possible to approach the goals of this vision.

2.2.2 A brief look at existing RIA technologies

There are currently 3 technologies that are of particular interest when we are addressing the term RIA. The three are Macromedia Flash/Flex, Java (Applet, Web Start) and AJAX.

2.2.2.1 Macromedia Flash/Flex

Being inventor of the term RIA, Macromedia Flash introduced amazing graphics, audio and video to the end user via a downloadable browser plug-in. After developing their own script language, Action Script, and a high end development platform Macromedia quickly established itself as one of the major actors on web technology.

The core architecture of the original Flash player is server – client oriented, where the user must download a Flash plug-in to the preferred web browser. Information from the server is downloaded in a compressed binary format as *.swf files, and run in the client web browser. On a mobile platform, using the Flash lite player, XML is used to structure the information.

Later Macromedia introduced their RIA Presentation Server Macromedia Flex. Flex provides an XML-based development platform that supports seamless middleware integrations such as Web Services, .NET and J2EE. Web development is also supposedly easier with the Flex platform than with the traditional Flash platform. Figure 2 shows the Flex architecture against a Flash lite player, on a mobile platform, and a Flash player inside a web browser. The server uses the XML format when communicating with the Flash lite client, and .swf if the client is a ordinary Flash player.

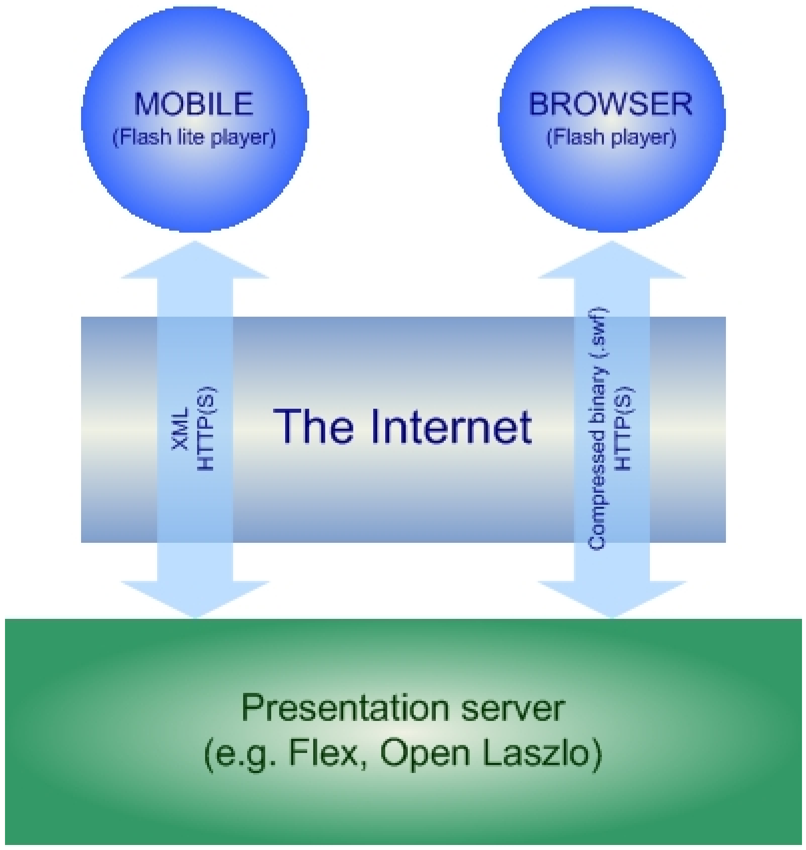


Figure 2: The Flash/Flex architecture

2.2.2.2 Java (Applet, Web Start)

Java has for years provided technology for RIA development. Even before Macromedia introduced the term RIA, Java provided enchantments for the web through Java Applets. Later on they have introduced the technology Java Web Start.

The Java web technologies Java Applet and Java Web Start operate on the Java Runtime library. In order to run Java programs the user would have to download and install a version of the Java Runtime Environment. While Java Applets are run in a container specific location in the web browser, Java Web Start is run via a Java Web Start plug-in from the operating system. Where Java Applets are prohibited to access the local system, Java Web Start provide full access to the Java library, and makes it possible to develop advanced applications that can be allowed to interact with the local system. Java Web Start applications are deployed on the web server, and are after download installed in a temporary folder on the user's machine. Internet connection is not a requirement to start and use the application, as they act like traditional Java applications. This allows the possibility to present them as shortcuts on the user desktop. When a Java Web Start application is executed, it will try to connect to the web server and look for newer versions. This is indeed a very effective way to distribute updates to the application users. Being run outside the browser Java Web Start applications use Java Applets to communicate with the web browser environment. This makes it possible to overcome the traditional barriers when using Java Applets. Figure 3 shows the communication between client and server for both Java Web Start and Java Applet. This is achieved using RIM, IIOP and HTTP. [5]

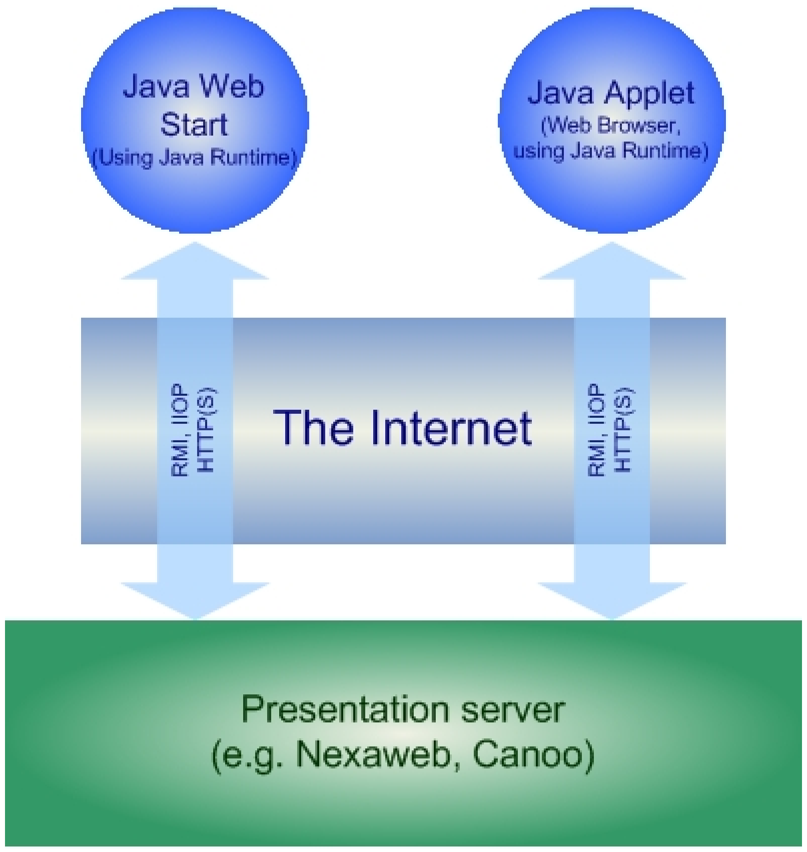


Figure 3: Java Web Start/Applet architecture

Using the UI library Swing, JSP, JSF and numbers of compatible open source frameworks, Java developers have a rich selection of web development tools available. Access to a wide and comprehensive open source environment, low cost development tools and environments like the Eclipse platform makes Java a popular and powerful choice.

2.2.2.3 Ajax (Asynchronous JavaScript and XML)

Being the newcomer among the RIA technologies Ajax has surely gotten its fare share of attention over the last year. Since Jesse James Garret, the founder of the web site adaptivepath.com baptized this new technological approach Ajax, Asynchronous JavaScript and XML, numbers of Ajax enabled web sites have been born. If one should try to explain the hype surrounding this technological approach we believe that one good explanation simply is the possibility to achieve asynchronous communication between the web server and the web browser without the need for a browser plug-in. Ajax makes it possible to refresh defined parts of the user interface without the need for a full refresh of the whole web page It is possible to update parts of the page while the user plunders with some other site functionality.

Another explanation is perhaps what Google has managed to do with applications like Google Maps and Google Suggest, where they use a variant of the Ajax technology to make web applications that has impressed web developers and users world wide.

More explanations could be the growing Ajax community, supportive frameworks, and broad attention from world leading companies like Sun, IBM, Microsoft and others. And let us not forget that Ajax build on well know technologies like JavaScript, Cascading Style Sheets (CSS), XML, XMLHttpRequest and Document Object Model (DOM) to take advantage of the unrealized potential already present in modern web browsers.

Being a newcomer Ajax is still fresh, and this is indeed a challenge when companies consider Ajax as a new technological approach.

2.3 INTRODUCING AJAX

In this thesis investigating Ajax and the standard relating to Ajax is important. This section will present a closer look at Ajax architecture and its relationship to other web standards.

As we already have mentioned, Ajax is a technological approach building on already existing technologies. The approach makes it possible to achieve asynchronous communication, and this way enabling the possibilities to update selected parts of the user interface while others remain at their current state. No more need for a complete page request, just the information you need or want to refresh. Since Ajax is built on well know web technologies like JavaScript, DOM and XMLHttpRequest there is no need for a browser adapted plug-in to make it work. Just JavaScript enable your web browser and Ajax will work as it is intended to.

The name Ajax was indeed defined by the now well known Jesse James Garret, but the core technological approach was invented earlier. The summer of 2000, Brent Ashley wrote the client-side libraries JavaScript Remote Scripting (JSRS) and Remote Scripting Lite (RSLite). The libraries were built to achieve asynchronous communication between the web page and the server without refreshing the page. The core technological approach was built on letting Dynamic HTML (DHTML) elements make hidden remote procedure calls to the server. The result was asynchronous communication. Ashley's work indeed made developers aware of the possibilities of asynchronous communication, and he is indeed on of the real founders of the Ajax approach. [6]

2.3.1 The Ajax architecture

Figure 4 visualizes traditional web communication versus Ajax enabled web communications.

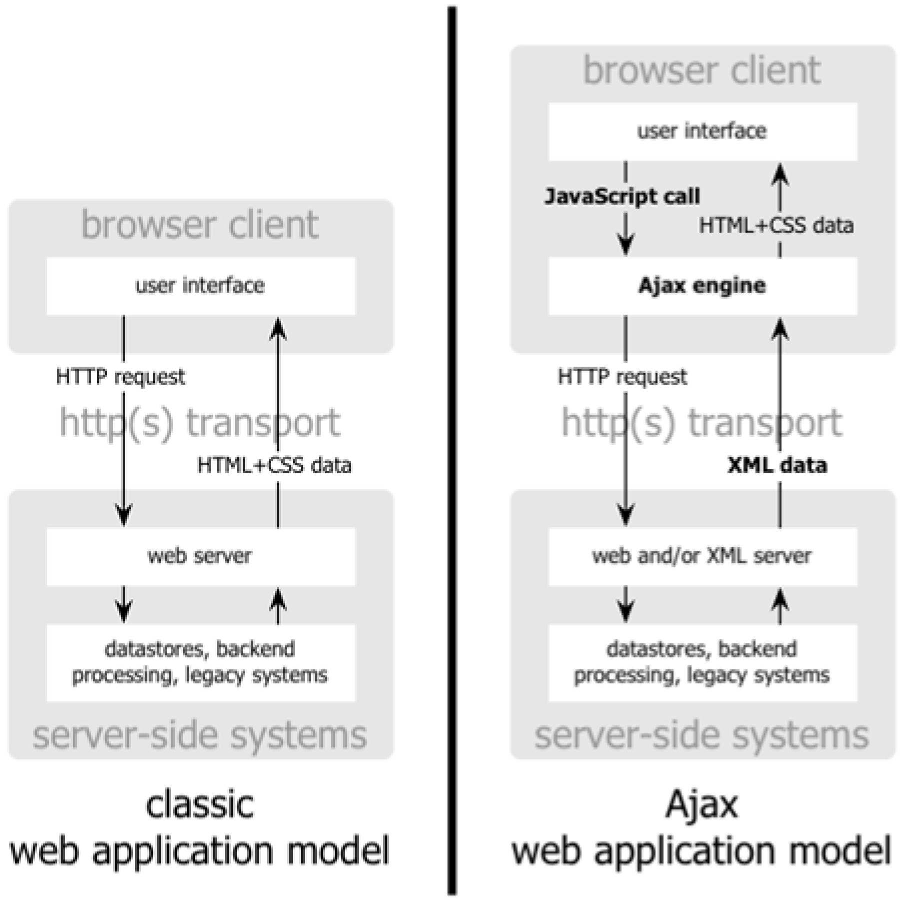


Figure 4: Classic versus Ajax web application model [7]

In the classic web application model we see a traditional web transaction between a web server and a web browser. The user connects to the web server by sending HTTP requests, and the server responds with the requested page. This interaction is repeated each time the web browser sends a new HTTP request to the server. Figure 5 visualize a typical synchronous communication between a web server and a web browser.

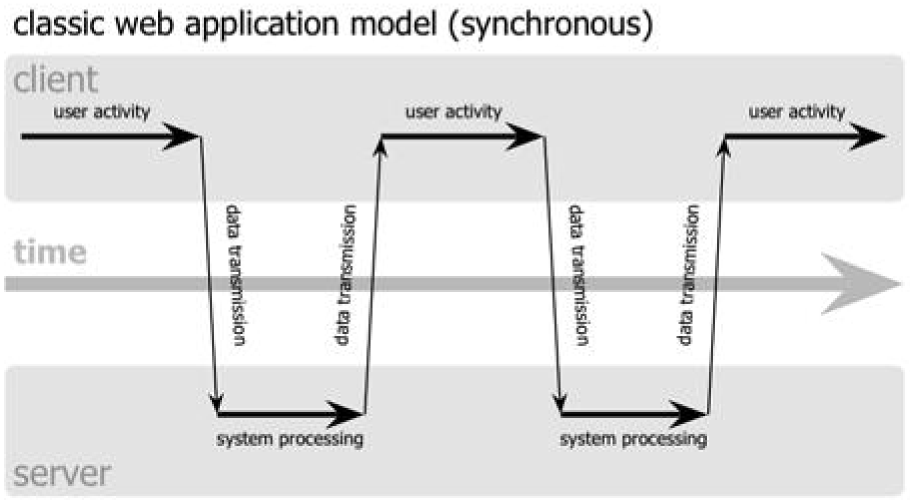


Figure 5: Classic web application model (synchronous) [7]

As the figure shows, user activity is only allowed between server response and the next server request. Usually a request-response action is executed very quickly, and not very problematic for the user. The problem emerges when the developer wants to refresh parts of the page, while the user is using the application and not disturb the user in this process.

Using the Ajax approach, it is possible to overcome this problem. Figure 6 visualize communication between the web server and the web browser, using an Ajax engine.

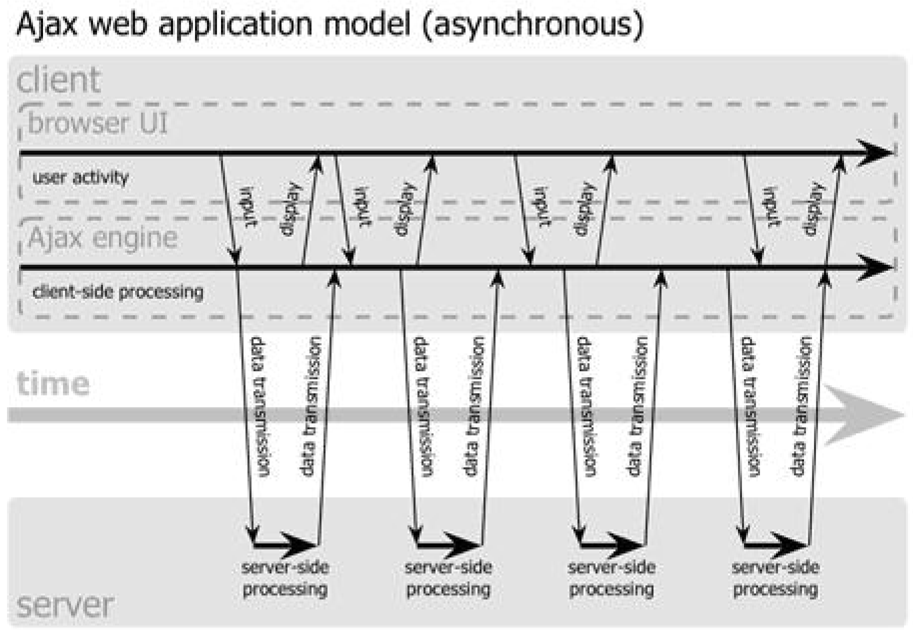


Figure 6: Ajax web application model (asynchronous) [7]

We see that the main difference from the classic model is the Ajax engine. While the user is working with the web application, the requests sent to and from the server are handled by the Ajax engine. To handle the requests and updates from and to the user interface, the engine uses JavaScript. This enables the possibility to alter information in a specific web control, and no page refresh is needed.

2.3.2 Ajax, under the hood

The fundamentals of Ajax are JavaScript, DOM, CSS and XMLHttpRequest. All four technologies have been around for a while, and all but the XMLHttpRequest technology has been standardized. We will look a bit closer at Ajax and standardization issues later in the report, but knowing that the W3C has put together a working group to make a standard for the XMLHttpRequest technology is in our opinion a great leap in the right direction.

So knowing the ingredients of the Ajax recipe, we draw the conclusion that developing Ajax enabled web applications demands some skill in all of the fundamental technologies. A closer look at them, and Figure 7, might reveal the complexity.

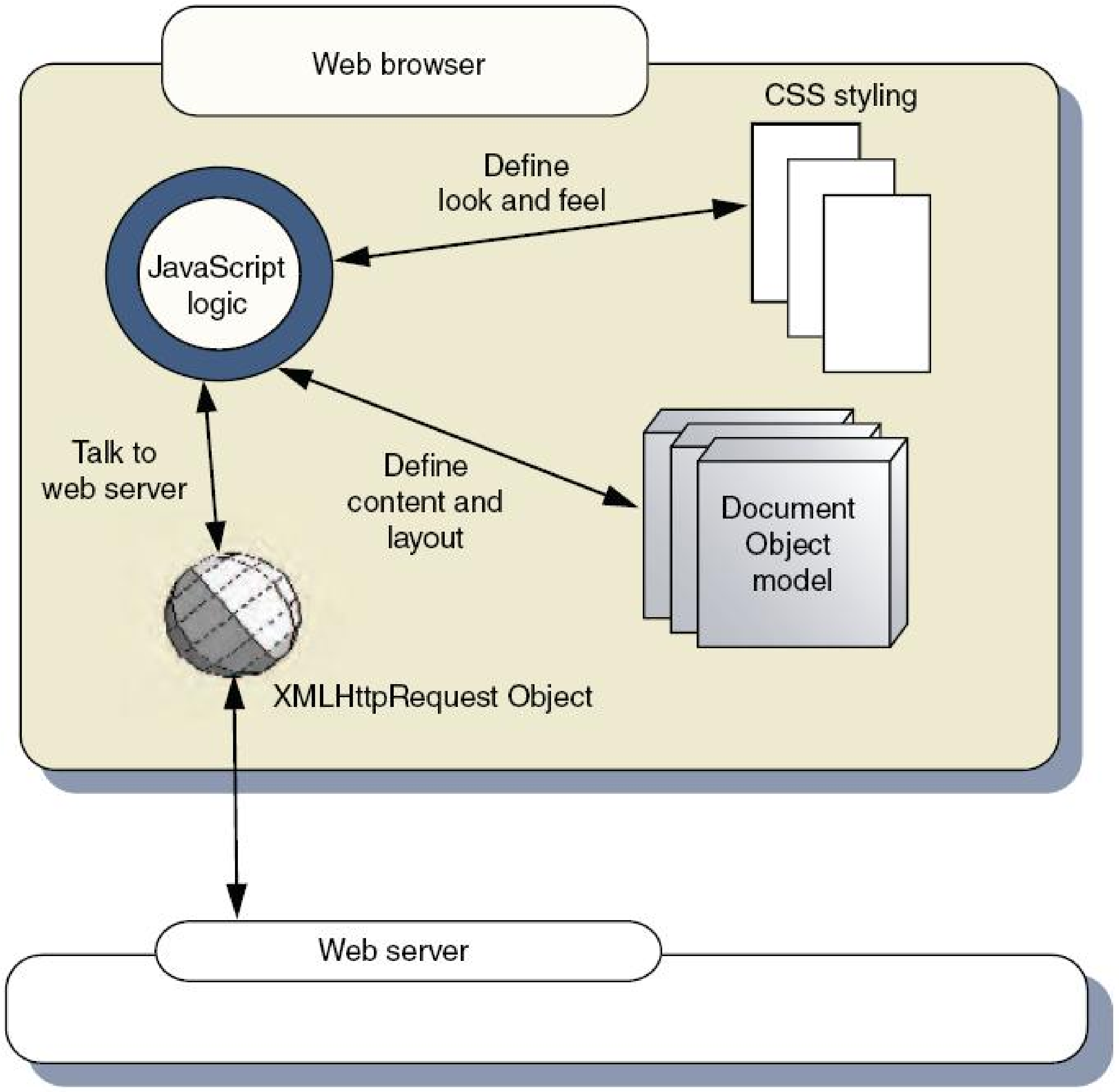


Figure 7: Ajax, the building blocks [9, p.63]

2.3.2.1 JavaScript and DOM

A vital part of Ajax is JavaScript, which is a prototype-based scripting language, with a syntax that is loosely based on C. The language is dependent on the host environment, and is executed directly in a supportive web browser.

Fundamental for the technology is the ECMA-262 specification, the standardization of ECMAScript, and the fact that all major web browsers currently support it.

JavaScript offers ways to interact with the DOM (Document Object Model) of a web page, and make it possible to execute script commands that can alter the presentation of the web content, and even build the page from scratch. DOM is standardized by the W3C as a standard, portable way to access all of the elements and text within and HTML document. This way JavaScript have access to all document content in a web page, and this is indeed needed when Ajax functionality is being deployed. So the first step when considering the core Ajax approach is to gain some JavaScript knowledge, and examine the possibilities of DOM. An alternative is to consider the various Ajax frameworks available. We will look closer at this in section 2.3.3, but can reveal that there exist several frameworks that can simplify Ajax development.

2.3.2.2 CSS (Cascading Style Sheets)

CSS offers the possibility to style a document by defining a set of standardized rules that can be applied to individual elements on a web page. CSS provide rules that can alter color, borders, background images, transparency, and size of elements. It is even possible to add simple user interactivity. In traditional web applications CSS has been used to provide a set of web pages with the same style. In Ajax applications this is not necessary changed, but some Ajax enabled web applications will be presented as one page, only updated by user interaction, and defined functionality that refresh that single page with information from the web server. Even so, CSS provide us with a comprehensive repository of predefined styles that can be applied to elements dynamically with a minimum of code.

CSS is currently standardized by the W3C and is constantly being enhanced with new functionality. The latest working draft, the CSS 2.1 Specification was published the 11th of April this year.[8] [10]

2.3.2.3 XML and XMLHttpRequest

To transfer data to and from the web server, Ajax applications use the XMLHttpRequest object. XMLHttpRequest is an API that can be used by a scripting language to transfer data in XML format or plain text if preferred. The real Ajax magic is dwelling on the asynchronous possibilities of data transfer the XMLHttpRequest object is capable of delivering.

As mentioned earlier W3C is also working on a specification for the XMLHttpRequest API. A set of minimum requirements will be defined, and it will be up to the browser manufacturers to follow them.

2.3.3 The Ajax engine, frameworks and tool support

Now that we have had a brief look under the Ajax hood we perhaps see that Ajax development might turn out to be a complex and time consuming affair. A complex Ajax site will be dwelling on loads of JavaScript code, and several XMLHttpRequest objects constantly demanding information from the server. Now this is where an Ajax engine would come in handy. And they exist, in form of several frameworks, developed to make the development process easier for you.

Now, in our report we are looking for frameworks that can help us develop Ajax applications using the Java platform. And in this process we have noticed that there exist several frameworks, not only for Java, but also for other platforms like DOT.NET, Pearl, PHP, Python and Ruby. In this process we have chosen one of these, Direct Web Remoting (DWR)

2.3.3.1 Direct Web Remoting (DWR)

DWR is an open source library with Ajax adapted tool support. The concept behind DWR is to make it possible to run Java functions from a web server via the web browser. This is done by letting a Java Servlet run on the server, processing requests and respond back to the browser. The browser contacts the server using JavaScript, and dynamically updates the page based on the response from the web server. DWR is marshalling the data communication between the browser and the web server.

2.4 AGILE SOFTWARE DEVELOPMENT

TDD is considered as one of the agile software methodologies. In this section we will introduce both agile methodology and TDD. We will also look at software testing, and have included information connected to the design phase of TDD.

Initializing a large scale software development process requires a defined goal, and guidelines on how to reach the goal. In some cases the process itself could be a part of the secondary goals, but for most software companies, the end result in terms of profit is what really matters. As an old expression say, there are many roads that lead to Rome, and there are also many ways to organize a software project.

In this section we will start looking at agile software development and continue with a look at TDD. In addition we will look at some of the testing phases of a software project, and take a closer look at TDD and the pre design phase.

2.4.1 Agile software development

TDD is defined as one of the agile software development methods, and shares its fundamental ideology. The agile methods try to minimize the risk by developing software in short time boxes, or in other words iterations. Each iteration is given a short period and must be completed before a new is started. An iteration can be viewed as a miniature software project on its own, and might consist of all standard software stages like planning, requirements analysis, design, coding, testing and documentation. The difference is that the result is just a part of the whole software project. Communication is preferably done face-to-face, and working code is more important than documentation. The agile teams are often organized in a bullpen, including all people necessary to finish the software. This includes at a minimum programmers and customers. It can also consist of testers, interaction designers, technical writers and managers. [11]

While the agile software development methods are often classified as adaptive methods, traditional and more disciplined methods are represented as predictive development methods. Figure 8 visualize by setting agile software methods up against the waterfall method.

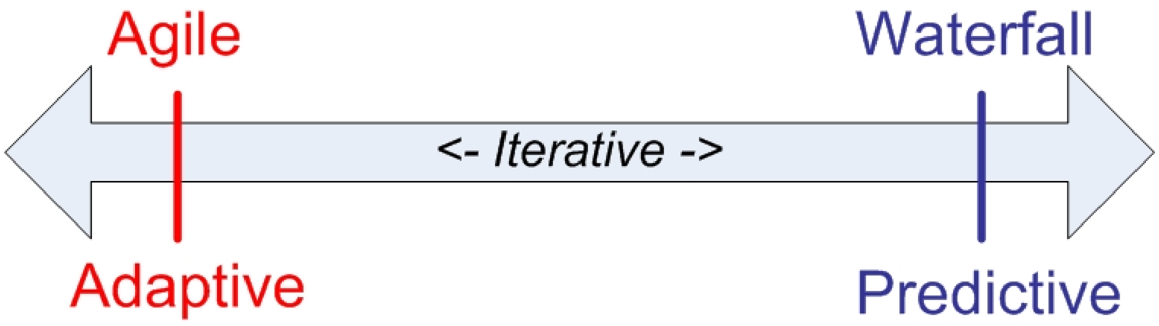


Figure 8: Adaptive versus predictive methodologies

On the figure we see the iterative arrow, with the adaptive development methods on the left, and the predictive on the right. While agile teams are focusing on code, verbal communication and small iterations, they can more quickly adapt to changes. But on the other hand, this might require that your staff has some experience, and have the fundamental knowledge to make the adaptive process successfully. On the right side the waterfall method represent the fulfillment of a predictive method. Using a step by step ideology, and doing this in a strict-planned sequence, waterfall is dwelling on order in form of analyzes and written documentation.

2.4.2 Software testing

Testing software is a requirement in most of the development methodologies. The question is really if you should test the code after you have implemented it, or before. As we see it, there is no final answer to this question, as it all depends on the developers working with the project, their experience, level of discipline and work routines. We believe a team of experienced and disciplined programmers will do quite well in either setting, and the reason is that both testing first or last demands discipline and experience. In a TDD scenario the programmer is strictly writing a test before the code is implemented, following the cycle shown in Figure 10. If this is done properly the result will be software, tested with a test framework, refactored code, and let us not forget the test suite, consisting of detailed information on all the classes and methods of the project. In a test later scenario the developers will typically start their implementation after doing detailed specification and requirement documents. The test phase will be done in the latter stages of the project. Now given that you don't need a very comprehensive documentation, TDD will leave you with enough to understand how the software is working down to the level of out/input values. If you on the other hand are dependent on a comprehensive documentation, the unit test suite might lack the weight of traditional specification and design documents. If the goal is to get the software shipped as fast as possible, TDD might be the thing.

Software testing is really a wide term, and if we cannot possibly cover it all in this project. In our investigation we will cover the phase of unit testing and look briefly at the integration test phase.

2.4.2.1 Unit testing

Unit testing is an expression often combined with the agile software methodologies. The philosophy is to write tests that are testing at a low level, e.g. a class containing of various attributes and methods. By using a unit test framework that supports your chosen programming environment, you will then be able to run the tests automatically. The feedback will be either success, or failure, with a response attached to the method failing. A unit framework will give you access to a set of methods, developed to test your code.

In Kent Beck's original testing framework paper on eXtreme programming, he describes unit testing like this: “I recommend that developers write their own unit tests, one per class. The framework supports the writing of suites of tests, which can be attached to a class. I recommend that all classes respond to the message "testSuite", returning a suite containing the unit tests. I recommend that developers spend 25-50% of their time developing tests.” [12]

As we have mentioned earlier, one of the advantages of TDD is that the produced test suite, the set of unit tests attached to the software, is working as documentation. In a scenario where you at a latter stage need to change some code, the test suite will let you know if you break dependencies and other code in the process. At the same time, TDD is about testing first, so the result will be a revised test suite, updated with all recent changes.

Figure 9 visualize the development and test cycle of a software project. If we study this model we see that each of the development phases is closely related to the test phases. In a TDD project most of the development will be done on the unit test phase. In a predictive software methodology like Waterfall the design phase will be adjacent to the unit test phase in a TDD project.

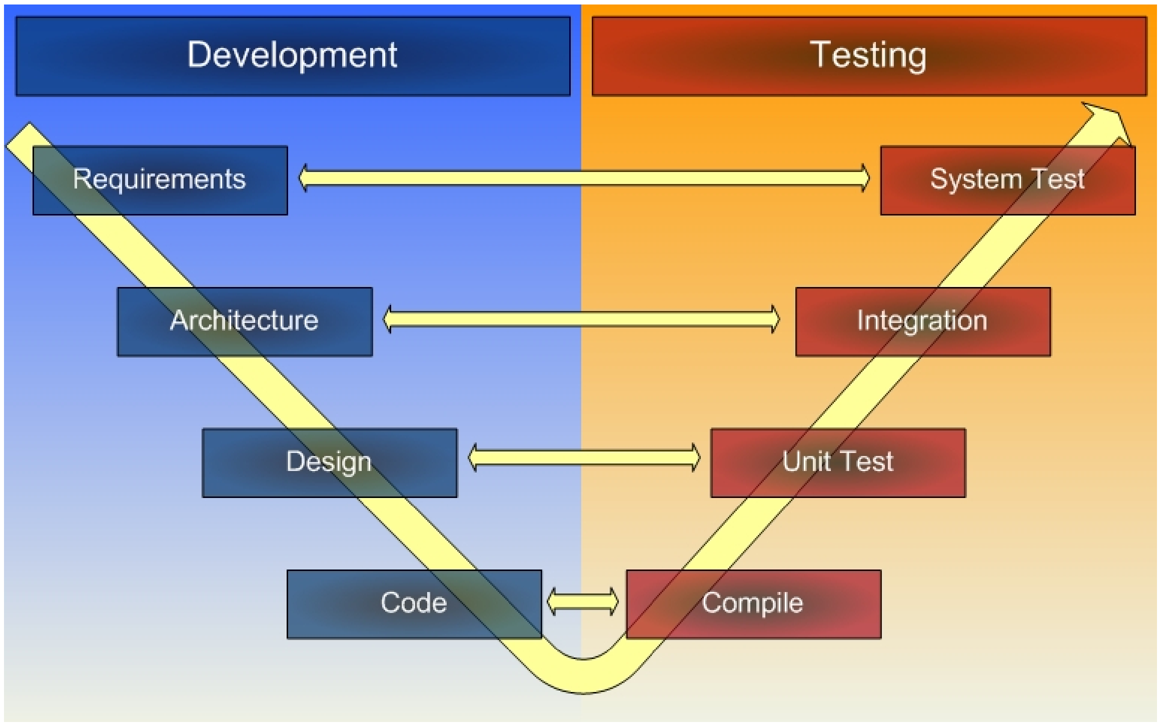


Figure 9: The development and test cycle

2.4.2.2 Integration testing

As unit testing is done on design level, integration testing is executed on architecture level. The modules produced in the unit test phase are grouped, and fed with simulated interaction. The results are analyzed based on the expected response from the tests. George Ellen is defining the goals of integration testing like this in his article: [13]

“Ultimately, the goals of integration and test are to:

- Bring together the multiple pieces of a system
- Find and fix defects that couldn’t be found earlier
- Ensure that the requirements of the system are met
- Deliver a product of predictable quality that meets the business’s quality objectives as well as the customer’s quality expectations.”

To reach the goals Ellen defines we need a framework that can help us test our software on this level. In this report we will focus on testing in the Unit test phase, but will recommend a testing framework that can help you achieve integration testing in Ajax.

2.5 TDD, BASIC INGREDIENTS

In this section we take a closer look at some of the basic ingredients we have found important to be aware of when considering TDD.

As TDD is among the agile software methodologies, we can draw some basic conclusions. The software development process is divided into short period iterations, and the goal of each period is to achieve working software. Functional code is preferred over written documentation, and communication between the team participants are mostly verbal and direct.

TDD is a way to define how to organize how to produce code within the development phase. In Figure 10 we see the TDD cycle.

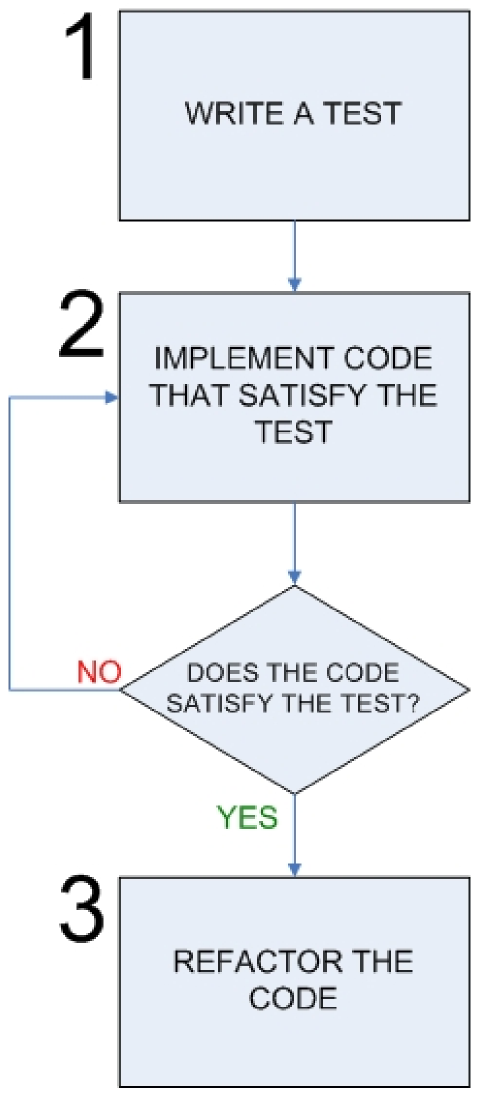

1
WRITE A TEST
2
IMPLEMENT CODE THAT SATISFY THE TEST
NO
DOES THE CODE SATISFY THE TEST?
YES
3
REFACTOR THE CODE

Figure 10: The TDD cycle

In an article discussing TDD, Dan North has this to say about it: “The point of TDD is to drive out the functionality the software actually needs, rather than what the programmer thinks it probably ought to have.” [38] With this in mind let’s explain the TDD cycle from Figure 10.

Instead of programming the specific code that is needed to implement the target software, we start with step 1, writing test code as if the implemented code already exists. Then we jump to step 2 and try to satisfy the test writing the minimum amount of code needed to compile. If the result don’t satisfy the test we do step 2 once more, if on the other hand satisfactory is achieved we move to step 3 and refactoring. The refactoring done in step 3 is all about making the code as simple and clean as possible. And out of the cycle is a part of the software, tested and quality checked. “TDD is not about the tests, it’s about seeing how little you actually need to do and how cleanly you can do it!” [18]

The next citation is taken from the web site agiledata.org and Scott W. Ambler: [14]

"Kent Beck, who popularized TDD in eXtreme Programming, defines two simple rules for TDD. First, you should write new business code only when an automated test has failed. Second, you should eliminate any duplication that you find. Beck explains how these two simple rules generate complex individual and group behavior:

- You design organically, with the running code providing feedback between decisions.
- You write your own tests because you can't wait 20 times per day for someone else to write them for you.
- Your development environment must provide rapid response to small changes (e.g. you need a fast compiler and regression test suite).
- Your designs must consist of highly cohesive, loosely coupled components (e.g. your design is highly normalized) to make testing easier (this also makes evolution and maintenance of your system easier too).

For developers, the implication is that they need to learn how to write effective unit tests. Beck’s experience is that good unit tests:

- Run fast (they have short setups, run times, and break downs).
- Run in isolation (you should be able to reorder them).
- Use data that makes them easy to read and to understand.
- Use real data (e.g. copies of production data) when they need to.
- Represent one step towards your overall goal."

This short presentation of TDD shows that it is part of the agile software methods, and inherits the fundamental behavior of the agile software methodology. We also see how the TDD cycle can help developers write clean and working code.

When we at first started this project we had very limited background knowledge about the TDD methodology. This led to a period where we tried to understand the concept of TDD. During this process we have identified what set of basic ingredients TDD require to achieve success. The list is based on our own experience and our experience is built upon literature study and prototype development.

We have identified the following basic ingredients:

- A simple pre design technique.
- A unit testing framework.
- A decent portion of TDD knowledge.

2.5.1 TDD and design

In our study we noticed that TDD was heavily based on the TDD cycle, see Figure 10, and left little room for a pre design phase. After a while this really felt a bit lacking. Our concern was that a software project, without a proper design paper, quickly could get out of control.

As we continued to study TDD, we were after a while made aware of a modeling technique named Agile Draw. We decided to look it up. What we discovered was a very simple modeling technique. Simple but yet seemingly powerful enough to make high level models covering all but the detailed part of design and code level phases of the project. Se Figure 9. This could be exactly what we were looking for.

The agile draw team vision is: “The primary goal of Agile Draw is to serve as the simplest modeling technique that is pleasingly natural, provides design freedom, requires the most basic tools, promotes modeling creativity, fosters better communication, and complements existing standards. A secondary goal for Agile Draw is to give developers the confidence to do freeform modeling.” The Agile Draw technique is closely related to the principles of Agile Modeling. [15] The founder of agilemodeling.com, Scott W. Ambler serves as one of the contributors to the Agile Draw modeling technique. Table 1 shows the core principals of the Agile Draw technique.

Table 1: The core principles of Agile Draw [39]

1	*Simple:* Many complex models can be communicated using a set of simple shapes, connectors and minimal notations. Shapes denote the type of entity, connectors denote the relationships and notations to decorate and add additional constraints and meaning.
2	*Less is more:* In many cases, models with large number of artifacts, relationships, and constraints eventually become so complex that most of developers simply refuse to use them. Furthermore, as software changes, it is hard to update the models. Keeping the models light and clean is an effective way to manage complexity.
3	*Light-weight tools:* Modeling should not require sophisticated tools and products. An easy to use diagramming tool and a pragmatic approach where the team can collaborate, exchange and modify the models is all you need.
4	*Easy to understand:* Agile Draw maintains its simplicity by using an intuitive approach to modeling that also makes it easy for users to understand the design and concepts behind the model.

So to achieve this, Agile Draw introduces a modeling technique that seems able enough to provide us with a light, high level conceptual design of the system. Let's look at the Agile Draw basics.

2.5.1.1 Agile Draw basics

The Agile Draw premise is that data can be identified in two ways:

1. In motion
2. Stored

Based on this premise, two shapes, circles and boxes are founding the basics to present data in motion. Communication and/or relation between the shapes are visualized through simple lines. Minimal usage of text is used to describe the shapes and the lines. The Agile Draw terminology refers to these as points, connectors and text. So as Figure 11 shows, the basic Agile Draw components are points (circles, boxes), connectors (lines), and text.

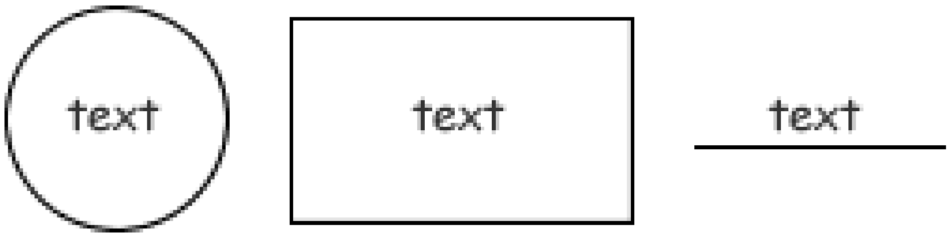


Figure 11: Agile Draw, basic components [16]

A deeper understanding of when to use what of the components is described in Table 2 and Table 3. In addition it is also important to remember that use of text should be minimal.

Table 2: Points, description

Shape	Depicts	When to use it
Circle	Movement/Infinity	“Listeners” such as application servers, web server, database server, messaging servers, listener classes.
Boxes	Fixed/Rigidity	Hardware, classes, entities/tables, user interface, or just about anything.

Table 3: Connectors, description

Style	Depicts	When to use it
Solid	Concrete, synchronous, tight association, etc.	Synchronous communication, entity-relationship, etc.

Dashed	Abstract, asynchronous, loose association, etc.	Asynchronous communication, extending abstract class, etc.

Agile Draw offers a simple way to design the project before you start writing your tests. We believe that this technique is here to stay, and the reason is the simple and yet working design techniques it offer to agile software development methods.

2.5.2 Unit testing, frameworks and mock objects

TDD is all about the tests. The most important of the TDD ingredients is a framework to help you run your tests. If we look back at Figure 9 we see that the second phase, just after the compiler has finished doing its job is the unit test phase. Unit testing is done on low level, and forms the basic when test are to be written in a TDD cycle, se Figure 10. To help us write and run our tests there are several frameworks available, supporting many languages. A common name on these is xUnit frameworks.

The xUnit frameworks offer a set of defined methods you can use to design your tests, and in addition, most of them also offer a graphical GUI, where you can receive feedback on the tests running.

To test Java code the main unit test framework is JUnit. When developing JavaScript there are several alternatives.

When performing unit testing you might be facing a situation where you have to make tests that use databases, communication devices, user interfaces or any external application. In these situations the solution could be a simulated object of the real instance, or a mock object.

We will introduce JUnit, the JavaScript alternatives and mock objects in the Sections below.

2.5.2.1 JUnit

JUnit is the main unit testing framework for the java platform. In our case it came with bundled with Eclipse, and we didn't have to do much to make it run. JUnit is shipped with methods to test code behavior, and a test suite to present test results to the developer. To use it you must know of the basic set of assertion methods it provide, and how to set up tests for your code. We have provided an overview of the basic methods in Table 4.

Table 4: JUnit, assertion methods

Method	**Description**

assertEquals(Object expected, Object actual)
assertEqual(String text, Object expected, Object actual)
	Check that two objects are equal by using the Object.equal() method.

assertTrue(Boolean condition)
assertTrue(String text, Boolean condition)
	Check that a value evaluates to true.

assertFalse(Boolean condition)
assertFalse(String text, Boolean condition)
	Check that a value evaluates to false.

assertNull(Object value)
assertNull(String text, Object value)
	Check that a value is null.

assertNotNull(Object value)
assertNotNull(String text, Object value)
	Check that a value is not null.

assertSame(Object expected, Object actual)
assertSame(String text, Object expected, Object actual)
	Check that two values are the same – that is, the same reference.

assertNotSame(Object expected, Object actual) assertNotSame(String text, Object	Check that two values are not the same reference.

expected, Object actual)	

fail()
fail(String text)
	Fail the test.

2.5.2.2 JsUnit

JsUnit is one of the unit testing frameworks developed for JavaScript testing. The development of JsUnit was initialized in January 2001. JsUnit is essentially a port of JUnit to JavaScript, and offer a platform for automating the execution of tests on multiple browsers. The framework also provides a plug-in developed for integration with the Eclipse platform. [17]

2.5.2.3 Script.aculo.us

Script.aculo.us is first and foremost a JavaScript/Ajax framework, and not a testing framework. The reason it fits in here is that it provides a set of classes and methods for JavaScript unit testing. To write unit test cases they have provided a utility class. The development of test cases is structured in a XHTML page, and run via a web browser. Testing is only supported via the Firefox web browser. [18]

2.5.2.4 J3Unit

J3Unit is an object-oriented unit testing framework for JavaScript. It is built on the work done in both JsUnit and Script.aculo.us. The test cases are written in the utility class provided by Script.aculo.us. J3Unit is currently in beta, and supports testing in Firefox 1.5 and Jetty 6.0.0 beta. [19]

2.5.2.5 Mock objects

Mock objects are fake objects made to simulate the behavior of a real object. Examples are databases, communications devices, user interfaces and external applications. Instead of ending up with long messy unit tests, a mock object can make a virtual instance, and simulate the behavior of the target instance.

2.5.3 A decent portion of TDD knowledge

We believe that the third ingredient for a successful TDD experience is a decent portion of TDD knowledge. You need to know what you are doing, what you should test and how you can implement the code to succeed the test. There are good literature and several web sites available. Our advice would be to have a look at some of these.

3 TDD AND AJAX

3.1 IDENTIFYING THE CHALLENGES

Based on our literature study, we started to analyze both Ajax and TDD to identify any possible challenges. To reach this phase we first had to understand in detail the architecture and building blocks of the Ajax architecture. At the same time we needed basic understanding of how to use the TDD methodology, not only in theory, but in practice.

In this chapter we will introduce the work we have done to identify challenges attached to TDD of Ajax enabled web applications. We will introduce the challenges identified and suggest possible solutions.

The solutions presented will only cover the phases where TDD and Ajax introduce challenges, and will not cover the web development phase and TDD as a whole.

3.1.1 Choosing a unit testing framework for JavaScript

When developing Ajax enabled web applications you will most surely end up with a good amount of JavaScript code. JavaScript is a script language, and can be rather messy to start working with, especially if you are used to stricter and object oriented languages like Java, C++ or C#. In section 4.3 we will look at ways to structure design through the usage of design patterns.

In the TDD development process we have already identified the need for a unit testing framework. When we started looking for a supportive testing framework for JavaScript we wanted a framework that was mature and easy to use. As testing framework we ended up with JsUnit.

Considering the alternatives, Script.aculo.us and J3Unit we chose JsUnit based on the maturity, the simplicity and the Eclipse plug-in tool support.

The JsUnit project was started in January 2001, and is unit testing framework for client-side, in-browser JavaScript. One of the neat features we found appealing with JsUnit was a plug-in developed for the Eclipse platform.

Even though JsUnit can be run through the Eclipse plug-in it demands a defined web browser to do the job. In Figure 12 we see how JsUnit works.

If you use Eclipse, and have installed the plug-in properly you will be able to run a chosen html page as a JsUnit test from Eclipse. If you are not using Eclipse you will have to run the testRunner.html page provided with the JsUnit distribution. As Figure 12 show, the plug-in will launch the testRunner.html page, run the testPage.html page, and then run the tests defined in the testPage.html.

With the Eclipse plug-in, this is automatically done. Without it you will have to define what page to test in the testRunner.html page.

The JsUnit testing suite then will look up the tests defined in your test page. In Figure 12 we have visualized the JavaScript to be tested as separate files, Collection1.js, Collection2.js and so on. Putting your JavaScript in separate collections is a good way to improve the design of your project. In a TDD perspective you will have high level figures, or at least a notion about what you are trying to implement. A suggestion is really to use information provided by either specification or ideas to build your JavaScript collections.

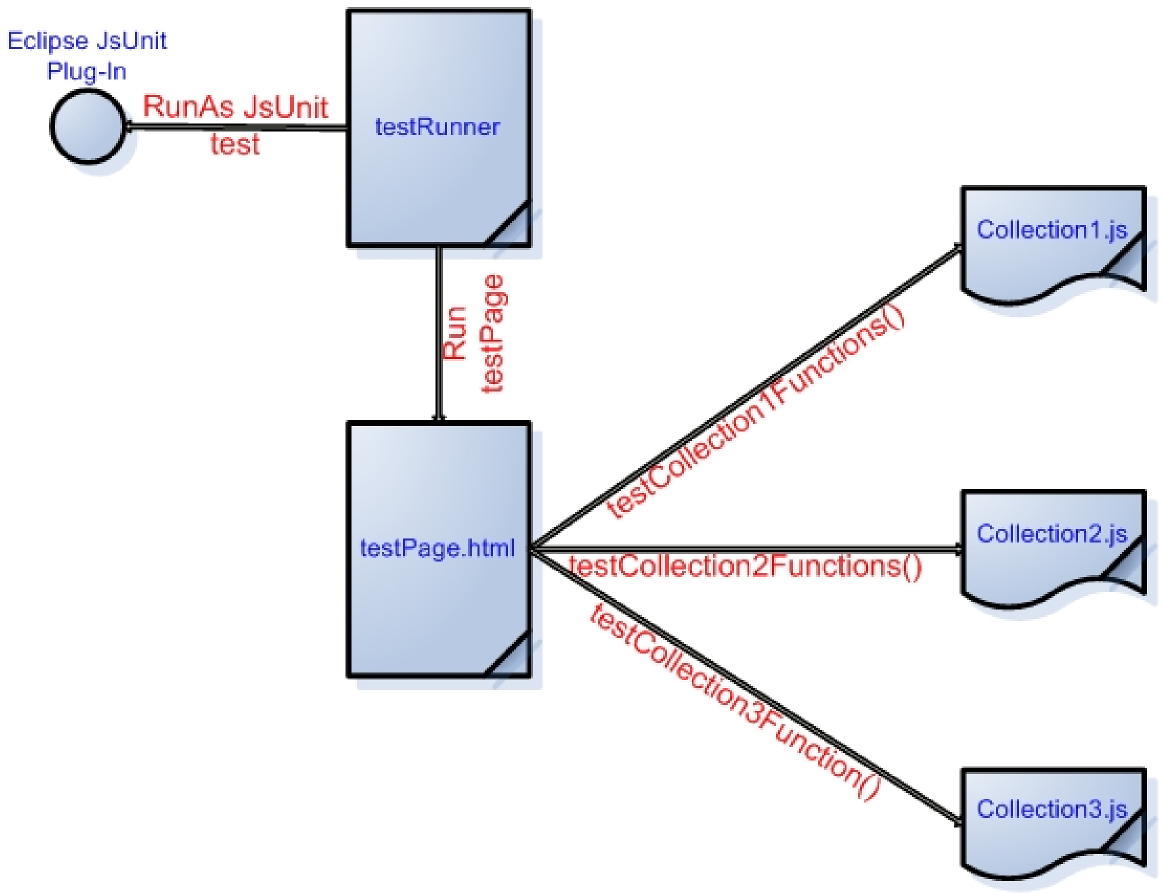


Figure 12: Running JsUnit tests from the Eclipse platform

To make a unit test case for JsUnit you first of all need an html page where you can write your unit tests. In Figure 12 we have called this page testPage.html. To make the JsUnit test suite aware of your test, their name must start with test, and then something of your own choosing. Let's say you have a function that initialize your XMLHttpRequest object, and want to check if it is created. In Code example 1, we show a function that initialize a XMLHttpRequest object based on the web browser that is calling the function. We have put this function in a file that we have called request.js.

Code example 1: Initialize a XMLHttpRequest object

```
function initXmlHttpRequest(){
      var XMLHttpRequestObject = false;
      if (window.XMLHttpRequest) {
            XMLHttpRequestObject = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
            XMLHttpRequestObject = new
ActiveXObject("Microsoft.XMLHTTP");
      }
      return XMLHttpRequestObject;
}
```

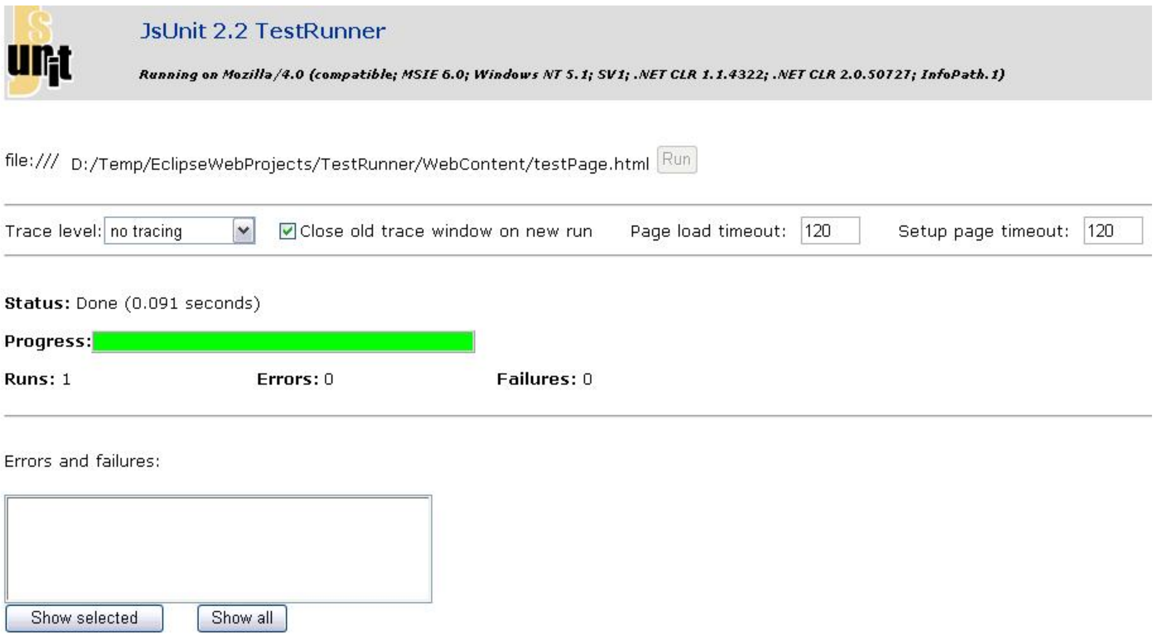
The code showed in Code example 2 shows how this simple test can be coded. Worth noticing is that you will need the jsUnitCore.js file from the JsUnit distribution defined in the file where you will create your test suite. First of all we create a variable that we call XMLHttpRequestObject. This is yet only a variable, and we have defined that it is null. The real test is happening when the JsUnit test suite discovers the function testXMLHttpRequestObject(). First we make the assertion that our created object should be null, and not have any value attached to it. Then we initialize the initXmlHttpRequest() function, see Code example 1, where the variable will be defined as real XMLHttpRequestObject. If the test is to pass now, the variable must have been instantiated, and we make an assertion that will fail if it still is null.

Code example 2: The testPage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>Test-Suite</title>

<script language="javascript" src="/jsunit/app/jsUnitCore.js"></script>
<script src="JS/request.js" type="text/javascript"></script>

</head>
<body>
<script language="javascript"  type="text/javascript">

var XMLHttpRequestObject = null;

function testXMLHttpRequestObject(){
      assertNull(XMLHttpRequestObject);
      XMLHttpRequestObject = initXmlHttpRequest();
      assertNotNull(XMLHttpRequestObject);
}

</script>
```

```
</body>
</html>
```

The result is shown in both browser view, se Figure 13, and inside Eclipse similar to JUnit, se Figure 14.


JsUnit 2.2 TestRunner

Running on Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.1)

file:/// D:/Temp/EclipseWebProjects/TestRunner/WebContent/testPage.html

Trace level: no tracing ✓ Close old trace window on new run Page load timeout: 120 Setup page timeout: 120

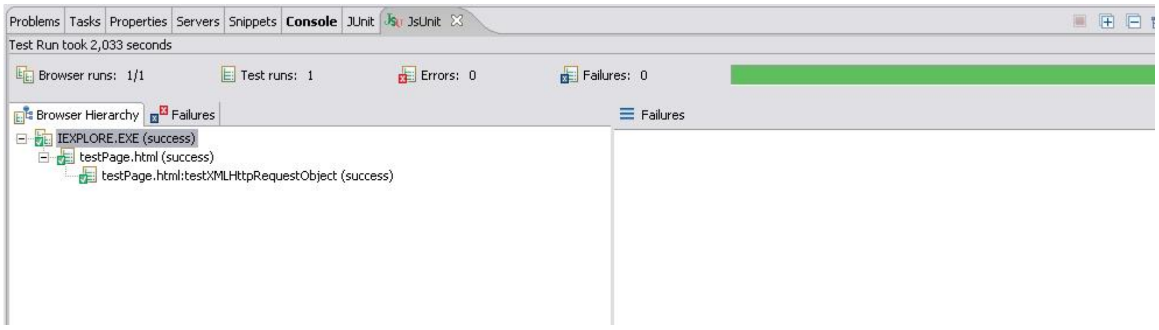
Status: Done (0.091 seconds)

Progress:

Runs: 1 **Errors:** 0 **Failures:** 0

Errors and failures:

Figure 13 JsUnit test suite, browser screenshot


Test Run took 2,033 seconds

Browser runs: 1/1 Test runs: 1 Errors: 0 Failures: 0

IEXPLORE.EXE (success)
testPage.html (success)
testPage.html:testXMLHttpRequestObject (success)

Figure 14: JsUnit test suite, Eclipse screenshot

3.1.2 Testing asynchronous communication in JavaScript

After choosing a proper testing framework for JavaScript we tried to test some basic Ajax communication, executed by JavaScript. Our concern was that it could be difficult to test that the response from the server was returned before the test was finished. As we tried to solve this problem we first tried to make a solution we have named, the waitForAWhile solution.

3.1.2.1 The waitForAWhile solution

The first thing that ran through our heads when encountering this problem was to force the test case to wait until the asynchronous response was done, and then try to analyze the result. After reading an article discussing this matter, but referring to a different testing framework, Selenium, we decided to see if it could be possible to make such a script for the unit testing phase. [20] [21]

The reason we could not use the Selenium testing framework instead of JsUnit is that Selenium is designed specifically for the acceptance testing requirements of agile teams. If we look back to the development and test cycle figure in Figure 9, Selenium will fit in under the integration and system test phases. We will look a bit closer at this in Section 3.1.3.

We then started working out a simple test for our JsUnit test suite, and made use of a script that did exactly what we wanted it to do, force the test to wait for a chosen time. [22] The script used is shown in Code example 3.

Code example 3: The waitForAWhile.js script

```
function waitAWhile(millis)
{
date = new Date();
var curDate = null;

do { var curDate = new Date(); }
while(curDate-date < millis);
}
```

We tried this out by requesting a simple text file from the server with only the string value, working.

As you see in

Code example 4, we call up the waitAWhile() script, and do an assertEquals() on our response, the variable asyncResponse. The asyncResponse variable is created in our pre made test page, and is used to represent the response from the server.

Code example 4: Testing the asynchronous request

```
function testAsynchronousResponse()
{
      getDataOnly('http://localhost:8080/TestRunner/data.txt',
XMLHttpRequestObject);

      waitAWhile(1000);

      assertEquals("Testing if the response from the server is
correct.","working",asyncResponse);
}
```

The reason this could even work out at all is the nature of the asynchronous request. As we se in

Code example 5, the function calls up a call back handler that is starting up a new process in a new thread. This thread is running until the XMLHttpRequest objects ready state = 4. We will look a bit closer at this in Chapter 4. So stopping the script would not stop the response from being created.

The results we got from this from this were quite confusing really. As we tried to run different tests we noticed that the response from the server often took quite some time to receive. At some occasions our test received a response from the server in less than 1 second. This seemed however not to be stable. Even a wait value set to 2 seconds sometimes gave test failure.

Another error we encountered was connected to the Eclipse plug-in. The JsUnit test unit runner didn’t manage to run the test properly at test launch. This is of most importance when using the Eclipse plug-in. The reason is that when a new test is run by executing a new JsUnit test on the chosen test page, the results are reported back to the Eclipse JsUnit view. If the test fails on the first try, the test will show as a failure in the Eclipse view. In our case the tests always failed at execution, but did work when we refreshed the web browser to try again.

Code example 5: Waiting for the XMLHttpRequest response

```
function getDataOnly(dataSource, XMLHttpRequestObject)
{
            XMLHttpRequestObject.open("GET", dataSource);
            XMLHttpRequestObject.onreadystatechange = function()
            {
                  if (XMLHttpRequestObject.readyState == 4 &&
                  XMLHttpRequestObject.status == 200)
                  {
                        asyncResponse =
XMLHttpRequestObject.responseText;
                  }
            }
            XMLHttpRequestObject.send(null);
}
```

After struggling for a while with this solution we realized that it was not satisfying. Realizing that this was no good solution we had to find a different approach to the challenge.

3.1.2.2 The mock object approach

If we look at our effort done in the last secion, we can only imagine how slow a test suite will run with many asynchronous request/response calls. In addition, in a large software project, ambiguous errors and unstable results is not acceptable. The developer should not have to worry about such matters, and in most languages he doesn't have to. To help developers cope with these kinds of problems there exist several mock object frameworks, able to simulate databases, Java Servlets and other objects that need a working and running instance to be able to test.

We have briefly explained what mock objects are earlier in the report. To fresh up a bit, mock objects are fake objects that's intension is to simulate the behavior of a real object. There are several defined patterns made for mock object creation, and each pattern try to solve a simulation of a specific problem. In a document by Matthew A. Brown and Eli Tapolcsanyi several patterns for mock object creation is introduced. Se Table 5 for a list of the mock object patterns presented in their document. [40]

Table 5: Mock object patterns

Pattern Name	**Synopsis**
MockObject	Basic mock object pattern that allows for testing a unit in isolation by “faking” communication with collaborating objects.
MockObject via Factory	A way of generating mock objects, utilizing existing factory methods.
Self Shunt	Unit Test code serves as the mock object by passing an instance of itself.
Pass in Mock Collaborator	Pass in a mock object in place of the actual collaborating object.
Mock Object via Delegator	Creates a mock implementation of a collaborating interface in the Test class or mock object.

An example could be using mock objects knowing that the class and the methods you are writing unit tests for will be dependent on other classes and methods that are not yet implemented. A solution will in this case be to make a mock object that wraps the interface of the class you are dependent on. This way you can override the internal methods, and define a set of response/requests manually.

In most of the platform environments, like the Java platform, it is possible to get hold of mock frameworks, and use these to implement mock objects that can simulate several elements. In JavaScript however, such frameworks seem to be of non existence.

We tried to analyze the challenge with Ajax, TDD and JavaScript again, and saw that we needed a mock object that simulated the XMLHttpRequest object. In Figure 15 we have tried to visualize the problem, and how it could be solved. If it was possible to encapsulate the XMLHttpRequest object with a wrapper function, we perhaps could override the methods that initialized the response/request communication with the server. Instead we could define a response being sent back to the requesting JavaScript function.

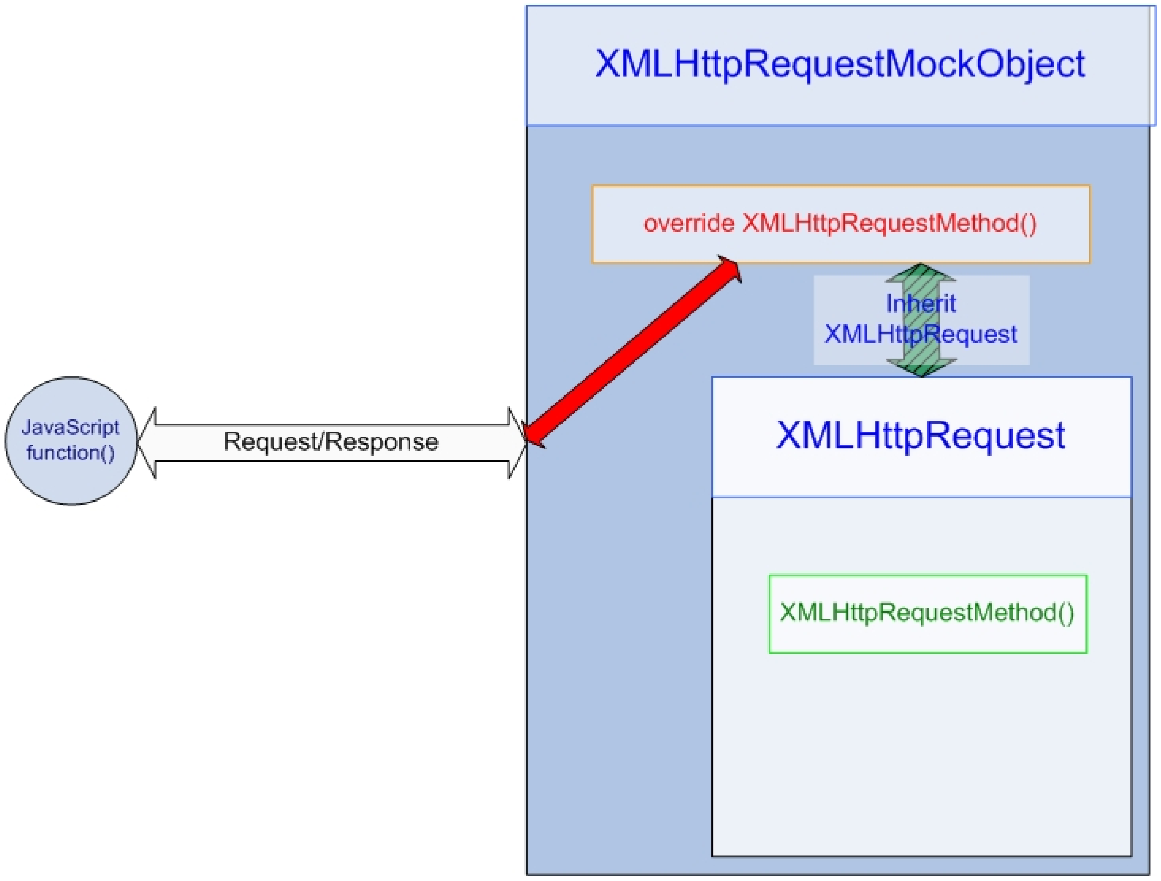


Figure 15: XMLHttpRequest mock object for JavaScript

So a very good solution to the Ajax – TDD challenge in JavaScript would be such a mock object, simulating the XMLHttpRequest object. Making the object, on the other hand, is another story.

Identifying the object to be mocked, we tried to see if we could develop a mock object for the XMLHttpRequest object. So we started to look at the possibilities in JavaScript. This was in fact quite encouraging, as we quickly were reminded on the limitations in JavaScript. First of all, JavaScript is a script language, there is in built support for object oriented development, inheritance, interface building and so on. As mock patterns heavily rely on these to work, we decided to only document this matter, and not try to develop a mock object on our own.

3.1.3 Ajax and integration testing.

As we encountered the challenge connected to the asynchronous request/response in the unit test phase, we also understood that this would make both integration and system testing of Ajax enabled web applications challenging.

Even though we have defined the unit test phase to be of importance in this report, we will suggest a possible solution to the integration test phase.

As with unit testing, integration testing is dependent on a proper framework that can help you test your software. In integration testing you might want to test components of the whole system, but in contrast to unit testing, the test has to be run against a working system. With a working system we mean the complete component, with its dependencies, web server, database etc. Integration testing of a web page or perhaps only a part of the page is done by simulating user behavior on it.

To overcome the asynchronous behavior produced in Ajax applications you will need an integration testing framework that can do the job. While we were searching for testing frameworks, we encountered the Selenium testing tool for web applications. [26]

Selenium is built to run directly in web browsers. It supports Internet Explorer, Mozilla and Firefox, and allow the developer to write tests in Java, .NET, Perl, Python and Ruby.

To test Ajax enabled web applications the developer must download and include an extension to the Selenium framework named waitForCondition. The extension allows the developer to run a chosen JavaScript, and set a timeout that terminates the condition after a chosen amount of time. If the script finished before the timeout is reached, the waitForCondition will be evaluated to true, and will stop waiting.

The waitForCondition script will make the integration test run as quickly as possible, and this is exactly what is needed when testing Ajax applications.

3.2 SUMMARIZING THE CHALLENGES

Our work done both through literature study and prototype development has helped us to identify the challenges connected to TDD and Ajax development. These are listed in Table 6.

Table 6: TDD - Ajax challenges

1	Ajax is heavily dependent on JavaScript, and requires a testing framework that is able to test JavaScript.
2	Ajax introduces asynchronous communication between the web client and the web server. In a TDD – Ajax scenario this cause problem connected to unit testing of the JavaScript code.
3	The asynchronous communication introduces similar challenge in the integration and system test phase.

4 AJAX

Previously in chapter two we presented a brief look at the technologies behind Ajax. In this chapter we will present the Ajax architecture in more detail and look at a sample prototype.

4.1 MAKING A AJAX REQUEST

A user would interact with an Ajax application through a web interface, so let’s start by looking at some html code.

Code example 6: JavaScript event handler example

```
...
<script language="javascript" src="Customer.js"></script>
...
<tr><th>Zip:</th>
   <td>
     <input onblur="getZipData(this.value)" type="text" name="zip"/>
   </td>
</tr>
<tr><th>City:</th>
   <td><input id="city" type="text" name="city"/></td></tr>
...
```

The code in Code example 6 defines two inputs elements with the names zip and city. To the zip-field we have registered the JavaScript event handler onblur which handles the blur event thrown by the web browser when another area gets focus. The onblur attribute is special from other html attributes in that it can take JavaScript code as its parameter. The code “getZipData(this.value)” is a JavaScript call to the function getZipData in the associated JavaScript file Customer.js. This function is listed below in code listing 2. Calling a JavaScript function in this way is how one would initiate the core of an Ajax application. After this we are ready to start making the request back to a server.

Code example 7: The getZipData function

```
var xhr;
function getZipData(zipCode){
      xhr = new XMLHttpRequest();
      xhr.onreadystatechange=processZipData;
      xhr.open("POST", "zip.do", true);
      xhr.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
      Xhr.send("zipcode="+zipCode);
```

The function getZipData will create an object that is used for communication with the server. To do this we use the XMLHttpRequest object and create a new object named xhr in the code above. When the send method of this object is invoked a new connection is opened in another tread. The invocation returns immediately and continues executing the rest of the code in the getZipData function. This means that the JavaScript code does not wait for the servers response, instead we have to register a callback handler using the onreadystatechange property of the XMLHttpRequest object. The callback handler in Code example 7 is a function called processZipData. This function is in its turn called when the value of the readystate property of the XMLHttpRequest object changes. Processing the request in another thread and the use of callback handlers is what makes Ajax asynchronous and vital to the architecture of Ajax applications. Unfortunately the XMLHttpRequest object is not a part of the official JavaScript standard [23]. The code in Code example 7 will only work in browsers like Mozilla Firefox and Opera. This will affect the way the way Ajax applications are written and possibly cause standardization issues. We will come back to this in later chapters.

4.2 PROCESSING THE REQUEST AND RESPONSE.

In an Ajax application you would typically need to process user data on the server or fetch new data from the server. In the example in Code example 6 and Code example 7 we send back a zip code to the server that the user has entered in the web browser. In Code example 8 we show the web service that the open method of the XMLHttpRequest calls when sending the data.

Code example 8: Java servlet example

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException {

      String zipcode = request.getParameter("zipcode");
      String city = getCity(city);

      response.setContentType("text/xml");
      response.getWriter().write(
            "<zip>" +"\r\n" +
            "<postalAddress>" +"\r\n" +
            "<zipcode>"+ zipcode +"</zipcode> " +"\r\n" +
            "<city>"+ city +"</city>" +"\r\n" +
            "</ postalAddress>" +"\r\n" +
            "</zip>" +"\r\n
      );
```

The java servlet in Code example 8 shows how a web service in java can handle Ajax requests. The servlet extracts the data that the user sends from the request, and process the data by using the zip code to find the matching city. An xml document is then created and sent back. While this is done the readystate property of the XMLHttpRequest object are changing and each time the callback handler is called. When the response has been received properly, this property will have the value 4. The other values are listid in table 7.

Table 7: The possible values of the readyState property [41, page 55]

Value	State
0	Uninitialized
1	Loading
2	Loaded
3	Interactive
4	Completed

Code example 9: XMLHttpRequest, callback handler

```
function processZipData() {
    if (xhr.readyState == 4) {  // if the request is completed
        if (xhr.status == 200) {  // and if the responese completed
                                  // succesfully
            var CityNodes =
                  xhr.responseXML.getElementsByTagName("zipcode");
            var city = cityNodes[0].firstChild.nodeValue;
            document.getElementById("city").value = city;        }
        else {
            document.getElementById("Error").innerHTML = "Error";
        }
    }
```

The JavaScript code in Code example 9 is the callback handler that was previously registered to handle the response of the XMLHttpRequest object. It is the task of the client side JavaScript to navigate the returned xml-document and find the data that will be used to update the webpage.

The figure below illustrates how the components that makes up an Ajax application relates to each other. The figure is based on [24] but modified to show the complete Ajax process and adapted to fit the prototype

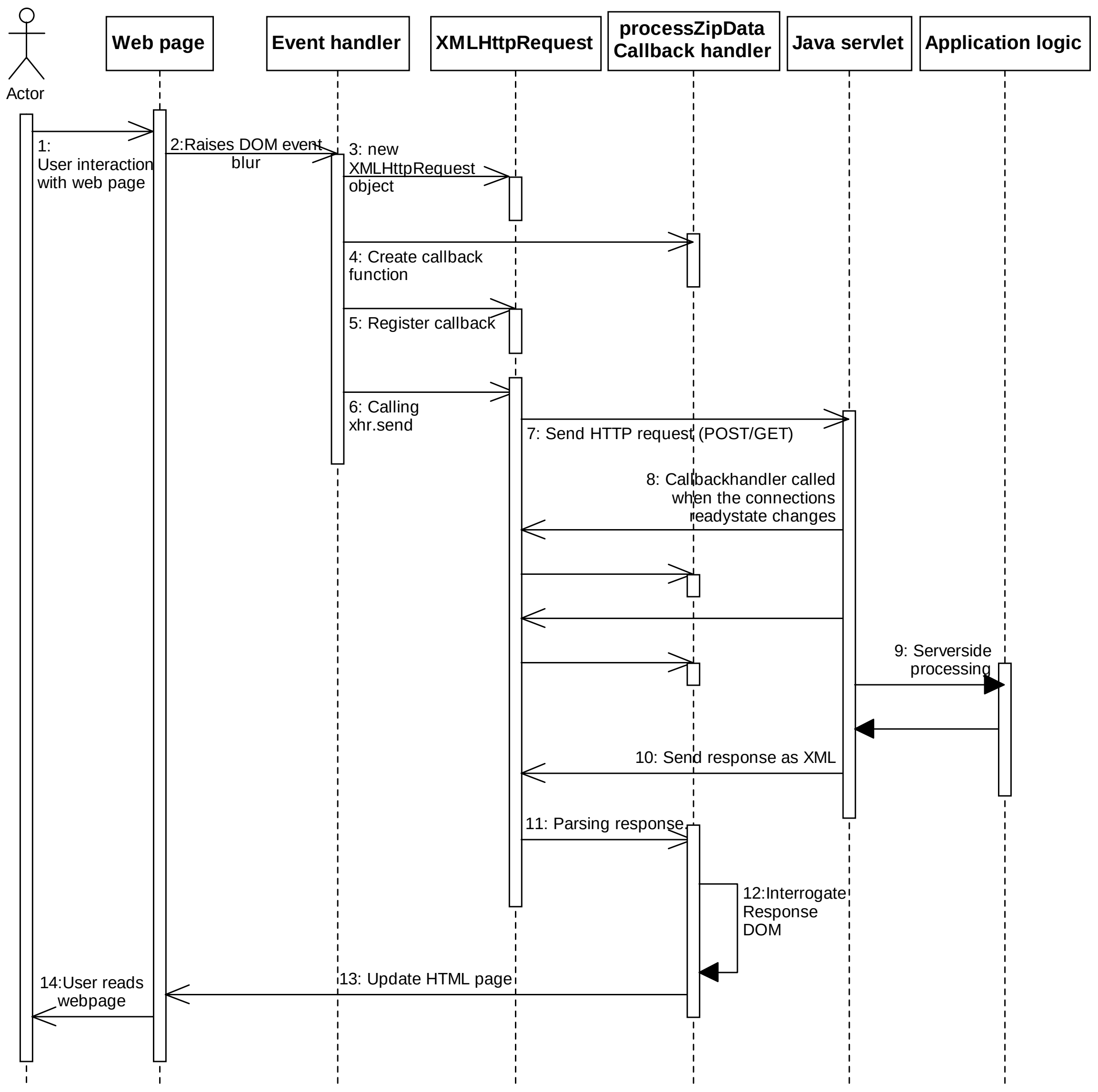

Actor
Web page
Event handler
XMLHttpRequest
processZipData Callback handler
Java servlet
Application logic
1: User interaction with web page
2:Raises DOM event blur
3: new XMLHttpRequest object
4: Create callback function
5: Register callback
6: Calling xhr.send
7: Send HTTP request (POST/GET)
8: Callbackhandler called when the connections readystate changes
9: Serverside processing
10: Send response as XML
11: Parsing response
12:Interrogate Response DOM
13: Update HTML page
14:User reads webpage

Figure 16: Detailed UML sequence diagram of the Ajax architecture.

4.3 DESIGN PATTERNS

Design patterns are an approach where one considers recurring design problems and tries to find a solution to these problems. In computer engineering a common approach is to break down functionality into classes. Patterns are often specific solutions to specific problems. According to E. Gamma; "Patterns are distilled from the experiences of experts. They enable you to repeat a successful design done by someone else. By doing so you can stand on the shoulders of the experts and do not have to re-invent the wheel" [25]

Implementing a pattern can help significantly to solve a programming problem in a particular context.

Many of the design patterns that exist for Ajax focus on the user interface part e.g. drag-and-drop where the user is allowed to rearrange elements on the page. Many of these patterns existed before the Ajax term was coined and the problems can be related back to DHTML. However, in this chapter we will take a closer look at patterns that are more important when creating Ajax applications.

4.3.1 The Facade design pattern

The Facade design pattern is about creating a class or function if you like so that a new class is offered to the programmer with a simpler interface. It can also be a way of abstracting away some code handling issues so that the programmer will have fewer objects to handle. In this section we take a closer look at how such a pattern can be applied in our prototype.

The most common non-gui pattern that we have found is the creation of the XMLHttpRequest object. As mentioned previously in this chapter this JavaScript object is not a standard part of JavaScript. It is up to the various browser makers to decide in which way they want to implement such functionality. For an Ajax developer this presents a problem because he can not any longer just create the needed object but has to consider what browser the user uses and how this browser has implemented the needed functionality.

In Code example 7 we showed how the XMLHttpRequest object was initialized and used. As discussed, the code would any work in some browser. To account for other browser we could instead implement an XMLHttpRequest object of our own which would return a usable object based on the users settings.

Code example 10: Creating a XMLHttpRequest object

```
function XMLHttpRequest() {
      var xmlreq = false;

      if (window.XMLHttpRequest) {
            // XMLHttpRequest object suited for Mozilla Firefox
            // and Opera
            xmlreq = new XMLHttpRequest();
      }

      else if (window.ActiveXObject) {
            try {
                  // XMLHttpRequest object suited for newer versions
                  // of Internet Explorer
                  xmlreq = new ActiveXObject("Msxml2.XMLHTTP");
      }
            catch (e1) {
                  try {
                        // XMLHttpRequest object suited for older
                        // versions of Internet Explorer
                        xmlreq = new
                             ActiveXObject("Microsoft.XMLHTTP");
                  }
                  catch (e2) {
                        // Unable to create an XMLHttpRequest
                        //  by any means
                        xmlreq = false;
                  }
            }
      }
      return xmlreq;
}
```

In

Code example 10 we show an implementation of a design pattern that allows the programmer to keep the code in Code example 7 unchanged. The new function in

Code example 10 will handle all the trouble of creating the proper object and returns it to the calling function. By using the facade design pattern the programmer no longer need to worry about the problem and can focus his effort on the actual task, communicating with the server. Using this pattern in the way we described resembles object oriented techniques something that simplifies the use of design patterns. If we would need our Ajax application to support other browsers this could easily be fixed bye replacing the function in

Code example 10 without changing the core of our application. Writing JavaScript in an object oriented way would also help write better structured and easier to understand applications.

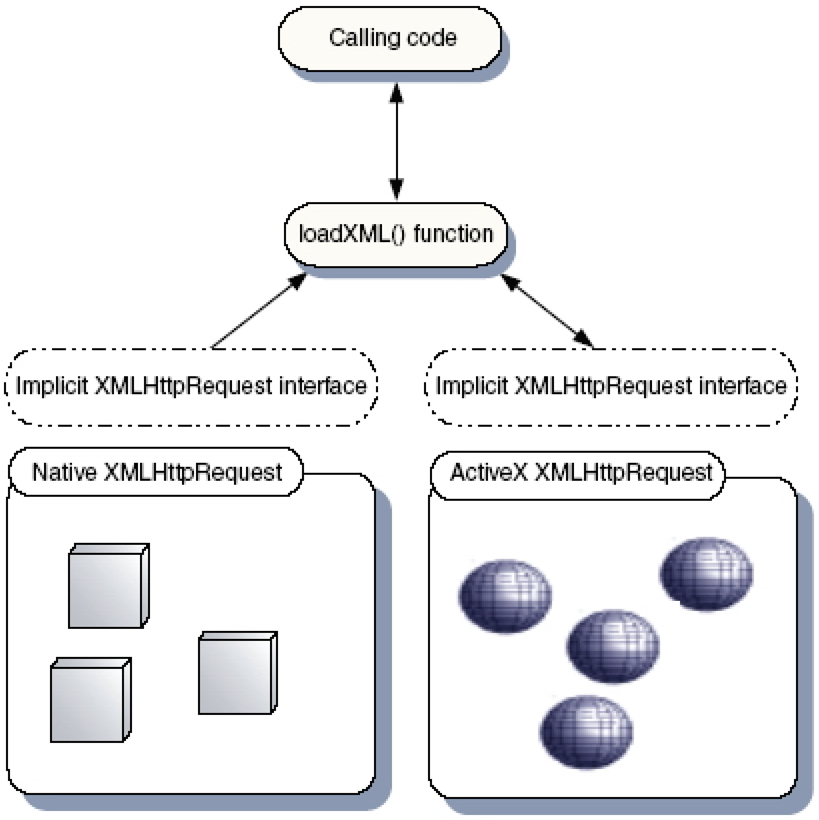


Figure 17: Facade Pattern diagram, [26, page 80]

4.3.2 The Model-View-Controller design pattern

In normal desktop applications the Model-View-Controller design pattern (MVC) is a common way of writing programs. As Ajax applications in time are expected to have many of the features that desktop applications have, applying the MVC pattern to Ajax becomes important. We will in this section see how MVC can be applied to the client side of an Ajax program.

The traditional MVC pattern divides responsibility for the application in three components, the View, the Controller and the Model. Each component is supposed to handle its own logic without interference from the other components.

If followed strictly this means that each part could be developed independently.

4.3.2.1 The View

The View component provides the user interface. The view is tasked with providing the user the possibility to talk with the Controller but also to update the user interface according to changes in the Model usually through communicated through the Controller.

In our prototype the View is the equivalent of the html-page sent to the user seen in Code example 5. One problem however with this code is that is mixes the View logic with the Controller logic. Separating these two could be a very good idea and would allow the designer of a web application to focus purely on what the application would look like. The html code as it is now, describes what event that is to occur and what function that should handle this event. This is part of the Controller logic.

4.3.2.2 The Controller

The Controller handles user input and is composed of event handlers. As we saw in the previous section our code mixes the View and the Controller because this html-tag: <input onblur="getZipData(this.value)" type="text" name="zip"/>

The onblur attribute tells the View which function that will handle the user event. Writing the code in this way requires that the designer is aware of the Controller components. Instead these could be separated by creating or adding the event handler dynamically in a separate JavaScript function, a Controller class if you like.

Code example 11: Separation of View and the Controller

```
* HTML
<input type="text" name="zip"/>

* JavaScript
Windown.onload=function(){
      var inputController = document.getElementByID("zip")
      inputController.onblur=getZipData(this.value)
}
```

4.3.2.3 The Model

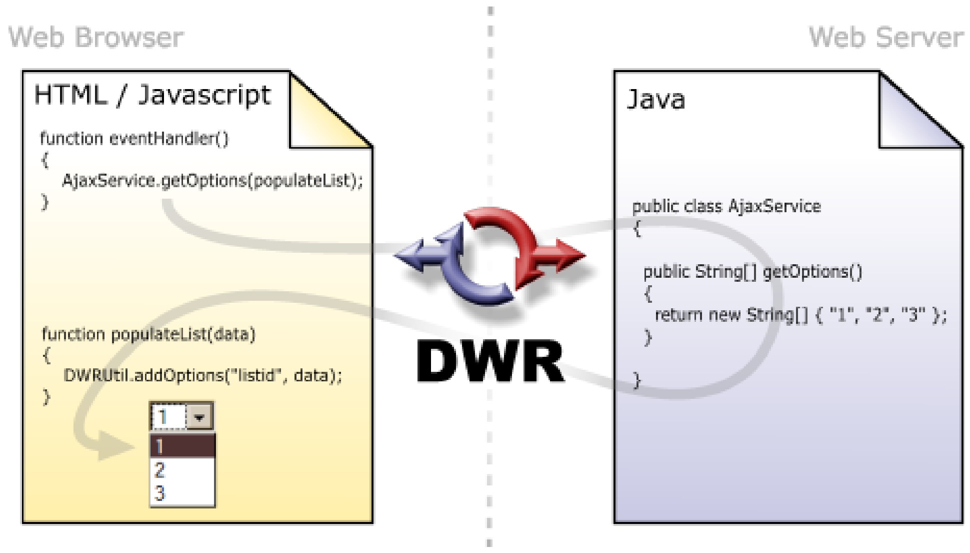
The Model is the component responsible for storing contents and state of the application. Another way of saying this is that the Model "consists of the business domain objects" [26, page 122]

Interacting with the server and passing information back to the Controller is another of the Models responsibilities. See Code example 9 for how we previously dealt with sending information to the server. What happens in Code example 6 is that we update the View based on changes in the Model.

4.4 FRAMEWORKS AND DIRECT WEB REMOTING (DWR)

As with design patterns, frameworks exist to simplify the programming job. The main difference is that a framework offers you methods or even a API based on the platform you are using, you then code using the provided framework API. With design patterns you will get the necessary code, or idea of how to program, so that you can modify it and adapt it yourself to your context.

A interesting framework that exist for Ajax on the Java platform is Direct Web Remoting, DWR. DWR aims to let the programmer use call java methods on the server from JavaScript functions in the browser. In order to use DWR it must first be installed on the web server that will host the web application. The framework will generate JavaScript files to be included on the client side. The generated JavaScript functions will be based on the java classes to the web application. The DWR framework will handle all the code required to communicate between the browser and web server, the developer can then use the java methods in the JavaScript functions as if it was java. DWR is a remoting framework as it handles all the calls to the remote web server. For the same reason DWR is known as a proxy-framework, as it creates the objects that communicate.



```
function eventHandler()
{
   AjaxService.getOptions(populateList);
}

function populateList(data)
{
   DWRUtil.addOptions("listid", data);
}
```

```
public class AjaxService
{
  public String[] getOptions()
  {
    return new String[] { "1", "2", "3" };
  }
}
```

Figure 18: Diagram of how DWR can be put to use [27]

5 WEB STANDARDS

In this chapter we will look at web standardization, and possible standardization issues connected to Ajax development. We will start with an introduction to the standardization in context of the World Wide Web history, and continue with a look at the Ajax approach, and its collection of fundamental technologies.

5.1 STANDARDIZING THE WEB

Since the World Wide Web architecture was proposed in 1989, and the first web browser written in NeXTStep (October-December 1990) saw the dawn of light, the term web has grown to be a term well known by most people. [28] The technological implementation of the web has over the past decades revolutionized the way literature and media can be presented to the end users.

It started with the proposal of the World Wide Web architecture, as shown in Figure 19, invented by the now famous Tim Berners-Lee. The WWW architecture was built on the combination of four basic ideas:

- Hypertext
- Resource Identifiers
- The Client-Server model
- Markup language

If we take a closer look at Figure 19 we see that the pink arrow shows the common standards: URL, and HTTP, with format negotiation of the data type. [28]

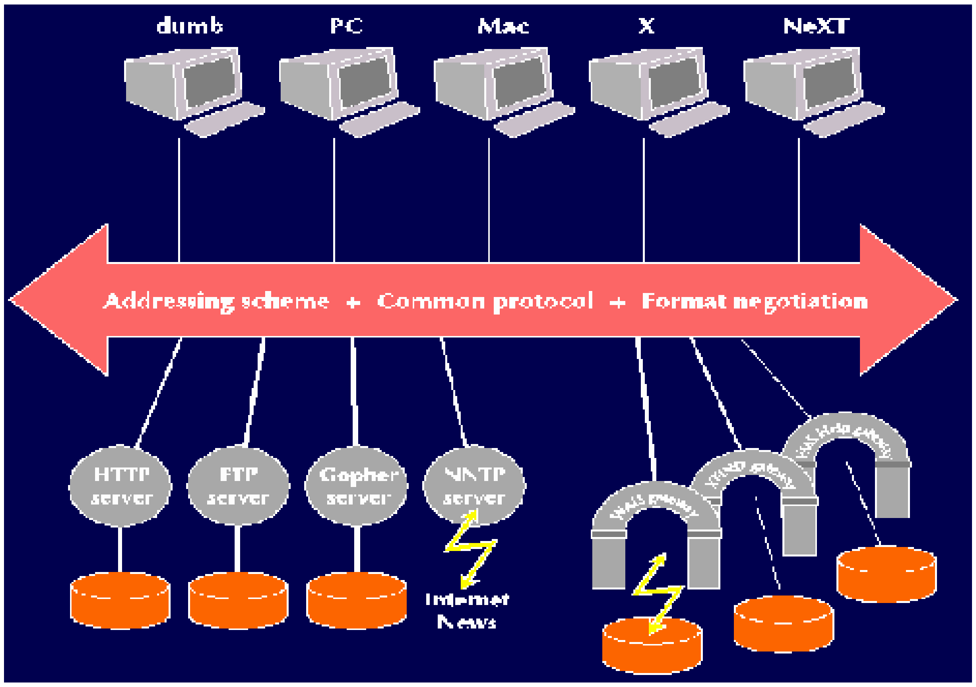


Figure 19: The original WWW architecture diagram, 1990 [28]

The four ideas evolved into the transfer protocol HTTP (Hyper Text Transfer Protocol) and URL (Uniform Resource Locator). Later on the IEFT (Internet Engineering Task Force) published HTML (Hyper Text Markup Language).

To begin with, web applications offered a static way to present text, pictures and sound. The hyperlink technology made it possible to navigate between web resources, and as various web applications were born, hyperlinks and search engines “webbed” it all together.

Supportive web browsers were developed, first the NCSA Mosaic browser, originally a UNIX based browser, but later ported to both Apple Macintosh and Microsoft Windows. Then, after the Mosaic success, Marc Andreessen, the leader of the Mosaic team at NCSA decided to give up his position, and formed the company Netscape Communications Corporation. The result was the well known Netscape Navigator, launched October 1994. [28] [29]

To ensure compatibility and agreement among industry members in the adoption of new standards, Berners-Lee the same year founded the W3C (World

Wide Web Consortium). Since then both Berners-Lee and W3C has had a significant role in defining web standards for the web.

Netscape Navigator was quickly challenged by Microsoft and their Internet Explorer, and resulted in years of war between the two major competitors. The battle was fought on fields where web standardization differences were used as bullets. This resulted in proprietary extensions to the HTML language, and they both launched technologies supported only by their own browser. This struggle did however result in the technologies CSS (Cascading Style Sheets) and JavaScript.

The introduction of JavaScript in the Netscape Navigator, December 1995, gave developers new possibilities developing their web applications. JavaScript was a prototype-based scripting language, and its syntax was loosely based on C. JavaScript is dependent on the host environment, or in other words, browser compatibility. JavaScript offered ways to interact with the DOM (Document Object Model) of the web applications, and made it possible to execute script commands that could alter the presentation of the web content. [30]

As expected, Microsoft did not sit on the fence for long. In 1996, with the Internet Explorer 3 they countered Netscape by launching Jscript, and CSS. CSS was originally proposed by Håkon Wium Lie, in 1994, and later standardized by the W3C. CSS made it possible to add defined styles and layouts to web applications, while Jscript offered ways to alter the DOM similar to JavaScript. The combination of CSS and Jscript was a golden combination for Microsoft. [31]

Netscape tried to counter CSS by launching their own style sheet language, JSSS, (JavaScript Style Sheets) in 1996, but gave this up for CSS as it was never accepted as a formal standard by W3C.

A little bit earlier, during 1995, Sun Microsystems developed the Java language, and introduced a new technology, the Java Applet for the web community. The Java Applet was developed to provide interactive features to web applications. Sun developed the JVM (Java virtual machine), and the idea was that every web browser should have JVM installed, and thus support the

Java Applet technology. This new approach made it possible to make interactive applications run via the web browser. The Applet was built to run in a sandbox in the web browser, secure from interacting with the local file system.

Making their own version, Microsoft for a while shipped their Internet Explorer with MSJVM (Microsoft Java Virtual Machine), and added some extra functionality into it. The MSJVM supported the Java Applet technology, but the additional functionality was closed, and thus not possible to run using JVM. Sun sued Microsoft, and the MSJVM project was frozen.

To make one historical leap in time, Microsoft won the battle, and Netscape Navigator slowly faded out of the picture. A resurrection of the Navigator is today however alive in form of the Mozilla Firefox. [32]

5.2 WHY STANDARDIZATION

If we look back at the history of the web in section 5.1.1, we first of all see that the World Wide Web is built on several competitive and collaborating technologies. It all started with the initial concept, the WWW and the fundamental ideas, the HTTP protocol, the HTML scripting language, URL, and the client – server model.

The entry of the competitive web browsers Internet Explorer and Netscape Communicator introduced new technologies, and compatibility problems. To fight each other, they used browser adapted technology as ammunition. The result was that internet users had to either have both Netscape and Explorer to be able to browse different web pages.

Dealing with such issues, the W3C, with Tim Berners-Lee started working with the web technologies to form common standards. The standards produced defined a fundamental concept of the technology being standardized. The standards were open, and developers could easily access the standardization documents.

The work of W3C, and others, has made it possible to browse the internet using a web browser by choice. There are still some compatibility issues that are not solved; the .NET framework from Microsoft is one of them. But if we look back at the 90'thies we se that progress has been achieved. As we now see the introduction of new technologies, new standardization issues evolve.

5.3 STANDARDIZING AJAX

Throughout this report we have introduced Ajax as both a new technology and a technological approach. If we again analyze Ajax, we see that the fundament is other independent technologies, see Figure 7. These technologies are together forming an approach that has been baptized Ajax. [7]

So what standardization issues does Ajax rise? To be able to answer this we will have to look at the fundamental technologies, JavaScript, XML, CSS, DOM and XMLHttpRequest. We refer to section 2.3.2 for additional information about the technologies.

5.3.1 JavaScript

JavaScript was first present as Jscript in Internet Explorer, and JavaScript in Netscape Navigator. The introduction of the standard ECMA-262 [30] composed by ECMA (The European Computer Manufacturer's Association) resulted in a standard with elements of both Jscript and JavaScript. The latest editions of the Internet Explorer, Mozilla Firefox and Opera are fully compliant with the ECMA-262 standard.

5.3.2 XML

XML was standardized the 10th of February 1998 as the XML 1.0 recommendation from W3C. This has later evolved into the 1.0 (Third Edition), and XML 1.1 the 4th of February 2004. Using XML to structure the data communicated in Ajax applications seem to be a good choice when considering eventual standardization issues.

5.3.3 CSS

CSS is at this point standardized by the CSS level 1 standard. The CSS level 2, revision 1, is still a candidate W3C recommendation, and CSS level 3 is under development. [33] Even thou the core Ajax communication is not reliable on CSS to work, we see that CSS can add visual enhancement by creating structured style sheets. We also notice that the technology has been around for a while, and is steadily evolving.

5.3.4 DOM

DOM was standardized as the Document Object Model level 1, the 10th of October 1998. The latest recommendation is the Document Object Model level 3, of the 7th April 2004. The DOM specification is supported in all the major web browsers.

5.3.5 XMLHttpRequest

As we briefly mentioned in section 2.3.2.3, the W3C has began working on a draft to define a basic recommendation for the XMLHttpRequest object. In their introduction section they introduce the standardization problem like this:

"The XMLHttpRequest object is an interface exposed by a scripting engine that allows scripts to perform HTTP client functionality, such as submitting form data or loading data from a remove Web site.

The XMLHttpRequest object is implemented today, in some form, by many popular Web browsers. Unfortunately the implementations are not completely interoperable. The goal of this specification is to document a minimum set of interoperable features based on existing implementations, allowing Web developers to use these features without platform-specific code. In order to do this, only features that are already implemented are considered. In the case where there is a feature with no interoperable implementations, the authors have specified what they believe to be the most correct behavior." [34, 1.Introduction]

A final specification for the XMLHttpRequest object will perhaps solve eventual compatibility problems, as it has with the rest of the fundamental Ajax technologies. But defining a specification is in our view not enough. The real issue is what the major web browser companies will do to follow the recommendation.

If we look at the current market situation we see that the Internet Explorer, with its versions IE7, IE6 and IE5 April 2006 had a marked share of 62,3%, see Figure 20. On the 2nd place we find the Mozilla based Firefox with 25,7% of the total marked share. [35]

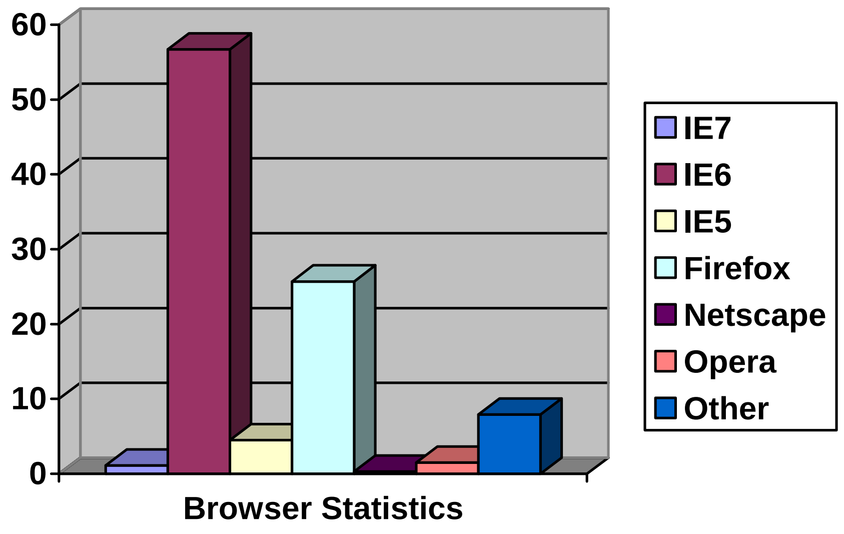


Figure 20: Browser statistics April 2006

The statistics show that there are two dominant web browsers on the marked, Internet Explorer and Mozilla Firefox. Based on the major share represented by the IE browsers we believe that possible standardization issues attached to Ajax development first and foremost will rely on what decisions Microsoft make in the latter stages of the IE7 development phases.

Looking at the feature list provided by Microsoft, we see that they indeed have decided to improve the browsers handling of the XMLHttpRequest object in the IE7 browser. Under the section improved platform and manageability in IE7 feature list, they have a defined feature improvement on Ajax support:

"IE7 improves the implementation of the XMLHTTP Request as a native JavaScript object for rich AJAX-style applications. While Internet Explorer 6 handled XMLHTTP requests with an ActiveX control, Internet Explorer 7 exposes XMLHTTP natively. This improves syntactical compatibility across different browsers and allows clients to configure and customize a security policy of their choice without compromising key AJAX scenarios."

Based on the information we have introduced in this chapter we conclude that Ajax introduce a challenge to current web standards by using a non standardized technology, the XMLHttpRequest object to make request/responses from and to the server. If we on the other hand should consider the future of Ajax and possible standardization issues, we see that this is evolving in a positive direction. W3C is working on a specification, and IE7 will be launched with improved Ajax support by releasing the XMLHttpRequest object from ActiveX. Our only concern is attached to Microsoft's participation in the making of the XMLHttpRequest recommendation. As we reviewed the list of participating organizations and authors of the W3C working draft, we could not find Microsoft on the list. [34, B]

Our suggestion to Ajax developers, based on this information would to monitor the progress done in the standardization process of the XMLHttpRequest object to avoid future lock-in and compatibility situations.

6 RESULTS

In this chapter we will introduce the results received from the literature study, and prototype development.

6.1 RESULTS, TDD AND AJAX CHALLENGES

In Section 3 we introduced the work we have done to identify challenges attached to use of TDD when developing Ajax enabled web applications. The results we have gained from this process are the identification of three main challenges, and two possible solutions.

6.1.1 The main challenges

In Table 6 we present the three main challenges to the use of TDD in an Ajax development phase. The three main challenges we have identified are:

1. Ajax is heavily dependent on JavaScript, and requires a testing framework that is able to test JavaScript.
2. Ajax introduces asynchronous communication between the web client and the web server. In a TDD – Ajax scenario this cause problem connected to unit testing of the JavaScript code.
3. The asynchronous communication introduces similar challenge in the integration and system test phase.

6.1.2 A testing framework for JavaScript

The first of our identified challenges was to find a proper testing framework that could help us write unit tests in JavaScript. The framework we decided to use was JsUnit. Using this framework we were able to develop unit tests to test our JavaScript code. JsUnit also offered a plug-in to the development platform Eclipse.

6.1.3 Asynchronous request/response in the unit test phase

The identification of the asynchronous request/response challenge resulted in the two following suggested solutions.

6.1.3.1 The waitForAWhile solution

Our first solution is shown in Section 3.1.2.1. The solution is based on the usage of a script, waitForAWhile.js, see Code example 3: The waitForAWhile.js script. The waitForAWhile script makes it possible to insert pauses in a test case, and define the length of the pause in milliseconds. In

Code example 4 we show the usage of this script in a test case.

We managed to test the request/response from the server using the waitForAWhile solution. The results was however unstable, and resulted in occasional test failures.

6.1.3.2 The mock object approach

The second solution is shown in Section 3.1.2.2. The solution is based on the theory of using mock objects to simulate the behavior of another object.

We identified the XMLHttpRequest object as the main challenge in the unit test phase, and tried to see if it was possible to make a mock object of the XMLHttpRequest object.

The result was not successful. We found JavaScript not suitable for mock object creation. In Figure 15 we have presented a possible mock object implementation of the XMLHttpRequest object. The lack of object oriented support in JavaScript made it difficult to develop the mock object.

6.1.4 A test framework in the integration test phase

In Section 3.1.3 we present a possible solution attached to the integration test phase of a software project.

We see that the challenges attached to unit, integration and system phases are related to asynchronous request/response. Our solution is to use a testing framework that makes it possible to test this communication.

The result from this investigation is to use the Selenium framework in the integration test phase. Selenium is possible to extend with the possibility to test asynchronous request.

6.2 RESULTS, AJAX ARCHITECTURE AND DESIGN PATTERNS

Ajax has a more complex architecture then ordinary web applications, it asynchronous nature makes it harder to know when the web client will receive responses from the web server. In chapter 4 we showed how an Ajax application could be structured including the implementation on the server side and how the data transfer relates to the architecture.

We have also seen the benefits of applying design patterns. Using design patterns would help structure applications and add a new level of abstraction so that the programmer can solve certain problems independently without changing much of the remaining code.

Various frameworks exists that could help the developer. We have particularly looked at one framework that allows the programmer to write the JavaScript code for the client in a Java-manner. Frameworks can be off great help but can hide parts of the architecture and make changes in the code dependent on the framework.

6.3 RESULTS, AJAX AND WEB STANDARDIZATION

In Chapter 5 Web standards we have presented the standardization status of the technologies Ajax is built upon. The studies performed show that all technologies except from the XMLHttpRequest object are standardized and well supported in the major web browsers. The XMLHttpRequest object is currently being standardized, and the W3C is organizing this process.

We have also identified that Internet Explorer 7 will include better support of the XMLHttpRequest object.

7 DISCUSSION AND FUTURE WORK

In this chapter we will discuss the results from Chapter 6. To form the basis of the discussion we will use the questions defined our thesis definition from Section 1.1:

1. What challenges does Ajax introduce to TDD?

2. Is there a need for Ajax adapted tool support to address these challenges?

3. What kind of architecture do Ajax applications have and which design-patterns are important to consider.

4. How does Ajax relate to current and future web/Java standards?

7.1 DISCUSSION, AJAX AND TDD CHALLENGES

In this section we will discuss the results introduced in Section 6.1.

7.1.1 The main challenges

In Table 6 we have presented the main challenges we identified attached to TDD and Ajax. Our motivation behind them is question 1 and 2 from our thesis background. Based on literature study and prototype development we identified three challenges.

7.1.1.1 Selecting a test framework for JavaScript

In our literature study we introduce TDD and the need for a unit testing framework in Section 2.5. In 2.3.2.1 we introduce JavaScript as one of the building blocks in the Ajax approach.

Based on our literature study, we started looking for a unit testing frameworks for JavaScript. The result was the unit testing framework JsUnit.

The reason behind the selection of JsUnit is the possibilities the framework offered. The unit test cases were intuitive and easy to build and the test suite was integrated in the Eclipse platform via a plug-in.

If we consider the alternatives, Script.aculo.us and J3Unit, we chose JsUnit based on the maturity status, the simplicity of unit test creation, and the tool support provided via the Eclipse plug-in. Script.aculo.us was compared to JsUnit not as easy to use, and was outmaneuvered by the simplicity and the Eclipse plug-in. J3Unit looked was a bit more appealing, but its beta status, and lack of tool support made JsUnit our choice.

7.1.1.2 The waitForAWhile solution

As we tried to test our JavaScript, we encountered a challenge in the asynchronous request/response communication between the web client and the server. In our first solution to solve this problem, we tried to force the JavaScript code to wait for a chosen time, and then see if we had received the response from the server.

This approach turned out to be unstable. In our literature study of TDD in Section 2.5, we see that unit tests should run fast with short setups, run times and break downs. In a large scale project, with several asynchronous request/responses the waiting time will increase for each one, and make testing a time consuming operation. Our unstable results using the waitForAWhile solution made us look for other solutions.

7.1.1.3 The mock object approach

In our first solution we have mentioned that one of the main factors of unit testing is that they can be run quickly. In our waitForAWhile solution, tests would increase the run time for each asynchronous request/response test included. As we started to look for other solutions we discovered a technique called mocking. The technique was built on the theory of making objects that could simulate the behavior of other objects.

After looking at existing mock objects we saw that they were built using the object oriented possibilities of languages like Java, C# and other object oriented languages. Making mock objects in JavaScript following defined mock object patterns was not achieved. We identified the object to be mocked to be the XMLHttpRequest object. The creation of a mock object of the XMLHttpRequest object will theoretically reduce the run time on unit tests as the tests not will have to wait for the server to respond physically to the asynchronous request.

In Figure 15 we have visualized a possible mock object for the XMLHttpRequest object. The figure present what was not possible for us to make. We could not inherit and wrap the XMLHttpRequest in JavaScript. The only thing we could do was to visualize the problem.

7.1.1.4 A test framework for integration testing

As integration testing is not unit testing, we did no real test experiments in this phase. We did however try to cover this phase in our literature study in Section 2.4.2.2, and suggest a framework that could be used to test Ajax applications on the Java platform.

We have suggested Selenium as a test framework in the integration phase. We encountered Selenium when searching for unit testing frameworks, and noticed its ability in the integration and system test phases.

7.2 AJAX ARCHITECTURE AND DESIGN PATTERNS

There are many ways an Ajax application could be written. We have described in detail one possibility that we think would fit most cases. Even if the approach could be slightly different our prototype describes every part that a working Ajax application needs. The various components could be changed, like listening on other or more events, parse more advanced XML documents, or even other data formats and so on. The same elements would however still be required in one form or another. The Architecture can become even more complicated when using frameworks. However if a framework is used, they may provide a simpler interface and a simpler abstraction level. We would warn against using too large frameworks as replacing part of the code could become much more challenging. We believe clearly defined smaller frameworks is favorable as fewer part of the code would depend on a specific framework.

The same is to a certain degree true for design patterns. Design patterns like the Facade Pattern helps abstract a problem to separate functions so that the programmer will have fewer objects to deal with. It could be better to implement smaller design patterns that solve a specific problem rather then patterns that try to solve too much.

JavaScript programming is often overlooked when “serious” programming is discussed. With the drive towards Web 2.0 this is no longer an issue that can be avoided. Having a clear structure for the web application and using design patterns, frameworks and other object oriented techniques could be just what future web applications need.

7.3 AJAX AND WEB STANDARDIZATION

We identified that there was standardization issues connected to the implementation of the XMLHttpRequest object in different web browsers. We tried to approach this problem by analyzing the standardization status of XMLHttpRequest, and by looking on future changes in Internet Explorer. Based on this information we have suggested Ajax developers to monitor the XMLHttpRequest object standardization closely to avoid compatibility and lock-in situations.

7.4 FUTURE WORK

In this report we have identified the need for an XMLHttpRequest mock object. As we have not been able to make our own implementation we see that further work could be done in this area. The challenge will thus be how to make an XMLHttpRequest mock object for JavaScript.

We would also recommend that different frameworks is analyzed and compared with each other to see what other benefits they can contribute and what weakness/strengths that the various approaches has.

It could also be interesting with a larger Ajax application and analyze how the Model on the client side interacts together with the Model on the server side.

In a common web application much of the processing is done on the server but in a large full scale Ajax application some of this processing could happen on the client side so a closer look at how this could be coordinated would be of interest.

8 CONCLUSION

We have during this thesis shown the fundamental theory behind both Ajax and TDD. As a response to the questions asked in the thesis background we have introduced the main challenges connected to the usage of TDD on Ajax enabled web applications and suggested tools to address these challenges. The report also introduces a challenge connected to mock implementation in JavaScript.

In the report we have introduced the fundamental architecture of Ajax applications attached to web development on the Java platform. We have also provided some examples on usage of Design Patterns, and discussed how usage of Design Patterns can improve the design.

Dealing with a new technology we have also identified a standardization issue connected to the XMLHttpRequest object. As a response to our study of this issue, we have suggested developers to monitor the future progress on this matter closely to prevent eventual compatibility and lock-in situations.

APPENDIX A GLOSSARY & ABBREVIATIONS

Action Script	- Script language developed by Macromedia
Ajax	- Asynchronous JavaScript and XML
Applet	- Java based program that runs in a web browser
ASP	- Application Service Provider
CSS	- Cascading Style Sheets
DHTML	- Dynamic HTML
DOM	- Document Object Model
DOT.NET	- Development platform from Microsoft
ECMAScript	- The official name/standard for JavaScript
Flash	- Macromedia based browser plug-in
Flex	- RIA Presentation Server from Macromedia
HMTL	- Hypertext Markup Language, describes web pages
HTTP	- Hypertext Transfer Protocol
IIOP	- Internet Inter-Orb Protocol
ISP	- Internet Service Provider
J2EE	- Java Platform, Enterprise Edition
JRE	- Java Runtime Environment
JavaScript	- Script language used to manipulate DOM
JSF	- Java Server Faces
JSP	- Java Server Pages
JSRS	- JavaScript Remote Scripting
JsUnit	- a Unit Testing framework for JavaScript
Junit	- a Unit Testing framework for Java
Mock objects	- “false” objects that simulate “real” objects
MVC	- Model-View-Controller Design Pattern
Pearl	- A development platform
PHP	- A development platform / language for web pages.
Python	- A development platform

RIA	- Rich Internet Applications
RSLite	- Remote Scripting Lite
Ruby	- A development platform
TDD	- Test Driven Development
W3C	- The World Wide Web Consortium
XML	- Extensible Markup Language

APPENDIX B REFERENCES

[1] "Bruk av IKT i husholdningene, 2005" *Statistisk Sentralbyrå*, http://www.ssb.no/emner/10/03/ikthus/
(Current May 29, 2006)

[2] "Internet usage in the EU25 in 2005" *Eurostat*, http://epp.eurostat.cec.eu.int/pls/portal/docs/PAGE/PGP_PRD_CAT_PREREL/PGE_CAT_PREREL_YEAR_2006/PGE_CAT_PREREL_YEAR_2006_MONTH_04/4-06042006-EN-AP.PDF
(Current May 29, 2006)

[3] Noda T. and S. Helwig (2005) "Rich Internet Applications, Technical Comparison and Case Studies of Ajax, Flash, and Java based RIA," *University of Wisconsin-Madison*, http://www.uwebi.org/docs/final_1.pdf
(Current May 29, 2006)

[4] Duhl, J (2003) "Rich Internet Applications," *IDC*, http://download.macromedia.com/pub/solutions/downloads/business/idc_impact_of_rias.pdf
(Current May 29, 2006)

[5] Steven, K "Java Web Start" *IBM*, http://www-128.ibm.com/developerworks/java/library/j-webstart/
(Current May 29, 2006)

[6] "Remote Scripting" *Ashleyit.com*, http://www.ashleyit.com/rs/main.htm
(Current May 29, 2006)

[7] Garrett, J. J. (2005) “Ajax: A New Approach to Web Applications,” *adaptivepath.com*, http://www.adaptivepath.com/publications/essays/archives/000385.php (Current May 29, 2006)

[8] “Cascading Style Sheets” *wikipedia.org*, http://en.wikipedia.org/wiki/Cascading_Style_Sheets (Current May 29, 2006)

[9] Crane D., E. Pascarello and D. James (2006) “Ajax in Action” Manning Publications Co., Greenwich, CT 06830.

[10] “CSS 2.1 Spesification” *w3c.org*, http://www.w3.org/TR/2006/WD-CSS21-20060411/ (Current May 29, 2006)

[11] “Agile Software Development,” *wikipedia.org*, http://en.wikipedia.org/wiki/Agile_software_development (Current May 29, 2006)

[12] Beck, K. “Simple Smalltalk Testing: With Patterns” *xprogramming.com*, http://www.xprogramming.com/testfram.htm (Current May 29, 2006)

[13] Ellen, G. “Managing your way through the Integration and Test Black Hole” *methodsandtools.com*, http://www.methodsandtools.com/archive/archive.php?id=13 (Current May 29, 2006)

[14] Ambler, S. W. “Introduction to Test Driven Development” *agiledata.org*, http://www.agiledata.org/essays/tdd.html (Current May 29, 2006)

[15] Ambler, S. W. “Agile Modeling (AM) Principles v2” *agilemodeling.com*, http://www.agilemodeling.com/principles.htm (Current May 29, 2006)

[16] “Agile Draw, semantics” *agiledraw.org*, http://agiledraw.org/index.php/Main/Semantics (Current May 29, 2006)

[17] “JsUnit Introduction” *jsunit.net*, http://www.jsunit.net/ (Current May 29, 2006)

[18] “Script.aculo.us: test.unit.runner” *wiki.script.aculo.us*, http://wiki.script.aculo.us/scriptaculous/show/Test.Unit.Runner (Current May 29, 2006)

[19] “J3Unit Overview” *j3unit.sourceforge.net*, http://j3unit.sourceforge.net/ (Current May 29, 2006)

[20] Gheorghiu, G. “Ajax testing for selenium using waitForCondition” *agiletesting.blogspot.com*, http://agiletesting.blogspot.com/2006/03/ajax-testing-with-selenium-using_21.html (Current May 29, 2006)

[21] “Open QA: Selenium” *openqa.org*, http://www.openqa.org/selenium/ (Current May 29, 2006)

[22] “JavaScript Delay/Wait/Pause routine” *sean.co.uk*, http://www.sean.co.uk/a/webdesign/javascriptdelay.shtm (Current May 11, 2006)

[23] Crane D., E. Pascarello and D. James (2006) “Ajax in Action” Manning Publications Co., Greenwich, CT 06830.

[24] "Standard ECMA-262, ECMAScript Language Specification" ecma-international.org, http://www.ecma-international.org/publications/standards/Ecma-262.htm
(Current May 29, 2006)

[25] "Build dynamic Java applications" ibm.com, http://www-128.ibm.com/developerworks/library/j-ajax1/
(Current May 29, 2006)

[26] "How to Use Design Patterns" artima.com, http://www.artima.com/lejava/articles/gammadp.html
(Current May 29, 2006)

[27] "DWR: Easy AJAX for JAVA" getahead.ltd.uk, http://getahead.ltd.uk/dwr/overview/dwr
(Current May 29, 2006)

[28] Duhl, J (1996) "The World Wide Web: Past, Present and Future" *W3.org*, http://www.w3.org/People/Berners-Lee/1996/ppf.html
(Current May 29, 2006)

[29] "World Wide Web" *wikipedia.org*, http://en.wikipedia.org/wiki/Web_browser
(Current May 29, 2006)

[30] "JavaScript" *wikipedia.org*, http://en.wikipedia.org/wiki/JavaScript
(Current May 29, 2006)

[31] "Jscript," *wikipedia.org*, http://en.wikipedia.org/wiki/JScript
(Current May 29, 2006)

[32] "Web browser" *wikipedia.org*, http://en.wikipedia.org/wiki/Web_browser (Current May 29, 2006)

[33] "CSS Specifications" *w3c.org*, http://www.w3.org/Style/CSS/#specs (Current May 29, 2006)

[34] "The XMLHttpRequestObject, W3C Working Draft 05 April 2006" *w3c.org*, http://www.w3.org/TR/XMLHttpRequest/ (Current May 29, 2006)

[35] "Browser Statistics" *w3c.org*, http://www.w3schools.com/browsers/browsers_stats.asp (Current May 29, 2006)

[36] North, D "Test driven development is not about testing" *java.sys-con.com*, http://java.sys-con.com/read/37795.htm?CFID=9725&CFTOKEN=D0C8A42E-1472-7BF3-2F81AD9E5BCC6934 (Current May 29, 2006)

[37] Walnes, J. et al. (2004) "Java Open Source Programming, With XDoclet, JUnit, WebWork, Hibernate" Wiley Publishing, Inc., Indianapolis, Indiana.

[38] North, D "Test driven development is not about testing" *java.sys-con.com*, http://java.sys-con.com/read/37795.htm?CFID=9725&CFTOKEN=D0C8A42E-1472-7BF3-2F81AD9E5BCC6934 (Current May 29, 2006)

[39] “Agile Draw, about” *agiledraw.org*, http://agiledraw.org/index.php/Main/About (Current May 29, 2006)

[40] Brown M. and E. Tapolcsanyi “Mock Object Patterns” *jerry.cs.uiuc.edu*, http://jerry.cs.uiuc.edu/~plop/plop2003/Papers/Brown-mock-objects.pdf (Current May 29, 2006)

[41] Gethland J, Galbraith B., Almaer D. (2006) “Pragmatic Ajax” The Pragmatic Bookshelf, Dallas Texas