The effect of Ajax on performance and usability in web environments

Y.D.C.N. op 't Roodt, BICT

Date of acceptance: August 31st, 2006

One Year Master Course Software Engineering

Thesis Supervisor: Dr. Jurgen Vinju
Internship Supervisor: Ir. Koen Kam
Company or Institute: Hyves (Startphone Limited)

Availability: public domain

Universiteit van Amsterdam,
Hogeschool van Amsterdam,
Vrije Universiteit

This page intentionally left blank

Table of contents

1 Foreword

First of all I would like to thank Dr. Jurgen Vinju for his support during the last year, during the writing of this thesis and his excellent insights. Furthermore I would like to thank Hans Dekkers, Jan van Eijck and Paul Klint for making this master course possible. Thanks also go out to Eric Smalley for correcting my English.

I have been a user of the Internet for more than 10 years now and have seen all the browsers, languages, protocols come and go. The dotcom bubble and all the Internet start-ups that came with it fascinate me. Currently a new 'bubble' that goes by the name of Web 2.0 is emerging on the Internet. One of the key factors in Web 2.0 is the use of Ajax. So why not combine my personal interest with research?

2 Motivation

In the last 10 years, the Internet has tremendously gained in popularity. In the same time, the application model of websites has changed. At first the web pages were very static, followed by dynamic web pages, database driven websites and online stores. Many technologies have contributed to this development, server side languages (Java, PHP, ASP), open source databases (MySQL, PostgresQL) and client technologies (JavaScript, Flash, Java). Most of these technologies made it possible to unlock vast amounts of information to the user, but the application model stayed largely the same.

Currently a new change is taking place, offering better usability and productivity to end-users. Ajax is the new use of existing browser technologies that makes a new kind of application model possible. Ajax is already widely being used on websites, although most of them are smaller, but nonetheless important. They are the websites that drive the technology rush on the Internet. It cannot be said yet that Ajax is proven technology but it is already used for varying reasons. Ajax will be discussed in detail in chapter 4.

2.1 Tasks and sources

The goal of the research is to determine the effect of using Ajax, in terms of usability and performance, in a very large web environment. Cross browser compatibility and integration into existing code are very important. The emphasis lies with Ajax frameworks and not with other Rich Internet Application (RIA) frameworks such as Adobe Flex and OpenLaszlo. Although RIA applications can be made with an Ajax framework they have a different philosophy. A part of the research covers the availability of Ajax frameworks and assessing their qualities based on predefined criteria.

The case study concerns Hyves, The Netherlands' largest social networking website (http://www.hyves.nl). Provided here is some data to get an impression of size: over 2 million unique users, over 12 million photos and videos, 10 million page views per day and more than 180 web, database and storage servers. The website has an enormous load, partly because of the amount of page views and partly because of the highly dynamic nature of the pages. Any saving in load on such a scale is very valuable, therefore the effect of using Ajax in general will be measured. Finally, another point of interest is improvement in usability of websites when using Ajax. The better the usability the more time users will spend on your web site and the more pages they will view, which is good form a commercial point of view. The theory is that both usability and server and network load can be improved by using Ajax and that there is a relation between the two.

An introduction to Ajax is given by [6], describing the history, and using [17] to explain the principles of Ajax. JavaScript is an important factor in Ajax frameworks and applications. All important JavaScript knowledge and information about the browser wars and the current state are discussed in [2]. Information on performance related to content manipulation is provided by [3] with performance benchmarks in [10]. An important reason to implement an Ajax framework is to improve the user experience. [1] Describes the difference between sovereign and transient applications (see section 4.2) and the direction that Ajax applications are headed in. Sources [16] and [21] discuss the user expectations regarding response times in software applications and the effect that different response times have on the way users react. These sources also support the expected results of the positive effect that Ajax has on usability.

The implementation requirements are mostly determined by Hyves. Because JavaScript is such an important factor in Ajax applications, an Ajax framework will be selected to intelligently deal with JavaScript and its shortcomings. A framework is software that takes care of low level implementation details and aids developers to write software at a higher abstraction level. A more complete definition of a framework and the available Ajax frameworks will be discussed in chapter 5.
JavaScript is notorious for it's debugging and testing. Although I have found [8] as a unit-testing framework, this will remain as future work. Event handling in JavaScript is mostly done in an obtrusive way and is considered unwanted from a developers view. Some frameworks introduce Aspect Oriented Programming like solutions to event handling, see [19]. Using event observers the event handling can be separated from the HTML code. Design patterns for Ajax are described in [1] and [4], as standard work on design patterns [5] will be used. Licensing and continuity of the product will be assessed on a rather subjective basis.

Some browser functionality will be broken when creating Ajax applications. The issues and their possible solutions are discussed in [9]. Another concern is browser compatibility and the ongoing implementation of incompatible browser models. [7] Gives some insight in the current situation.

The frameworks possibly useful in the case study will be discussed from section 5.7 onward. The most promising frameworks for the case study will be evaluated using [4], [18] and [20] in chapter 6. Application prototypes will have to be developed for Hyves using an Ajax framework. This will concern improvements of existing functionalities, in order to make a comparison between the two versions. During development best practices will be extracted from leading websites that use Ajax (Flickr, GMail, del.icio.us).

The research will deliver quantitive and qualitative results. The quantitive results regard the savings in server load and network bandwidth. The qualitative results regard the usability tests [16] and will provide information on usability and experience improvement. Also, the quality of the code written with a framework is important. The

final conclusion will determine whether it is worth implementing an Ajax framework into an existing website in terms of usability and server load.

2.2 Research question

The study will research the effects of Ajax in terms of performance and usability in web applications. How and why are performance and usability related in web applications? This main question can be subdivided into four questions:

- Which Ajax framework suits the case study?
- How will Ajax affect network traffic?
- How will Ajax affect server load?
- Will Ajax offer better usability and productivity?

Choosing an Ajax framework is very important. There are many frameworks available, but not all of them are suitable or just don't provide enough functionality. The network traffic and server load are important to measure in order to see how it will affect the scalability of your application. Better usability will keep users on your website longer and, which is good for commercial reasons.

These questions will be answered in:

- Selecting the framework (chapter 6)
- The effect on network traffic (chapter 7)
- The effect on server load (chapter 8)
- The effect on usability (chapter 9)

3 Research method

To be able to test the effects of using Ajax in a large web environment, an Ajax framework will be implemented into the Hyves website, http://www.hyves.nl. Hyves is The Netherlands' largest social networking site with over 2 million unique users, over 12 million photos and videos, 10 million page views per day and more than 180 web, database and storage servers and counting. While the quantitive effects of Ajax will go unnoticed in a small web environment, with a website as large as Hyves' the effects will be clearly visible. Because of the scale and it's highly dynamic and modular pages it is clearly a good test case. The website is constantly experiencing very heavy loads. Reducing the load and network traffic will help in further scaling the website. Results can be found in chapters 7 and 8.

Please note: due to the sensitive nature of this information, the description of the "page" is very abstract. But in order to get an impression of what we are dealing with here is some information about the specific page.

The page that was modified to use the Ajax application model is one of the most frequently requested pages of the website. The page contains user placed content, along with meta information like comments, number of views and for example. Almost all content is dynamically generated. When requesting another page of the same type, roughly one third of the page content has to be updated to create a new page. The other two thirds remain the same, like on most pages on the website. In general the findings in this research apply to web pages similar characteristics and are regarded as typical pages. This means that a variable portion of the page can be updated in order to present new content or a new page.

The modified page was deployed in a live situation and observed for one month. This page is often requested in sequences, only changing a small portion of the page. With the modification, only an incremental part of the page is retrieved using Ajax and is inserted into the existing page. The first page requested in a sequence will be almost identical to the unmodified page and the advantages will be seen in the incremental updates. Because of this incremental update, we will likely see a reduced load on the server and a reduced amount of network traffic. The results will be measured for two periods of one month each, one month with the unmodified page, one month with the modified page.

In the evaluation the research questions will be answered. Which framework is suitable for the case study? The effect of Ajax on performance is split in two. What is the effect on server load and what is the effect on network traffic? Finally, what is the effect of Ajax on usability?

After choosing a suitable framework it will be implemented in the website. Aiding

developers in writing JavaScript more efficiently is one thing but more interesting are the quantitive and qualitative results. For quantitive results I have measured the amount of network bandwidth required for a non-Ajax and an Ajax page. Also the server time saved by only requesting incremental (or delta) page updates was measured. Last but not least, how do users react to Ajax enabled web applications? The application model is different from traditional web applications and users might be unaware of the working of the controls. It is expected that users generally complete their tasks faster and more efficient but will they instantly learn how to use them and become more productive right away?

3.1 On implementation

The implementation of an Ajax framework requires modifications on the client and server side. The modifications on the client are the most obvious, while the server side modifications largely depend on the underlying architecture. The client will require custom JavaScript functions to post data to the server or request data from the server and update the user interface accordingly instead of requesting a whole new page. This will cause a slight overhead due to the extra code on the client.

An Ajax request is just like any other HTTP request, but is performed from JavaScript. As far as the server is concerned it is just another GET or POST operation and will be handled accordingly. Most requests enter the server through the main index script, which then executes the desired script as defined by the request parameters. Because of this, requests for Ajax and non-Ajax pages can co-exist on the server and could be changed at the flick of a switch. Because of this independence the test cases for Ajax and non-Ajax are well isolated and have no effect on each other.

4 Background and context of Ajax

Ajax is the key technology is this thesis and therefore it is essential to know what it is, how it is used and what the caveats are.

4.1 Background

In the last year to year and a half, the name Ajax has started to appear on the Internet more and more. It is often wrongly said to be the same as web 2.0, although it is true that most of the web 2.0 companies and websites make heavy use of Ajax. Ajax itself is not a new technology but rather a combined use of existing technologies, you might call it a design pattern. Ajax is an acronym for Asynchronous JavaScript And XML. The name describes quite well what it is and does. Ajax uses JavaScript to make asynchronous request to a (web)server with XML as mark-up for the data being received. Instead of XML, plain text can also be used. The key element in Ajax is the arguably misnamed XMLHttpRequest (XHR) JavaScript object, which introduces the ability to do server requests from JavaScript. Essentially this has been possible in Microsoft's Internet Explorer 5 browser since 1998 when Microsoft started using it in the web-version of their Outlook mail client [23]. The XMLHttpRequest object thanks its existence to Microsoft, but was later natively implemented in Mozilla's, Apple's and other browsers. So why is it that while Ajax has been possible for the last eight years it was so little known? Until recently the name Ajax was only associated with Greek gods, soccer teams and household cleaning liquids but not with software technology. The name was thought up and popularized by Jesse James Garrett of Adaptive Path [6] in the beginning of 2005 and has been spreading like fire ever since. It seems like it needed a simple name to describe the strategy, much like design patterns.

4.2 Rich Internet Applications

Web pages have had an interaction model that was defining for the Internet. You see a page, click a link and wait for the next page to load. It is a rather static interaction model and the user workflow is interrupted all the time, with every new request. The industry has been speculating for years that, in time, web based applications would be favoured above the desktop applications as we know them. Attempts have been made but nothing really took off, maybe because the available technology was not mature yet, maybe because the Internet penetration was too low. Whatever the reason, we are now at the start of a new type of website. There is a shift going on. Websites are becoming more like desktop applications, moving from transient to sovereign applications. In [1] usability expert Alan Cooper is quoted on the two usage modes. Transient applications might be used every day but only for a short period, where sovereign applications require the user's attention for several hours at a time. The common name for these types of applications is Rich Internet Applications (or RIA).

Rich Internet Applications can be created in many different ways using different technologies. Flash, Java Applets and Ajax are all legitimate technologies for Rich Internet Applications [17], each with their own advantages and disadvantages. The key point of Rich Internet Applications is creating a much better, more intuitive, user experience and having users get more work done in less time.

4.3 JavaScript

JavaScript has acquired a rather disputable reputation over the years. From the end-users' perspective, JavaScript accounted for annoying pop-ups, disabling right-clicking to prevent saving images from your website and creating downright stupid features. From the developers' perspective, JavaScript has had its security issues, suffered from incompatible browser DOMs (Document Object Model, a tree-like object-oriented representation of a web page) and lacked integrated development environments and debugging tools. Many consider JavaScript to be too "simple" while in fact it's a rather powerful scripting language and even the most popular programming language on the web. The success of JavaScript lies in the fact that it has relatively low barriers for use. No compiler is needed, and scripts can be run in any browser with little editing. At the inception of JavaScript in 1995, the main target audience of JavaScript were the thousands of web developers who had no substantial programming background but needed some way of creating more interactivity in their websites. In ten years time, most, but definitely not all, browser incompatibilities have been solved resulting in more and more web pages using JavaScript for useful things.
JavaScript is an interpreted, loosely typed, object-oriented programming language that is mainly used on web pages and runs in the browser. It can also be incorporated into other environments like XSLT parsers or as a scripting language in varying frameworks.

4.4 The XMLHttpRequest object

This little known object of the JavaScript standard library plays a key role in Ajax. The object makes it possible to do asynchronous HTTP requests to the server, within the same domain. From the initiation of the request the object will report the state of the object, the so-called readystate, through a call-back function. This information can be used to track the progress of the request until it has completed. The response of the request can either be XML or plain text, additionally the plain text can be JSON. JSON stands for JavaScript Object Notation, which is a literal representation of JavaScript functions and objects that can be evaluated at run-time. This is because JavaScript is an interpreted language. The choice in data carriers offers great flexibility to both server and client side developers. The XMLHttpRequest object was introduced by Microsoft in 1998 as an ActiveX component, but was later implemented natively in most other browsers and as of April 5th 2006 it is a draft standard specification of the W3C. In the upcoming Internet Explorer 7, there will be a native implementation next

to the existing ActiveX component. Most would agree that it's a welcome addition to the JavaScript language allowing much more interactivity among other benefits.

4.5 Content manipulation

Ajax allows server requests without completely reloading the web page, and in order to be of any use, the requests response should be returned to the end user in some way. Visible feedback is an important requirement for Rich Internet Applications, indicating that something happening by for instance showing the newly added content or removing some content. Frequently used types of visual feedback are highlighting, movement of elements or animated indicators like hourglasses and progress bars. Every web page is internally represented by a Document Object Model (DOM) which essentially is a tree-like structure that can be accessed through a JavaScript interface. In modern browsers, almost any object of the DOM can be accessed and changed: styling, browse history, images, elements, form data and so on. Ajax applications are different from traditional web applications because some application logic is moved to the client and therefore the manipulation of the page elements is essential.

4.6 Caveat emptor

Ajax has quite some impact on the traditional application model of web applications. Certainly Ajax has a very positive side but there are some setbacks that come with the introduction of this new application model. I will discuss the issues a developer should be aware of and provide solutions where needed.

4.7 Bookmaking and the back button

Browsers were initially not developed for the kind of applications made possible with Ajax. People have become accustomed to their browser and the Internet and are expecting certain behaviour from the browser. One very important feature is the ability to set bookmarks to later return to the exact same page. Browsers keep bookmarks as URLs, because they should be portable to another computer, user or time. As noted by [9], when changing the web page with Ajax and DOM manipulation, the URL stays the same but the content doesn't. If you would bookmark a page that was modified using a content manipulation technique, returning to that page at another time will not result in the same page. This essentially renders your bookmarks useless, the same goes for passing on the URL to someone else by mail or instant messaging.
The type of solution suggested by [9] is the most widely used technique to maintain bookmarkability. When content is changed in a web page, a "#" mark is added to the URL in the address bar followed by one or more unique identifiers. When the URL of the page is transferred and the page is reopened, the client side script will have to check for the "#" mark and then reconstruct the page from these unique identifiers. This technique works in the most popular browsers, but not in all browsers like [9] suggests.

Some browsers will force a page reload when adding the “#” mark. The “#” mark is normally used for scrolling to an in-page anchor. But when the anchor is not available nothing noticeable for the user will happen and thus information can be stored after the “#” mark.
Another important feature in browsers is the browser history, a chronological track record of pages you have visited. Users expect to return to the previous page, or page state, when pressing the back button. Then pressing the forward button should bring you to the original page you were viewing. Most browsers update their history when adding the “#” mark and thus allowing easy navigation. However, not all browsers implement the history the same way and so browser specific workarounds are needed. A common solution is to use a hidden iFrame in the page and force the page history there, without changing the actual page.

4.8 Page footprint

As with all web applications, a great deal of the program code is executed by the client. Because of this, code has to be transferred to the client. Depending on the framework the size of the code varies between a few kilobytes and a few megabytes. When network bandwidth is critical, there are techniques to decrease the amount of data to be transferred. Most web servers support the gzip content encoding method. When the client does so too, there is an opportunity for significant size reduction equivalent to compressing data with the gzip command found on most Unix-like operating systems. Alternatively or additionally, a more aggressive approach is to remove all formatting characters (spaces, tabs and new lines) or even shortening function and variable names in order to reduce size. The drawback is that the code becomes much less readable and thus harder to debug in a live situation. Of course the original code will be maintained, but will have to be compacted before every deploy.
Saving some bandwidth is good, so gzipping the content will suffice.

4.9 Browser wars

The term “browser war” can be interpreted in different ways. Often it refers to the quickly changing market shares held by the respective browsers. In a way this is important as to make decisions about which browsers to support. From a technical point of view the “browser war” can be interpreted as the incompatibility of different implementations of certain browser features. Some years ago, the differences between browsers were huge, making full cross-browser applications out of reach for most developers and often it was chosen to only support certain browsers. A key requirement for the success of Ajax is browser compatibility. Browser compatibility has become better in recent years because of the standardization of implementations. There are still differences in implementations though, but most of them can easily be coded around. Frameworks usually take care of browser incompatibilities. If you are going to push the

limits of rich Internet applications and cross browser compatibility, be prepared for obstacles you might not be able to overcome. When choosing a framework, determine which browsers you want to support and choose accordingly. For the case study, the widest support possible is needed because of the enormous amount of visitors and many different clients.

5 Framework requirements

To make a good judgement on which Ajax framework, if any, is suitable to be incorporated in a very large website you need requirements. What is the framework expected to offer and what are the needs for this specific case study? In the case study an Ajax framework will be implemented in the Hyves website. An existing page in the website will be modified so that it takes advantage of the Ajax application model. The effect on server time required to process a page and the amount of data transferred will be studied for traditional pages and Ajax enabled pages.
Chapters 5.2 to 5.6 each describe a framework requirement.

5.1 Defining a framework

The word "framework" requires some explanation. Ajax by itself is not a framework it is a design pattern. A framework in software development can be defined as a support structure consisting of support programs and code libraries, used to build new software. Frameworks are designed to facilitate software development, allowing software developers to spend more time on the software requirements rather than dealing with the low level details to get a system working. In literature frameworks are defined in many ways such as a reusable and extendable set of objects for related functions. Or a set of related functions that define the area of expertise or competencies of the framework [14]. When looking at Ajax it makes sense to use a framework. Classes and functions for making Ajax requests, manipulating DOMs and visual indicators are all related in the context of rich Internet applications. This is exactly what the reviewed frameworks are. In general they offer a high level of code abstraction and let you focus on what you really want to accomplish instead of handling all sorts of trivial operations. Common complaints against frameworks are that they set you back with too much code that could make your system unnecessarily slow or complex. Programmers will also have to invest in actually learning the framework. This is a trade-off however and many software developers will agree that generally using a framework will make you more productive.
Although not all the reviewed "frameworks" call themselves frameworks, looking at the definitions of a framework they fall into this category and will be referenced as such.

5.2 High level code abstraction

Performing an Ajax request seems like a rather simple thing to do but in fact it requires rather complicated code for different browsers. As it is an asynchronous request you will have to check incoming data for a readystate (see chapter 4.4) and perform the appropriate action. The chosen Ajax framework should provide a high level of abstraction for performing Ajax requests. As mentioned earlier, client side content manipulation is essential for the Ajax application model and therefore the framework

should also provide other high level abstractions to simplify common routines and lift cross browser issues off the developer's shoulder.

5.3 Non restricting licensing model

Ajax/JavaScript frameworks are available under many different licenses, from expensive commercial licenses to open source licenses with very few restrictions. The company in the case study is a start-up where money matters. Some of the more extensive frameworks carry per server or even per CPU licenses, which would be very expensive for more than 70 web servers and double the amount of CPU's! For that reason the focus will lie on freely available frameworks, although commercial frameworks are not excluded.
From a developers point of view the licensing model also has impact on the ability to modify source code and indirectly the continuity of the product. When choosing third party software there is a risk of lack in continuity. With commercial products this may be a problem as the providing company might discontinue a product and might not open up the sources. Free or open source products are different in the way that you can only depend on them to a certain extent. The sources are mostly open but the development roadmap can be very erratic. The licensing model is of importance when the creator or company that made the framework might become defunct. In order to guarantee the continuity of the product, the licensing model should allow for modification when needed. In general, the more flexible the license the better.

5.4 Documentation

The quality of documentation is important, as is community support. Commercial products are likely to be shipped with adequate documentation, but the quality of documentation of open source products varies widely if there is any documentation at all. Quite often the product is developed while the documentation is lagging. We require either decent documentation from the creator or sufficient support from the community, like forums and third party documentation.

5.5 Back-end independence

The available frameworks differ strongly in what they are trying to achieve and what they have to offer. Some are only Ajax/JavaScript frameworks that focus only on the client side while others offer end-to-end solutions including a fully integrated development environment. The Hyves website is a very large commercial website with around 500.000 lines of code. When adding new Ajax functionality to the website the existing backend will have to be used, as rewriting the back-end would have too much impact on all the developers and would require a long feature freeze. Developing a new backend for use with a framework will call all other development to a screaming halt, which is simply unacceptable from a commercial perspective. The chosen framework

has to be back-end independent, making most end-to-end frameworks unsuitable solutions.

5.6 Broad level of browser compatibility

With a large scale website like Hyves comes a wide variety of browsers that visit the website. When using an Ajax framework you don't want to worry about cross browser issues, this is something the framework should take care of for you. It is nearly impossible to fully support all browsers but luckily most of them can be categorized by their rendering engine. Below is an overview of the most important browsers that visit Hyves. The overview also includes browser versions and their respective rendering engines. Other browsers than the ones mentioned, that use one of the above rendering engines, are likely to have no problems.

Browser name	Rendering engine	Should be supported
Microsoft Internet Explorer 5.5 Microsoft Internet Explorer 6.0 Microsoft Internet Explorer 7.0	Trident/MSHTML	Yes
Mozilla FireFox 1.0.x Mozilla FireFox 1.5.x	Gecko	Yes
Apple Safari 1.x Apple Safari 2.x	WebCore	Yes
Opera 7.x Opera 8.x Opera 9.x	Presto	Yes

The browsers mentioned above account for 99.9 percent (literally) of all browsers that visit the website, supporting these browsers is sufficient.

5.7 Framework preselection

Since the inception of the name Ajax, many frameworks have rapidly become available. Even in the last few months, new frameworks were released that I could not include in the evaluation. It is not possible to evaluate all frameworks in the limited amount of time. Using the criteria mentioned earlier, a subset of frameworks will have to be made. An important criterion is the cross browser compatibility. In [22] a grading system for Ajax frameworks is explained and also a vast amount of frameworks are rated. This is also a good starting point for determining which frameworks to select. Obviously there are even more frameworks available, but these are mostly unfinished or very small and unknown products. The rating system categorizes frameworks in five

categories (from A to E), ranging from the broadest level of browser compatibility to the least browser compatibility. We focus on grade A Ajax frameworks as Internet Explorer 6+, FireFox 1.0+ and Safari 1.2+, Opera and other DOM-compliant browsers are supported. They also best represent the browsers that visit the Hyves website. Please note that Ajax frameworks are constantly being re-evaluated and may move from one category to another. At the time of writing the following frameworks fell into the grade A category:

- Dojo Toolkit
- Echo 2
- JavaScript/Ajax Toolbox
- jQuery
- Moo.fx
- Prototype
- Rico
- Sardalya
- Script.aculo.us
- Tacos
- TurboWidgets
- TwinHelix
- Wicket
- Yahoo! User Interface Library

As this is a preselection phase, the frameworks will not be discussed in depth. All the framework information here is gathered from their respective web sites, documentation and demonstrations.
I will walk through these frameworks. For any of the licenses mentioned below, please visit http://opensource.org/ for the most recent version of the complete license texts.

5.7.1 Grade A frameworks

Dojo Toolkit

The Dojo Toolkit is a well-featured JavaScript Toolkit that provides many functionalities for creating rich internet applications. Some effects and many controls are available as well as a lot of advanced features. The framework footprint is rather large but supports a lazy loading mechanism. It is available through either an Academic Free License or BSD license. There is a roadmap for the coming years and they have substantial backup from the industry, IBM and Sun Microsystems for example. It is a very advanced framework and looks very promising. Not all controls work on all browsers though.

Echo 2

This framework is an end-to-end solution, which is available under the Mozilla Public License or alternatively LGPL. Your web applications are written in Java and the framework generates your web application. This construction will not fit into an existing environment, although it is an interesting option to consider when building from scratch and your chosen programming language is Java.

JavaScript/Ajax Toolbox

A library that simplifies frequently used controls in web pages, it has limited Ajax capabilities. The library does not require a specific back-end to work. In general this library is too limited.

jQuery

This library is not really a new creation but more of a custom implementation of different libraries, most notably Prototype, Moo.fx and Script.aculo.us. Its implementation and configurability is very limited. On the positive side, the library has a small footprint.

Moo.fx

Moo.fx is an effects library written on top of Prototype. It's a very small library with limited functionality but is promoted as such and is very easy to use in existing environments. It's available under the MIT license.

Prototype

The name of this framework can be confusing as the property in JavaScript to extend object classes is also called prototype. Prototype takes away all cross browser annoyances in JavaScript and also extends existing objects with "missing" and very useful methods. It was originally developed as the client side counterpart to Ruby On Rails, but does not require it. Although documentation from the authors is virtually non-existent, the community provides plenty of documentation, tutorials and examples. The framework is available under the MIT license and has a limited footprint. It can easily be integrated into an existing environment, but a little care has to be taken because of possible conflicts with existing JavaScript code.

Rico

Rico offers JavaScript effects and controls written on top of Prototype. It's an easy to use library and easy to integrate into existing applications if you use Prototype. It was developed initially for Sabre Airline Solutions, but is now available for free under the Apache 2.0 license.

Sardalya

The Sardalya library is inspired by the Prototype library. It has not been tested on the Macintosh platform, it is hardly used and requires a license fee (be it a small one). The documentation is messy and it doesn't have anything to offer over other frameworks.

Script.aculo.us

Script.aculo.us is a controls and effects library written on top of Prototype, that is very configurable and fully cross browser. Not that many controls are provided though. The ones that are provided are very useful (drag and drop, slider, auto completion). The effects are very well thought out and provide excellent visual feedback for Ajax applications. The documentation is of good quality and many examples are provided. The library is becoming very popular among web developers and new controls and effects are being contributed. The library was developed for the commercial Fluxiom project, a web based digital asset manager, but most of it is now freely available under the MIT license. The library adds a fair amount of data to be downloaded, but offers the ability to retrieve only certain parts to save bandwidth.

Tacos

While most other frameworks are client based, Tacos is a back-end framework and is written in Java. For the client side it relies on other frameworks like Dojo, Prototype and Script.aculo.us. The documentation is still under development, the framework is available under the Apache Software License.

TurboWidgets

Like the name suggests, this framework offers widgets. The implementation is well done and relies on the Dojo Toolkit. The license however is restricting and requires a license fee.

TwinHelix

TwinHelix offers controls that have been around for ages, like menus, pop-ups and tool tips. TwinHelix is a collection of independent controls that each require a fee if you intend to use them.

Wicket

This is another Java based framework, publicly available under the Apache 2.0 License. It is a fully featured WYSIWYG framework for designing Ajax applications, requiring no HTML or JavaScript skills and offering full cross browser compatibility.

Yahoo! User Interface Library

This library was only recently released, under the BSD license. It has a full set of features such as animation effects, widgets and Ajax functions. The documentation and examples are of good quality and so is the code itself. Yahoo! is dedicated to this project and is actively developing.

5.7.2 Preselection conclusion

Although there might be frameworks available that have better features or more complete support, they do not offer enough cross browser compatibility for me to consider them. More and more frameworks are becoming available making it even harder to make a decision.

First of all, all frameworks that are not truly cross browser were not even considered in this evaluation. Frameworks that rely on a particular back technology or that are back end only are ruled out. Frameworks that are closed source implementations or require license fees are the next ones to be excluded from further evaluation. The overall design of the framework is a final criterion on which the frameworks will be preselected.

An interesting observation is that quite a few of these grade A frameworks (Script.aculo.us, Rico, Wicket, Taco, Moo.fx, jQuery) are built on top of or inspired by Prototype. Why is this? The following reasons come to mind. Prototype has been around for almost a year and was one of the first Ajax frameworks available, it was popularized by the Ruby on Rails framework that uses Prototype as its client side counterpart and last but not least because it is very well written and thought out. The Dojo Toolkit also looks very promising and has a very good roadmap and industry support, but the framework footprint is quite large. The Yahoo! User Interface Library (YUI) is very new but offers great functionality and documentation. Yahoo! made this library available under an open source license. Prototype, Dojo and YUI are the only frameworks suitable for use in the case study and will be discussed in depth in chapter 6. Script.aculo.us will be evaluated together with Prototype, instead of Moo.fx and Rico, because of the very tight integration with Prototype and because it is very well designed.

Table 1: framework usability

	Back-end independent	Open source / no license required	Overall design	Conclusion usability in case study
Dojo Toolkit	+	+	++	+
Prototype	+	+	++	+
Script.aculo.us	+	+	++	+
Rico	+	+	+	+
Yahoo! User Interface Library	+	+	++	+
Moo.fx	+	+	+	+/-
JavaScript/Ajax Toolbox	+	+	-	+/-
jQuery	+	+	-	+/-
Sardalya	+	-	-	-
TurboWidgets	+	-	+	-
TwinHelix	+	-	-	-
Echo 2	-	+	+	-
Tacos	-	+	+	-
Wicket	-	+	+	-

6 Selecting the framework

The three frameworks (Dojo Toolkit, Prototype and Yahoo! User Interface Library) that where pre-selected in chapter 5.7.2 will be assessed based on the selection requirements as described in chapter 6.1.

6.1 Selection requirements

Some requirements are more important than others when selecting a framework for this case study. Some apply to frameworks in general, others apply more specifically to this case study. The requirements can be divided into two groups: functional and non-functional requirements.

For the functional requirements we will not be looking at the ability to perform Ajax requests, because they all do. This was a requirement to be evaluated in chapter 5.7.1.

The functional requirements are:

- Useful visual effects
- Widgets (user controls) for greater efficiency
- Good event handling abstraction

The visual effects play a key role in Ajax applications as they actively keep the user informed of what is happening. Without these visual indicators the user is left in limbo on the current status of the application. The purpose and usefulness of these effects weighs heaviest.

Advanced user controls (widgets) are often associated with Ajax. These widgets can be anything from drag-and-drop to sliders and from in-place-editing to auto-completing text fields. It can be a complex and tedious task to write good cross-browser versions of these controls yourself. Useful and cross-browser widgets are important in selecting a framework.

The more advanced your Ajax application becomes, the more complex it will become to write good code for event handling. Instead of coding all events inline, a more AOP style approach is preferred. Most Ajax framework projects have correctly identified this area that needs attention and have thought up solutions to make event handling more manageable in complex applications.

The non-functional requirements are:

- Effective use of the possibilities of JavaScript
- Good ease of use for developers

- Good documentation and examples
- Flexible licensing

As the three selected frameworks are client side only, they are written entirely in JavaScript. As JavaScript is an interpreted language it offers great possibilities and more flexibility to your programming style. Do the frameworks take advantage of the possibilities of JavaScript?

As noted earlier an arguable drawback of using a framework is that developers will actually have to spend time to learn to program in the framework. In order to minimize the overhead that is created by introducing a framework, the ease of use is of importance. Does the coding style resemble another well-known framework or language? Isn't the framework over engineered? These questions will help to determine how quickly developers will adapt to the framework.

With many open source projects the documentation is still incomplete. But the documentation should help developers learn to use the framework more quickly. When it comes to debugging errors in your code or maybe the frameworks code, if you have no documentation it feels more like black box testing. Having documentation available will give far better results. The availability of the documentation is needed for possible future debugging and currently for assisting the developers.

All of the selected frameworks are open source. The licensing model of a framework matters when it comes to continuity. With the high paced development on the Internet it is not unlikely that every now and then standards will be broken and the framework might possibly not function the way it was designed anymore. Usually the creators of the framework will fix the framework. If the framework might become obsolete or abandoned it is essential that you have the right to make modifications to the framework in order to fix it yourself.

The requirements for the frameworks were evaluated by reading documentation, checking forums for user experiences and by writing prototypes to get simple tasks done.

6.2 Dojo Toolkit

The Dojo Foundation is a non-profit organization that currently hosts two projects, the Dojo Toolkit being one of them. It is getting support by major companies in the industry like IBM, AOL, OpenLaszlo and most recently Sun Microsystems. In essence the Dojo Toolkit is a dynamic HTML (DHTML) toolkit with Ajax support written in JavaScript. As with most JavaScript frameworks it aims to simplify development in JavaScript by solving browser incompatibilities with a new layer of abstraction. Although it's only at version 0.3.1 and still has many bugs to be fixed and features to be

implemented, it offers a great amount of functionality. The event system, I/O APIs, widgets and generic language enhancements are only a few of them. The toolkit is not required to run in a browser environment but can run in most JavaScript engines.

JavaScript

The Dojo Toolkit offers a Java like package system, which requires you to specifically include packages to be able to use certain functions. While you could write JavaScript in a non-object oriented way, the Dojo Toolkit is all object oriented and offers a very high level of code abstraction.

Event-handling

The Dojo event system uses an aspect oriented programming (AOP) approach to event handling. [19] describes the practices of AOP. When using a likewise approach for event handling, one can isolate the events added to objects without changing the objects themselves. This is a very good solution to the problem of event handling.

Visual effects

The visual effects are most valuable as an indicator that something has changed or needs attention, though Dojo focuses mostly on animation (movement). In general this is useful but when you look at existing web applications that use Ajax, this kind of animation is rarely used. Of course it is possible to write your own visual effects and that's where Dojo comes in handy because custom effects are very configurable but will require some effort. Very useful out of the box effects are not provided though.

Widgets

The toolkit offers many user controls that are difficult to implement without a toolkit. While trying out the different controls, quite a lot of them came out differently in different browsers or simply didn't work at all. This is probably a work in progress, but it seems the focus lies on sheer quantity of widgets instead of true cross browser compatibility for now. When using a framework you don't want to worry about widgets not being compatible with all browsers!

Ease of use

Because of the great amount of functionalities that Dojo offers it will require some time to become proficient at it and the unfinished state of the documentation makes it more difficult to get started. The Dojo Toolkit is very well structured though.

Documentation

Documentation [20] is largely in the works. This means that there is no complete reference guide or that documentation for some parts might be unavailable. There is a very large repository of example code, which might be used to see how things work. Most of these examples are rather limited though and should be used with care.

Licensing

The Dojo Toolkit has a dual licensing system, it is either available under the BSD license or the preferred Academic Free License v2.1. The BSD license should be applied when the Dojo Toolkit is being used in software products that fall under the (L)GPL. Both licenses are incredibly permissive and give you royalty free rights to use or modify the original work. For the full license texts, see http://opensource.org/licenses/.

6.3 Prototype (with Script.aculo.us)

Prototype was developed to ease the development of highly interactive web pages and rub out browser incompatibilities. It is also the client-side part of the very popular Ruby on Rails web development framework, but it also works independently or with any server-side software. It is an object oriented JavaScript library that offers easy Ajax requests and many useful functions for content manipulation.

JavaScript

A criticized [12] aspect of Prototype is that it extends many standard JavaScript objects with additional methods using the prototype property, hence the name Prototype for the library. This may cause conflicts with existing code that also extends the standard objects with equally named properties or methods. As described in [12], Prototype also breaks the standard behaviour for the Object type that is often used as an associative array in JavaScript. Code that relies on "for(var i in object){...}" control structures may have issues when using prototype. Not all is bad though, for these control structures can be rewritten with very little effort. So why accept this discomfort? Prototype adds many methods to existing objects and arrays which makes writing JavaScript code a lot more comfortable. These additional methods are inspired by the Ruby programming language. Prototype also offers object inheritance and functions for easy access to document elements. It is a very inspired work of art.

Event-handling

There a two approaches to event handling in Prototype, one is by adding event observers to certain elements on the page. This is comparable to the Dojo system, although the Dojo system is far more powerful. By using the additional

"Behaviour" library, a Prototype related project, it is possible to create rules based on CSS selectors. CSS is more than only styling of elements, it offers a very smart and extensive mechanism for selecting elements in your document tree. With Behaviour you have an unobtrusive way of adding events to elements on your page, selected by CSS rules.

Visual effects

The Prototype library doesn't provide visual effects on its own, which is why most often it is used in combination with Script.aculo.us. Script.aculo.us is built on top of Prototype and its syntax seamlessly blends with Prototype's. It offers the so very important visual indicators for when page content is updated. Highlighting and the appearing and fading of elements are very useful effects for drawing attention and showing the user what is happening. The effects system is very configurable and has a queuing mechanism for chaining effects. It is very easy to add effects and will generally provoke a "Wow!" reaction from the user.

Widgets

There are not a lot of controls in Script.aculo.us, but the ones that are provided are very useful and are difficult to implement otherwise. The key widgets are drag-and-drop support, sortables (creating sortable lists of nearly any collection of elements), sliders and auto-completing text fields. More controls are being added every now and then.

Ease of use

Prototype is very easy to use, even if you haven't written in Ruby before, and Script.aculo.us follows suit. All that needs to be done is include the required JavaScript libraries and everything is at the tip of your fingers. It's very intuitive and has a clean syntax.

Documentation

Documentation from the developer of Prototype is almost non-existent, but because of the enormous popularity of Prototype there are dozens of websites explaining its functions and providing samples. Some go in depth on certain features, while others provide complete function references. Most of them are collected at http://www.prototypedoc.com.

Licensing

The MIT license under which Prototype and Script.aculo.us are licensed is arguably one of the most flexible license types and gives the right to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the

software. For the full license text, see http://opensource.org/licenses/mit-license.php.

6.4 Yahoo! User Interface Library

The Yahoo! User Interface Library is relatively new and was not available at the initial selection process for possible candidates. The library looks very promising and could provide many developers with very high quality code to enrich their internet applications.

JavaScript

As in any programming language, using global variables is considered to be a bad practice. There is a chance you will be overwriting that variable when you write more code, and thus breaking your existing code. Chances increase when using more JavaScript. This is why all of the features are accessible through a single YAHOO global variable, which is very unlikely to be overwritten. The library is fully object oriented, and has a Java like package system.

Event-handling

The YUI facilitates event handling in a few ways. It has a flexible method for attaching event handlers to one or more elements, it provides browser abstraction for the event object and automatic listener cleanup just to name a few. The event part of the YUI is very well implemented.

Visual effects

The effects that are provided by the YUI are very basic effects: movement, resizing and opacity. Of course this allows for the creation of custom effects but, like the Dojo Toolkit, it doesn't provide useful effects by default. More thought should have been put into the effects library to make it more valuable.

Widgets

The amount of user controls is decent, the library offers auto completion, calendar, slider and tree views. The controls are very well implemented and work on all major browsers.

Ease of use

Depending on which features you will be implementing you will have to include extra JavaScript files for each additional feature you will be using. Because of its well-engineered nature and Java like setup most developers will have little trouble using it.

Documentation

Yahoo! provides extensive documentation of the user interface library, API references, cheat sheets and examples are all provided.

Licensing

The BSD license is a very flexible license and is equivalent to the MIT license except for the fact that it includes a no-endorsement clause. This prohibits you from using the name of the organization or its contributors to promote products derived from the original product.

6.5 Table overview of the frameworks

	Dojo	Prototype	YUI
License	AFL or BSD	MIT	BSD
Footprint	232Kb bootstrap (max 2Mb)	56Kb + 132Kb for Script.aculo.us	Depends on used libraries (max 1,3Mb)
Development roadmap	Yes	No	No
Documentation	Incomplete	No, good third party documentation available	Yes
Bookmarking	Limited	No	No
Effects	Limited usefulness	Yes (with script.aculo.us)	Limited usefulness
Object extending	No	Yes*	No
Cross browser	Yes (except some widgets)	Yes	Yes
Ease of use	Good	Very Good	Good
Widgets/Controls	Yes (many)	Yes (few)	Yes (few)
Community adoption	Very limited	Very high	Very limited

* This creates a risk as existing code might break, see Appendix B for a solution.

6.6 Analysis

Eventually all three of these frameworks will do the job just fine, but still one has to be chosen for implementation so I will give a rationale for my decision. Looking at the sheer footprint of the frameworks, Prototype clearly is the best option as it will take the least time to be downloaded. In the case study, however, static JavaScript files are served separately and once download are cached on the client. The broadband penetration in the Netherlands is also very high and the additional data of client scripts will require little time to download. Prototype is the easiest to integrate into an application, you don't have to think about what files to include in order to access certain functions. Dojo offers a lot of functionality and is a functional overkill for this particular case study, which require more time and effort from the developers who would be using it. Dojo and YUI will not break existing code as everything is written in its own namespace. Prototype can break existing code but workarounds are available and it will not really be an issue as long as you know how to write your code.

Bookmarking and history support is only partially supported in Dojo, so custom solutions will have to be written anyhow. In terms of visual effects, Script.aculo.us is clearly the winner. Everything you need to give your application more appeal and visual feedback is at your fingertips, where the other frameworks only offer limited usefulness. All frameworks are of similar quality when it comes to widgets and easy of use, although Dojo is a bit more complex and it's widgets do not work on all browsers. The YUI library is very new and therefore Prototype is favoured. Last but not least, Prototype and Script.aculo.us are being used in many websites already and has an overwhelming support from the community. These websites are the frontrunners in the Web 2.0 movement

6.7 Conclusion

We will be using Prototype together with Script.aculo.us as the framework of choice. The good third party documentation, very useful effects and controls, community adoption and high easy of use are the main reasons for this decision.

Since Prototype was chosen tests were executed to check where existing code in the Hyves website would break due to the extending of objects. This came down to only two pieces of code that could easily be fixed as described in appendix B. All new JavaScript code extensively uses the Prototype framework, freeing developers from quite some cross browser issues.

This concludes the first research question "Which Ajax framework suits the case study?". Next the effect on network traffic will be researched.

7 The effect on network traffic

An important use of Ajax is to enable incremental updates to the web page. When requesting a new page only certain parts will have to be updated. Depending on the type of page and the amount of data needed to be updated, it should be possible to reduce the amount of network traffic generated. This assumption was tested in the Hyves website.

7.1 Method

To get quantitive results on the effects of Ajax on network traffic, a web page in the site was modified in order to update contents using the Prototype framework. A similar approach is described in [13]. JavaScript functions were written in order to request only parts of the page and updating the current content instead of requesting an entire new page.

Using the OneStat web analytics tool used by Hyves, the amount of page views generated are compared over two periods of one month each. This analytics provides information like when and how often a certain page is requested, the amount of (unique) users and the client used by the user. Due to the sensitive nature of this information the page description is very abstract, see chapter 3 for more information about the page.

In the original set-up a page is always served as a whole. Every new item to be viewed is a complete round-trip to the server, returning the whole page. Even though much information on the page stays the same it is still being fetched from the server. Using Ajax it is possible to update only certain parts of the page, so called delta or incremental updates. The first page always contains the complete content and even a little overhead for the extra JavaScript needed for Ajax. Consequent views require only a partial update that include just the changes. With delta updates the absolute amount of data to be transferred can indeed be reduced significantly as we will see.

7.2 Results

The original page only consists of one page, while the Ajax version has one base page and a delta update page. The results are taken from two consecutive months.

Page	# Views	Average size	Total Kilobytes	Page	# Views	Average size	Total Kilobytes
Base	3,970,490	275Kb	1,091,884,750	Base	799,544	275Kb	219,874,600
				Delta	4,293,439	98Kb	420,757,022
		Total	1,091,884,750			Total	640,631,622

Figure 1: number of page requests and the amount of data generated

Serving the normal page in the old situation produces 1100 Gigabytes of data in this particular period, while in the new situation only 640 Gigabytes need to be transferred. A saving of about 450 Gigabytes, or 41.3 percent. In addition to the reduced network bandwidth, the effective number of page views for this page has increased by over a million. The difference in number of unique visitors over these monthly periods was less than 1 percent, so this can only mean that unique visitors viewed more pages of this particular type.

7.3 Analysis

As we have seen, significant amounts of data can be saved by using incremental updates. Although these results will vary strongly depending on the amount of content to be updated, the figures above are a good representation of a typical use of Ajax for delta updates. In the future more modifications can and will be made to other pages on the website in order to save even more network bandwidth.
Something more interesting however is the reason why there are over a million more page views in the Ajax situation. It cannot be explained by the 1 percent difference in unique visitors. Is it because of the increased usability and reduced latency introduced by Ajax? And how can these extra million pages be served while the servers are already dealing with very high loads? The next chapter will give insight in the server load of the Ajax and non-Ajax situations.

7.4 Conclusion

For a certain single page it can be expected that using Ajax an average X% of network traffic can be saved, where X is the percentage of page content that remains the same when updating the page. This means that 100% - X% of page content needs to be transferred over the network, where in traditional web pages all 100% needed to be transferred. A saving of 100% is not possible, because this would mean that no page content needs to be updated and performing a request would be useless. This conclusion applies to web pages that match the characteristics mentioned in chapter 3.

Even when the number of requests increase in the Ajax situation there is still a significant absolute saving of network traffic.

8 The effect on server load

In addition to the reduced amount of data that has to be transferred when using Ajax, it is also interesting to see how much time the server needs to actually prepare the data before it is being sent to the client. Server load is purely a factor on the server whereas the amount of network traffic impacts the server network connection, the user's Internet connection and the costs that network traffic might generate.

The incremental updates have a reduced complexity compared to the original pages and should therefore take less time to process. The measurements were taken from the same pages as described in the previous chapter.

8.1 Method

For the measurements of the non-Ajax page, 10 series of 4 unique pages were requested entirely. The server measures the time needed to process the page. The timer is started at the moment the script (of the requested page) starts executing and stops when nothing more has to be done.
For the measurements of the Ajax page, again 10 series of requests were done. The first request is a complete page, similar to a page in the non-Ajax version. The consecutive requests are delta updates to the current page. These delta updates only contain certain elements that need to be replaced in the current page in order to create a new page with new content. As we have seen, these delta updates contain far less data than a complete page.

The execution time reflects the difference in system time (also called wall-clock time or real-world time) between start and end, not the actual processor time to execute a script. In multitasking environments this can have a negative effect on the time to complete a script as other processes might require time from the processor. All testing however was done on an isolated machine used only for running these performance tests. As multiple series of runs were done, any other processes that might have affected the results would be clearly visible.

For the database a development server was used, which is shared between developers. When many developers would actively be using the database at the same time this could affect the time to complete a script. This is not bias but actually better represents a real-world situation. This would also affect both non-Ajax and Ajax situations.

8.2 Results

Figure 2 shows the average time for the server to process a page in both Ajax and non-Ajax situations as collected from the test runs. The maximum and minimum times were removed from the results to get a better representation of the average. The numbers 1 to 4 represent consecutive requests that were always performed in order.

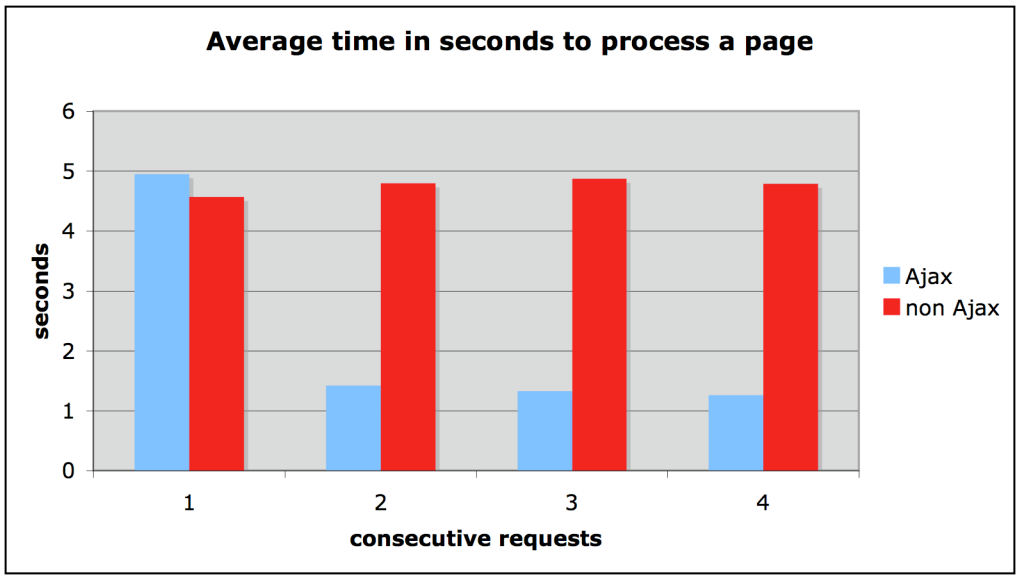


Figure 2: difference in processing time

As we can see, the initial page requires slightly more time to be processed because of the overhead of the extra Ajax code. Compared to the consecutive requests however, this is really insignificant. The reduced complexity of the delta updates requires far less time to be processed. For this specific page it shows that the delta updates require about one third of the time to process when compared to the first request. The results from web analytics tool used by Hyves in Figure 1 show that there are far more delta updates being requested at a 5.37 to 1 ratio. This represents a very significant saving in server processing time and future modifications to other pages will further improve savings. The long processing times of the pages are due to their highly dynamic nature.

8.3 Analysis

By using incremental updates for your web pages you can achieve significant savings on server load. However, this strongly depends on how dynamic your pages are. It speaks for itself that a page with 80 percent dynamically generated content of which 50 percent is incrementally updated offers more advantage than a page with also 80

percent dynamically generated content of which 90 percent is incrementally updated. In the case study roughly only one third of the page is changed with an incremental update, explaining the strongly reduced server time required. See chapter 3 for more information on the specific page. It is easy to imagine that less complex page updates will result in even greater improvements.

8.4 Conclusion

Even though you will need to add extra code in order to enable the use of Ajax, this code is executed on the client and causes no overhead on the server. Consecutive page requests after the first one offer a significant reduction in load on the server. Again, this conclusion can be applied to pages with characteristics similar to those mentioned in chapter 3. To get exact numbers on how much load can be saved you would need to measure the complexity of all parts of a page. Then you know which parts will need to be updated and how their complexity measures compared to other parts of the page, that do not need to be updated. This is a very tedious task and does not contribute to this conclusion in general terms as it is different for every page. The order of magnitude is far more interesting.

9 The effect on usability

Maybe the single most important effect of using Ajax is the improvement of usability. According to Jakob Nielsen [15], usability is defined by five quality components: learnability, efficiency, memorability, errors and satisfaction. In this study all these components are measured except for memorability as all tests were done in one session. Memorability is how easily users become proficient in an application after a period of not using the application.

The efficiency of web pages can be improved in different ways. With Ajax you will mainly be focussing on the availability of user controls (widgets) for more intuitive use of the application and reduced server response latency, which translates into better overall responsiveness.

9.1 Method

To test the effects of improved usability, I have made two versions of a sample application and conducted tests as described in [16]. It suggests what measurements to take, how many users you need, how to prepare the users and how to avoid possible bias.

The application lets you make simple to-do lists, from now on it will be called "Listmaker". A number of tasks were predefined for the user to carry out. The users were asked to make a to-do list with 10 items prior to the actual test in order to reduce possible bias for one of the applications. These were the tasks:

- Enter all 10 items on your to-do list;
- Put the last item in the list first;
- Move item 4 to position 8;
- Move item 5 to position 2;
- Change the text of 3 items in the list;
- Remove 3 items from the list.

The following measurements were taken during and after the tests:

- Total time to completion;
- The number of user errors;
- The number of times the users expressed clear frustration;
- The preferred version of the application.

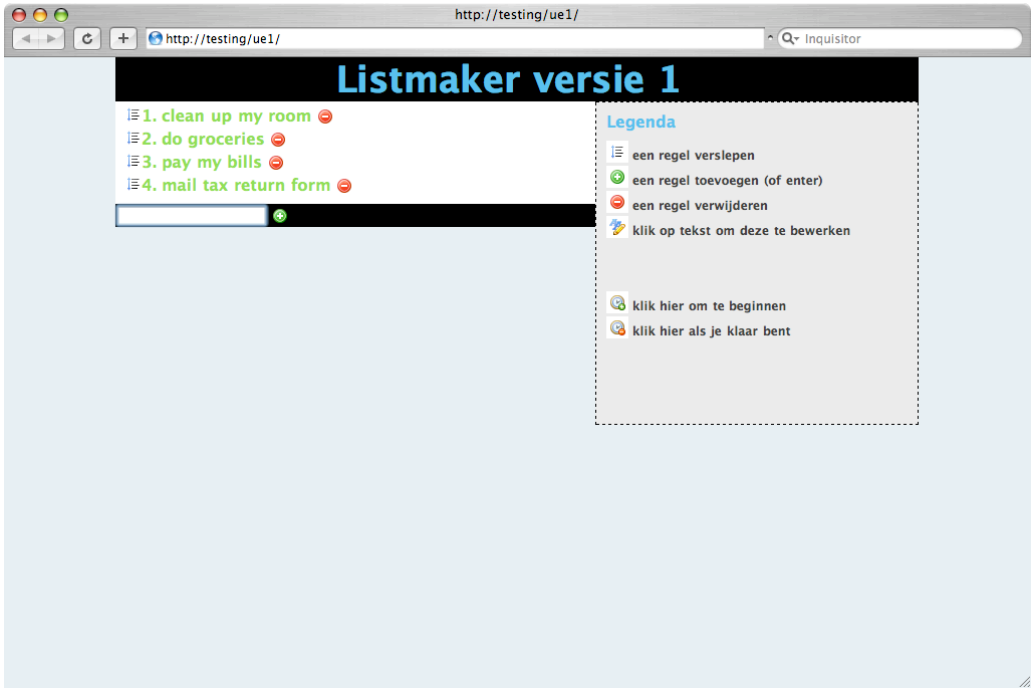


Figure 3: Ajax version of Listmaker

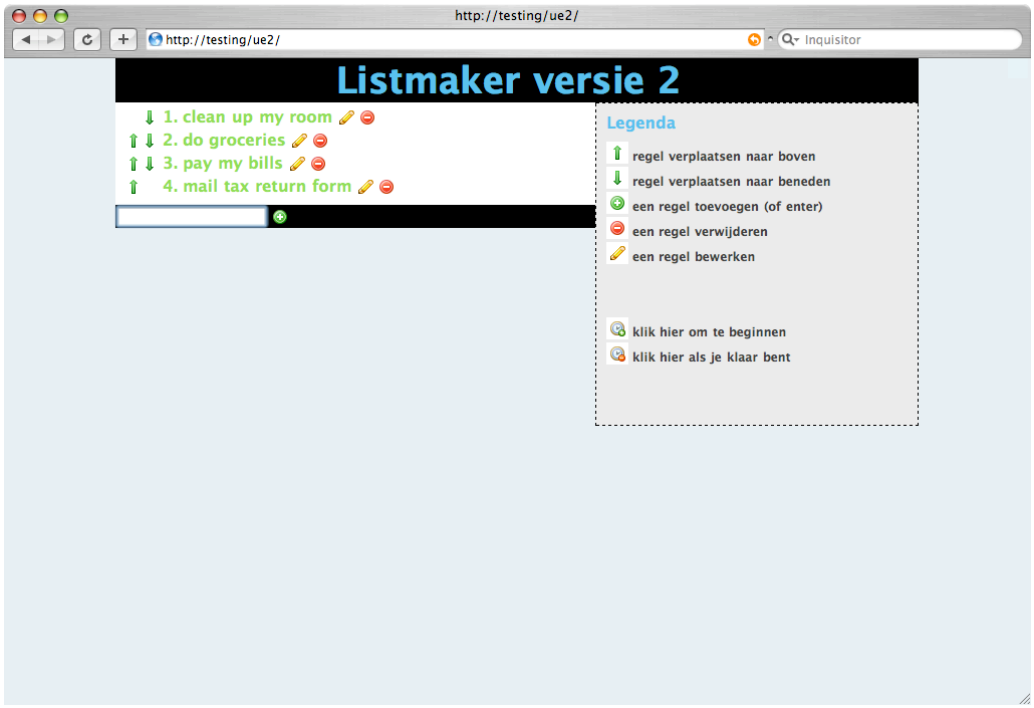


Figure 4: non-Ajax version of Listmaker

Two groups of five test persons where asked to complete both the task list in both applications. One group started with the non-Ajax version, the other group started with the Ajax version. After the completion of both tasks, the subjects were asked to perform the tasks in the Ajax version of Listmaker one more time. This was done to see if they could perform tasks more efficiently now that they had learned how to use this new kind of web application that uses Ajax. People are generally familiar with how websites work but Ajax is something else. Before actually starting the tests, users were briefed with the following:

- The tests will remain anonymous;
- During the test, it is not allowed to ask questions;
- Questions may be asked and answered afterwards;
- Carry out the tasks exactly as instructed;
- Take your time;
- Stay focused and turn off your phone.

With the Ajax version of the Listmaker, users are expected to complete their tasks in less time due to the fact that their workflow isn't interrupted when adding or changing data. In fact, the user perceived latency is almost non-existent as data is being sent to the server behind the scenes while the user interface might be updated instantly. This near instant response time is very desirable, as Nielsen describes [16]. Response times longer than 10 seconds require progress indicators and generally cause users to lose their attention. In traditional web applications the user perceived latency depends on several factors: server throughput, the server's connection, the Internet, the user's connection and the speed of the browser. With Ajax you effectively only have to deal with the browser speed. In [21] it is said that it is likely that users make more errors when response times are long, so this is a possible time saver.

Furthermore the controls are expected to be more intuitive, as it supports drag-and-drop and in-place-edit capabilities, further improving the productiveness. Drag-and-drop is the ability to grab an element and drop it somewhere else, which is perfect for the reordering task in the test. In-place-edit lets you change text values at their current position. The non-Ajax version of the Listmaker does not have any of the advantages mentioned above and thus is likely to require more time in order to complete the tasks.

The non-Ajax version of the application creates round trips to a server and therefore an artificial load (on both network and processing time) is created on the server to better represent a real world environment. This load was present for both versions of the application. If no artificial load had been created the server would have performed optimal. The user perceived latency of the non-Ajax application would then be very low, similar to the Ajax version of the application. This means the user activity would hardly be interrupted. See appendix A for a visual representation of the difference between user activity in Ajax and non-Ajax application models.

Optimal server performance is not a good representation of a real world example as you are not likely to be the only one using the server.

9.2 Results

Most of the expectations were actually met after conducting the usability tests. Some observations were not anticipated beforehand, but could have been expected. Experienced users (mostly having a technical background and being regular users of the Internet) for example proved to make no or hardly any errors and would carry out the tasks without any trouble. This was true for both types of the application. Less experienced users (who occasionally use the Internet) proved to make more errors, but very evenly spread between the two versions of the applications. With these users the recovery time between errors was far greater with the Ajax version of the application, resulting in a longer time to complete all the tasks. The recovery time between errors with the non-Ajax version of the application was shorter, probably because users are more familiar with this kind of application and the available (limited) user controls.

First	TTC1	Errors	Frust.	TTC2	Errors	Frust.	Pref.	Rep.	Ratio
2	105	1	0	180	0	0	1	-	1.71
1	249	2	1	234	3	1	1	116	0.940
1	268	2	1	247	2	1	1	161	0.922
2	213	1	1	377	1	1	1	133	1.77
2	113	0	0	200	2	1	1	-	1.77
1	170	0	0	273	2	1	1	-	1.61
2	134	1	0	199	1	1	1	-	1.49
2	128	0	0	212	0	0	1	-	1.66
1	165	1	0	274	2	1	1	-	1.66
1	230	2	1	244	2	1	1	175	1.06

Figure 5: measurements from usability study

First	Which version of the application was used first, 1 being Ajax, 2 being non-Ajax.
TTC1	Total time to completion in seconds of the tasks for version 1.
TTC2	Total time to completion in seconds of the tasks for version 2.
Errors	Number of errors (mistakes) made by the user. This can be clicking on the wrong elements without anything happening or using controls the wrong way.
Frust.	Times the user expressed his or her frustration with the application.
Pref.	The preferred version of the application.
Rep.	Total time in seconds to complete version 1 afterwards, in a repeated try.
Ratio	How many times faster the tasks completed in version 1 compared to version 2, in the initial run.

Figure 6: legend for usability study measurements

To rule out a possible advantage of knowing how the application works after one version of the application half of the users performed the tasks with the Ajax version first while the other half started with the non-Ajax version. After completing the tasks with both versions of the application the users were asked which version they preferred and why. Although initially some users actually needed more time to complete the tasks in the Ajax version, all users preferred the Ajax version of the application. When asked why they preferred it, it came down to the high responsiveness and the advanced controls (drag-and-drop, in place editing) of the application. Some users claimed to be more familiar with drag-and-drop although they could not name a single website that used it. They intuitively performed the operations they recognized from regular desktop applications.

Users that suffered from the long recovery time in the Ajax version were afterwards asked to perform the tasks again now that they knew how it worked. They would then perform the tasks somewhere between one and a half to and three times faster with the Ajax version. Users that did not suffer from errors or quickly recovered from them performed the tasks one and a half to two times faster with the Ajax version.

9.3 Analysis

The preliminary conclusion is that Ajax really does help create a better user experience as all users preferred the Ajax version and in terms of productivity it offers a significant improvement. But is it realistic to draw this conclusion? Wasn't just the responsiveness of the server measured instead of the effects of Ajax?

Of course the responsiveness of the server plays a part in performing the tasks but this is one of the key factors of using Ajax. By using Ajax you can actually decouple the user interface from the server, making the responsiveness of the server far less important for the performance of the application. Everything that needs to be sent to the server is done from JavaScript and doesn't interrupt the workflow. A user can continue to do his or her task while the results might come in later. With a non-Ajax application the user will have to wait for the server until the new page appears. The server response will affect the total time to complete the tasks but in fact these results are realistic and better represent a real world situation for comparing.

Recovery times from errors were not recorded. It would be interesting to see if the same users would recover more quickly from errors in another test session. But as I did not test the memorability of the applications, the recovery times were only an interesting observation.

9.4 Conclusion

Ajax does indeed increase the usability because of two things. One is the higher responsiveness of the application, which is made largely independent of server performance. The other thing is the more advanced user controls that provide better learnability and efficiency. The results show that all of the 4 quality components that were measured have improved compared to the non-Ajax version of the application, but most significant is the improvement in efficiency.

One might argue that using a larger group of test subjects would affect the results. I chose to follow Jakob Nielsen's suggestion of using 5 subjects for qualitative testing (5 for both versions of the application tested first). With more test subjects you might find only marginally more problems, errors and different times to complete the tasks. The results of testing with more than 5 subjects don't justify the investment in time or money you make.

10 Conclusion and final thoughts

So let's recapitulate, what are the effects on performance and usability? The effect on performance was measured in two ways: network traffic and server load. Chapter 7 shows that using Ajax for incremental page updates can lead to very significant savings in network traffic, depending on the amount of page content that needs to be updated. The server load is also reduced because of the reduced complexity of the incremental updates. This is shown in chapter 8. The absolute savings in server load increase as more pages of the same type are being requested in sequence, as shown in Figure 2. Finally the effect on usability is examined in chapter 9. All test users said they preferred the Ajax version of the Listmaker application above the non-Ajax version. The main reasons for their preference were the advanced user controls made available through Prototype and the very fast response times of the application. The users could complete their tasks more quickly and thus became more productive.

At first the title of this thesis might seem a little unorthodox: "The effect of Ajax on performance and usability in web environments". Why research the effect of performance *and* usability, what have they got to do with each other? There is a relation between the two, which I will explain using the data collected earlier.

The number of page views increased when Ajax was used, this would automatically mean more load on the servers. They were not introduced because of the use of Ajax, but because users simply requested more pages. The servers are already under heavy load, so where does the extra capacity come from? The results on the server load indicate that significantly less time is required to process the incremental updates, allowing for more pages to be served. Why were there more page views anyway? It might just be the increased usability of the Ajax pages. Users feel more comfortable, explore more and thus request more pages (updates). Both the better response times of Ajax pages and the advanced user controls available contribute to this increase.

We can start from another perspective as well. If we would say that Ajax improves the usability, can we prove it? Yes, looking at the number of pages requested users do more in the same time. This is an increase in efficiency and productivity. The savings in server load compensate for the extra requests being generated to keep things in balance.

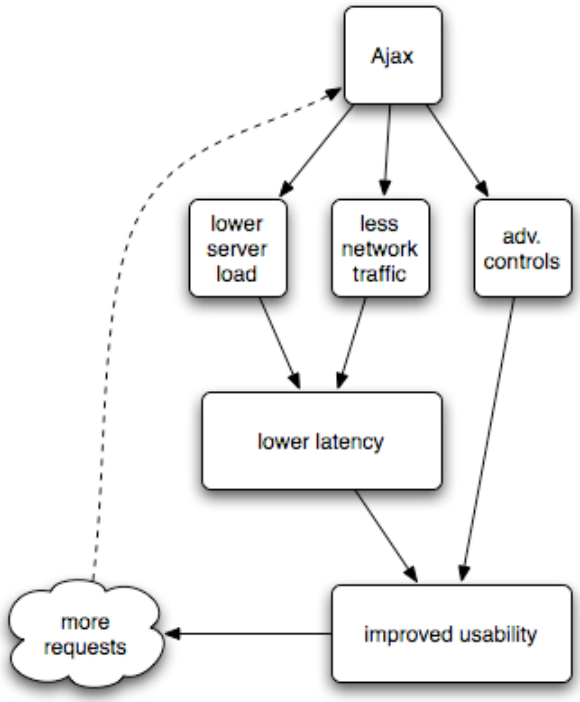


Figure 7: Ajax / usability diagram

Figure 7 illustrates this nearly circular behaviour, although it's very hard to connect all the lines correctly. We have seen that Ajax has positive effects on server load, network traffic and user controls. The lower server load and less network traffic directly affect the latency and indirectly has a positive effects on the usability. The advanced user controls directly affect the usability. This improved usability leads to more page requests and updates. More requests would traditionally lead to more load on the server and more network traffic, but since we're using Ajax these are heavily reduced. This last part is where the diagram might lead to wrong conclusions. It should *not* suggest that your server load and network traffic keep reducing *because* you generate more requests.

It's almost like an ecosystem where everything is related, if one thing changes other things change too to compensate. In a real ecosystem, the system only works in a large enough environment, not in an ant-farm. The same goes for websites. In a small website it probably doesn't really matter if you have generated a bit more data or server load. In very large web environments every request is done millions of times so it is important to keep the system in balance.

11 Future work

11.1 Testing Ajax

An important aspect of programming is writing tests for your software. For developers one of the most important forms of testing is unit testing. When testing Ajax applications you are not only testing the software but also the implementation of the JavaScript environment. The testability of Ajax applications is not one of the topics covered in this thesis but it should get more attention in general. For a typical application you should test the remote server, the functional aspects, visual aspects and the JavaScript engine. One testing framework probably can't do it all. Currently there are a few promising options to do functional testing. One is Ghost Train, from the developer of Script.aculo.us. It is a sort of macro recorder, to check application functions. Although there is only a very early alpha release that works only under FireFox it seems like a very convenient solution to functional testing. Another is a port of JUnit to JavaScript, which allows for unit testing in JavaScript, called JsUnit [8].

11.2 The future of Ajax

Ajax is changing the web as we see and use it, but on the web technologies come and go. Although Ajax is only at the beginning of its up rise, it is likely that it will be surpassed in the future. Because of the short living HTTP request nature of browsers and web servers everything is done in a kind of polling based fashion. There is nothing wrong with this but it sets limitations to the types of applications that can be realised and their efficiency. The next big thing on the web could be a technology dubbed "Comet". Comet uses long living HTTP connections that are initialized by the client but are kept alive. Because of this, the server can easily push data to a client. On the other hand this poses a real threat to server scalability as connections are kept alive. Currently Comet is only available for those who want to use bleeding edge technology, as not many web servers currently support pushing. Also certain modifications or libraries have to be added to the web server. The latest version of the Apache HTTP Server (2.2) does support Comet and the Dojo Toolkit also offers some support already. For a comparison of the Ajax and Comet application models, see Appendix A.

12 Bibliography

[1] Dave Crane, Eric Pascarello, Darren James, "Ajax in Action", Manning Press, January, 2006.

One of the first books on Ajax, recommended by many developers. It focuses on the software architecture and design patterns, like in other programming languages, for creating Ajax applications in the web browser. JavaScript doesn't natively support programming principles like for example information hiding and inheritance. The book gives solutions on how to face these problems.

[2] David Flanagan, "JavaScript: The Definitive Guide, 4th Edition", O'Reilly Media, Inc, December, 2001.

This is the standard work on JavaScript, JavaScript as implemented in the different browsers to be more specific. When reading this book it becomes more obvious that JavaScript is in fact a real programming language and not a seriously flawed one.

[3] Thomas Fuchs, "Audible Ajax Episode 12: Thomas Fuchs of Script.aculo.us", ajaxian.com, January, 2006.
http://media.ajaxian.com/podcasts/audibleajax-show-12-interview-ThomasFuchs.mp3

In this podcast Thomas Fuchs, the developer of Script.aculo.us, describes the different ways of content manipulation and data transfer (JSON, XML, HTML) with Ajax. He comments on performance of different ways of content manipulation and the future of Script.aculo.us. Script.aculo.us was originally developed for use in a commercial product but is now available as open source software.

[4] Thomas Fuchs, "script.aculo.us", http://script.aculo.us, January, 2006.

All information on the effects and controls of Script.aculo.us comes from this website. It has documentation and examples.

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Pearson Professional Education, 1995.

A well-known standard work on design patterns, used as a reference for developing JavaScript applications.

[6] Jesse James Garrett, "Ajax: A New Approach to Web Applications", Adaptivepath.com, February, 2005.
http://www.adaptivepath.com/publications/essays/archives/000385.php

Jesse James Garrett is the person who first coined the name Ajax and caused the rapid growth in popularity and press attention of Ajax. This article discusses the workings of Ajax as opposed to the traditional web model and its inception.

[7] Danny Goodman, “Object Detection”, O’Reilly, October, 2001, http://www.oreillynet.com/pub/a/javascript/synd/2001/10/23/ob_detect.html

To ensure cross browser compatibility you need to know what browser your script is running in and what features are supported. The easiest way to achieve this is to check for the browser name and version, but this is not fool proof. Object detection is a different approach to check for the actual existence of features. This method has a better potential for surviving when new browsers or versions of browser are released.

[8] Edward Hieatt, Robert Mee, “Going Faster: Testing The Web Application”, IEEE Software Engineering Journal, pp. 60-65, March/April, 2002.

In the development of software for the desktop, testing is a structural part of the process while in web development testing has less priority, especially on the client side. This article discusses both client and server side testing for web applications. For client side testing of JavaScript JsUnit is being suggested. JsUnit is a port of JUnit to JavaScript, Edward Hieatt is one of the authors of JsUnit.

[9] Laurens Holst, “Bookmarks and the back button in AJAX applications”, Backbase.com, August, 2005.
http://www.backbase.com/go/dev/tech/002_bookmarks.php

In traditional websites every page has a unique URL, but in Ajax applications at least a part of the page will be generated by the client, without changing the URL. This poses a problem for bookmarking a page and navigating with the back button. This article discusses these issues and provides solutions for the Backbase framework. These solutions can be translated to other frameworks in order to keep the bookmarking ability and the correct functioning of the back button.

[10] Peter-Paul Koch, “Benchmark - W3C DOM vs. innerHTML”, http://www.quirksmode.org, Augustus, 2005. As seen on February 10th, 2006.
http://www.quirksmode.org/index.html?/dom/innerhtml.html

This article has benchmarks on different ways of content manipulation. The most important are manipulations through the W3C DOM and the innerHTML property. Directly setting the innerHTML property with prepared HTML is the fastest way of manipulating data, according to these benchmarks.

[11] Steve McConnell, “Code Complete: A Practical Handbook of Software Construction”, Microsoft Press, 2004.

A practical work on software development that covers many aspects like testing, design patterns, code standards and maintenance.

[12] James Mc Parlane, “Why I Don't Use The prototype.js JavaScript Library”, http://blog.metawrap.com/blog/CommentView,guid,42b961d5-b539-4d9a-b1e0-108e546ae3e6.aspx, December 29th, 2005.

Comments on why extending the standard JavaScript objects is bad. By understanding the impact of using Prototype, one can make a better judgement how it will affect existing code.

[13] Christopher L. Merrill, "Performance Impacts of AJAX Development: Using AJAX to Improve the Bandwidth Performance of Web Applications", Web Performance Inc., 15 January, 2006.

This article describes the possible savings of network bandwidth with Ajax applications and how this can be realized. The modifications in the case study are based on the findings in this article.

[14] H. Mili, M. Fayad, D. Brugali, D. Hamu, D. Dori, "Enterprise Frameworks: issues and research directions", Software—Practice & Experience Volume 32 , Issue 8, July 2002, Pages 801-831

[15] Jakob Nielsen, "Usability 101: Introduction to Usability", http://www.useit.com/alertbox/20030825.html, August 25th, 2003

As the title suggests, it provides a quick and brief introduction to what usability is.

[16] Jakob Nielsen, "Usability Engineering, Chapter 5: Usability Heuristics & Chapter 6: Usability Testing", Morgan Kaufmann, San Francisco, 1994.

Chapter 5 describes the common rules on response times of applications and what the users' expectations are. When the amount of time before a response is expected to be very high or varies strongly, the importance of visual indicators or progress bars is stressed. Chapter 6 describes the testing and usability of applications, how tests should be conducted, what should be measured during the test and what test-subjects should be used.

[17] Linda Dailey Paulson. "Building Rich Web Applications with Ajax", Computer (IEEE Computer Society), vol. 38, no. 10, pp. 14-17, October, 2005.

This essay discusses the technologies used for Ajax and suggests alternatives for creating rich internet applications. She also provides certain pro and con arguments on using Ajax.

[18] Sergio Pereira, "Developer Notes for prototype.js", February, 2006. http://www.sergiopereira.com/articles/prototype.js.html

The original documentation of Prototype is virtually non-existent and therefore third party documentation is essential. Sergio Pereira has made a very extensive reference on Prototype that is very valuable and is widely seen as the starting point for any developer starting with Prototype.

[19] Gary Pollice, "A look at aspect-oriented programming", Worcester Polytechnic Institute, February 2004.
http://www-128.ibm.com/developerworks/rational/library/2782.html

The 'best practice' in JavaScript to add event handlers to elements is to do this in a non-obtrusive way. This is very much like aspect oriented programming and most Ajax frameworks have a similar solution. This article describes AOP for Java but the principles are applicable to event handling in JavaScript.

[20] Alex Russell, "Dojo documentation", Dojo: the browser toolkit, January, 2006.
http://dojotoolkit.org/docs/

This is the official website of the Dojo Toolkit. It offers a roadmap, examples and some documentation. There is not so much information on Dojo apart from this website.

[21] Ben Schneiderman, "Response Time and Display Rate in Human Performance with Computers", Department of Computer Science, University of Maryland, College Park, September, 1984.

This researches the effect of varying response times on the reaction of users when dealing with human computer interaction. Response times of less than one second are desirable and productivity tends to increase with lower response times. With very short or very long response times, users tend to make wrong decisions. Ajax certainly has an effect on response times and therefore you must know what the effect is on the user.

[22] Leland Scott, "Ajax/DHTML Library Scorecard: How Cross Platform Are They?", http://www.musingsfrommars.org/2006/03/ajax-dhtml-library-scorecard.html, March 4th 2006.

This website keeps track of most Ajax frameworks in a broad sense. It also grades them based on browser compatibility. Frameworks are graded from A (most compatible) to E (least compatible), and is frequently updated.

[23] Aaron Swartz, "A Brief History of Ajax", http://www.aaronsw.com/weblog/ajaxhistory, December 22nd 2005.

This article describes the history of Ajax, beginning with the implementation of the ActiveX component in Internet Explorer 5.

[24] Lauren Wood, "Programming the web: The W3C Dom Specification", IEEE Computer Society, January, 1999.

Every web page is represented by a document object model (DOM) and can be accessed through JavaScript. When writing Ajax application you will have to deal with manipulation of the DOM. This article describes the history of the DOM and how to perform modifications to it from JavaScript.

13 Appendices

13.1 Appendix A: Classic, Ajax and Comet application models

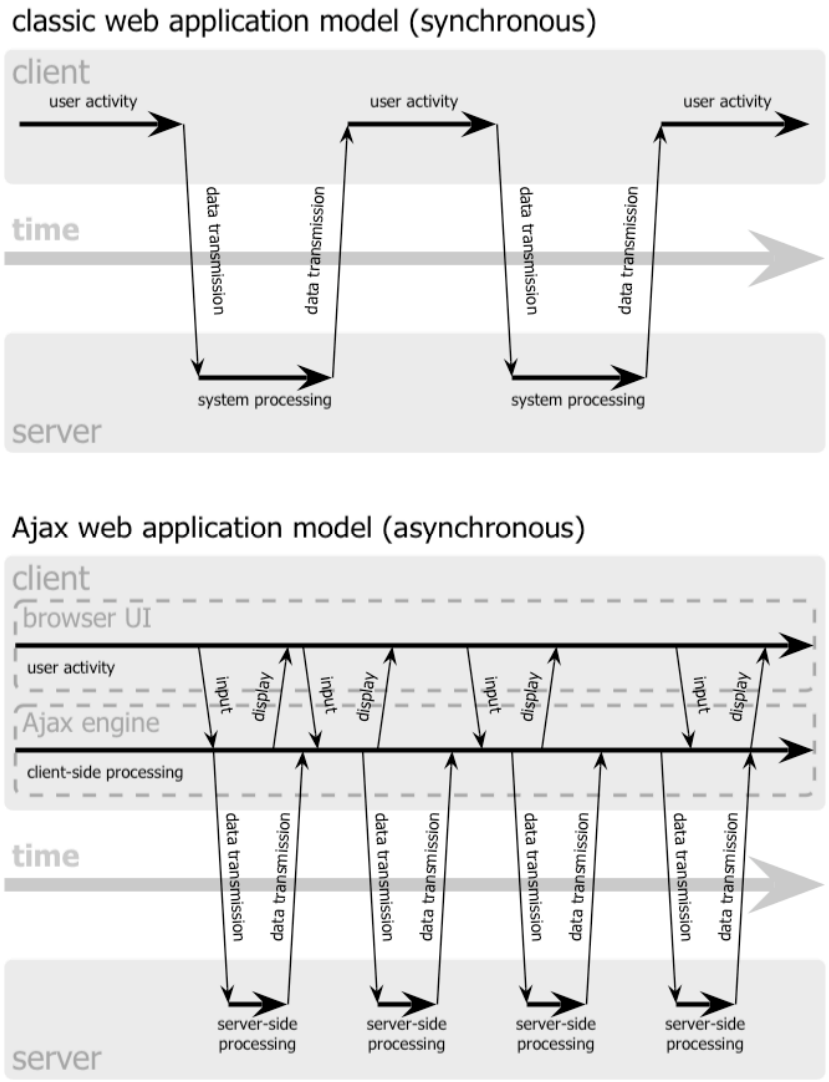


Figure 8: Classic versus Ajax application model

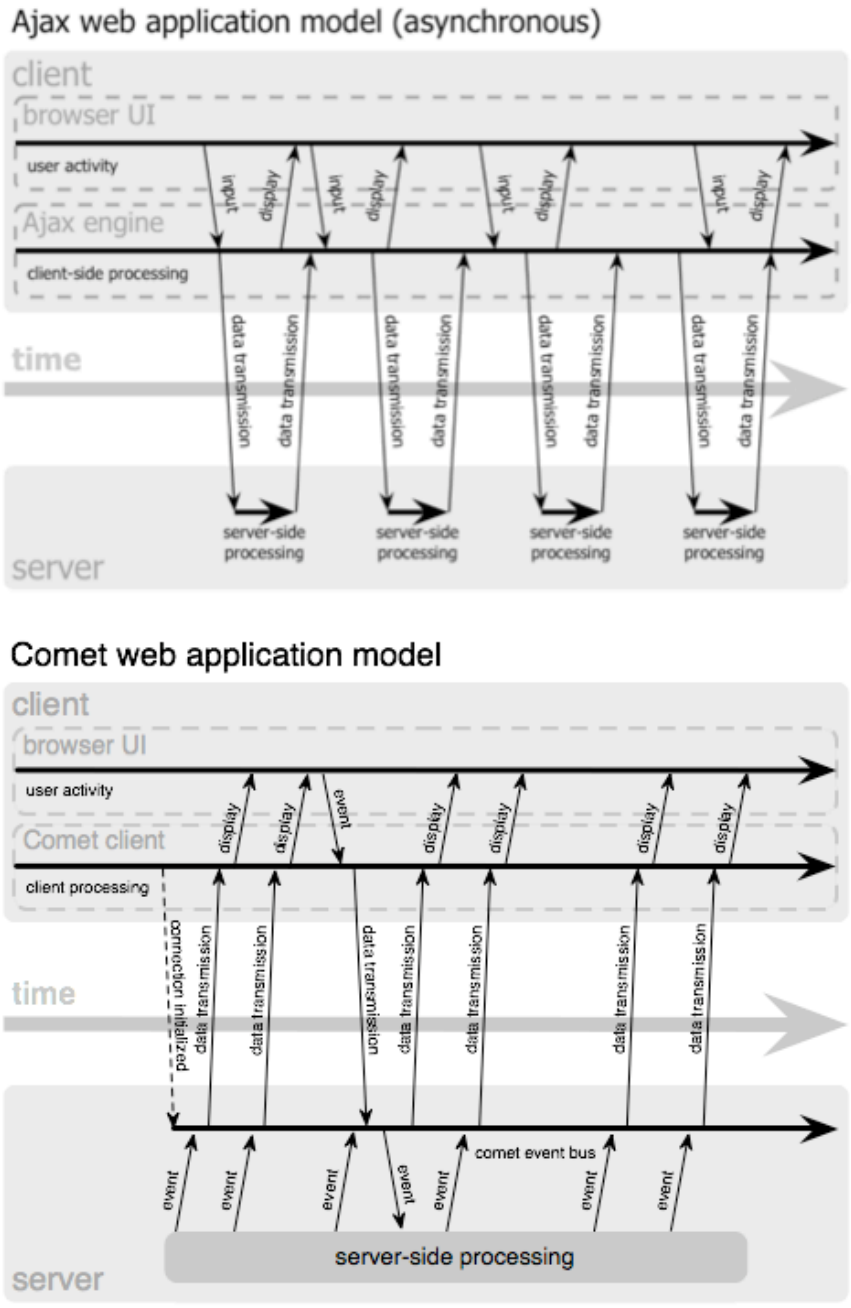


Figure 9: Ajax versus Comet application model

13.2 Appendix B: Prototype's object extension

Prototype extends the standard JavaScript objects, which may cause existing code to break. Below is an example.

First, we initialize some variables:

```
var sourceArray = [];
sourceArray.push(0);
sourceArray.push(1);
sourceArray.push(2);
var listOfElements = "";
```

The pass condition of the listOfElements variable is:

```
listOfElements = "0 1 2 "
```

The "for(i in object)" control structure:

```
for(var arrayElement in sourceArray){
      listOfElements += arrayElement + " ";
}
```

Results in:

```
listOfElements = "each all any collect detect findAll grep include
inject invoke max min partition pluck reject sortBy toArray zip inspect
map find select member entries _reverse _each clear first last compact
flatten without indexOf shift 0 1 2 "
```

The alternative:

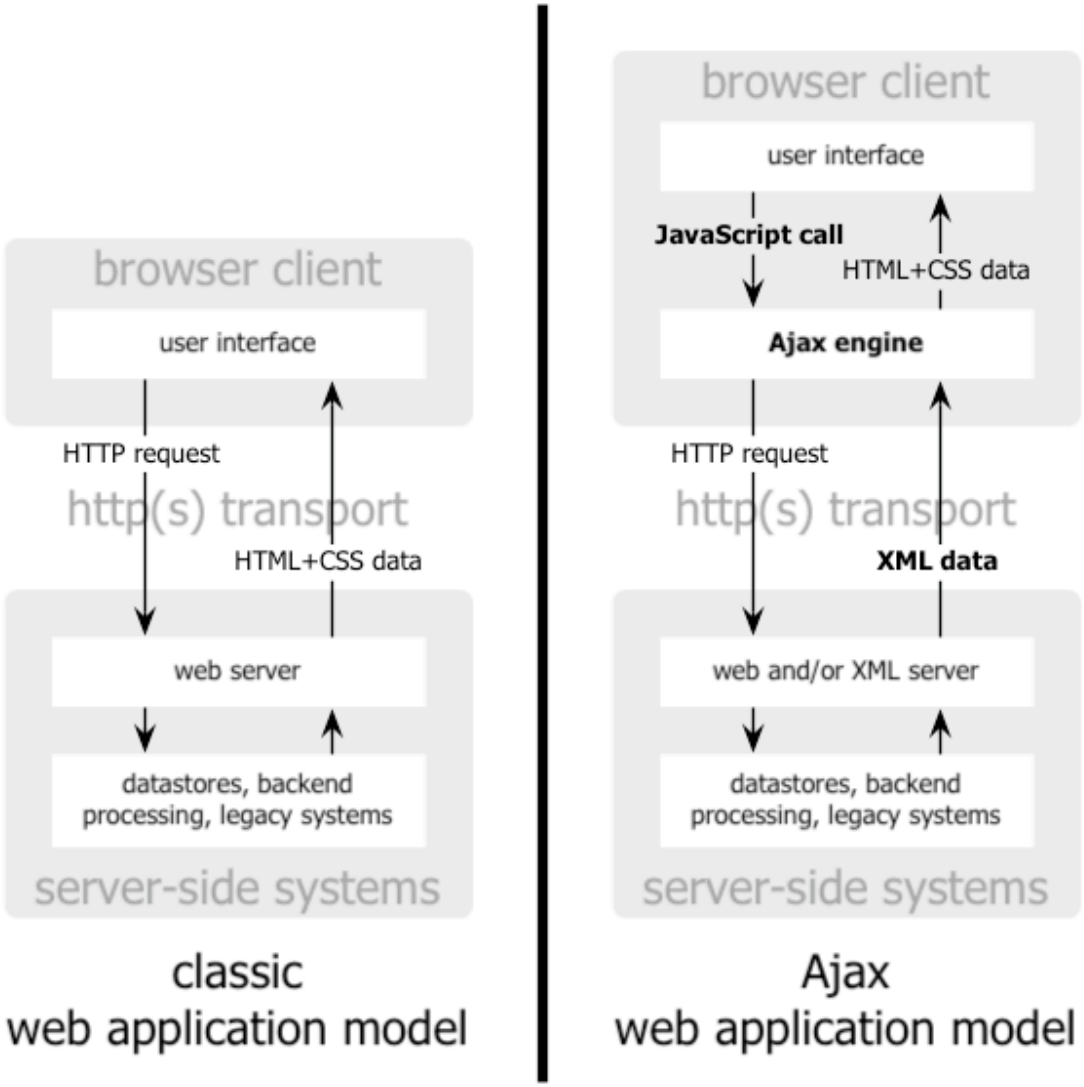
```
for(var i = 0; i < sourceArray.length; i++){
      listOfElements += sourceArray[i] + " ";
}
```

Results in:

```
listOfElements = "0 1 2 "
```

So with a simple rewrite of the control structure you can use Prototype without any trouble, this was also the case with the case study.

13.3 Appendix C: Classic versus Ajax application model



14 Glossary

AOP
: Aspect Oriented Programming. An approach to solve cross cutting concerns in software development.

CSS
: Cascading Style Sheet. A selecting and styling language mostly used to define layout and style of web pages.

DOM
: Document Object Model, a tree like object oriented representation of web pages as interpreted by the browser. The DOM allows for developers to interact with the web pages and make client side modifications.

RIA
: Rich Internet Application, a fully featured software package that runs in a browser.

Web 2.0
: A paradigm shift on the Internet where the focus moves towards Ajax, democracy and user centeredness. See http://www.paulgraham.com/web20.html

Widget
: A basic building block for creating user interfaces. Often used to reference user controls such as buttons, check and radio buttons, drop down lists and sliders.