AJAX:
A Methodology for Delivering Web Applications Via A Browser with Near Desktop Efficiency

By

Michael Lajoie

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Computer Science

Algoma University College
Sault Ste. Marie, Ontario
April 2007

ABSTRACT

In February 2005, Jesse James Garrett gave a methodology, slowly growing in popularity, a tremendous kick-start into high gear by providing it with a catchy name in the form of AJAX. Short for Asynchronous JavaScript and XML, the name encapsulated some of the key aspects of a methodology with the promise of creating web applications with the look, functionality, and near-power of traditional desktop applications. The Ajax methodology holds the promise of enhancing the users overall experience and productivity while adding value to the organization.

One of the most appealing aspects of Ajax is its ability to achieve its objectives using only the technologies included in every modern browser. It relies on JavaScript, DOM events, XHTML, CSS, and asynchronous communications using the HTTP protocol. All of these technologies are present in every modern browser and have been present since 2000.

This thesis will present a survey of the Ajax methodologies including: its advantages and disadvantages; some of the issues associated with its use; and a survey of the support currently available for developing applications using the methodology.

ACKNOWLEDGEMENTS

I wish to extend a heart-felt thank-you to my thesis supervisor, Dr. Yi Feng. Her encouragement, guidance, advice and input contributed significantly to the completion of this thesis. Additionally, I would like to thank Dr. George Townsend for consenting to be my second reader for this thesis, and for all his support as the faculty supervisor of the thesis class this year.

I would also like to express my gratitude to Dr. Jay Rajnovich for his encouragement and guidance when I initially approached him for advice about pursuing this degree in Computer Science. The entire Computer Science and Mathematics department has been nothing but supportive and I have enjoyed the entire experience. And to my other professors and instructors, I want to extend a great big "Thank-you", in particular: Sofia Silberberg, Simon Xu, Danny Reid, Klaus Peltsch, Charlie Whitfield, and Gerry Davies.

Thank-you everyone.

TABLE OF CONTENTS:

LIST OF FIGURES AND CODE

CHAPTER 1 - Introduction

1.1 - What is AJAX?

AJAX – an acronym first coined in Feb 2005 by Jesse James Garrett of Adaptive Path Inc. [23] – stands for Asynchronous JavaScript and XML. The name is somewhat misleading as it is so much more than just JavaScript and XML; it's a blending of long established technologies, scripting languages, and data formatting practices into a methodology that addresses several shortcomings of the traditional approach to web site / web application development. Some of the issues addressed include:

a) Eliminating the need to constantly refresh the web page by reloading a new version with every processing change. Using AJAX, portions of a web page can be refreshed without reloading the entire page.

b) Improving web application performance, which has been traditionally been too slow for most uses. Using the AJAX methodology, many web applications can now be delivered at speeds approaching desktop applications

c) Providing much needed web standards. This is in part the drive behind Web 2.0 – a term that collectively encompasses all of the new emerging technologies and methodologies aimed at improving the web experience for users and adding value for the organizations who own them.

d) Implementation of rich controls like those found in desktop applications. The lack of controls similar to those found in desktop applications has long been an obstacle to developers wanting to deliver feature packed web applications. The conventional form-submit-refresh approach to web development was incapable of adequately replacing complex grid controls and sortable lists common in many applications. HTML was never designed to supply such functions and features. With the advent of Ajax, there is an abundance of controls now available in many of the 100+ AJAX frameworks currently on the market.

e) Reducing the processing demands on the server and communications bandwidth requirements of traditional web applications. This is one of the key areas where the Ajax methodology has the potential to shine – and one of the principle reasons its adoption and popularity has escalated rapidly in recent years. Ajax encourages the distribution of processing requirements between the client and the server and reducing bandwidth by requiring only new content and results to be returned by the server. Both of these changes reduce the burden on application servers.

AJAX is a combination of many well-established technologies and programming languages. For the most part it includes: JavaScript used for client side processing and manipulation of the Document Object Model (DOM) which basically means the HTML code of a web page, some method of asynchronous communications, and server applications capable of returning text-based messages using the HTTP protocol. By far, the most popular method for achieving asynchronous communications between the browser and the application server is the XMLHttpRequest Object but other methods exist including the use of hidden frame pages and iframe elements. [12]

Additionally, the use of Cascading Style Sheets (CSS) play an important roll in the development of nearly all Ajax applications by providing an efficient means of managing content formatting and style – but their use is not required. For the server side programming, Ajax is very accommodating. The only requirement is that the communications between the browser and the server occurs using the HTTP protocol – mandated because browsers understand and communicate using this protocol. In light of this, the application server can use any programming language which can communicate using HTTP and so far language such as Java, PHP, .NET, C, Perl, Python, C++ have been used to build the server-side of Ajax applications. This openness makes it easy for developers to adopt this methodology while retaining a lot of the skills and knowledge they have acquired developing more traditional web implementations.

1.2 - Web 2.0 and Ajax's Role:

Wikipedia.org defines Web 2.0 as "a term often applied to a perceived ongoing transition of the World Wide Web from a collection of websites to a full-fledged computing platform serving web applications to end users. Ultimately Web 2.0 services are expected to replace desktop computing applications for many purposes." In a nutshell, Web 2.0 is about improving the users experience and adding value to the organizations behind the web sites. Standards are at the core of Web 2.0 – standards about the proper use of XHTML, about the integration of other technologies into web sites, about the application interfaces, about browser-server interaction, and about the overall functionality of this new breed of web applications. How many and what kinds of applications will find a suitable home on the internet has yet to be seen – but Web 2.0 is about finding a way to deliver whatever those applications may be in an appropriately thought out manner.

Like Web 2.0, Ajax is a methodology and not a technology. Additionally, neither Web 2.0 nor Ajax try to limit themselves to a predefined set of technologies but are open to the use of anything and everything that will help achieve their goals. And like Web 2.0, Ajax is also focused on improving the users experience and increasing value to the web site owners.

While the Web 2.0 is more of a generalized ideal of what developers are looking for, Ajax is a more specific method of implementing the objective. At its core, Ajax is an open methodology – having really only one named required component in the form of JavaScript – and even this technology could be replaced should a suitable alternative emerge. Ajax is not married to a specific technology – but rather to a methodology. Its about asynchronous communications between the browser and an application server whereby only the rested information, and not an entire web page, is passed between the two, and where client side processing manages the interaction by altering the current web page in order to provided the required content or results.

As one of the Web 2.0 proposed implementations, Ajax stands out in a couple of ways from the alternatives. First, it is based on what already exists in practically every browser – JavaScript, XHTML, Cascading Style Sheets, and some form of the XMLHttpRequest object. Second, Ajax is not tied to any proprietary technology and is therefore not driven by any single advocate. Given the current Open Source movement, this feature is extremely appealing to many technology groups who see Ajax as friendly way for everyone to work towards a common goal. As a result, there has been an explosion in support for Ajax from both commercial and open source organizations.

1.3 - Characteristics of Ajax Applications:

It is important to understand that the definition of an Ajax application is not a precise collection of technologies applied in a well-established manner – but rather a fluid application of any number of technologies from a highly flexible and expanding pool of resources. Ajax applications, may or may not include server interaction, and may or may not include DOM interaction. Ajax definition encompasses a spectrum of applications ranging from a page with a Flash widget built into part of it (and thus Flash providing any client/server interaction) at one end all the way to web sites that incorporate a heavy use of server based databases and utilize JavaScript to manipulate all aspects of the web pages through manipulation of the DOM.

With consideration of the aforementioned, the characteristics of an Ajax application [09] generally include:

a) It is about Applications, not Web Pages: In the beginning, there was the web page – a static representation of information, which, if the user was fortunate and the author skilled, was presented in a pleasant format. But we did not care – the web was young and we were amazed at how much information was now available at our fingertips. But times have changed, and users expect more from a website these days than just static information pages. They want services; they want to interact; and they want to be engaged; and in some cases, they want to be entertained. Developers formally accustomed to producing web pages are now being told that is about web applications – and sites are evolving into an interactive environment.

b) Smooth, Continuous Interaction: Traditional web applications involved a fill in the form, submit to the server, wait for results, present a full page refresh as the response cycle. Since this was a step up from static pages, users accepted this as better because it offered some interaction. But as a replacement for traditional desktop applications, this form of service was frustrating to use and plagued with efficiency issues. Of course, once users were exposed to the smooth and continuous interaction offered by Flash and Ajax interfaces, their expectations were elevated. Ajax, through the use of asynchronous communications, enables users to see results without a full-page refresh.

Additionally, users are left free to continue working while those results are being obtained from the application server.

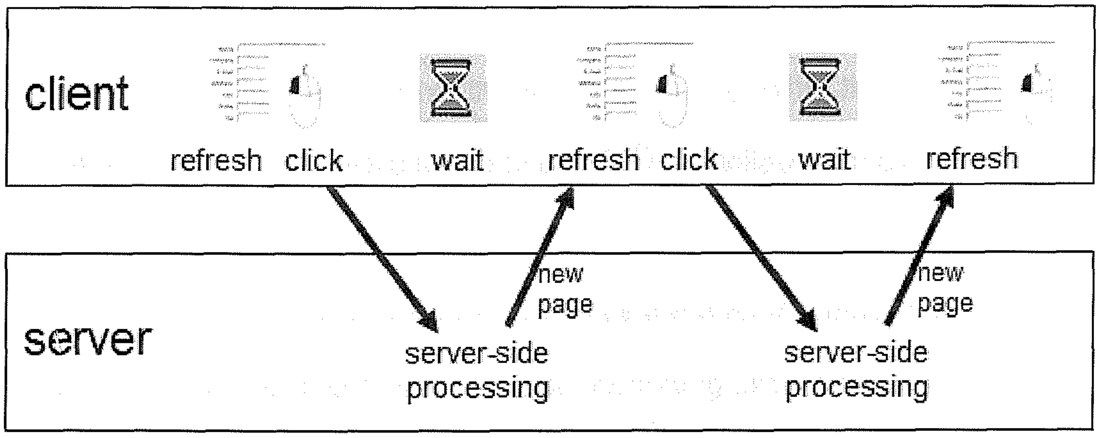

Figure 1.1 – Traditional Web Application Client-Serer Communications – new page every time.

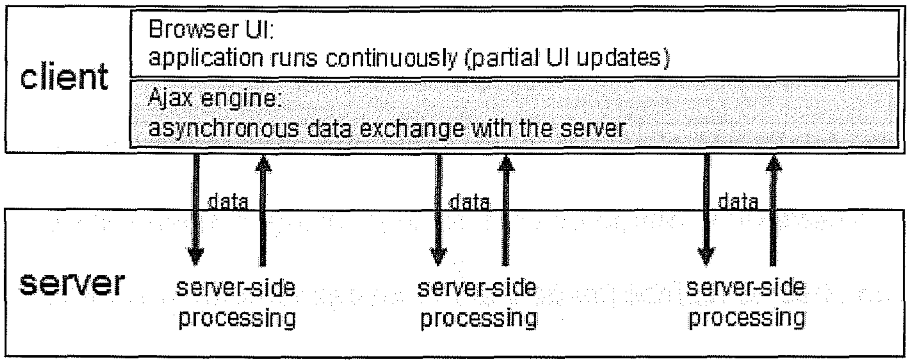

Figure 1.2 – Ajax Web Application Client-Server Communications – only data exchanged.

c) Live – With Ajax, the application no longer has to wait for the user to specifically initiate communications with the server – a process typically occurring in response to a submit button click. Communications can now occur as a result of simple user interaction such as behind the scenes population of a control in response to an entry in another field – or timed JavaScript events, which seek to periodically poll the server for updates such as news feeds or new data related to the currently displayed information.

d) Supportive – Ajax apps have the ability to monitor what the user is doing an initiate functionality designed to assist the user without an explicit request. Examples include the implementation of a suggestion feature on a search box which provides a list of possible topics as the user is typing based on the keystrokes already entered, or simply validating entries after each field has been populated so as to keep the user informed as he progresses through the form rather than waiting until a submission request is made.

e) Visual Effects – Ajax varies slightly from desktop applications in that many sights rely on animations and other visual effects to convey information to the user. Such effects might include causing an control to momentarily glow as result of a change in another aspect of the web application so as to draw the users attention to something which has been effected by the recent change.

f) New Widgets – Ajax does not rely on the old HTML control standards for building interfaces. Instead, a wide range of new controls, developed to mimic traditional desktop controls, have been introduced by 100's of companies and organizations. It is not uncommon for Ajax applications to present data in grid controls which offer sorting and editing features as opposed to the traditional use of tables, input boxes, and buttons.

g) New Styles of Interactions – Many features previously available only in desktop environments are now finding their way into web applications – particularly Ajax apps. Examples include Drag and Drop capabilities as well as extremely powerful grid controls.

h) Standards Based – Ajax applications rely on standard browser features that have been present for many years and were common place in still oft used older browsers. By avoiding browser specific features, Ajax attempts to maintain a high degree of environment compatibility allowing developers to more easily build for a wide audience.

1.4 - Advantages and Disadvantages:

"(AJAX) has not only helped revolutionize the look of Web applications, but also enhance usability of such applications..." [21]. Many mid to large size corporations are including Ajax capabilities into their intranet products. By allowing for the inclusion of sophisticated features, intranets are now not only helping to express the company's culture, but also servicing the needs of individual employees by providing such features as: customizing the colour schemes and fonts; allowing control over how information is presented and organized within the browser window; and allowing for the control and selection of what information is being presented as opposed to what has to be navigated to. Ajax is allowing for individual customization in a way that emphasizes utility and pragmatism of these intranet products.

Ajax is offering a large number of advantages, some of which are captures by the following [09] [12] [21] [23]:

a) As mentioned several times previously, Ajax offers a smooth and continuous interactive environment for the user when dealing with server communications. The page by page approach of traditional web applications is a more abrupt and fragmented experience as compared to changing just a portion of the page without reloading under the Ajax approach.

b) "AJAX is raising users' baseline expectation for the usability and responsiveness of Web applications..." This in turns makes collaboration amongst designers, developers and architects more important – and more communication should lead to better final products and a better overall experience for users.

c) The browser has traditionally functioned as a semi-smart "dumb terminal"; AJAX upgrades the browser to a lightweight application container.

d) Use of Ajax opens the eyes of developers to the possibilities of offering more extensive personal features to users – a task in the past considered to be far to demanding on the servers. The types of personalized features can include customized information delivery (getting the right information to the right people), dynamic content presentation (presenting information to different people in a format best suited for their needs), and making it easier to bring information to the user rather than making them navigate to it

e) Providing a more fluid experience to the user. This advantage is more about developing web applications which are more responsive to the actions of the user as they occur such as changes in one area automatically updating information in another, helpful suggestions based on current actions, or validating entries as they occur and not just when they are submitted. In effect, Ajax helps developers deliver web apps that behave in many ways like desktop apps.

f) Goodbye page flicker. Because AJAX moves web apps from the realm where submitting a form, clicking or following a link all meant a new page load to an asynchronous communication process whereby the DOM is manipulated rather than replaced, users are less frustrated by the wait and flicker interaction offered by traditional applications. Additionally, from an operations point of view, the elimination of full page reloads means less bandwidth used per server interaction. For highly used applications, this reduction can mean a significant savings and provide an opportunity to use the resources for service improvements.

g) Related to the two preceding items, a user can keep working while waiting for results from the server. Whether the server based communication involves validating some entry, or updating a record, an Ajax application will allow the user to continue working with the page while the browser waits for a reply to its recent server request. Ajax improves the efficiency of the user.

h) The majority of AJAX toolkits and frameworks are built around the open W3C standards that are supported by nearly all the popular browsers in use today. The foundation of the Ajax methodology is to build upon the broadly supported technologies and utilities what is standard in their implementations while avoiding what is specific to just a narrow offering. The advantage here relates to the broad compatibility of Ajax applications across a wide range of browsers, hardware and operating systems as well as the likely continued operability of current Ajax applications by future browsers.

i) AJAX can provide instantaneous feedback, using client side scripting that responds to changes as they occur. The validation can be handled in some cases strictly on the client side or can be passed to the server for validation asynchronously. An example of useful feedback would be a client side function which detects changes in a city / postal code combination and passes these values to a server side function which uses a large city information database to confirm that the postal code appears to be valid for the supplied city name. (ie for Sault Ste Marie valid postal codes start with P6A, P6B or P6C). With traditional web applications, such validation would not occur until the entire form was submitted. Even when some client side JavaScript validation was provided, this type of check, which involves a large amount of data to check many possible combinations, could not be reasonably handled on the client side. Ajax make client side validation both more responsive and more robust.

j) Ajax use can range from minor to extensive, as well as subtle to obvious. The users experience can be improved without the user even recognizing that AJAX is being employed.

k) The Ajax methodology is very open and encourages the inclusion of as many technologies as possible. This encourages development from a wide range of participants. The synergy of so many varied participants and supporters has resulted in a plethora of ideas, tools, and frameworks. In many ways, the openness of Ajax has been the fuel driving its growth in use and popularity.

Ajax is not without it detractions. Like every methodology, it has its weaknesses and some of the more obvious ones are listed below [09] [12] [21] [23]:

a) Ajax can be abused; it could create new problems as easily as it can solve old ones. SPAM and pop-up advertising are just two areas where Ajax may very well contribute to new issues. While the methodology provides new ways to achieve improvements for web applications, unscrupulous advertisers could exploit the same technologies for their advantage.

b) Many of the features of Ajax that appeal to designers and developers might result in distracting them from the real purpose of the project. We have seen this happen in the past where form over function became a big problem for many sites being developed. The fancy features of Ajax are meant to enhance the user experience and add value to the organizations, but if not implemented with the consideration of the overall purpose in mind, the resulting product could fail to satisfy the needs of the end user. This is of course a Software Engineering issue.

c) AJAX programming is complex and requires a broad range of understanding in many technologies. Developers need to be well versed in XHTML, CSS, JavaScript and some suitable language for building the server side application. Additionally, developers need to understand the principles of good programming design and software engineering concepts. This skill set requirements is well beyond those of the typical web designer and has been an impediment to Ajax developments to date.

d) Ajax implementations can be complex. Related to the point above, large Ajax applications are difficult to implement and maintain. It has been found that Ajax presently better to implemented as a gradual series of changes. In its current state of development, small Ajax applications and projects are more likely to be successful whereas may attempts to develop or convert larger applications have proven difficult to impossible. At the heart of this problem is a lack of available tools for appropriately managing the development of larger applications. While organizations like Google and Yahoo have achieved success with larger Ajax applications, they have also had the resources to develop the in-house tools required to properly management the development of these applications.

e) Lack of tools, which make development and debugging easy, are the two largest barriers to making AJAX mainstream. Related to the disadvantage above, this disadvantage actually relates to development at all levels. Integrated development environments and testing tools are still in the early stages and the few products that have been introduced are typically more useful for diagnosing issues rather than assisting in developing solutions. The current state of available tools has a direct impact on development time, testing and quality assurance – all of which is considerable.

f) Too many toolkits and frameworks. Because Ajax is still a relatively new methodology and with the explosion in contenders who have entered this market space with product offerings, a market shake down has not yet happened. As such, there is no clear indication as to which toolkits are likely to be supported in the future and which will disappear into obscurity.

g) Ajax presently has no standard definition of what the User Interface (UI) controls should look like. As a result, grid controls, sliders, calendars, etc… all vary in appearance from one toolkit to another. While the objective of the OpenAjax

Alliance is to create an environment where controls from the various frameworks will work in harmony within the same application and/or web page, the lack of standards with regards to the appearance of the UI components will result in applications looking pieced together rather than harmonious – or rely on the developers to make modifications to controls in order to achieve a uniform appearance.

1.5 - OpenAJAX Alliance – Providing Standards and Education:

On February 1st, 2006, less than one year after Jesse James Garrett first coined the term Ajax [23], fifteen industry leaders, including such notable names as Borland, Eclipse, Google, IBM, Novell, Oracle, Red Hat and Yahoo, announced the formation of the OpenAjax Initiative [19]. Renamed as the OpenAjax Alliance (OAA) at the official kick-off meeting in May 2006, the organization is dedicated to the successful adoption of open and interoperable Ajax-based technologies. The primary object of the Alliance is to provide the support needed to insure customer success in developing Ajax projects. The Alliance hopes to foster a rapid acceptance of the Ajax methodology by promoting the ability to mix-and-match solutions from a variety of Ajax technology vendors and actively encouraging the rapid development of the Ajax tools [19].

Value to the software development community from the OpenAjax Alliance will take several forms:

i) Marketing and Communications: Through education and communication activities, OAA intends to increase awareness of the benefits of Ajax developments, define a standard vocabulary for the technology, and foster the development of new Ajax projects.

ii) Technical Committee Work: At the kick-off meeting, technical committees were formed with the intention that these committees would produce most of the technical work of the Alliance. The product of these committees will include whitepapers, technical specifications and source code. The committees are:

a. Marketing and Architecture Committee – This committee is responsible for producing the architecture documents intended to help IT Managers and developers understand the Ajax landscape and the various technology options. Additionally, this committee will publish the standards, which define what makes an Ajax technology product OpenAjax compliant. Additionally, the committee promotes the vendors with compliant products. Compliance is dependent upon meeting the policies and standards established by the Interoperability and Declarative Markup Committees.

b. Interoperability Committee – Issues relating to the interoperability of competing Ajax vendor products are addressed by this committee. The objective of openness is to allow for competing products to be used within the same web application. The Interoperability Committee deals with defining standards and policies relating to: JavaScript name collision prevention; toolkit loading; and event management co-ordination between multiple Ajax products within the same application.

c. Declarative Markup Committee – This committee manages issues relating to HTML and XML markup interoperability that can arise when markups from different Ajax toolkits are used within the same application. Like the Interoperability Committee, the Declarative Markup Committee produces a set of standards, policies and recommendations.

CHAPTER 2 – A More In-depth Look at Ajax

There is a general consensus amongst web designers, developers and architects that Internet users expect more from web applications than what has been offered in the past. They want responsive, effective, interactive and intuitive applications. The adoption of such services as Gmail and Google maps, both of which entered the market place long after well established contenders like Hotmail and MapQuest, are both examples of how quickly users will embrace a new offering that provides a superior level of functionality.

Additionally, while many users are willing to install plug-ins from trusted sources, there are just as many users who perceive any attempt by a web site to install software as a threat. For web applications to be delivered to the masses, developers need to build upon readily available functionality. There are many benefits to such an approach, including [18]:

a) Appealing to users who are adverse to installing plug-ins or other software components;

b) Reduced compatibility issues because using standards available on all systems reduces the likelihood that some plug-in is not available for a given browser, platform or operating system;

c) Building of applications that are easier to maintain because changes are required only in the code which is downloaded into the browser as part of the web presentation or code which resides and runs on the server which is under the total control of the developer.

The AJAX methodology holds the promise of delivering the solution needed to achieve all of the goals previously stated. Some of the more important benefits of using Ajax are expanded upon in the sections that follow.

2.1 - No More Page Reloading for Every Server Request:

Some big name players have adopted the AJAX approach and they have constructed some very popular web-based applications using the AJAX approach including: Gmail, Google Maps, and Yahoo FlickR to name but a few. It's easy to see the advantages when you compare these two popular mapping programs:

a) MapQuest – a long-standing fixture of the World-Wide-Web, this mapping service has been in popular use for year. It was originally built using the original web-application deployment pattern. When navigating a map, each change resulted in a complete reloading of the HTML in order to change the map representation. As an example, unfortunately, it has recently been updated to use AJAX technology – a clear sign that AJAX is the way web-applications will be expected function in the future. To see an example of the older deployment method for mapping, I direct you to Steetmap – a UK based map service, which at the time of writing this paper was still using a traditional approach to delivering maps.

http://www.streetmap.co.uk/

b) Google Maps – surprisingly, it was the leader in search engine technology who has become a role model for this new revolutionary approach to developing web-applications. There mapping service was revolutionary not because it introduced or promoted some new programming language or API, but rather because it was built using long established technologies: JavaScript, XML, XMLHttpRequest Object, cascading style sheets, the Document Object Model and a server based application (the language in which it is written is irrelevant) which returns XML and/or text messages rather than XHTML. The navigation of the map is what is truly revolutionary: moving left, right, up or down; zooming in and out – these tasks do NOT result in the reloading of the HTML page, but rather change only the map portion of the page without the need to refresh. The task is accomplished by the combining of the aforementioned technologies and refreshing portions of the existing HTML document by manipulating it using server side JavaScript and the DOM. Check out http://www.google.com/maps and you will quickly see what I mean by revolutionary.

2.2 - Achieving Near Desktop Performance from a Web-Application:

AJAX separates user interaction from server interaction enabling the development of very fast Web-based applications. Both the client (browser) and server portions of the application operate asynchronously in parallel – this enables the AJAX application to operate at speeds comparable to many desktop applications. This approach makes development of Web-based applications a practical alternative to the traditional desktop application provided the application properly lends itself to delivery through a browser. Additionally, the use of Ajax improves the ability of the developers to provide highly interactive, functional applications to the masses via a corporate intranet or the Internet [21].

Communication between the client and the server is handled through the XMLHttpRequest Object. Implemented by Microsoft in 1999 as an ActiveX object, the XMLHttpRequest object enables JavaScript to make asynchronous requests to the server and receive responses without forcing a reloading of the current document. Combined with JavaScript's access to the DOM, this enables JavaScript to communicate with the server and update the current document without interrupting the users experience by waiting for the server responses or for a newer version of the HTML document to be reloaded and presented. The asynchronous communication leaves the browser free to continue responding to the user – stopping only once the server response is actually received. JavaScript offers many of the event driven functionality found in traditional desktop programs, which provides the power behind the high level of interactive services provided by Ajax style applications.

2.3 - Standards for Web Application Development:

As the web grew, the inconsistencies amongst the browsers, the introduction of new technologies, and phenomenal pace at which the World Wide Web (WWW) grew resulted in a complete lack of standards in the development of both websites and web applications. JavaScript, XHTML, CSS and the XMLHttpRequest object are all implemented in slightly different ways by the different browser developers. Variations have even been known to exist between browser versions from the same software company.

Fixes and ingenious solutions used to create fancy formatting or compensate for browser shortcomings have resulted in a lot of bad HTML/XHTML coding. Most modern browsers compensate for many of these shortcomings but how they handle each individual issue varies from browser to browser – and version to version. The result is that many well-crafted sites are now beginning to fall apart as newer browser enter the market and some specially engineered behaviours are either no-longer functioning or functioning erratically. Some estimates say that upwards of 99.9% of the current websites are obsolete – they just don't realize it yet [19].

JavaScript is another area facing standardization issues – and this is a problem for AJAX development. Microsoft's brand of JavaScript, known as Jscript, is similar to the official version of JavaScript, but has its own peculiarities. There are some minor and not so minor differences in the various flavours of JavaScript, which web developers must account for in their designs. Additionally, the XMLHttpRequest object, which forms the core of the asynchronous communications between the browser and the server, is instantiated differently on different browsers. Internet Explorer includes the object as an ActiveX object while all the other browsers include it as a native object. Also, the new IE 7.0 will include it as a native object – thus changing the rule relating to IE in general.

While all of these points actually highlight problems, they also reinforce the need for standards more than ever - standards in how the new web sites and web applications are developed as well as standards in how core technologies interact and interpolate. Careful construction is becoming more and more crucial since websites are now viewed on more than just the typical desktop computer. Websites and web applications need to also function properly on web-enabled devices such as palm pilots, cell phones, off-brand browsers, and in-screen readers used by the disabled. Following standards will make it just plain easier to accommodate everyone – both today and tomorrow.

2.4 - Features and Functionality:

Desktop applications are loaded with features ranging from a wide variety of controls to specialized display modules such as grids and graphs. To become comparable with desktop applications, users will expect to see all of these same features. Web programmers will not want to be creating custom controls, grids and graphing programs for each application they design. The AJAX movement has already addressed this issue by delivering more than 100 frameworks (some estimates say over 200) providing everything from slider controls and specialized selection boxes to grids and graphs. The number of choices is currently a problem because it is highly unlikely that all of these frameworks will survive in the long run. This creates a problem for developers who must weight there needs for a specific set of features against the likelihood that the framework or toolkit will continue to be supported after the winners and losers of this market space is decided.

Frameworks serve two purposes: first they save the programmer time by providing a set of reusable, tested tools to make development faster and consistent; and secondly, the accompanying JavaScript will have been tested on multiple browser and web-enabled platforms (at least the good ones) so that developers have fewer issues to deal with when testing their final product.

2.5 - Sharing the Load – Having the Client Do Their Share:

Under AJAX methodology, the idea is to have the client's browser do as much processing as is feasible and practical both in terms of time and convenience. As you can imagine, an application being shared on a server over the Internet can easily find itself doing a lot of work – managing the processing needs of many clients simultaneously. The more processing it can offload to each client, the faster it can process the requests requiring its specific resources and the faster everyone's application will respond to user requests.

Additionally, the communication between the client and the server is reduced substantially in an AJAX development because only the first page of the application will be the full HTML code. This will often be a slightly larger file because of the additional overhead in the form of JavaScript code files that must be downloaded. Once these files are on the client's browser, the application will not require very large exchanges of information. In most cases, the transfer will be either small text messages or small exchanges of XML data. The client will handle the job of managing changes to the interface displayed in the browser window through direct modification of the XHTML code through DOM manipulation functions. Whenever possible, these changes will be accomplished without reloading a fresh copy of the HTML.

2.6 Attributes of an Ajax Application:

At the core of Ajax, is the premise of improving the users experience while providing value to the organization. There are several key attributes for an ideal Ajax application – but in practice, delivering on all of the attributes is not feasible. There are tradeoffs – and the web developer must make decisions as to which ones are more important for the specific application. The key attributes are [09]:

a) Usability - The experience of using the application should not leave the user feeling frustrated. The application should be intuitive and productive to use. If it is also enjoyable or fun without compromising these two goals, all the better. In the book "Ajax Design Patterns", the author provided seven usability principles. They are:

i. Follow web standards: Users have expectations with regards to how various features should work. With Ajax, the developer may become tempted to reinvent how certain tasks will be handled. Instead, developers should use Ajax to make the web experience better, building an enhanced layer while respecting existing conventions.

ii. The browser is not a desktop: A decade ago, Netscape predicted the future was applications delivered over the network or internet and accessed through the browser. It seems to me that Microsoft took them dead serious, because it wasn't very long after this that Microsoft gained dominance in the browser market – displacing Netscape's near monopolistic hold on the market. But as of today, the browser still remains primarily a browser and not the replacement for the desktop envisioned by Netscape. Some future version of the browser combined with Ajax may one day bring Netscape's vision of the future to reality, but as of today, developers must view the browser as a means of delivering web content and not as a container for running applications. Browsers are just not designed to provide heavy computational power and as wonderful as the asynchronous communication capabilities of the Ajax methodology are, they are not sufficient to supplement the demands which would be needed to duplicate many current desktop products. In general, Ajax should be used to deliver desktop like applications which lend themselves naturally to delivery over the web rather than making the application fit into the web paradigm.

iii. When you have to change the expected, make the change really noticeable: This is the exception to the first principle. Sometimes it makes sense to change the expected behaviour of a control or expected behaviour. When this occurs, it's important to make the difference noticeable to the user – and sufficiently different so as not to confuse them while changing their expectations of what final behaviour will be.

iv. Known features have potential value: Its one thing to build into your application a fancy new feature or behaviour, but if the user is unaware of it, then what value does it add to their experience. Make sure the user is properly notified of their options and the benefits of their use.

v. Smooth, continuous interaction: From the users perspective, this is what Ajax is all about – a seamless progression from one state to another. This is characterized by the elimination of the full-page refreshes with their start-stop effect and the introduction of the ability to continue working within the page while waiting for results.

vi. Customization: While customizing the look and feel for individual web users has proven to be more work than the benefit derived from the experience for traditional websites, this is not necessarily the case for Ajax applications. Users of Ajax applications typically return frequently, especially with applications designed for special purpose use. It makes sense to allow users to customize their experience. Customization can take the form of colour or font adjustments or could extend to such what information is displayed on the users entry page to the company portal.

vii. Make it fun: What some call gimmicks and eye candy, others call motivation. Use of visually impressive effects, such as drag and drop, slider controls, and automatically updating content, when used in the right context can add value to an application. At the same time, employing such techniques can keep the user interested in the task at hand while providing a mild form of entertainment.

b) Developers' productivity – Good software design is suggested for smaller Ajax applications and can become essential to the successful creation, deployment and maintenance of large applications. The goal is efficient development – keeping in mind that efficient development directly effects maintenance.

c) Efficiency – Even thought full page refreshes practically disappear in an Ajax application, the efficient use of bandwidth is still important. If a newsreader was to update its content every 15-30 seconds by downloading the displayed content from the server as opposed to downloading only new content and having the browser update the displayed content, the advantages gained by the automatic updating of the content will be displaced by the large amount of redundant content wasting bandwidth.

d) Reliability – It goes without saying that the user expects accurate and timely information from its applications. As well, the application must preserve the integrity of the user data it handles and stores on their behalf.

e) Privacy – As will traditional applications, users expect to be kept informed about the information being collected about them, what will be retained, how it will be used, and with whom it will reside between the applications use. Keep the user informed and respect their right to privacy.

f) Accessibility – This simply translates into understanding the needs of your users. If your target users include those with disabilities, then Ajax applications should include features to accommodate them. For example, allowing for fonts to be increased so as to make it easier for the visually challenged to better read the sight, or including code which will allow a text reader software the ability to read your additions to the fancy news feed your application provides.

g) Compatibility – When developing the Ajax application, make it compatible with all the browsers, function on all the operating systems, and work with all devices it might want to interact with. Today, you may only plan to run it on the IE 7.0 browsers that are installed on all the Windows based office computers, but maybe tomorrow the applications deployment may be expanded to include your clients, many of whom don't use IE 7 or run Windows. Incorporating compatibility issues during development is far easier than making the fix deployment – especially when the problems might also reflect back poorly on the image of the organization.

2.7 – Ajax for Enterprise Implementations:

Three main areas where AJAX may offer opportunities to an enterprise: a) web applications to replace desktop applications; b) dynamic intranet apps to replace portals; and c) embedded web-applications where other sites can integrate the functionality of the enterprise (for free or a fee) using mash-ups [21]. Additionally, AJAX will help enterprises to create rich online self-serve applications for their clients that will reduce service costs while meeting customer needs for more effective services. An improved customer experience means more potential revenue. Enterprises will find AJAX helpful in modernizing existing heavy desktop applications; browser based deployment will mean less IT services installing new app versions.

AJAX use for business web developments is on the rise. A recent surveys of overseas developers has shown a marked increase in the use of AJAX in web developments – increasing from 21% to 36% use in the past 6 months. Additionally, the use of open source technologies (which goes hand-in-hand with AJAX) has also been steadily climbing – up and additional 10% from 59% to 69% during the same time period. [13]

2.8 - Alternatives to Ajax:

Not the only technology option: Flash / Flex from Adobe, Applets/JSP from Java camp – but its currently the favoured. It is just the beginning of this process and there is lots of time to see what develops. Still much of the traditional programming model features to be worked into Web Apps: such as clear separation of model from view; multitasking; and event-based programming [09].

1) Conventional Web Apps: There are several advantages to conventional web applications over Ajax apps, not the least of which is browser compatibility especially with older browsers. Users know what to expect from conventional apps as well and with that comes a sense of comfort. Finally, developers don't need to learn new techniques to build this type of application – and for the inexperienced, the skills required are significantly less than those needed for Ajax development.

On the other hand, Ajax applications offer the following benefits:

a) No full page refreshes with a higher degree of user interaction

b) Superior controls combined with visual effects and dynamic browser interaction allow for more expressive experience which leaves the user feeling empowered

c) Reduced waiting time and the ability to continue working while waiting for the server to respond – both of which result in a more productive interaction and leaves the user less frustrated

Depending on the intended audience and expected frequency of use, conventional application development may still make sense.

2) Flash Apps – Flash can be used as a component to an Ajax application – but it can also be used as a full-blown application in and of itself. Flash has its proponents and offers several benefits over Ajax applications particularly in the areas of video and graphics. Flash has the significant drawback that it requires a plug-in to work. While all of the popular browsers are supported, there still remain some for which the technology is not compatible making Ajax applications more likely to work with a broader range of browsers. Further, Flash is tied to this proprietary plug-in while Ajax is dependent on common open-source technologies not controlled by any one organization. Ajax also works better with search engines, which find the content of Ajax application pages easier to index whereas the majority of Flash content is obscured within the application itself.

Finally, the openness of Ajax to a wide variety of technologies is attracting developers far faster and is leading to a rapid development of libraries, techniques and patterns.

3) Java Applets – In the 1990's, Java applets were the hot technology and were expected to accomplish pretty much exactly what Ajax is now trying to deliver – desktop functionality within the browser. But the technology has long fallen out of favour and is now only rarely seen. While there has been a recent rekindling of interest in this technology, in part because it can be included as a component of either Ajax or Flash applications, it is too early to be sure if it will regain any real presence. The advantages of Java applets center around its ties to Java: the fact that Java programmers can quickly adapt to the technology, that Java can be used for the server side applications providing a 100% Java solution and that the technology includes extensive support for desktop-like user interfaces.

Ajax however offers several advantages over Java applets – primarily overcoming many of the limits of the applet technology. These advantages include: faster load times because applets must completely load often large libraries before they can start; Ajax offers a controls that look and feel like part of the browser as opposed to the desktop like appearance of Java applet controls; Java applets now suffer from browser incompatibilities issues as well as incompatibilities with current versions of Java; and finally, Ajax works with what is commonly available in every browser, where some environments require Java to be installed or updated with more recent versions in order for the applets to function. This is often an extreme deterrent to many users who would prefer not to deal with installing anything which must be acquired over the Internet.

4) Desktop Apps – Desktop applications offer capabilities well beyond that which can be provided by Ajax applications, including local file access, use of sound, rich graphics, more direct access to the local hardware, OS interaction, faster and more robust processing power, and faster, more varied communication protocols.

Ajax does have its advantages and when applied to the appropriate types of applications, can be far more usefully to the organization and end user. Some of these advantages have been discussed previously so they are listed here just briefly:

a) Ajax apps are delivered through the browser; as such, they do not need to be installed and are available from any computer connected to the server on which the application is hosted.

b) Users are comfortable with web interaction paradigms, and Ajax exploits this, making it easier for user to adapt to new applications.

c) Ajax applications are upgraded easily; changes are made only at the server. There is no need to run around changing installations at each users computer.

d) Developers of Ajax are free to select the server side programming environment that works best for them, while clients receive the client side application in the form of XHTML and JavaScript designed to run in any modern browser, on any computer, using any OS.

5) Mobile Apps – Writing apps for the mobile environment has been a challenging one because the operating environments have been so varied. However, it appears that the market is coming into sync on the issue of browser compatibility and to that end, there as been a move to ensure that Ajax capabilities are maintained in the browsers offered within Mobile devices primarily through the use of the Opera browser. If this support becomes standardized, then we will likely see Ajax type applications becoming the norm for Mobile devices.

CHAPTER 3 - An Overview of Implementation Patterns

In his book, Ajax Design Patterns [09], Michael Mahemoff has compiled an extensive collection of design patterns relating to design, development, maintenance and testing of Ajax initiatives, which can be applied in the development of web applications. He organized his patterns neatly into the following four categories:

3.1 - Foundational Technology Patterns:

Foundational Technology Patterns [09] describe the interaction between the basic building blocks from which all Ajax applications are constructed. These patterns describe the patterns of interactions between the raw technologies required for Ajax developments. The raw technologies include: XHTML, CSS, JavaScript, DOM manipulation functions, DOM events, and asynchronous communications typically in the form of the XMLHttpRequest object.

Applications can be built as full-blown Ajax developments where everything is *Ajaxified*. For such apps, everything appears to happen within just one or a few pages, JavaScript is used to manage all the interaction between the browser and the server, and none (or a very few) page refreshes occur. On the other hand, it doesn't have to be all or none with Ajax – the developer is free to use Ajax for just a portion of the web application and rely on conventional techniques for other parts. Some functionality might result in a full-page refresh while other features may take advantage of the Ajax methodology and update only a portion of the page.

At the core of every Ajax application is the "Ajax App Design Pattern", which is the most fundamental of the Foundation Technology Patterns. The following figure provides a graphical explanation of the pattern:

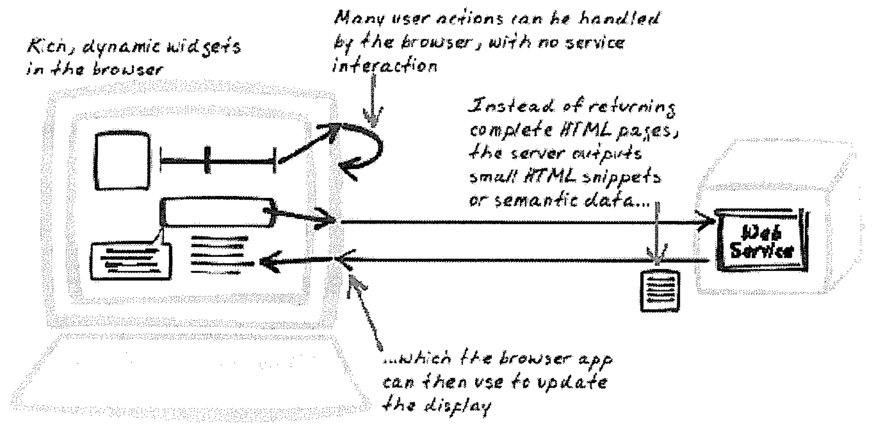
Rich, dynamic widgets in the browser
Many user actions can be handled by the browser, with no service interaction
Instead of returning complete HTML pages, the server outputs small HTML snippets or semantic data...
Web Service
...which the browser app can then use to update the display

Figure 3.1 – Ajax Application Design Pattern – core of every Ajax app.

Basically, this pattern describes the fundamental principles of the Ajax methodology – the inclusion of rich controls in the browser display; the use of client side processing to reduce the processing burden on the server and update the display with the information contained in server responses; and the user of asynchronous communications between the browser and server which include requests from the browser and responses which are reduced to just text snippets relating to the information requested only.

As part of the implementation of this pattern, the following questions need to be addresses by the web designer and developer:

i) What are the browser requirements? A decision as to which browsers to support must be made. The size of this subset of support, the larger the task of programming will be in terms of both coding and testing. If the project intends to rely on the XMLHttpRequest object, then the browsers will be limited primarily to those released after 2000 and even then, variability will exist between the different implementations.

ii) How will inadequate technologies be handled? This question is an extension of the first. The developer must decide if testing for specific features will be incorporated into the project. If not, the project may be subject to unexpected crashes. When testing is included, the developer can than handle the lack of a specific feature by: a) not including it; b) replacing the functionality with a simpler alternative; or c) requiring the user to upgrade to a compatible browser in order to continue.

iii) How much processing power will be required? JavaScript is an interpreted language; it does not execute quickly and can place a lot of demand on the processor for some types of functionality. While the goal is to take the processing demands away from the server, especially for public sites, placing too much demand on the client side can result in degrading the users experience. The application should be tested under conditions similar to what the user might experience – such as the web application running in the browser along with other browser windows being open, music applications running in the background, a mail client being open, and a few more applications.

iv) How much networking capacity will be required? With networks, there are two issues to contend with: bandwidth and latency. Ajax minimizes the bandwidth demands generally, especially after the application is up and running. However, some Ajax functionality, such as pre-fetching of documents and data, can result in using a great deal of bandwidth supporting anticipated usage that never materializes. Latency, on the other hand, is the greatest concern of the two issues when determining network capacity. Most Ajax messages are small – involving a simple request and a small piece of information representing the reply. When the network has high latency, then the delay caused by each small server request can become very noticeable and will impact upon the users experience. For Ajax features such as "Suggest" functions for which nearly every keystroke is sent to the server, latency can directly impact on the responsiveness of the feature and can turn the users experience into a frustrating one.

3.2 - Programming Patterns:

Programming Patterns [09] focus on the architecture and code features as they apply to software design principles specific to Ajax application development. They detail techniques intended to make the Ajax applications easier to maintain and of a higher quality. Additionally, the proper prevention and handling of error as well as techniques to improve the performance and responsiveness from the users perspective are addressed by these patterns.

The significant software design principles relating to the development of Ajax applications include:

i) *Empower the browser with JavaScript* – This scripting language is not only powerful, but also essential to the development of client side functionality. One of the main goals of Ajax is have the browser, primarily through JavaScript code, not only manipulate the DOM in response to server replies but also provide practical client side processing to take some of the burden off of the server. It is essential for developers to learn about the idioms, patterns and idiosyncrasies related to using this mature scripting language.

ii) *Accept that browsers are inherently flawed* – No matter how much we may wish it wasn't so, each browser comes with its own quirks, issues, and unofficial features. Even when it comes to the W3C standards that a browser claims to be implementing, inconsistencies exist in the actual implementations from browser to browser. All of these issues represent the constraints imposed on the developer when trying to deliver useful functionality, which will be consistently presented regardless of the client's choice of browser. The developer must become educated on the idiosyncrasies of each browser and deal with the technology, including all its flaws. Additionally, embracing trick and hacks in order to achieve a consistent representation to the user is a valid and expected approach every developer must take.

iii) *Manage asynchronous communications* – To some degree, nearly every Ajax application relies on communications with its server for information or processing of user supplied information. Asynchronous communications form the basis for this browser-server interaction. Issues of timeouts and failed calls should be accounted for in the design of an Ajax application. Additionally, care must be taken to ensure that when a reply is received from the server, the browser will know what the reply relates to and how to properly handle it. For example, a request for content is made to the server. For some reason beyond the user or browsers control, the request is delayed. Maybe the message was delayed in transit – or maybe the server processed many other requests before processing this request. There is generally no guarantee with regards to the order or timing of message processing. By the time the content arrives, the user has already decided to navigate to a different page within the application (yes, Ajax apps can have more than one page). The request handler for this response must not only know what do with the content it receives but also needs to be designed to recognize that the application still needs the content. Because the application has more than one page, the developer should be incorporating safeguards into the client-side JavaScript processing to ensure that the correct page is still in use before trying to process the content. Otherwise, who knows where the content will end up – or if it is actually processed, if adding it (or attempting to add it) to the current incorrect page will cause unexpected and undesired side effects.

iv) *Compatibility Issues are the Norm – Plan for Them* – JavaScript fell out of popular favour because of the plethora of small but annoying compatibility issues between the various versions of the language. While today the issues are less pronounced between all of the major browsers, this was not always the case; at one time the differences required substantial additional coding to manage a large number of inconsistencies. Originally developed by Netscape, Microsoft released their own version of the scripting language called Jscript. While both were basically the same at the time, they slowly started to introduce new features and changes to their implementations which differentiated the two sufficiently that some scripts did not behave as expected when run on the two browsers. ECMA International, an international body responsible for setting and documenting the standards for many technologies, endorsed JavaScript as developed by Netscape as the standard, but of course, they changed the name to ECMAscript. At the time, Netscape was the browser of choice, but times have changed and Microsoft's Internet Explorer is presently king. Further complicating the JavaScript issues, other less popular but still well used browsers, who claim to follow the ECMA standard, have implement the standard in slightly different ways and in doing so have introduced a few irregularities.

JavaScript is only part of the compatibility problem. XHTML and its predecessor HTML, Cascading Style Sheets (CSS) are two other areas where incompatibles exist between the various browsers. Adding to all of this further, the means of implementing the XMLHttpRequest object, developed by Microsoft and forming the core of nearly every Ajax implementation today, is also handled differently by IE and the other browsers. Even IE itself does not manage this object the same with all its browsers, having creating a new version of the object with nearly every version of its browser.

Compatibility must be a central concern and component in the design and development of nearly every Ajax application.

v) *Keep bandwidth traffic to minimum*: From the users perspective, one of the greatest benefits of Ajax involves the elimination of page refreshes between ever server interaction. From a bandwidth perspective, this means that a lot of traffic related to the passing of redundant HTML code is eliminated under the Ajax methodology. But the developer should still strive to keep messages to a minimum. It's a balancing game – since every HTTP message comes with its own inherent overhead relating to the HTTP headers and TCP/IP message handling. Ideally, all the required information should be packaged and sent in the same message. At the same time, the developer should avoid asking for redundant information. For example, asking for an updated version of some content would be more efficiently handled by sending back a "no change" message rather than an unchanged version of some 2K document – especially if the client is constantly automatically polling for such changes. Additionally, when using polling to enquire about changes, the frequency of such polling should be done at reasonable intervals. If one check every minute would satisfy the users needs, than use an interval close to this. Just because you can poll for changes every 5 seconds without effecting the efficiency of the application, doesn't make such an action a good thing. Such polling typically involves very small messages that are dwarfed by the HTTP and TCP/IP overhead. Additionally, your browser must then deal with the extra 11 messages per minute and while it may not effect your Ajax application while it is the only app being run, it could be affected when other applications are also being run in the background. In short, send the messages you need, and keep the messages to just what is required.

vi) *Consider Latency, not just Bandwidth*: A 5MBps connection to the network is great for downloading large amounts of content – but that is just a measure of throughput. When connecting to a server, especially in the case of small messages, the real factor in determining time is latency. When that server is thousands of miles away, you can be sure that latency can become significant. Engineering an Ajax application requires the developer to keep latency in mind – by making each request do as much work as possible and not sending every single change to the server as soon as it happens – but applying techniques to combine requests and predict client needs. For example, a user may have requested an article and the first page of a 10 page article has been delivered to the browser. The server delivering the pages has a 100ms round-trip latency and the page content can be delivered in 4ms. Further, delays due to TCP buffering make the entire process seem more like 2 seconds. If the user is still reviewing the first page after 5 seconds, there is a good chance that it is being read, and that that second page will be requested. A predictive application of Ajax will request the contents of the second page and possibly even the third before the user even makes the request. The content can be stored in hidden areas of the DOM – and when the user actually requests the next page, if the content has already been successfully fetched already, the application will respond immediately with the next page. A similar treatment of additional pages will make the users experience smooth, and the issue of latency will have been managed and the users experience improved.

vii) *Separate Business Logic and Presentation Logic* – Building rich Ajax applications is the goal, but organizing your project into manageable parts will leave it maintainable in the future. By breaking your applications into separate related components, the job of building more complete interfaces will become more manageable. JavaScript should not be imbedded in the main page if it can be avoided, and related code should be separated and grouped into individual files. This works to modularize the project and will serve developers later when it comes time to make alterations to the code. As well, it fosters code re-use. Cascading Style Sheets (CSS) should similarly be incorporated as separate files and not hard coded into the main page – again for ease of maintenance. There is nothing wrong with loading several JavaScript and CSS files in a single web document. Finally, the separation of business and presentation logic will help the developer in identifying portions of the code, which require a higher degree of attention to security. Passwords should not be hard coded into a JavaScript file as such files are downloaded and remain present on the computers of every user of the web page, and JavaScript files are in plain English – meaning the password or any other sensitive information can be read easily by anyone who cares to take a look.

viii) *Mind Your Environment* – Ajax applications, on the client side, are JavaScript / XHTML based applications running in a browser on a computer which could be state of the art or long past needing to be replaced. In many cases, the developer has little control over the client's operating environment. JavaScript and XHTML are both interpreted languages – one a scripting language providing programming functionality and the other primarily a presentation language for delivering static information in an orderly fashion. Both JavaScript and XHTML are slow relative to other languages. Finally, the user is likely to be running other applications in addition to the browser, so there will be competition for resources including processing power. With all this in mind, developers should try to make their implementations as efficient as possible and keep the "Flashy extras" to a minimum. Users expect a quality presentation but flashy features are not a priority and can detract from the quality of other aspects of the experience if they result in slowing down the primary functionality of the Ajax application.

ix) *Manage Degradation of Services* – As already mentioned, compatibility is a major issue with varying browser functionality and differences in client side environments. When developing an Ajax application that relies on specific technology being present for some special feature, the programmer should include provisions to use some alternative method to deliver some version of the functionality or politely advise the user of the missing requirement and inability to deliver the feature. A technical error message delivered to the client will likely be meaningless and just lead to frustration with the application. A wellphrased notification such as one explaining that a more current version of the browser will resolve the technical issue, will be received far more positively than a dump of the error stack.

One example of a Programming Pattern would be the management of a Periodic Refresh Pattern, which is part of the Brower-Server Pattern Collection described by Mahemoff. A following diagram depicts this pattern graphically:

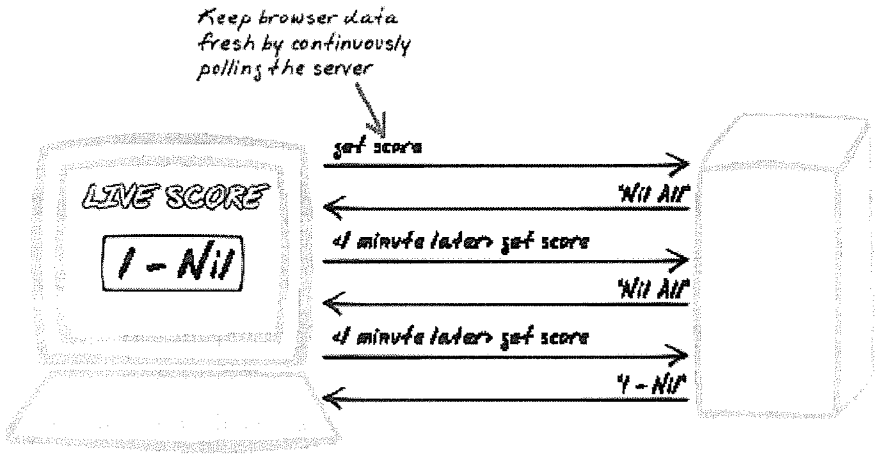

Figure 3.2 – Periodic Refresh Pattern – Ajax client polls server periodically for updates.

As depicted, the browser periodically issues a request to the server for a current piece of information – in this case a score – and upon receipt, it either displays the current result or determines if an update is required and then displays the new information. Such a service can easily be implemented using a timed interval request, which calls a function to make the request and then establishes the next interval request.

There are issues with this pattern:

a) While the user would ideally want updates to be as close to instantaneous as possible, latency plays a direct role in determining the maximum frequency.

b) The frequency of the updates has a cost in terms of network resource usage. If it is too high, it could be more expensive than the benefit derived while too infrequent and the changes being monitored may be reported long after they occurred resulting in some consequence to the user.

c) A fixed frequency may result in requests being sent when the user is not actively using the application. Further, the network or server performance may have degraded and the requests may be occurring faster than replies are being received.

The key when using this pattern is to find the appropriate recall frequency for each period task, and to set independent recalls for each one. Ideally, the interval should be self-monitoring so that it adjusts itself to reflect actual client activity and current server responsiveness.

3.3 - Functionality and Usability Patterns:

"Ajax is fundamentally about delivering value to users and their organizations." [09] Functionality and usability patterns are those primarily concerned with the issues relating to how the interfaces of the Ajax application will appear and how the users will interact with them. The introduction of widgets, which were previously seen only in desktop applications, and visual effects designed to add value to the uses experience by conveying information about changes in state are two of the more well known aspects of Ajax applications which differentiate them from traditional form-based offerings. But usability involves more than just what controls the user sees; it is also concerned with architectural issues such as page layouts, content management and content presentation. Finally, the functionality patterns attempt to expose some useful theory about how to make the users experience more valuable.

An example of a Functionality and Usability Pattern is the Suggestion Pattern. This pattern deals with the functionality provided by controls that attempt to provide a selection of options (suggestions) based upon the partial input supplied by the user. It is an interactive widget, which acts as an input field while providing a higher level of functionality while the user is using the control. The following diagram depicts this pattern graphically.

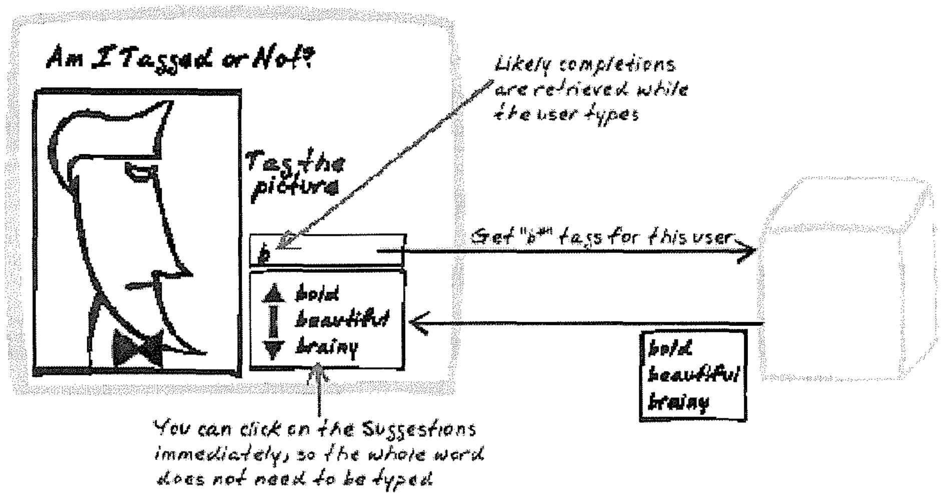
Am I Tagged or Not?
Tag the picture
b
bold
beautiful
brainy
Likely completions are retrieved while the user types
Get "b*" tags for this user
bold
beautiful
brainy
You can click on the Suggestions immediately, so the whole word does not need to be typed

Figure 3.3 – Suggestion Pattern – server suggests possible values based on partial client input.

Suggest functionality can be used to assist user in many ways: from speeding up input, assisting in correcting typing errors, to narrowing down choices and helping to validate input at the time of entry. It works by taking the portion of the input already entered by the user, sending it to the server who uses the partial input to find matches against a database of known acceptable entries. The list of suggestions is the sent back to the client and displayed as a selection box below the entry field. At this point, the user can: a) continue to type and entry, or b) select one of the offered suggestions. The real benefit of Suggest is found in situations where there is a large, but fixed set of possible entries but the list is too extensive to include in a traditional pull-down list. Through the use of the Suggest Pattern, the users partial input can be used to narrow the list until it is sufficiently small enough to be practical to present in a list.

The Suggest Pattern can be implemented to create some very powerful and useful features. One of the key decisions for the developer applying this pattern is to determine the parameters of its implementation. The questions to be answered include:

a) How will the suggestions be produced? This is where a great deal of the power of Suggest materializes. Suggestions could be simply based upon previously acceptable values for this given input or it could be more complicated such as offering matches used most recently, or even matches used most recently by this particular user.

b) What order will the suggestions be presented? Ordering could be based on historical information, alphanumeric ordering, or some application specific ordering.

c) How many entries to show? Obviously, if one of the reasons to use Suggest is because the number of choices available are too great to show all at once in a list box, then some determination must be made as to how many suggestions is appropriate. Also effecting this decision is the fact that all of the suggestions must be transmitted back to the client, so a shorter list is faster to deliver than a longer one.

3.4 - Development Patterns:

This final set of patterns focuses on the issues of testing, troubleshooting and monitoring. While the Ajax methodology is built on a relatively simple set of technologies, they are not without their issues, complications and idiosyncrasies. I have already elaborated a little about the compatibility issues between the browsers. Many issues are well known but being a known issue and the developer knowing about it are two different things. There is no central repository for all the known issues and research is often needed to find a workable solution for problems and bugs that crop up from time to time. The benefits of the Diagnostic and Testing patterns include: aiding in identifying issues during the development of the application; troubleshooting issues when new browsers are introduced; and monitoring the general health of the Ajax application after it has been deployed [09].

For Diagnostic Patterns, there are four general categories of techniques that can assist in identifying and isolating problems. They are logging, debugging, DOM inspection and Traffic Sniffing. The latter three Diagnostic Patterns generally involve the use of third party tools which are installed in the developers browser and generally will control the flow of execution within the browser, report about the state of the application at a given point; and show the contents of messages passed between the client and the server. For logging however, if third party software is included, it is used to insert code into the JavaScript in order to provide progress reporting as the code is executed. The developer can also manually add the logging code; the use of tools simply makes it easier to manage and remove the logging code, if desired, before using the JavaScript in production.

Logging is a technique which involves including lines of code in the JavaScript that records various events by writing entries to a DIV tag specifically intended to gather such information. The DIV tag could be included at the bottom of the application page and set as hidden. By setting the DIV tag as visible when logging is desired, the log entries will become readable and the developer will be able to use the entries to trace the activity within the application – with the possibility of isolating a particular issue. Logging can also be implemented by having the log messages asynchronously sent to the server where they can be collected and stored for later analysis. This is particularly useful for monitoring for problems once the application has been put into production. Since the JavaScript is delivered to the client when the application is instantiated in their browser, different versions of the JavaScript code can be used depending upon whether or not logging is desired. Since logging client events on the server will consume network resources, it may not be desirable to leave the logging code intact. Similarly, when logging uses the DIV tag method, it is of no benefit to leave the code operational once the system goes into production because the developer will never see the results of these logs and the logging process will simply consume client processing resources without any beneficial return.

The final set of Patterns from Mahemoff's book deals with Testing. As he describes it, there are four Testing Patterns that can be employed during development of an Ajax application:

a) Simulation Service Testing – Under this Pattern, a dummy or stub web service is created to simulate the behaviour of the server side programming. Such a service would return a predetermined response for each client request allowing the developer to test the functionality of the client side coding.

b) Browser Side Test – This type of testing is Unit Testing for JavaScript. Basically, each function and module is tested by creating a test suite which automatically runs specially coded methods designed to test the functionality of the JavaScript for the Ajax applications. The benefits of this type of testing is that regression testing can be preformed as development progresses, and unit testing each function forces the developer to carefully consider what the function is supposed to do by defining the tests which indicate it is working propertly.

c) Service Testing – This type of testing is the compliment to Simulation Service Testing in that it creates a dummy client side module designed to test the server side functionality by sending HTTP requests and validating the results returned.

d) System Testing – Once an application is completed, a thorough testing of its functionality, performance, and robustness is typically undertaken. Such a testing is referred to as a System Test. This type of testing is often done manually with a tester clicking their way through a sequence of activities designed to test the applications performance from a users perspective. System testing can also be automated – although the tools for assisting in doing so are difficult to use at this stage of their development.

Once considered a final last stage of development, testing has now become a fully integrated step in modern programming – particularly in larger projects. Ajax web development can benefit from such integrated attitude to testing as well – particularly in light of the compatibility issues imposed by the slight inconsistencies between browsers. Developers would be well advised to include Browser Side Testing in their developments as they incrementally develop their projects because the use of a unit testing approach will help quickly identify issues which arise as a result of new features being introduced as they are developed.

CHAPTER 4 - Implementation Issues:

4.1 - Database Updating Issues:

In a desktop application, the ability to lock a record in a database for the duration of an update is a given. Whenever a user wished to alter a database record, and the possibility of multiple users accessing the same record during an update is even a remote possibility, the software developer will request a lock for that record which will be maintained by the DBMS until the update is completed and the lock is released. The lock is associated with a user connection; if that connection is broken, a good DBMS system will terminate the lock.

For web applications, record locking is a much bigger issue. While it is still possible to have the DBMS lock the record, it cannot be associated directly with a user connection because the client (browser) and the server communicate using messaging passing over HTTP. This type of communication lasts only for the duration of the message passing and results in locks that last only for the duration of the transaction.

In order to achieve record locking for web applications communicating using HTTP, the server application needs to maintain records about the various activities of each user. There are number of issues which arise due to the nature of the communications between the client and the server.

i) Only the client can initiate communications with the server – never the other way around. So when a lock occurs, the server cannot advise the client directly. The developer must decide when the client needs to know about the lock: a) when the lock occurs and subsequently when it is released; or b) only when the client tries to edit the locked record.

ii) The developer must decide the level of granularity for the locks. The locks can be at the table, record or field level. Table level locks may be impractical in applications where there are large numbers of users – or the editing process can last long enough to affect the performance of other users. Field level locking would allow multiple users to edit different parts of the same record – but managing such a locking scheme could be a strain on the server especially when there are a large number of concurrent users who often edit information. Record level locking is the most typical level of locking and can usually be easily managed provided each table for which a lock is required has a small primary key associated with it. Larger keys make record level locking more inefficient to manage for large numbers of concurrent users.

How long to maintain a lock becomes an issue. The issue becomes whether or not the user is just taking a long time to complete the editing process or if the update transaction has been abandoned. In most cases, an expiry is appropriate and each lock should be assigned a lease when created. The developer must then decide: a) how long the lease should be; b) if leases can be extended; and c) if other methods of determining client abandonment are appropriate. The nature of the application and the level of concurrent client usage will be determining factors in making these decisions. A long lease may be appropriate where extensive editing may be appropriate – but a short lease combined with periodic lease extensions triggered by activity in the client's browser may server to detect abandoned processes and free up records sooner.

David Perelman-Hall, in his article entitled "Ajax and Record Locking" [17] detailed a plan for providing up to date record locking status to clients of a multi-user web application where the possibility of multiple users attempting to update the same record existed. Briefly, his approach was as follows:

a) Initially, every record entry is presented in the client entry in a read-only format. Users authorized to edit a record are provided with an Edit button that retrieves current record information when selected and presents it in an editable mode.

b) When a user elects to edit a record, the current information is retrieved and an entry is make in a lookup table identifying the record as locked and the user holding the lock.

c) When the user completes the edit process either by sending the updated information back to the server for recording or by abandoning the edit process, the server is notified and the lock entry in the lookup table is removed.

d) The client browser, using an Ajax technique for periodically refreshing information, polls the server at some preset interval to ascertain if the current record being displayed has a lock entry in the lookup table. If unlocked, the Edit button is shown; if locked the Edit button is removed. The client who owns the lock is sent a message confirming ownership.

e) Lock entries have an expiry associated with them. If a client establishes a lock and then abandons the application, the lock will eventually expire. At this point, the client is notified that the lock has expired on its next polling cycle so that the browser can respond accordingly. Once the lock owner has been notified of the expiry of the lock, the lock entry is removed and the record is free for editing by other clients.

For the most part, I like the solution provided above, but it does present some issues:

i) If the client displays 20 records at a time in a grid control, then each call for lock update information will involve 20 lock status requests per interval. If the client makes such a call every 10 seconds, and there are 10 concurrent users at any given time, then the server must deal with 1200 lock status requests a minute. Obviously, for heavily used systems, such a polling can easily add up to a lot of bandwidth usage as well as processing strain on the server.

ii) Also connected with the above issue, JavaScript is not very fast. Having the client do so much checking and processing relating to the checking may degrade the performance of the client.

iii) The solution fails to consider total abandonment by the client – such as when the client closes the browser without properly releasing the lock. Even the client may have a difficult time re-establishing a connection to the lock if a session was attached to the lock and the client is unable to reconnect with the original session after closing and re-opening the browser.

These points merely server to illustrate just how complex the issue of managing record locks can be for a multi-user web application.

4.2 - Security Issues:

Web 2.0 technologies such as Ajax are leading to the creation of more user-friendly applications. At the same time, they introduce the potential for security vulnerabilities. Security watchdogs at "SPI Dynamics said that the two main reasons behind security vulnerabilities [to Web Applications] are Rich Internet Applications and increased criminal activities." [14] In other words, RIA is creating new opportunities and criminals are more likely than ever to exploit these vulnerabilities. For Ajax applications, a few of the potential security issues include:

a) Lack of validation on input. In part this problem is because programmers are not paying attention (or are just too lazy) and in part because programmers have not been trained to recognize security vulnerabilities. SQL injections are just one type of attack commonly associated with improperly validated input. One way to protect against such attacks is to process input parameters against a function such as the PHP function *real_escape_string(),* which escapes special characters in a string for use in a SQL statement. Using such a function on an input parameter, which is not expected to contain these special characters, is a good way of guarding against malicious code being intruding through otherwise harmless text inputs.

b) Logic in Plain Sight. JavaScript is written in plaintext and it is not difficult to read even for those not versed in computer programming. Including business logic such as database structures or even server access information such as logins and passwords within a JavaScript file is a good as handing over your server to a resourceful hacker. Such logic needs to be kept server-side, with access granted through interfaces with validated input parameters. Its one thing to know how to make a request – its completely another to understand how that request is being handled. Sensitive logic and server side resource access passwords need to be hidden from the average user.

c) More Points of Attack. Generally, Ajax applications have a lot more server call points than a traditional application. In a traditional app, the user completes a form, hits enter, and the entire collection of values is sent to a specific call on the server for processing.

With Ajax applications, each entry could result in a call to the server for immediate validation of the recent entry. Each call could be made to a single point, but it is very common for a number of different types of calls to be used for different functionality within the application page. With each additional processing call point on the server, the number of potential entry points for hackers to exploit increases.

d) Plain Text Communications. HTTP transmits its messages in plain text. Unless the application takes steps to encode the data before transmitting, all communications are at risk of being easily understood if interrupted. Typically, most web applications protect their sensitive data by encrypting it at the point of storage. Unfortunately, anyone with access to the server will also likely have access to the decoding mechanism for this data, as it will also reside on the server. If the user wants to encode the transmission, they would need to handle the encoding and decoding on the client side – but as already mentioned, JavaScript is in plain text and including such code on the client side would in fact expose the mechanism for managing the encryption. One solution is to have the client supply a key which will used in the encryption process – making it difficult to decode any intercepted transmissions even with foreknowledge of the encoding scheme. Of course, if encrypted HTTP communications are essential, the use of Secure HTTP could be employed to provide the necessary security. This process is not without its cost and for small applications, such a cost may not be warranted on a cost-benefit basis.

These are just a few of an every growing list of potential security vulnerabilities for web applications.

4.3 - Managing the Back-Button:

First, let me state that there is presently no way to actually disable the back button on the popular modern day browsers. There are however a host of techniques that can be employed to make it so the button is not available or so that it continues to load the same page [22]. Each of these techniques have there own issues:

a) New Window Technique: There are two versions of this technique: i) open the application in a new window, in which case the previous page is available in another window; and ii) open a new window and close the old window, which eliminated the browser history completely. Users are not crazy about applications that open new windows. The technique is also slow as opening new windows is a time consuming process. Both techniques do have the effect of leaving the user in a new window for which the back button has no prior history and is therefore disabled. [22]

b) Use a Frame Trap: This older technique involves the creation of a two page frame – with one page set at 100% and the other set at 0% (width or height, it doesn't matter). Next, the 0% frame is loaded with a blank page and the 100% page is loaded with a page whose content is nothing more than a META tag redirect such as <META HTTP-EQUIV="refresh" CONTENT=".0; URL=realpage.html">. When the frame is loaded, the blank page and the redirect page are initially loaded into the frame. The redirect page immediately loaded realpage.html. Now, if the user clicks the back button, the redirect page is reloaded – and it again executes to load realpage.html. Of course, all the user needs to do is hit the back button twice and the effect is defeated. It should be noted that when tested in IE 6 and FireFox 2.0, this technique was no longer working. [22]

c) Using the Window History: Most browsers provide the ability to move forward and backwards through the browser's history using the functions window.history.forward() and window.history.back(). There is a technique that takes advantage of the window.history.forward() function. For Internet Explorer, the technique is as simple as calling the window.history.forward() function in the onload handler of the <body> tag. However, this technique fails for most of the other browsers because they do not execute onload handler function calls when navigating to the page using the back-button. However, these browsers, along with Internet Explorer, will continue the execution of JavaScript setTimout() call. Therefore, one technique is to use the onload handler to execute a function that we will call PageForward(), which does two things: first, it uses a setTimeout() call to call itself again at some future point which must be at least 500 milliseconds in the future; and secondly, it executes the window.history.forward() function. If there is no forward history, nothing will happen. Once the user moves forward to a new page, the function's real benefit will become active. If a user clicks the back-button to go to the earlier page, either the onload handler will immediately execute the PageForward() function, or the callback established by the setTimeout() will Forward() function. Either way, the window.history.forward() function will be executed and this time there will be a page in the forward history. Again, this method can be defeated with multiple clicks of the back-button. [25]

d) Use the location.replace() function: Basically, this technique calls a function using the onload handler which removes the most recent history entry – eliminating the previous page from history. It can only be used to remove one history entry and if the user navigated through a number of pages before starting the application, then the back-button will still an entry in history. Unless the user starts the window by opening the application and nothing else before it, this technique will not be able to eliminate all the history entries as it moves forward through the application. If combined with the New Window technique, however, it can be very effective. [24]

Of all the techniques, I like the window.history.forward() one the best for use with Ajax applications. However, the technique can be defeated with multiple clicks of the back-button.

Practically, the developer should not try to completely disable the back button, as this is still a browser environment and the back-button is a feature of this environment. Developers should not be able to disable this feature as it belongs to the user and not the application. Rather, the developer should approach the back-button as an activity that might be accidentally used by the user for a purpose other than moving back one page. For example, because the user is in a browser, they tend to hit the back-button almost as a reflex when they want to undo some event. In traditional form-based applications, this has often proven effective but in Ajax applications, it is not.

As already mentioned, I prefer the window.history.forward() technique for Ajax applications. When tested in a number of browsers, it works consistently. Additionally, it relies on JavaScript and if JavaScript has been disabled, the Ajax application is not going to run and we have nothing to guard against. The major drawbacks to this technique are two fold: first, there must be at least one initial page executing the window.history.forward() function in fixed intervals; and second, the technique can be defeated easily with a double click – a common behaviour for users. My solution to these problems would be to create a series of introductory pages to the application, each of which would provide an additional layer to counter the back-button effect.

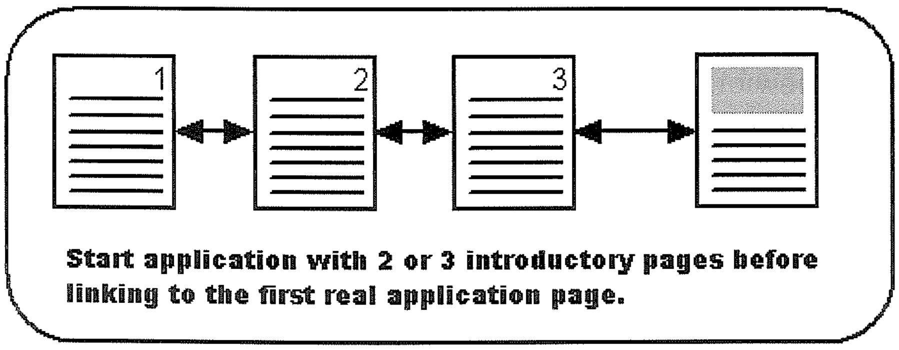
1
2
3
Start application with 2 or 3 introductory pages before linking to the first real application page.

Figure 4.1 – Defeat Back-Button with combination of Introduction Pages and JavaScript

Code for the intermediary pages would be as follows:

```
<html><head>
<script type="text/javascript">
<!--
function moveForward()
{
        window.history.forward();
        setTimeout("moveForward();",500);
}
-->
</script>
</head>
<body onload="moveForward();">
<a href="./nextPage.html">This way to the application</a><br />
</body>
```

Code 4.2 – JavaScript Code for Introduction Pages to defeat Back-Button

At least two introductory pages should be used before leading into the first page of the actual Ajax application because this will provide insulation from the effect of a double click on the back-button. In my tests, I found three pages to be best when dealing with the FireFox browser, which relies on the setTimeout() callback function in order for the counter effect to function. In Internet Explore browsers, which execute the onload() handler of the <body> tag when the page is loaded after the back-button, the effect works extremely well and two pages are more than sufficient to defeat any clicking by the user.

For the user, this method does not eliminate their ability to use the history to go back to some previous page. This ability is still available by selecting the page from the browsers list of previously visited pages often found either at the tab for the page or by using the View/Go To menu options. In this way, the users right to go back in history is still maintained, while the application provides a counter measure to an inappropriate operation.

4.4 - Cross Browser Issues:

Writing Ajax applications that will perform as expected across as many browsers, platforms, and operating systems as possible requires an understanding of the compatibility issues between the various implementations. For some issues, it is better to completely avoid using the feature because alternative functionality may be too difficult or impossible to achieve. For many other issues, well known alternatives do exist and can be managed very effectively.

4.4.1 - XmlHttpRequest Object:

The XmlHttpRequest object was introduced by Microsoft as an ActiveX object to allow asynchronous communication of XML data between a server and a browser. Of course, it always functioned equally well for passing plain messages – as such, basically any message whose format was text based could be used. Other browser vendors immediately recognized the value of this object and by 2001, nearly every commercially available browser supported the object as a native component. Up until just recently, with the release of Internet Explorer 7.0, Microsoft continued to include the XmlHttpRequest object as an ActiveX control. Now with IE 7, it is supported as a native object just like every other major browser. Unfortunately, this does not eliminate the need for developers to provide code for all the possibilities when using this method for asynchronous communications. Here is a sample of how a developer might compensate for the instantiation of this object: [03]

```
//Instantiates an XMLHttpRequest Object
function createXHR()
{
        var XHR;        //local variable

        try {   // first try to create XMLHttpRequest object
                // (works for IE7 and most other non-IE browsers)
                XHR = new XMLHttpRequest();
        }
        catch(e) {      //assumes IE6 or older

                //for IE6 or older versions, an activeX object is required for the XHR
                var XHR_IE_Versions = new Array("MSXML2.XMLHTTP.6.0",
                                                "MSXML2.XMLHTTP.5.0",
                                                "MSXML2.XMLHTTP.4.0",
                                                "MSXML2.XMLHTTP.3.0",
                                                "MSXML2.XMLHTTP",
                                                "Microsoft.XMLHTTP");

                //try each version, newest to oldest till we get one that instantiates
                for (var i=0; i<XHR_IE_Versions.length && !XHR; i++)
                {
                        try { //try  a version
                                XHR = new ActiveXObject(XHR_IE_Versions[i]);
                        }
                        catch (e) {} // ignore errors
                }
        }

        if (!XHR) {     //if the XHR not created, report an error
                alert("Error creating the XMLHttpRequest object.");
        }
        else {
                return XHR;
        }
}
```

Code 4.3 – JavaScript Code for Instantiating XmlHttpRequest Object on all browsers.

4.4.2 – JavaScript:

The basic implementation of JavaScript between the browsers is consistent with the standards set by ECMA between all of the popular browsers. They all offer the same set of control structures and variable support. Where the differences appear most between the implementations revolve around implementation issues with the browser and the Document Object Model (DOM). The DOM issues are described below, so for the remainder of this section, the focus will be on the differences in handling events.

As mentioned earlier, in the section 4.3 – Managing the Back-Button, the onload() event is not handled consistently. Only IE can be relied upon to fire the onload() handler of the <body> tag, most of the other browsers will execute it when linking into the page or as result of the window.history.forward() method. Opera distinguishes itself as the only browser which will not execute the onload() handler of the <body> tag for any window.history method. [25]

Some events are unique to only a specific vendor such as the onMouseEnter and onMouseLeave events belonging to Internet Explorer browsers. For the most part, these specialty events represent a subtle difference from some already common event handler or they describe some obscure behaviour, which developers would rarely even consider using.

When developing an Ajax application with the objective of achieving the widest possible browser compatibility, a developer must become well versed with the known event compatibility issues so that these issues can be either be avoided or appropriately compensated for with additional code.

The website, QuirksMode.org has done a remarkable job testing and documenting the differences in event handling compatibility amongst the leading browsers. A link to their results table can be found at: http://www.quirksmode.org/js/events_compinfo.html.

4.4.3 – Cascading Style Sheets:

Of the four areas of compatibility, this is by far the one where diversity in implementation exists. The core definitions of cascading style sheets appears to be consistent with the standards set by the World Wide Web Consortium (W3C), but when it comes to some of the specialty behaviours such as effects during hovering or when active, the implementation can vary widely. When using such features, the developer must decide just how important the failure of such a feature will impact the users experience.

4.4.4 – Document Object Model:

The Document Object Model (DOM) is the current web page organized into a tree structure, which can be accessed programmatically by JavaScript. Information can be retrieved, altered and replaced within the DOM with the effects being represented instantly within the browser window. For the most part, the core functions of the DOM as defined by W3C – the World Wide Web Consortium – have been consistently implement by the current modern browsers, but when it comes to the less used functions related to this Object, the browsers differ significantly.

Developers working extensively with the DOM would be well advised to familiarize themselves with the contents of their test results of QuirksMode.org. Once again, this organization has done an excellent job of itemizing the differences between the various modern browsers with respect to the DOM features and functions supported. The details of their findings can be found at: http://www.quirksmode.org/dom/w3c_core.html.

CHAPTER 5 - AJAX for Desktop Application Delivery over the Web

5.1 – Controls and Widget:

The lists of controls and widgets available to Ajax developers number in the 100's. For many of these, there exists dozens of variations available in dozens of frameworks and toolkits offered by a large number of vendors. Additionally, there are no standards associated with each control other than the expectations of the users as to what features must exist and might be included. [01]

The specialized controls and widgets of the Ajax frameworks enable developers to build applications, which look and function very similarly to traditional desktop applications. These are an extreme improvement to the controls offered by HTML – adding features and functionality never before seen in a web application. Users find it easy to use Ajax websites because of their familiarity with the controls and their functionality. Of course, the underlying assumption is that the Ajax versions of these controls behave just like their desktop counterparts. When developers or vendors build in alternative behaviours into a control, which has a desktop counterpart for which users have preconceived expectations, these additional features will typically result in confusion or frustration for the user. Additionally, users who have no expectation of the new functionality often will overlook the inclusion of such special functionality. [04]

Some of the more popular specialty controls and feature enhancements found in Ajax toolkits and frameworks include:

i. Grids: Basically, a grid is a tabular representation of data, typically records from a table, which often allows for the resorting of data on a column by column basis; pagination through sets of results rather than displaying all results at once; and the selection of records for editing. Many grid controls even go so far as to allow editing of data from within the control itself.

ii. Drag and Drop: A technique whereby items in a list can be rearranged simply by selecting a item and dragging it to a new location within the list. Drag and drop can also be used for migrating items from one list to another; or removing items by dragging them out of a list and depositing them into a region assigned for deletion.

iii. Suggest: This is a feature built into text entry boxes whereby the user is presented with a selection of known matches to the partial entry already provided. The feature is useful in helping users clarify an entry – such as in a search box – or in narrowing the selection to a known list of acceptable entry values – such as in the case where the list of acceptable values is too large to accommodate in a traditional list box control.

iv. Auto Complete: This feature is also built into text entry fields. It uses a partial entry to determine a match against a list of known acceptable results. It can also be used to provide alternative methods of entry – such as in the use of date fields where an entry such as today would be replaced by the current date.

v. Calendars: These controls are useful in visually selecting a date value and often include month-by-month pagination controls. Often, the calendar information is presented in a tabular fashion with the days of the month associated with their corresponding weekday name.

vi. Sliders: Useful when a range of numeric values is appropriate as an entry, this control presents a bar with a movable button that can be slide from one end of the bar to another – representing a value from one end of the range to another. A label is typically used to display the actual value being selected by the user.

vii. Tabs: Often seen in desktop applications, this control allows the developer to build an Ajax web app that can provide transitions between a variety of interfaces from within a single web page. This control is represented as a set of file-folder style tabs, typically shown at the top of the page or top of a section. One of the tabs will appear as if part of the current pain of information, while the others will appear to be connected to layers below. By selecting one of the tabs for a lower layer, the control will change the DOM of the pain to show the content related to the tab as well as change the selected tab so it appears to be part of the current pain, while the formally connected tab will appear as part of some lower layer.

5.2 - Frameworks and Toolkits – Speeding Up Development:

Designing Ajax controls can be difficult. Not only does the developer need to figure our what features are expected by the user for the particular control, they must also solve the issues related to implementing those features. Additionally, there are the issues of cross browser support and interaction with other controls. Frameworks and toolkits are useful in reducing and removing many of the tool design issues, freeing the developer to focus on the core functionality of the application.[19]

5.2.1 - Benefits of Using Frameworks and Toolkits:

A framework is a collection of reusable classes, methods, and functions designed to be incorporated into the development of other applications. In the case of Ajax frameworks, the collections typically represent both client side and server side components which are coded to interact with each other. Some frameworks are libraries representing dozens of sophisticated controls; they are extensive enough to meet most of the needs when building web applications. Others represent just a few related controls which can form just part of a web development project.

Toolkits are more than just a collection of controls – they include utility programs designed to assist the developer in constructing the application and customizing the controls for specific uses.

Both frameworks and toolkits offer Ajax developers some significant advantages:

i. Obviously, not having to build the control saves the developer considerable time.

ii. The components are typically well tested which can save the developer considerable time not only because they don't have to build these controls but can rely on them working as promised.

iii. Issues relating to feature expectations will already have been studies and considered in the design. The developer should compare the various implementations by different vendors of a control they wish to use to insure that the one they plan to select has considered everything appropriately.

iv. Most frameworks and toolkits offer components that have been tested for compatibility across a wide range of browsers. Most vendors attempt to provide controls that are compatible with every modern browser currently in use.

v. With the creation of the OpenAjax Alliance and this organizations certification of compliance, many vendors have made their controls meet the expectation of this standards body. Such compliance means that mixing controls with other compliant frameworks and toolkits will not pose any issues.

5.2.2 – Types of Frameworks and Toolkits:

Frameworks and toolkits come in a variety of flavours divided along two fronts: client vs. server; and single vs. dual DOM models. The first is concerned with where the majority of validations and transformations occur; while the later is concerned with where the Ajax delivered data is stored. [19]

a) Client Side: With client side frameworks, the bulk of the work done by the Ajax engine is performed within the browser. In these cases, the server returns messages concerned strictly with the results of a request and does little or no work with regards to formatting or transforming the information into its final presentation form. Most, if not all, of the transformations are managed by the client's browser. On the server side, no additional Ajax specific software is required. In many cases, legacy web service applications can continue to be used with little or no modifications. With client side transformations, the client remains independent from the server making it possible to make changes on either side without affecting the other.

b) Server Side: With sever side frameworks, the focus is on having the server process and transform as much of the input as possible. Such frameworks typically provide a large set of widgets and controls, which encapsulate all the necessary JavaScript, required to communicate input and process the replies. This JavaScript code is delivered with the initial web pages to the user from the server and is configured by the server for its specific use. Server side programming is typically handled in some higher-level language like Java or C++, which can be factored and optimized for performance.

c) Single-DOM vs. Dual-DOM: With a single DOM approach, the web browsers DOM representing the web page is used for both the original page representation and to contain any JavaScript and additional HTML supplied by the Ajax application. With a Dual-DOM approach, the Ajax applications uses a second DOM to hold any JavaScript and HTML data it needs. Typically, bi-directional listeners are set up between the two DOMs in order to maintain synchronization. Additionally, the Ajax DOM can be housed on either the client or server side.

5.3 - Tools for Development:

The popularity of Ajax is being supported by an increasing number of frameworks, toolkits, and development tools. Most recently, the Eclipse Foundation released two tools to support the development of AJAX applications at the enterprise level using Java: the AJAX Tools Framework dubbed ATF and the Rich AJAX Platform, or RAP for short. ATF is an Eclipse tool that provides debugging and development features such as XML and DOM inspection, syntax highlighting and code validation. The ATF debugger offers the user the ability to watch and inspect packets as they go back and forth between the browser and the server. RAP on the other hand, allows developers to create applications in Java, with all the robustness you expect to find in Java apps and then have the functionality deployed in AJAX. RAP is still not complete, but it has now been publicly released for developers to use. [15]. There release is symbolic of the commitment by vendors and the industry to the Ajax methodology.

Tools for Ajax development come in many forms. Most are independent, one purpose utilities, while a few more comprehensive tools presently exist. Some of the different categories of tools useful for building Ajax applications include:

a) IDE: ATF, described above, is but one of the early entries into the IDE market place; many more are expected soon.

b) JavaScript Tools: There are a variety of tools available for documenting JavaScript code and checking JavaScript syntax. Since JavaScript has been around for more than a decade, there is no shortage of good (and bad) tools available.

c) Validation Tools: HTML and CSS validation tools have long existed and many of these validation tools come standard in most Web Authoring tools. Dreamweaver and HomeSite are just two such products. The DOM is part of HTML; as such, it is typically validated for structure by most HTML validation routines. However, some features of the DOM are not checked as closely as Ajax developers may require. Because Ajax relies extensively on the DOM being correct, it is important that developers seek out and use a good DOM validation tool.

d) Comprehensive Tools: Because of the growing popularity of Ajax, comprehensive developer tools are starting to emerge. One of the more interesting tools is FireBug – a plug-in for the FireFox browser, which provides support for validating and identifying issues with Ajax applications as they run in the browser. FireBug provides tools for validating HTML, CSS, and the DOM. It includes a tool to graphically represent the container configuration of a web page. Also included are communications capturing capabilities which records both requests and replies and saves this information so that developers can examine the contents of the client server message passing for their application.

e) Enterprise Level Tools: The recent releases by the Eclipse Foundation are an indication that support for enterprise level Ajax developments is underway. At the same time, the state of development of these two tools are also indicative of how much work still needs to be done in the area of tools before Ajax can truly be considered a serious contender in the coveted Enterprise Web Application market.

CHAPTER 6 - CONCLUSION:

The Ajax methodology is based upon mature, well-implemented standardized technologies. Its objectives of delivering an improved user experience while adding value to organizations by providing web applications that provide functionality and performance comparable to desktop applications seems to be achievable.

The methodology is still relatively new and had been experiencing a rapid growth in frameworks and toolkits. At present, there are no clear indications as to which of these frameworks will dominate the market and which will fall by the wayside. Additionally, the Ajax market lacks the sophisticated tools necessary to propel it forward from an development methodology applicable to partial implementations or small to medium size applications up into the realm of enterprise web application development.

There is a lot of excitement around Ajax. The formation of a standards organization in the form of the OpenAjax Alliance; an industry magazine (AjaxWorld); and a conference dedicated to Ajax development community are three solid indicators that the web development marketplace is expecting Ajax to be around for a long time to come.

Having examined the current state of the Ajax methodology and its applications to date, I can conclude that it is indeed a suitable approach for developing desktop type applications for delivery over the Internet through a web browser. It is, in the opinion of the writer, the best approach for developing web applications presently available.

Bibliography:

1) Book Resources:

[01] Babin, Lee. *Beginning AJAX with PHP: From Novice to Professional.* Berkley, CA: Apress Publishing. 2007. ISBN: 1-59059-667-6

[02] Crane, David, Pascarello, Eric, James, Darren. *AJAX in Action.* Greenwich, Connecticut: Manning Publications, 2006. ISBN: 1-932394-61-3

[03] Darie, Brinzarea, Chereches-Tosa, and Bucica. *AJAX and PHP: Building Responsive Web Apps.* Birmingham, UK: Packt Publishing. 2006. ISBN: 1-904811-82-5

[04] Eichorn, Joshua. *Understanding AJAX: Using JavaScript to Create Rich Internet Apps.* Upper Saddle River, NJ: Prentice-Hall. 2007. ISBN: 0-13-221635-3

[05] Flanagan, David. *JavaScript: The Definitive Guide, 5th Edition.* Sebastopol, CA: O'Reilly Media. 2006. ISBN: 0-596-10199-6.

[06] Hadlock, Kris. *AJAX for Web Application Developers.* Indianapolis, Indiana: Sams.Publishing. 2007. ISBN: 0-67-232912-3.

[07] Heilmann, Christian. *Beginning JavaScript with DOM Scripting and AJAX.* Berkley, CA: Apress Publishing. 2007. ISBN: 1-59059-680-3.

[08] Keith, Jeremy. *DOM Scripting--Web Design with Javascript and the Document Object Model.* Berkley, CA: FriendsOfEd Publishing. 2005. ISBN: 1-59059-533-5

[09] Mahemoff, Michael. *AJAX Design Patterns.* Sebastopol, CA: O'Reilly Media. 2006. ISBN: 0-596-10180-5.

[10] White, Elliott III, and Eisenhamer, Jonathan D. *PHP 5 in Practice.* Indianapolis, Indiana: Sams.Publishing. 2007. ISBN: 0-672-32888-7.

[11] Woychowsky, Edmond. *AJAX: Creating Web Pages with Asynchronous JavaScript and XML.* Upper Saddle River, NJ: Prentice-Hall. 2006. ISBN: 0-13-227267-9

[12] Zakas, Nicholas C, McPeak, Jeremy, and Fawcett, Joe. *Professional AJAX.* Indianapolis, Indiana: Wiley Publishing. 2006. ISBN: 0-471-77778-6.

2) Magazine Resources:

[13] Evans Data. *AJAX Use Climbs in New Markets.* Software Development Times, March 15, 2007.

[14] Feinman, Jeff. *New Security Threats for Web 2.0 Detailed.*Software Development Times, January 15, 2007.

[15] Handy, Alex. *New RAP Hits to Debut at EclipseCon.* Software Development Times, March 1, 2007.

[16] Myer, Thomas. *"Building a Simple AJAX Application: Learning to Use AJAX together with PHP"* International PHP Architect Issue 06-2006. Pages 26-29.

[17] Perelman-Hill, David. *AJAX and Record Locking.* Dr. Dobb's Journal, September 8, 2006.

[18] Seabrook, Conor. *Bringing the Desktop Application to The Web.* Dr. Dobbs Journal, March, 2007.

[19] Sys-Con. *AJAX World Magazine.* Volume 1, Issue 1. Entire magazine

[20] Sys-Con. *AJAX World Magazine.* Volume 1, Issue 2. Entire magazine

[21] Taft, Darryl. *A Better Experience: AJAX-Enabled Apps are Springing Up on Intranets…* Software Development Times, February 15, 2007

3) World Wide Web Resources:

[22] Burns, Joe. *Disabling the Back Button.* January 4, 2005. http://www.htmlgoodies.com/tutorials/buttons/print.php/3478911

[23] Garrett, Jesse James. *Ajax: A New Approach to Web Applications.* Feb18, 2005. http://www.adaptivepath.com/puplications/essays/archives/000385.php

[24] Reddy, Akhilesh. *A Thorough Examination of "Disabling the Back Button".* 2006. http://www.4guysfromrolla.com

[25] WWW FAQS: *How Do I Disable the Back Button on a Web Page.* Nov 21, 2006. http://www.boutell.com/newfaq/creating/backbutton.html

APPENDIX A – Glossary

Ajaxified – A term commonly used within industry books and articles which refer to the use of the Ajax methodology of asynchronous client-server interaction to deliver a function within a web site.

CSS (Cascading Style Sheets) – A style sheet language that enables authors and users to attach style, such as fonts, spacing and aural cues, to structures such as HTML and XML documents.

DOM (Document Object Model) - The official standard for representing structured documents in a platform- and language-neutral manner and used as the basis for a wide range of application programming interfaces.

HTML (Hypertext Markup Language) - The authoring software language used for creating World Wide Web pages.

HTTP (HyperText Transfer Protocol) - The primary method used to convey information on the World Wide Web who original purpose was to provide a way to publish and receive HTML pages. Ajax uses this protocol extensively for browser-server communications.

Latency – With regards to networking, it is the measure of how much time passes between a message being sent and its arrival at the destination.

Throughput – With regards to networking, it is the measure of how much data can be transferred in one second.

Widgets - The collective name for buttons, sliders, menu bars, title bars, and all the other paraphernalia that windows can have or contain

XHTML (eXtensible HyperText Markup Language) - The next generation of HTML that is a hybrid between HTML and XML.

XML (Extensible Markup Language) - A format that allows information and services to be encoded with meaningful structure and semantics that computers and humans can understand. XML is great for information exchange, and can easily be extended to include user-specified and industry-specified tags.

APPENDIX B – Ajax Project and Source Code

An AJAX demonstration project was built which demonstrated several features of AJAX style applications. The project is a To-Do Task Management Program which displays the tasks assigned to each unique use. The program demonstrates the following features:

1) Logging into the Application results in the center panel being populated by a list of tasks belonging to the user.

2) The list is provided in a DRAG-AND-DROP style, which uses server-side functionality to maintain and record the order the items as well as re-present the items after re-ordering.

3) The application allows for the changing of the background colour schemes from the default to another selected scheme. The scheme is stored with the user – customizing their experience.

The code for this project is included on the accompanying CD.