Date of acceptance Grade

Instructor

Performance of Ajax applications on mobile devices

Mikko Pervilä

Helsinki February 5th, 2008

UNIVERSITY OF HELSINKI

Department of Computer Science

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion –Faculty/Section Faculty of Science	Laitos – Institution – Department Department of Computer Science

Tekijä – Författare – Author

Mikko Pervilä

Työn nimi – Arbetets titel – Title

Performance of Ajax applications on mobile devices

Oppiaine – Läroämne – Subject

Computer Science

Työn laji – Arbetets art – Level M.Sc. Thesis	Aika – Datum – Month and year 5.2.2008	Sivumäärä – Sidoantal – Number of pages 88 pages + appendix 20 pages

Tiivistelmä – Referat – Abstract

The Ajax approach has outgrown its origin as shorthand for “Asynchronous JavaScript + XML”. Three years after its naming, Ajax has become widely adopted by web applications. Therefore, there exists a growing interest in using those applications with mobile devices.

This thesis evaluates the presentational capability and measures the performance of five mobile browsers on the Apple iPhone and Nokia models N95 and N800. Performance is benchmarked through user-experienced response times as measured with a stopwatch. 12 Ajax toolkit examples and 8 production-quality applications are targeted, all except one in their real environments. In total, over 1750 observations are analyzed and included in the appendix. Communication delays are not considered; the network connection type is WLAN.

Results indicate that the initial loading time of an Ajax application can often exceed 20 seconds. Content reordering may be used to partially overcome this limitation. Proper testing is the key for success: the selected browsers are capable of presenting Ajax applications if their differing implementations are overcome, perhaps using a suitable toolkit.

ACM Computing Classification System (CCS):
C.4 [Performance of Systems]: Performance attributes;
H.3.5 [Information Storage and Retrieval]: Online Information Services---Web-based services;
H.5.4 [Information Interface and Presentation]: Hypertext/Hypermedia---Architectures, Navigation, User issues;
H.5.2 [Information Interface and Presentation]: User Interfaces;
D.3.2 [Programming Languages]: Language Classifications---Concurrent, distributed, and parallel languages, JavaScript;
C.2.4 [Computer-communication Networks]: Distributed systems---Client/Server, Distributed Applications

Avainsanat – Nyckelord – Keywords

Ajax mobile devices performance web applications DHTML JavaScript

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Content

1 Introduction

During the past decade, the popularity of the World Wide Web has grown tremendously from its humble origins as the Hypertext Markup Language (HTML) for scientific texts [W3C03]. The low learning curve of the HTML language, a host of WYSIWYG editors, and the will to publish are all factors that have enthralled the public as well as enabled it to participate. Eventually, both small and large organizations and companies caught on *en masse*. These new participants brought new requirements into the picture – requirements that did not always match with HTML as it was defined. As the number of interested parties grew, the need arose to make each new web site more special in order to distinguish it from the existing ones. To meet the increasing demand, new versions of HTML were specified, adding to the language's power of expression. Later on, its increasing layout capabilities were split and endowed to Cascading Style Sheets (CSS).

The new specifications were eagerly interpreted and implemented by browser developers, whose aim was to populate the market with their products. But soon it became clear that the demand far exceeded the supply. The major standards organization, W3C, could not keep up with the rapid pace of web designers everywhere. While many new design techniques became available, critics yelled out that some of these techniques wantonly misinterpreted the *intent* behind the standards (see any discussion on HTML tables). To aggravate the problem, browser developers started to extend the standards with their own, browser-centric features. These extensions started out as guesswork on which new features the users would next like to see. Some were successful, others forgotten. However, the use of such tools forced designers to dedicate their projects to a specific browser platform. One of the new extensions was JavaScript [W-J07a]; its main competitor Visual Basic Scripting Edition (VBScript) [Mic07a].

Very early on in this history, another problem emerged as well. As new features and techniques started to become available, both the designers and the public became aware of differences in the layout of the pages. Even web sites written to follow the standards did not always look exactly as intended. The human capacity for error manifested as faults in the interpretation of the standards, and the faults were reflected in the browsers. Instead of writing once and displaying everywhere, the designers were now forced to take into account a number of small discrepancies. The flaws were not fatal, but it took more time to test the code on several browsers. Warnings and disclaimers became common on web sites, announcing that the site in question was “best viewed” [Yah07a] with the designer's browser of choice. This struggle was called *the browser wars*, and it continues to this day [Wei06].

Gradually, it dawned that these problems were not going to disappear with the next generation of browsers. Most designers calmed their minds and stopped neglecting the other platforms. But not all of them. Ever since, each web site's testing cost has been paid many times, once for each *major* browser. The strategy is to categorize the public of the web site, and try to support their chosen browsers. Outside of intranet environments, this task might well be impossible to perform conclusively. The number of choices is greater today than ever before, with new browsers released bi-monthly for open source platforms, home entertainment systems, and mobile devices. Commonly, the same browser might behave differently when installed on two operating systems. Automated testing suites [GEO07d] do exist, but their cost and other restrictions remain prohibitive factors for some designers. Before all the combinations can be tested by hand, a new browser version might be released, bringing new features and failures into the mix.

Fortunately, the hardy designers have not given up. Instead, they have created web communities dedicated to avoiding the inconsistencies and writing the best possible code. These communities have constantly employed the newest technologies to evangelize the word: first forums, then wiki sites and web logs (blogs). A more experienced web designer will typically have to maintain his palette of techniques by following several sites. Scientific articles are published only exceptionally, making references scarce.

Writing around the problems has met with partial success. Even though the amount of code required for exception handling grows, the available network bandwidth and client-side processing power seems to grow faster. After the so-called fifth generation browsers [W-J07b] were released, web sites written in XHTML 1.0 and CSS could be labeled as *adequately* accessible. The vast majority of the users see the same page with little or no differences in layout, and the minority of users with dated browsers are urged to upgrade as soon as possible. Even mobile devices have begun to support multiple browsers and upgrading.

The current trend is towards more responsive and desktop-like web applications [Nie05]. The first web applications used forms and links in order to receive input from the user. Combined with server-side scripting techniques, the input would be validated and then either processed or the errors returned to the user. From this simple technique, quite elaborate user interfaces became possible. But these applications were always forced into the lock-step of the input-validate-process cycle. Usability studies were quick to declare the validation delays harmful, as they stopped the work flow of the user [ShM97]. The users felt that after every minor error, they were interrogated with the same questions over and over. On a larger scale, the bag of tricks created by the ingenious use and abuse of the form tag was beginning to thin. Some UI solutions remained simply unreachable as long as every input-output request required a page redraw, causing its signature interruption to the user.

In the rapid fashion typical of web development, multiple solutions were soon published by the community. Some of the early ones stem from the first rounds of the browser war and are dependent on browser-specific capabilities. However, with the heightened interest into writing code accessible for all, only the more compatible solutions have gained popularity. Common to most of these techniques is the need for JavaScript. JavaScript has been both the core factor in current Dynamic HTML techniques and, often simultaneously, infamous for its leaky security implementations. With just cause, a number of security experts have gone as far as to recommend keeping JavaScript disabled in security-focused environments [CER00], if not everywhere. Luckily for web designers, users do not always listen.

As JavaScript was soon defined as its own standard, new browsers kept the scripting language enabled by default. For several years now, even mobile devices have featured improving support. Through this extending deployment, one method for achieving responsiveness has gained significantly more popularity than its predecessors. Once a newcomer, this technique was dubbed Ajax in 2005 [Gar05]. It has since reached a respectable level of maturity, and an explosive coverage in current web services. But Ajax is not yet beyond the influence of the browser wars. Writing code for the current browsers requires would-be designers to learn where browser implementations differ. As mobile devices gain in performance, the number of implementations increases, not the opposite. The W3C Document Object Model (DOM) aims to give designers a common interface for all browsers, but accessing this interface requires both new knowledge and discipline. In order to further lower the learning curve, help has arrived from several sources. In two years, a multitude of Ajax toolkits have been released to aid designers master XHTML, CSS, JavaScript, and the DOM. If and when the browser wars will finally end, the designers can do away with these crutches and really start writing universal web applications for all devices. Until then, it is a question of choosing the right toolkit amongst the contestants.

The purpose of this study is to measure how well newer mobile devices perform when handling Ajax requests. Chapter 2 is a review of predecessor techniques, some of which still exists hand-in-hand with Ajax. The chapter also shows how the component technologies of Ajax play their part from a mobile point of view. The research questions, methodology, and the selected mobile devices are presented in Chapter 3, along with the browsers available for them. Chapter 4 explores the most popular toolkits currently and evaluates some sample applications developed with them. Chapter 5 examines real, production-quality applications in their normal environment. Chapter 6 describes still unresolved problems and critique against using Ajax on a mobile, or any, device. The results are concluded in Chapter 7, followed by a short acknowledgment of people partially responsible for this thesis.

2 The History of Ajax

Ajax is a loosely defined collection of web design techniques used to make web services more responsive by minimizing waiting periods through asynchronous server communication. The name Ajax is based on the abbreviation[1] of its core components: Asynchronous JavaScript and XML. Ajax has become very successful in a relatively short amount of time. In the two years since its naming [Gar05], the number of both scientific and popular publications has sky-rocketed. However, an easily overlooked fact is that similar solutions have existed for almost a decade. As many of its predecessors, Ajax builds on standardized and mature technologies. The main difference is that while many of the other techniques employ more creative uses of (unintended) side effects, Ajax relies on browser features that are designed for communication.

Ultimately, all of these techniques rely on browser support. The flood of Ajax-enabled web sites could not have happened without the gradual shift towards newer browsers. The common requirement is a so-called *fifth-generation* browser [W-J07b]. This moniker is borrowed from Microsoft Internet Explorer version 5.0, as it was the first browser that implemented the W3C DOM interface. Other manufacturers were quick to add support for the new recommendation. Today, it is hard to find a desktop browser that belongs to any of the previous generations. But on mobile devices, external limitations

[1] The capitalized form “AJAX” has quickly fallen out of fashion.

might force the amount of features to decline, not increase.

This chapter reviews the motivation towards more desktop-like web applications. Later, the component technologies of Ajax are reviewed specifically with mobile devices in mind. Finally, the last parts of this chapter present alternative techniques and how they may be combined with Ajax. Program examples are omitted in favor of the multitude of tutorials already available.

2.1 From Web to Desktop Applications

As both the public and the features of the World Wide Web grew, it became first fashionable and then efficient to design services specifically for the web. Commercial ventures have met with varying success, but public organizations have slowly but surely spread onwards. Reaching as large a crowd as possible by easily accessible documents is a notable principle, for it matches well with both advertising and non-commercial services. As the web itself is either free or relatively low-cost, it has become a reasonable development platform for applications previously only released for operating systems.

This transformation from the desktop to the web has also become a limiting factor for applications. The first hindrances lay in program logic, for client-side scripting support was riddled with inconsistencies, but also because any client-side processing was by nature open and unreliable. These problems were solved by shifting control away from the clients using server-side scripting. Ajax can be seen as an effort to move some of the processing back towards the client.

Secondly, the web protocol presented an uniform interface for clients everywhere and every time. Each document request was processed as unique and totally independent of any other. However, this anonymity logically prevented user authentication necessary for identification purposes. Thus, cookies and sessions were bolted on the protocol. Their use met up with heated debate from the web community [StS02], but eventually the necessity was accepted.

The third problem is the user interface. The request-wait cycle is seen as both disruptive and disorienting by the users. It is disruptive because the indeterministic delay caused by other users and network traffic. These factors make it difficult to decide when the request has failed to reach the server, which may lead to duplicate requests. The cycle's disorienting effect is caused by the necessity to redraw the browser window after new output from the server. It requires the user to remember the previous page, read the new one, and try to determine what changes have been made.

These three problems are not random by nature. They closely resemble a common design pattern, the model-view-controller approach [CPJ05 ch. 3]. Design following this pattern splits the content of the page into a business model, a view of the the functions available, and a controller intermediating between the two. On web pages, the output of the browser is the view, the business model is held within the web server, and controller functions are traditionally handled by server-side scripting. This split requires every input to pass into the server to be processed, causing both the delay and the redraw.

Logically, responsiveness improves if we can avoid some of the delays and redraws. To achieve this, two requirements must be met: a communication channel and an interface for partial redraws. The communication channel must enable delayed or event-driven requests to the server, or push transmissions from the server to the client. Partial redraws enable the elements of the browser window to be changed individually, either through a standard interface like the DOM or a more proprietary programming layer. Further requirements for productivity in web applications are listed by Yu *et al.* [Yu06].

2.2 Ajax Builds on Existing Technology

Ajax is not defined in any specification. There exists no validation service to stamp a specific web site for Ajax compatibility. Therefore, defining an “Ajax application” is a somewhat ambiguous task. When an application is said “to be Ajax” or to “use Ajax”, the application employs a *design pattern* [Gam95] that is recognizable despite implementation differences. The implementation depends upon lower-level components or technologies, some of which may be validated against existing specifications. In addition to the core components, a number of other technologies are implicitly assumed, because the primary platform for Ajax applications is the World Wide Web. For example, pages may be written in (X)HTML and CSS, JavaScript may use the W3C DOM to manipulate partial redraws, and XMLHttpRequest objects may be chosen as the asynchronous communication channel. However, using any single component technology is not strictly required either. The following sections will review the components and present some ideas on how they could be exchanged with suitable alternatives.

An important design principle with Ajax is the bias towards open technologies, i.e., those that are specified and governed by non-profit organizations. At the same time, Ajax builds on existing technologies in the sense that its components have been proven to be adequately supported by existing browsers. This implies that the technologies have been around for a while, but note that the support is still not uniform. As Ajax depends on the underlying technologies, differences in their implementation cause variation in the execution of the applications.

Ajax components are selected for different purposes. Their tasks can be divided into the layers describing the content (alt. semantics), presentation and behavior (alt. functionality) of the web application [Yan07]. In this division, (X)HTML is used to mark the content, CSS to express the presentation, and JavaScript to change the presentation's behavior. Refactoring and upkeep are simplified by also separating the code in different files [Ant07]. The division may be emphasized by writing the application gradually, beginning from the content and continuing through presentation to its behavior. This iterative approach is labeled *progressive enhancement*, and it is very closely related to its sibling, *graceful degradation* [Cha03, Ols07]. Common to both is the idea that the browser can “decide” which parts of the application to display, by simply ignoring those layers that the browser does not support. On mobile devices, progressive enhancement may be the better solution, since it tries to guarantee that even the most limited viewers are served a working solution.

Without persistent connections (see Section 2.3.4), each HTTP file request yields a separate TCP connection to the server. Each connection requires its own TCP handshake, which takes time [Yah07e, Sou07]. On high-latency links this delay becomes clearly noticeable, and therefore, undesirable. This means that if persistent connections are unavailable, the benefits of content separation have to be re-evaluated in mobile environments where the high latency is typical. Placing the layers into a single file may be the better choice, but note that the idea of progressive enhancement need not suffer. An example of this technique is the Google default search page, as it embeds both script and CSS into a single file.

2.2.1 JavaScript

JavaScript (JS) is a client-side scripting language originally designed for use in Netscape web browsers. JavaScript has been widely adopted for use in other browsers, most notably Microsoft Internet Explorer. In August 1996, Netscape let JavaScript be defined in an open standard by Ecma International[2] [And98]. It is important to note that there are currently three different versions of the ECMA-262 standard [Ecm99]. All browser

[2] Electronic Computer Manufacturers Association, abbreviated ECMA, before its renaming in 1994.

implementations offer differing degrees of support of the corresponding version. Netscape, Microsoft, and other developers have extended their implementations with browser-specific features. At times, browsers have also manifested reduced functionality in parts of the implementation.

There is considerable confusion regarding JavaScript's naming. JavaScript is a registered trademark of Sun Microsystems[3], but the language bears little resemblance to Java. The project was code named both LiveScript and Mocha during its infancy [Cro01]. This confusion has not alleviated, as current browsers have chosen to call the language JavaScript, ECMAScript, or by more local titles. The renaming is done to avoid potential trademark issues. Internet Explorer's variant is officially called JScript [Mic07b] whereas Adobe Flash employs a version called ActionScript [Ado07a]. Please note that Flash is not a browser, but a browser plugin. ECMAScript implementations have begun to surface in other areas outside the browser sandbox as well. Wikipedia makes an effort to list all dialects on its ECMAScript page [Wik07a].

Looking at the amount of names, one expects the language to be a dear child of developers everywhere. In fact, JavaScript's reputation has been quite the contrary. A number of security concerns once made the language infamous [StS02, NVD07], compromising its use in web design. The merit that balances this flaw is JavaScript's deployment. Barring disabling by the user or a local policy, JavaScript is available wherever a browser is installed. As the ECMA-262 core has been implemented in mobile browsers as well, the 80 million [IDC07] mobile devices shipped last year alone make JavaScript one of the largest programming languages in the world.

2.2.2 XMLHttpRequest

The XMLHttpRequest (XHR) object was implemented by Microsoft in March 1997 [Esp00, Wil03]. XHR was originally an Internet Explorer 5.0 ActiveX object designed to work with Outlook Web Access. Mozilla-, Safari-, and Opera-based browsers were quick to catch on to the idea and implement their own versions of XHR. Recently, it has also been accepted as a W3C Working Draft [Kes07]. XHR:s purpose is to combine an asynchronous communication channel to the server with the XML document format. Despite its name, XHR also supports plain text as the message format. As JavaScript objects can be encoded into plain text through JavaScript Object Notation (JSON), it is clear that XHR provides a reasonable level of versatility for the designer. Even though

[3] http://www.sun.com/suntrademarks/

self-defined message formats may yield performance benefits, the use of XML or JSON is recommended for reasons of convenience, maintainability, and robustness. In desktop environments, a well-tested and maintainable interface is worth the performance cost.

An existing limitation with XHR:s communication channel concerns its source domain. By design, XHR requests may be directed only to the same host name from where the original document loaded. This is done to avoid cross-site scripting attacks (XSS), where a malicious host uses the client browser's credentials, e.g., stored passwords, to request services from a trusting host. XSS will be more thoroughly examined in Section 6.1. Server-side request forwarders may be used to bypass this limitation locally [CPJ05, ch. 7.2], but special care has to be taken in limiting the hosts addressable through the forwarder.

2.2.3 Asynchronous Events

Mobile devices are used in varying types of networks, which implies that the network properties may also vary. The development of mobile carrier networks has enabled connections with increasing amounts of bandwidth, but the latency of current networks has not decreased at an equal pace [CCP02, Cat05, VRP05]. This means that mobile users are susceptible to undeterministic amounts of latency, depending on the available network types. A heightened latency quickly yields equally heightened response times, injuring the sought-after property of responsiveness.

Whereas shortening latency might remain the long-term development goal, an already available solution is to employ asynchronous requests for the communication channel. An asynchronous request will yield a response after an unknown round-trip time, but the user will be able to continue viewing the current page while waiting. The key element here is to write the client-side script to react to events from the client browser. An interface event, e.g., click of a button, might cause the original request to be sent. Another event will be generated when the request is sent, and it may be used to pacify the user by letting her know of the ongoing process. A third event is generated by the arriving request. A call-back function could then be registered with the third event type. The function will process the response content, displaying the output or a suitable error message.

Taking a look from the usability perspective, the second event may be as important to the user as the two others. As the whole point of using the asynchronous call is to avoid the now-familiar refresh delay, the user might easily become alarmed that something went amiss with the original request [Dis06]. In a normal request-reload operation, the browser's loading icon is animated to signal communication in progress. Due to limitations in display area, a mobile browser might not contain a loading icon at all, or it might not respond to XHR requests. It is therefore necessary to inform the user with a separate display element, suitably positioned to convey the meaning but not cause undue irritation.

2.2.4 XML and Verbosity

The Extensible Markup Language (XML) [Bra06] is the X in both XHR and Ajax, but it is not required *per se* of any Ajax application. As noted, an XHR object is capable of handling responses in multiple formats through its support for plain text. Further, there has been grounded concern [KLT05] regarding the verbosity of the XML document structure and the amount of processing power it takes to read and write XML. These problems have made it questionable whether XML is suitable for use in mobile devices. The answer to that question has recently turned out to be positive, with some reservations [Kan07]. The space and time requirements for XML processing can be much alleviated through the use of binary XML [GoL05]. Also, HTTP might not be the optimal transfer format in wireless environments, due to similar concerns about the protocol's inherent verbosity.

An important side note here is the necessity of server-side support when responding to an XHR call with a XML document. The originating browser does not always know the response type when making the call [Eer06]. Instead, the server-side script must set the appropriate content type before starting its output. Without an explicitly set content type, the output will be received as plain text by the browser.

2.2.5 Displaying CSS

Cascading Style Sheets [LiB99] are the most common method for specifying the layout of a web page today. A long-time division of current layouts is between the *fixed* [Hol99] and *fluid* (alt. liquid or dynamic) methods. Please note that the following definitions mainly address window width, whereas the height is indirectly determined by the amount of content.

A layout is said to be fixed if it makes assumptions of the browser window size. For example, fixed layouts use more invariable measurement units, usually pixels or points, to specify element sizes. This is done to ensure that the layout is rendered equally among the clients. If the browser window is shrunk beneath the intended display size, the user will be forced to scroll the window in order to view all of the content. It might hamper readability a great deal, as forcing the user to scroll the window for every line read is clearly not a sign of great usability.

If no assumptions on the browser window are made, the layout is said to be fluid. This means that the browser's layout engine is left with the responsibility of rendering the page as it sees fit. If the browser window shrinks, the elements may be repositioned in order to retain readability. The fluid technique presents an alternative flaw: when the browser window grows, there exists no maximum width for elements. This will, again, hamper readability if the text lines stretch to become overly long.

Another problem with both fixed and fluid elements concerns the use of images. As bit-mapped images contain intrinsic width and height properties, they will not scale with the rest of the page. The use of vector images [FFJ03] has been proposed, but they have not yet reached wide-scale deployment.

A newer technique designed to answer some of these problems is the use of *elastic* layouts [Gri04]. Elastic layouts contain both fixed and fluid elements. Picture dimensions may be specified in units that are relative to the selected font size. As a bit-mapped picture will not scale without some artifacts, so this is only a partial solution.

As we will see later on, mobile browsers have responded to the layout problem with browser zooming. Zooming the display window allows the user to catch the overall layout of the page, and then concentrate on the selected level of detail. Pictures are then typically viewed with a resolution smaller than the inherent dimensions of the image, causing no artifacts but, instead, some redundancy in the transfer size.

2.2.6 Document Object Models

The abbreviation “DOM” is used interchangeably with “W3C DOM”, even though the former version is somewhat imprecise. Alternative document models are sometimes called browser object models (BOM) [Ler06] to distinguish the use of a browser-centric version. As briefly mentioned earlier, the DOM was first supported by Microsoft Internet Explorer's fifth version. Before Microsoft's decision to support the DOM, competing browsers had very much contained their own models [Koc04]. An important early competitor was Netscape Navigator's *layer* interface [Sco06a], although it supported a more limited set of functions. When Internet Explorer 5 became the dominant browser, due to reasons not discussed here, the DOM consequently gained much in popularity.

Both the DOM and different BOM:s were first used to implement DHTML-like techniques, which are now reused in Ajax. The idea of the W3C DOM [Hég04] is to model the displayed document as a hierarchical tree structure, which may be manipulated by adding or removing elements as child nodes. Changes are then propagated to the visual representation. Using a programmatic approach is desirable, for it allows the designer to manipulate whole sections of the currently visible page. More advanced techniques also become available, such as automatically generating repeating elements by just passing the generating function to the client.

Critique is mainly directed towards the necessity of strict programming techniques, lest elements be removed with their parent by accident, or other such unwanted side effects ensue. Also, statements that access the interface can become quite lengthy, depending somewhat on the programming style chosen for the browser. With desktop browsers, a tertiary flaw exists, and it becomes more important as we move on to mobile devices. This issue concerns the speed of the browser's implementation of the DOM. The normal behavior for a browser is to build the hierarchical model from server output, parsing and interpreting the text into a tree structure. After this, only small modifications to the presentation are expected. It is wise to concentrate speed optimizations on the most common tasks. When the application's behavior changes and more advanced functions are accessed, speed may quickly become an issue [Geo07c].

A shortcut exists in the form of the *innerHTML* property, which allows designers to replace a node's complete subtree with a text string, subsequently interpreted as code. Multiple sources [Rah06, Hed07, Leb06, Koc06] have commented that the innerHTML method is significantly faster on some, but not all, browsers. A similarity exists between this style and the avoidance of XML for its more processor-intensive interpretation. Advocates have gone as far as to coin abbreviations such as AHAH, AJAH, and AXAH, where the replaced letters underscore the use of (X)HTML. The position of this thesis is that the use of innerHTML may indeed be valuable for mobile devices, but these techniques are already encompassed by the existing name Ajax. XML has never been a prerequisite, only a suggestion.

2.3 Alternative Techniques

The motivation for this partial review of alternative techniques lies within browser differences. The techniques are not disjunct in the sense that one is prohibitive of using any other. Using two different methods to achieve the same goal might improve compatibility with different browser versions. In the case of Google Maps [Goo07b], the amalgam of Ajax with the Hidden IFrame technique enables the use of the back button within the *same* browser [Web05b]. In fact, Google Maps is one of the most famous Ajax applications, and it does not use XHR as its main communication channel [Web05a].

A rigorous listing of early communication techniques is by Wilton-Jones [W-J07c]. Many of the solutions listed there have already disappeared from use. In addition to the more inventive approaches, even a W3C recommendation [StH04] has been reduced to having nothing but the most rudimentary browser support.

2.3.1 Dynamic HTML

Dynamic HTML [Goo02] is the moniker for the combination of HTML with client-side scripting and a browser interface. DHTML originally appeared at the height of the browser wars. Early DHTML pages used browser-specific extensions for scripting and interface calls, so that their compatibility was severely limited. Later DHTML techniques have embraced JavaScript in combination with CSS and the W3C DOM. Thus, DHTML today has much in common with Ajax. Both employ the same core technologies to generate a wider variety of user interfaces.

With DHTML, the goal was to make the *content* more dynamic, not the *interface* more responsive. Beyond those labels, there was little difference in the effect. The major difference to Ajax was that DHTML, as a technique, did not contain a communication channel for data transfers. If page redraws were to be avoided, DHTML-enabled pages had to be populated in advance with the necessary data for partial redraws. This method was less than optimal, since making each client download irrelevant portions of the application database can quickly become either inefficient, insecure or both.

It is worth noting that some designers [Ash00, CPJ05 preface] were craftier than this, and combined DHTML with a communication channel to avoid the prepopulation of data. However, without a catchy name, targeted publication to professionals, and popular applications to capture the audience, these solutions did not reach the fame of Ajax.

2.3.2 Delayed Inclusion

The technique of delayed inclusion is similar to many other patterns that achieve reductions in the initial loading time of the web application [Aja07a]. These resemblances stem from the same basic assumption: not all users will employ every service available. A classic example is a web forum were users might read posts anonymously, eventually register to the service, and then post messages themselves. If the forum is popular, it is reasonable to assume that the majority of users will mostly read and occasionally post. Therefore, it is inefficient to force every user to download the program code required for posting functionality. Different patterns solve this problem by varying the exact moment the additional functionality is downloaded by the user.

The common factor behind all these patterns is the fact that when new JavaScript files are loaded by a browser, their contents are immediately evaluated. This means that the additional JavaScript runs within the context of the current page. By defining a script element to load a file from the server upon receiving a suitable event, the developer may control which file is loaded and when it is loaded. This event may be triggered by a client-side timer [Lun02] to achieve periodical refreshing of content. It is a small task to parse the parameters of the request at the server side and generate client-specific content, if necessary. Further, the communication is not limited to the same server that supplied the original page – any accessible URL will do. This seemingly minor feature bypasses the “same domain” -limitation of the XHR, which is revisited in Section 6.1.

Overall, the technique of delayed inclusions is very compatible with the vast majority of browsers. Therefore, it is helpful in several situations where it is beneficial to let the server perform parts of the computation. From the client's perspective, responding in JavaScript offloads the burden of parsing the response. Finally, delayed loading is not strictly limited to JavaScript. For example, image loading may be fine tuned in order to present input queries as early as possible, or to smooth transitions from one page to another [Alm07].

2.3.3 Hidden IFrame

The name IFrame stands for ”inline frames”, paying tribute to its predecessor, HTML frames. The major difference between the two is that while IFrames embed the specified page within the current document, frames split the browser window into multiple pages [RHJ99, ch. 16]. Frames and their correct use became a hotly debated topic because they could easily cause a number of user interface problems [Nie96]. Most notably, using frames made the back button notoriously difficult to predict for the user. Designing with frames quickly became *passé*. IFrames have not met with the same explosive fame, but their success has turned into a more lasting kind.

The hidden IFrame approach is one of the most widely adopted communication channels for asynchronous communication. It enjoys browser support beginning from the *fourth* generation of browsers, i.e., those preceding support for the W3C DOM. It is relatively easy to understand, suffers from no restrictions on the source servers, and has been around since at least 2002 [Cos02]. Even before that, the same technique was possible through the use of frames, with some visual differences.

Despite all the praise, the hidden IFrame approach is essentially a hack, meaning that it employs an unintentional side effect of the way IFrames are loaded and displayed by most browsers. Hiding an IFrame involves specifying an invisible window element into the current document, and then loading a page into that element. When the page has finished loading, its content may be parsed and passed back to the IFrame's parent, the current document. All this is achieved with a suitable scripting engine. The IFrame source is not limited to the original server, and the loading may be either caused by an event from a window element or by any script executing in the browser.

IFrame has an additional *forte*. Targeting content with the hidden IFrame causes another side effect – the browser catches on what is happening. This means that the functionality of both the browser's history and back button are easily accessible to the IFrame approach. The cost is an increase in memory usage relative to the amount of content loaded into the hidden IFrame. Essentially, the same content is loaded twice. As the IFrame element is trivially reusable, this cost seems low by many designers.

2.3.4 Comet and HTTP Pipelining

Common to all current communication channels based on HTTP is that transmissions are always initiated by the client. Clients imitate push behavior by relying on events triggering requests for updates from the server. By choosing event timers or update conditions intelligently, this approach may be quite adequate for the task at hand. Still, some of the responses might indicate that no new changes have been made. The most efficient communication method is to avoid these unnecessary requests altogether. Logically, this happens when the server can tell the client to update its content. This can be specially desirable in conditions where each message has a resource cost, be that in battery time or financial units.

HTTP Streaming [Aja07b], also known as "Comet" [Rus06], is a technique designed to remedy the restriction of one-sided connect initiation. Comet extends HTTP [Rus07a] by allowing the server to truly push content to its clients. It works by letting the server keep the TCP connection open, sending delayed responses to the original request. In this manner, the original synchronization request serves as a client registration, giving the server permission to push all changes from that point onwards. Except for this initiator identity, Comet has a resemblance to pipelining, introduced in version 1.1 of the HTTP specification [Fie99]. With pipelining, the client is able to send multiple requests through a single TCP connection. Both of these techniques help the client to avoid making spurious TCP connections, which can be costly in high-latency environments [LaH03]. With Comet, the responses just take a little longer to arrive.

Pipelining is an old extension, while Comet is quite new. Their problems resemble each other, however. Current browsers seem to contain lenient settings for server timeouts [Bur07, Car07], so employing Comet requires only added server-side support. HTTP pipelining requires support from the client as well, and its deployment is patchy at best. The most recent Microsoft Internet Explorer is version 7, and it still does not support pipelining. Mozilla Firefox has featured support for a while, but it is disabled by default. The reason for this is the need for heuristic tests for pipelining support on the HTTP server. Such support can be still seen as experimental, as there are many alternative servers and their development is volatile. Notably, Opera currently features an heuristic test for pipelining support, and thus installs with pipelining enabled.

The interest for Comet seems to be increasing [Arc07, IBM07, Kne06], which makes it conceivable that Comet might become better supported than its predecessor. In environments where Comet is supported, it may be incorporated with both the IFrame approach and the standard XMLHttpRequest communication method [Gul07]. But even with the browsers that support pipelining and/or Comet, there are problems with combining the technique with multiple windows. As tabbed browsing has become popular, persistent connections make it easy [Rus07b] to hit the roof of maximum two concurrent connections to each server in the HTTP specification [Fie99, Section 8.1.4]. One simple fix is to direct the persistent connections to a host name dedicated for this purpose.

Another problematic area is the need for support in HTTP proxies, which are still popular due to their lowering effect on bandwidth consumption [Dav07]. Moreover, some mobile browsers employ proxies for content transcoding, e.g., reducing picture resolution to match screen size [Law01, Leh06]. Leaving the TCP connection open would require features similar to Network Address Translation (NAT) from the proxies, essentially keeping tabs on each and every client requiring support for persistent connections.

2.3.5 Flash and Java

Adobe Flash [Ado07b] is a browser plugin that excels in vector-based graphics, animation and sound. It is commonly used on web sites that rely heavily on multimedia. Typically, Flash is adopted by games and movie advertisement services. Despite its proprietary origin, Flash has reached an unparalleled level of deployment. Plugins exist for each *major* combination of operating system and browser. Flash enables partial redraws and asynchronous communication, among a host of other features. Albeit this overlap, Flash and Ajax are not strictly competitors. Popular sites like YouTube [You07] have demonstrated how the two techniques' separate strengths may be successfully combined. YouTube employs Ajax for the its interface and switches over to Flash for the video content.

With the release of Java Web Start [Sri01], the runtime environment required by Java may be bundled together with the application itself. Java Web Start is a step away from the applet paradigm, and it has been criticized because of its loose integration with web content [Rob07]. In the field of mobile devices, Java has typically been supported through the use of the Mobile Information Device Profile (MIDP) [Sun07]. The major problem is that different platforms have small but relevant disparities concerning the actual implementation of many interfaces in the Java API [Jao06]. These disparities have typically required extensive testing to ensure that the cross-platform program truly works on all devices. The magnitude of testing required has proven to become an incentive for designers to look into other possibilities. Here, Ajax enters into the mobile device scene.

3 Measuring Mobile Devices

For the first time ever, users are voluntarily downgrading to browsers with fewer features than the earlier generation. These mobile browsers feature less display area, less processing power, lower network bandwidth but increased latency, and the completely new factor of a limited battery lifetime. Simultaneously, the use of techniques like Ajax involve browser capabilities that have only become common enough during the past few years. It seems questionable if the mobile devices can actually support Ajax in the needed capacity. And indeed, it is correct to question this, since many devices can not.

As astute reading may have revealed, the selected statistic of 80 million devices manufactured in 2006 [IDC07] is limited into a subset of all the mobile devices shipped. This subset is called the *converged mobile devices*, and it is loosely defined as the set of devices that combine features from mobile phones, personal digital assistants (PDA), cameras, tablet computers, and other electronic aids. Here, converging means merging multiple separate units into a single, more intelligent device. The selection is limited to this subgroup because the majority of the devices sold last year, the more old-fashioned ones, are still incapable of supporting all the component technologies of Ajax.

3.1 Capability and Performance

On the other side of the spectrum, the smallest laptop computers have reached form factors almost comparable to mobile phones of yesteryear. While testing laptops running the same browsers and operating systems as their desktop brethren would probably be indicative of variations in network properties, in this thesis the focus is on the handheld category of devices. More specifically, the tests concentrate on the browser performance of the *least* capable hardware platforms that still are able to support Ajax.

It is important to realize that because of the selection criteria, the constrained browsers also feature lessened levels of *software* capability than their desktop versions. Browsers are separated into a graphical user interface (GUI), sometimes titled the *chrome*, and the background *engine*, which handles the interpretation and execution of the content passed to it. While the browser's GUI shares its name with the fully-functional version, some features may have been removed from the constrained engine. These lacking features may greatly benefit rendering performance, as unsupported program code is simply ignored. The fastest browser is the one that constantly displays a blank page.

Therefore, it is important to not only measure the performance of the constrained browsers, but also to evaluate how well the browsers are doing the work they are supposed to. This calls for a combined analysis of both the quantitative and qualitative properties of each browser. Combined with the results of each measurement is an overall grade (see Section 3.3.1) of the the tested browser. Comparisons between browsers with a different overall grade are sternly discouraged – they should only be performed by the most informed readers, e.g., the browser developers.

The selected devices are presented as educated guesses of what the “long tail” of mobile devices might be like in the near future. At the time of writing, the devices are clearly in the enthusiast-level of the market, with local price tags ranging from 320 € to 640 €. Preceding a more thorough description, we present a short motivation for using Ajax in mobile devices, followed by the research questions that these measurements seek to answer.

3.2 Motivation for Mobile Ajax

Four easily identified benefits with the use of Ajax are user input speed [Nok06], lesser display processing requirements, smaller amounts of transferred data [Whi05], and better compatibility between devices [ByH07]. These benefits are not unique to mobile Ajax, but their rewards are relatively greater when there is less processing power and bandwidth available.

First, writing text on a handheld device is categorically slow. The reason for this is that lugging a 102-key keyboard around is uncomfortable for most of us. Handheld devices employ smaller input methods: both reduced alphanumerical pads and stylus-based virtual keyboards are well known by now. Predictive text input has become the proverbial helper wheel. With Ajax, prediction may be performed twice – once on the device, then once more using a more application-specific vocabulary on the server. The trade-off is doing a (costly) network call, which should finish faster than the user can enter text.

Second, using the partial redraws customary for Ajax applications allows the mobile user browser to avoid full page redraws, sparing processing power, and thus, battery life. The user experience is simultaneously enhanced, since on slower devices, the white-out between complete redraws is visible for a longer period of time.

Third, when delays are considered, newer wireless techniques like EDGE and UMTS are categorically high-bandwidth but also high-latency. Returning responses in a verbose format like XML may remain feasible, but XML may also be replaced with JSON or basic HTML. When compared with a complete page transfer, communicating only partial changes reduces the amount of transferred data [SmS07]. If both the browser and the server support pipelining, even the relative overhead of performing the triple TCP handshake is reduced.

Fourth, using the mobile browser as a programming platform offers an additional level of middleware for application developers. Instead of having to port each application to every mobile operating system, the effort can be directed towards making the browsers more compatible. While current browsers are restricted by a security sandbox preventing access to much of the device's services, e.g., audio, contacts, and GPS, crossover Java/JavaScript libraries are a suggested remedy. At least one such project has begun development [ByH07]. Ajax functionality by itself might be improved by using toolkits specifically designed for mobile devices [Geo07a].

3.3 Research Questions and Methodology

Better than the previous motivation for mobile Ajax, the sheer number of devices available makes the time seem ripe for a performance measurement of mobile browsers. While surveying the field of mobile devices, it quickly becomes clear that there are already several browsers available for most platforms. A newer development is the onset of small web-enabled applications or *widgets* [Geo07b], which offer more narrowly defined functionality than browsers. Collectively, these mobile *user agents* reuse components from each other and desktop browsers. It is therefore interesting to see how the components perform on the much more constrained properties of a mobile device.

Wilton-Jones has performed meticulous application measurements for a suite of desktop browsers [W-J07e], noting significant variance between different browser/platform combinations. Further, he has duly noted that browser differences make some measurement techniques unreliable [W-J07d]. For example, using JavaScript to measure JavaScript may yield inconsistent results due to a different ordering of processing and presentation phases among the compared browsers.

Following the emphasis on practical testing set by Lilja [Lil00], the measurements in this thesis try to avoid synthetic benchmarks. The benchmarked hardware units are commercial, off-the shelf (COTS) products, not samples or development units. No modifications have been made to the hardware or their user agents, unless explicitly specified otherwise. The exceptions to this rule are some cache settings, explained in detail with the browser descriptions. All of the device measurements were run without an attached charger to accommodate for power-saving features.

Since Ajax employs so many of the available technologies, the newer Ajax applications offer a diverse real-world test suite for browser performance. In addition, the number of development projects is impressive. Since an important reason behind the ubiquity of the World Wide Web is audience participation, a selection of development libraries are included in the measurements. The rationale is that for Ajax to become successful, the users must have tools that mask browser differences. At the same time, the libraries must attract users. Therefore, the benchmarks target twelve sample applications promoted by the developers as showcase examples on what Ajax is capable of.

3.3.1 Grading and Results

Before the performance measurements, the applications are first tested for capability issues on the selected browser. All defects are noted and explained in the corresponding analyses. Each browser is given a grade on the scale from A to D, explained as follows.

- A represents the highest possible grade, with no defects detected during use.
- B represents minor problems with the layout, or missing functionality.
- C represents major problems that clearly are a hindrance to the user.
- D represents a failure in the workflow, meaning that the user does not receive the requested service from the application

Note that application speed is considered subjective, and is not factored in the grade, but may be noted in the analysis (when relevant). This grading is separate from the actual time taken to load an application, which is presented by the later graphs.

3.3.2 Measurement Method

After possible defects have been determined, the application is measured using ten iterations of the initial loading time or time taken to perform a selected function. The results reflect user-perceived wall clock time, measured using a stopwatch. Although this method limits the precision to 1/10 of a second at best, it is fully adequate to represent human reaction time [Kal04].

Results are separated into cached and cache-cleared categories to calculate an approximation of the upper and lower bound of the (perceived) delay. Cached measurements try to indicate the best case, i.e., the one where most parts of the application are already loaded. Conversely, the cache-cleared measurements try to indicate the worst case, i.e., the one where no parts of the application are already loaded. Please note that the worst case still ignores the WLAN connection setup time, as explained below. Preliminary testing includes that the connection time may greatly vary, and indeed be vastly longer than the time taken to load an application. Additionally, some users may have trouble distinguishing these components of the overall delay.

Because of the very mobile nature of said devices, performance measurements are tied to a multitude of real-world factors, the type of the network selected not being the least. It has been shown earlier that in slow networks, the transmission time easily dominates the overall wait time for a page load [LaH03]. In this thesis, I have tried to minimize the dominant factor of the transmission time and concentrate on the browser performance alone. In all of the following cases, the network connection type is WLAN, with the access point connected to a very high-bandwidth, low latency university network. All of the targets were accessed through a (very plain) XHTML link page so that the WLAN connection was set up during the first request, not the measured one. To reiterate: these measurements **do not consider** network connection properties.

In the cache-cleared category, the disk (flash) cache is cleared between each test, using the functionality offered by the browser. To further minimize the effect of memory caches, on the N95 and N800 the browsers are measured by interleaving the tests. In other words, in each iteration, browser A's cache is cleared, then browser A is measured, and allowed to completely exit. Then browser B is executed, its cache is cleared, the measurement is performed, and B is allowed to exit. On the iPhone, memory caches were flushed by executing other applications between iterations, e.g., viewing photo slide shows, taking pictures with the camera, or browsing the calendar.

Many of the details in the measurements have been omitted in the written text by necessity. All of the raw data gathered, along with the notes taken during the measurements, have been included in the appendix. In the electronic version of the appendix, the benchmarked values are recorded with the precision measured. The values are normally presented with single decimal precision.

The following browsers have passed initial testing of their basic Ajax functionality. This means that using short example programs, the browsers have been verified to support the component technologies of Ajax.

3.4 Nokia N800 Internet Tablet

The Nokia N800 Internet Tablet (N800) [Nok07c] is the oldest device of the three, having been released in January 2007. An important first observation is that the device is not a mobile phone – it lacks any and all functionality required to connect to a carrier network. Despite this drawback, the N800 includes a host of internet-related features that make it fit into the category of converged mobile devices. To mention a few, the device supports multiple web browsers, can run its own web server, and offers the possibility to use both Skype [Sky07] and Google Talk [Goo07a] for voice-over-IP calls. Mapping services are available through the maemo mapper project. In order to use the Global Positioning System (GPS), a separate GPS receiver must be connected.

As of writing, the current operating system is titled the "Internet Tablet OS 2007" (OS-2007). The tested version was 4.2007.26-8. Most notably, it is based on the open sourced components GNU/Linux and GNOME [Jaa06, Mae07]. The OS2007 contains its own package manager, allowing the device to connect to multiple repositories in order to fetch updates and community-developed software.

N800's input method is based on a touch screen, either through a virtual keyboard or handwriting recognition. The display size is 4.2 inches, 800x480 pixels with a color depth of 65536 colors. The device has 128 MB of RAM. Storage capacity consists of 128 MB of permanent flash space, plus the possibility to insert two removable flash memory cards, maximum size 2 GB each. The RAM memory may then be extended with virtual memory on the removable flash card. With a total storage area of over 4 GB, the N800 provides ample space for caching purposes.

As the measurements do not concern applications with huge cache requirements, only a single 2 GB flash card was used. The maximum virtual memory of 128 MB was extended through the device's control panel. All system sounds were silenced, which may have benefited battery life. We feel that this decision has not corrupted the results, since turning off the system sounds seems to be normal behavior for observed N800 users. Display brightness and volume level were otherwise left on default levels. Additionally, a Google Talk account was entered into the presence settings, and four speed contacts were configured. This is not a synthetic decision, as these are the actual settings for everyday use.

3.4.1 Opera

For the OS2007, the default browser is based on Opera's version 8. Its code base is closed and strongly related to the commercial desktop browsers developed by the company. Opera users are normally forced to purchase a license after a trial period. On the

N800, the Opera browser is part of the purchase fee, and therefore licensed by Nokia.

The relevant application packages for the Opera browser were osso-browser-opera-dynamic, version 2.0.43-1, and osso-browser-opera-eal, version 2:1.6.8-1. No application updates were downloaded during the testing period. From the browser settings menu, the memory cache size was extended to 4096 KB, the maximum value selectable.

3.4.2 Mozilla based browser for maemo

Thanks to OS2007's engine abstraction layer (EAL), multiple browser engines may be used through the same user interface. The *Mozilla-based browser for Maemo*[4] was released in July 2007 for the general public [Kin07]. The browser claims no shorter official title so, in this work, it is henceforth referred to by its package name “microb”. The installed version of the package microb-browser was 0.0.8-3. By editing the file /home/user/.browser and changing the value of “hidden” to “true” [sic], the browser menu was extended to contain a “Set engine” option. This option made switching between browsers notably easier.

The upcoming newer internet tablet, N810, will be released with the 2008 version of the operating system. Nokia has announced that OS2008 will ship with Mozilla engine enabled by default [Nok07a]. The OS2008 will also be available for the N800 since both contain the same CPU and memory hardware. However, the new version was not released in time to be included in these measurements.

3.4.3 GTK+ WebCore

A third browser engine based on the GTK+/WebKit [Kin06] code base has been in development since at least August 2007 [Tok07, Wri07]. It is perhaps best known for its (promised) SVG [FFJ03] support. There was an incentive to include the browser in the measurements due to the shared code base with the Safari browser described further on.

While the project was briefly reviewed, installation candidates were not yet available to the general public. Therefore, the browser was left out. This decision is based on the dual emphasis on currently available technology and the repeatability of the measurements.

[4] Not to be confused with the “Minimo” or “Moblin” projects, also based upon Mozilla's Gecko engine.

3.5 N95

The Nokia N95 is a “true” converged mobile device in the sense that it is no longer marketed as a mobile phone, but as a multimedia computer. It began shipping in March 2007, which places it between the two other devices. The N95 features an internal GPS receiver, enhanced support for video playback, and a 5 megapixel (2592x1944 pixels) camera. The display size is 2.6 inches, 240x320 pixels, and capable of displaying 16 million colors. Notably the display size is much smaller than on the other two devices. Browsing input is handled by a 12-button numerical/lettered keypad plus a four-way navigation button, with the selector in its middle. Additional buttons do exist, but they are reserved for menus and as shortcuts to the device's camera and multimedia applications.

The operating system is the Symbian Series 60 (S60) 3rd edition with feature pack 1 [S6007]. Firmware version was 12.0.013 (19-06-07). Due to its widespread nature, there are plenty of mobile browsers available for this platform. In addition to full browsers, Nokia has begun promoting the development of lightweight internet applications or widgets [Geo07b]. For example, the N95 ships with support for multiple image-sharing web sites. The photo application may be configured to transmit pictures directly to these services through HTTP.

For the context of this work, the decision was made to concentrate the measurements on just a select few of the available user agents. The rationale is that this should improve the level of detail possible, as every browser multiplies the number of observations required. Therefore, in addition to the S60 web browser, the only other user agent is the Opera Mobile browser. The selection is based on the browsers available for the N800.

A functional SIM card was inserted in the N95 to accommodate for processing power required to keep the phone subsystem operational. No other background applications were left running. The local carrier network supports 3G/UMTS [UMT07] and its signal strength was displayed as the maximum possible. The screen saver was set to activate after 45 seconds, whereas the backlight timeout was 30 seconds. The light sensitivity detector was left at its default middle position. Key click sounds were disabled. Wireless LAN scanning was left inactive, and the Bluetooth [Blu07] interface was kept disabled.

3.5.1 Nokia Mini Map Browser

The default browser for the S60 operating system is simply titled the “Web Browser for S60” or sometimes the “S60 Web Browser”. In this work, the latter form is used in its abbreviated form (S60WB). The S60WB is based on the WebKit [Web07a] project, which has been open sourced by Apple. To extend the project, modifications made by Nokia are passed back to the community in the form of the S60WebKit branch [Nok07b]. WebKit's WebCore and JavaScriptCore components in turn derive from the Konqueror browser's KHTML and KJS components, also released as open source.

Despite these similarities in their background and naming, it is worth remembering that the browsers are continuously and aggressively developed. S60WebKit forked from WebKit already in 2005. As we shall see, in practice their interpretations do differ quite significantly.

Like all other browsers selected in the benchmark, the S60WB includes zooming functions designed for applications using fixed layouts. Due to the “limited” display size on the N95, the zooming function was very helpful. In addition, the N95 also features the ability to turn the display into landscape mode by rotating it 90º counter-clockwise. However, in these measurements, the S60WB was run in its default portrait mode, with the full screen option inactive.

3.5.2 Opera Mobile

For the S60 operating system, there are two different Opera browsers available [Ope07a]. Opera Mini depends on a server-based proxy and is aimed towards less capable devices [Dav07]. The browser engine is split between the proxy and the browser, allowing the browser to offload processor intensive tasks. Unfortunately, this system does not support Ajax. Opera Mini was therefore not an option.

The other Opera browser is titled Opera Mobile, and the N95 supports its newer S60 3.x version 8.65. It is referred to as the “S60OM” in this thesis. The installed build number was 9730. All of the measurements were run during the 30-day trial offered by the company. From the settings, cache size was set to “large” and render mode to “quality”. Opera Mobile contains a feature that rewrites web pages into a single column mode for easier viewing on a constrained display size. This “fit to screen” function was disabled. Finally, for easier access, Opera was added to the list of applications available from the N95's active standby mode, i.e., the device's main screen.

3.6 iPhone

The Apple iPhone [App07a] is far from the first mobile device developed by the manufacturer, but it is the first with the ability to connect to a carrier network. The operating system is a variant of OSX, developed by Apple and only available on the iPhone. The system is heavily targeted towards web-enabled widgets. Several of the included applications use HTTP to fetch data, e.g., YouTube, Stocks and Weather. The device also contains a two megapixel camera and enhanced media player functionality.

Display size on the iPhone is 3.5 inches, with a resolution of 320x480 pixels (163 pixels per inch). Input is handled by a finger-operated multi-touch screen, with three special buttons for locking the device, accessing the main menu, and disabling the ringer. This unit had a storage size of 4 GB but no selectable options for the cache size. Keyboard clicks were turned off during testing.

As of writing, the iPhone has not yet been released in Europe. The device used in these measurements is from the American batch, with the firmware version 1.0.2 (1C28). Due to the fact that the American models are currently locked to only operate with SIM cards from AT&T, there is no carrier network available for the device in Finland. Therefore, the measurements were performed without the phone function operational. This may have skewed the results somewhat, but the magnitude should be lost within the greater one caused by human reaction time.

3.6.1 Safari

Safari is the only browser currently available for the iPhone. As mentioned earlier, Safari has been developed from the open sourced components in the WebKit project. It is therefore interesting to see how it compares with the S60WB on the N95.

The iPhone's operating system, OSX, also features a landscape mode usable by some applications, Safari being one of them. The view can be activated simply by rotating the device to its side. To maintain compatibility with the results on the N95, and avoid multiplying the amount of observations by having to go through all the combinations, the iPhone was also kept in its portrait mode.

4 Libraries, Toolkits, and Frameworks

As briefly introduced in Chapter 1, the major problem with current web design is *browser fragmentation*. This deviation routinely forces designers to acknowledge the individual perks of the different browsers and to find ways to circumvent these issues. In the practical field, much kudos is given out to designers who can invent the cleverest workarounds for existing problems. Ideally, these helpers will only be used until the relevant cause has been fixed in an upcoming browser release, and that release has become sufficiently widespread among the users.

Before proceeding into the field of Ajax development, another naming problem must be solved. The names of these "Ajax helpers" have split into three categories. The projects presented in this chapter are described alternatively as libraries, toolkits, or frameworks [Pay07]. For this thesis, the following definitions are adapted.

1. A *library* is a collection of one or more functions, aimed for reuse, and relating to a specific task. Examples of this are the browser detection technique incorporated in XHR object creation, as well as DOM calls for finding a specific element by its identifier.
2. A *toolkit* is a collection of one or more libraries related to a specific set of tasks, designed to follow a common programming style. A toolkit may customize its selection of libraries, possibly obfuscating or compressing the code contained.
3. A *framework* is a collection of libraries and toolkits. Using a framework lends structure to the developed project, possibly forcing the project to adapt one or more preselected design patterns, e.g., MVC.

Following this definition and the preceding example of Anttila [Ant07], the projects described in this section are titled toolkits. Google Gears, which is only briefly mentioned, may actually constitute a framework, but as the definition is inclusive, this level of imprecision can be accepted. Note that many of the developers will probably continue to call their projects as frameworks, conceivably due to the more "official" impression of the title. This is acceptable, because splitting the application into separate files can be loosely categorized as a structural decision (see Section 2.2).

In addition to the defined categories, toolkits may be classified according to their style of programming and intended purpose. Different toolkits have been designed to work with specific server-side programming languages and environments, and some have adapted their counterpart's style into the client-side as well [Web06]. Regarding intended purpose, there has been a trend to develop separate projects aimed for “pure” animation and graphics display. Two of the toolkits presented are designed specifically for visual effects, the rest also incorporate methods for the selection and manipulation of events, XHR, and the W3C DOM.

4.1 Graphics display

DHTML techniques have not been forgotten with the onset of Ajax. On the contrary, visualizations developed earlier with the help of browser-centric document object models and other incompatible interface calls are now being reinvented into more standards-compliant versions of themselves [Sco06b].

In this chapter, the toolkits script.aculo.us and moo.fx are briefly presented. They are both visual toolkits and require support from a secondary toolkit to function. The supporting toolkits are usually Prototype and MooTools, although other choices may be adapted. Complementing the supporting toolkit seems to be a popular decision, since script.aculo.us and moo.fx show up regularly in Ajax-related tutorials and introductions. Splitting the functionality into two different projects may also better maintain the development focus on the selected task, i.e., employ visual-minded and more algorithm-oriented developers separately.

4.2 Selection and Manipulation

Without exaggerating, it can be said that events are the heart of any Ajax application. Writing listeners and handlers for the many event types implemented by modern browsers allows the designer to truly adapt the static documents into functional models with many, if not all, of the workings of desktop applications. But event implementations do vary, and it is unrealistic to expect all designers to be intimately knowledgeable of all browsers and their differences.

Therefore, most Ajax toolkits involve easily accessed functions for the selection and manipulation of events and the W3C DOM. These functions mask browser differences and abstract interface calls, perhaps adding a programming style. Toolkit selection may very well be based on the style, allowing the designer to more easily fit in through preceding knowledge of a programming language other than JavaScript.

Of the toolkits presented, script.aculo.us and prototype have been developed along the Ruby on Rails framework [Han07]. Naturally, they borrow idioms from Ruby, even though the toolkits can be used outside of the framework. Google Web Toolkit (GWT) is designed work as an Java to JavaScript gateway – the programmer writes Java and GWT translates the code to JavaScript for the browser. Java is popular with the developers of Direct Web Remoting (DWR) as well. Microsoft's ASP.NET AJAX follows their own ASP.NET framework closely.

Not all of the toolkits may be easily categorized by their likeness to an existing language other than JavaScript. Some projects have chosen their own programming style or have decided to be true to JavaScript itself. Strongly related to the reuse of JavaScript are Widgets, programs that borrow functionality from the installed browser.

4.3 Widgets

When discussing Ajax, “widgets” may refer to at least two separate concepts. First, widgets can be taken to mean semi-independent software modules that can be included in larger web applications. These *web widgets* are commonly developed as parts of the toolkit projects and may require parts of the toolkit to function correctly. Examples are date pickers, rich text editors, and time line visualizers [SIM07, Mar07]. Secondly, widgets can be understood as lightweight browsers, limited into a single application or a set of similar applications. The *widget applications* have already appeared on the desktops of modern operating systems, and they are now beginning to get ported into mobile platforms[5] [Nok07d, App07b]. Examples of widget applications are Really Simple Syndication (RSS) feed readers [RSS07], weather notifiers, and exchange rate monitors.

In mobile devices, widgets can work as a gateway between the functions offered by the device itself and web applications [Gra07a]. Widgets can thus benefit from the browser libraries, e.g., (X)HTML/XML parsers, JavaScript interpreters, and security models. Simultaneously, as the widget exists outside of the strict browser sandbox, it may access converged subsystems unavailable from the sandbox. Such subsystems may include GPS reception, contacts and calendar items, and SMS messages.

4.4 A Look into the Most Popular Toolkits

Two years is a long time in web development. During the two years of fame that Ajax

[5] Apple markets iPhone widgets as “Web Apps”, whereas Nokia has coined the name “WidSets”.

has enjoyed, the number of web sites employing Ajax has grown explosively. Concurrently with the rapid web design, several toolkits have been released as open source in order to benefit others in their work. The wiki site Ajax Patterns [Aja07c] maintains a combined effort to list all of the toolkits. The list has grown to over 200 alternatives.

Browsing through the list, it is easy to notice that many toolkit projects have already been closed or abandoned. This is possibly in favor of the more complete alternatives or just due to lack of time, interest, or both. It would seem possible that other projects will follow, so that the palette will converge to a few toolkits with different strengths and weaknesses. Ajaxian[6], a web site dedicated to Ajax development in general, has performed yearly polls on its readers to find out which toolkits are most commonly used [Alm05, Gal06, Gal07a]. The results show some support for the theory of convergence. On the other hand, 15 years of web development have not been enough to overcome browser fragmentation. There is a chance that the toolkits will not fare any better.

	2005	**2006**	**2007**
Prototype	23,1%	43,1%	68,4%
script.aculo.us	17,7%	32,9%	58,7%
jQuery		7,2%	47,5%
YUI		5,0%	40,3%
Dojo	10,2%	18,7%	38,3%
Ext JS			33,8%
Google Gears			22,0%
GWT		3,4%	17,2%
DWR	11,0%	11,6%	12,7%
MooTools			*11,3%*
moo.fx		11,0%	
ASP.NET AJAX	8,3%	*4,4%*	
total votes	763	865	826

Table 1: Ajaxian.com reader surveys 2005-2007.

The toolkits measured in this chapter have been selected on the basis of their survey representation. Table 1 shows the ten most popular toolkits, along with their historical figures from the previous two surveys. Most of the toolkits reviewed in this thesis were not yet included in the October 2005 survey [Alm05]. 40,0% of the voters proclaimed that they were not using a toolkit at all, but were developing "directly" with XHR. It is conceivable that both home brewed libraries and not yet released projects have been in-

[6] http://www.ajaxian.com/

cluded in this statistic. The early fragmentation can be clearly observed in the results, as many toolkits have been voted less than ten times. Interestingly, Ajax was proclaimed to be in production use as often as in development, with 31,2% and 32,8% of the votes.

By September 2006 [Gal06], the situation had changed somewhat. 25% of the voters were still using XHR directly. The amount of production and development use was still balanced at 61,7% and 67,2%. This would seem to underscore the continuous development cycle of web applications. Prototype and its sibling script.aculo.us had already taken a clear lead over the other toolkits.

The latest results are from October 2007 [Gal07a]. Production and development are still in balance, with 81,2% and 79,3% of the votes. It would seem that the convergence continues towards a small number of popular toolkits. Prototype and script.aculo.us keep their top positions, but the gap between them and the next rankings have narrowed considerably. There are still a great number of toolkits with only a few votes each. Please note that the unintentional omission of the toolkit MooTools [Gal07b] may have thoroughly skewed the results for this choice. The percentage shown is gathered from user-entered free text inputs, i.e., the “other” category.

In addition to the most popular choices, this chapter contains reviews of the Frost and ASP.NET AJAX toolkits. Frost is an upcoming project dedicated on Mobile Ajax, developed by Rocco Georgi of PavingWays. Georgi is one of the authors of the Mobile Ajax FAQ [JGR07]. ASP.NET AJAX is Microsoft's product, which may explain why it is poorly represented by Ajaxian's user base.

4.4.1 How to Read the Results

For those readers that already know the background information and have skipped straight to this section: it is **strongly** recommended to read Section 3.3.1 concerning the capability grading before trying to interpret the following visualizations. Due to space constraints, the graphs include information for all browsers, although their presentations differ significantly. The capability grades are listed in the legends next to both graphs.

The graphs are presented as pairs of cached and cache-cleared measurements. In each measurement, the X-axis shows the 10 sequential measurement iterations. Each Y-axis shows time in seconds. Measured results are marked by the icons presented in the legend, and plotted lines are added to enhance visibility. The legend also lists the grades given to the browsers during each evaluation. As outlined in Section 3.1, these grades are the key element of an unbiased comparison between the measured loading times. If the grades are ignored, the charts can be easily misread. This is a known limitation of the chosen presentation format.

4.4.2 Prototype

The Prototype project [PCT07] was created in February 2005 by Sam Stephenson, and it has been well noted in both literature and web sites. Prototype is possibly best known for its terse writing style, as it uses only single characters for some function calls. Much work has been done to improve upon the project's initially criticized level of documentation. The web site now contains both an API description and tutorials.

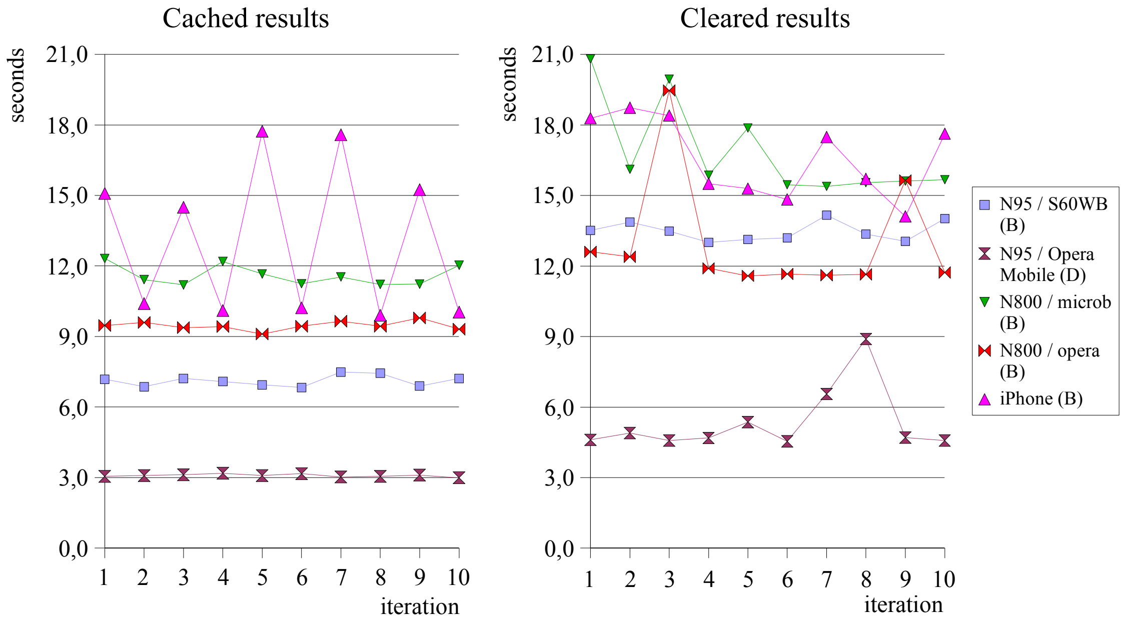


Figure 1: Prototype PWC-OS demo application, http://prototype-window.xilinus.com/.

In contrast to many of the other toolkits, Prototype offers no demo applications on their web site. This disputable lack is balanced by the featured list of production web sites using Prototype. The target for this Prototype test is a “virtual desktop” application, which shows how content may be separated into windows and manipulated in the browser user interface [Gru07]. The PWC-OS sample is prominently a more complex demo, included herein to seek at least some of the limitations of the mobile browsers. As it is the very first measurement, it also serves as a yard stick on what may be attempted in the following cases.

The S60WB shows a respectable level of capability while presenting the virtual desktop. Drag and drop is actually possible with this browser, despite the device's input limitations. The S60OM fails the capability, as none of the windows are actually presented. It also clocks the fastest time in the measurement, which is possibly caused by ignoring portions of the JS code entirely. As we will see, this hypothesis is supported by the other cases, as the worst representation of the application is often rendered in the fastest time. But please note that there are counterexamples of this as well. Sometimes it seems to take longer to render the page wrong.

On the N800, microb features only a minor problems with the PWC-OS. The most prominent one is the overall sluggishness of the interface. Opera has more easily perceived difficulties, as drag and drop is not available at all. This seems to be a limitation of the Opera interface, since drag and drop is not possible in any of the other cases either.

Safari on the iPhone features no drag and drop either. In this test, the desktop dimensions were drawn too large, causing unnecessary scrolling. Examining the performance graphs presented in Figure 1, S60WB and Opera seem to be a bit faster than Safari and microb. microb is the slowest browser, but also the most thorough in its representation. This effect is repeated multiple times in the following cases.

Curiously, Safari exhibits regular variation of the values in the cached measurement, prominently visible in Figure 1. Due to yet unidentified factors, every second fetch of cached content takes significantly longer than the first. As the chosen methodology is black box -oriented, further diagnosis based on these measurements may be impossible. The most credible speculation is that Safari might check for cache freshness every second time, ignoring the expiration headers of HTTP [Fie99] for the content. This suggestion might not be entirely insensible, since cache freshness has been noted as a problematic area (Section 6.4). Of course, it is also logically possible that all *other* studied browsers interprets the same information falsely.

4.4.3 script.aculo.us

script.aculo.us [sic] is an add-on toolkit for Prototype [scr07], designed to extend it with visual effects such as animation and drag-and-drop interfaces. As can be expected, script.aculo.us follows the same programming style as Prototype. The project leader for script.aculo.us is Thomas Fuchs, and the first public version was released in June 2005.

For script.aculo.us, the choice of a measurement target was easy. The project web site offers a demo of a shopping cart interface where the user may drag and drop objects in order to select them for purchase. This “drag and shop” application is also included with the microb browser on the N800, as a part of its default home page.

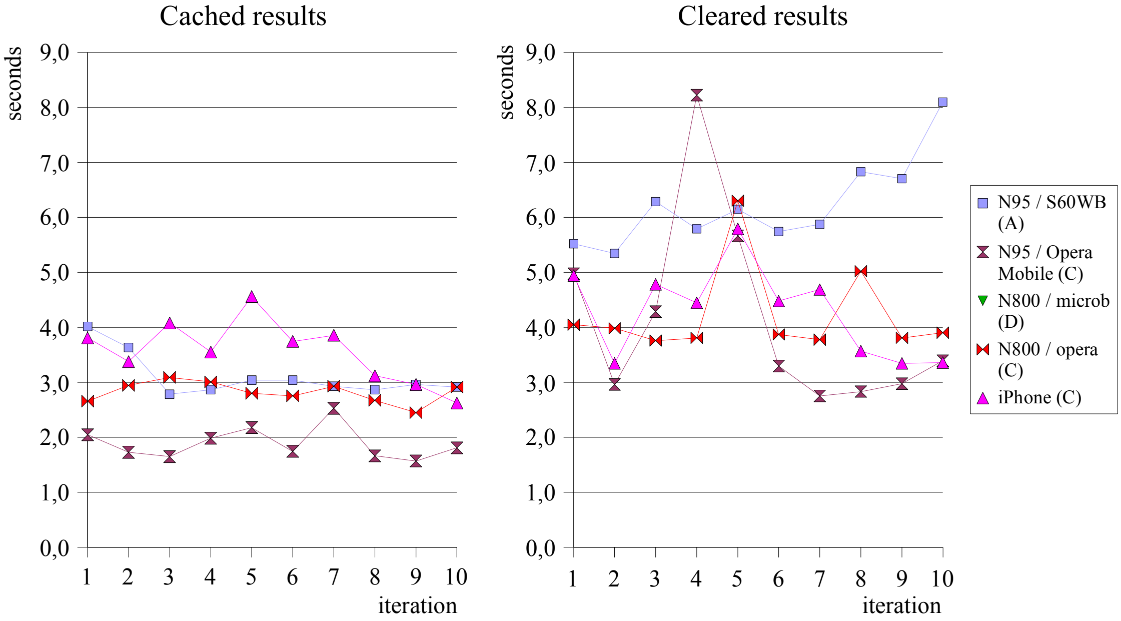


Figure 2: script.aculo.us shopping cart, http://demo.script.aculo.us/shop.

A bit unsurprisingly, as script.aculo.us was already used in the previous case, the S60WB renders the application quite well. Drag and drop is possible, with the same difficulties as earlier. The S60OM has major problems with the layout of the page, causing it to almost fail this test. As noted, drag and drop seems to be a limitation of the Opera interface, so both Operas pass by handicap. The microb does not, as every page fetch causes the browser to crash. Safari has the same limitation as the Operas, so it is also ranked as a near-failure. Drag and drop -centered interfaces are avoided in the following cases.

Figure 2 shows that performance-wise, all of the browsers render the application in a sensible time frame. The iPhone seesaw effect is just barely visible, due to the minimal time scale. The outliers on iteration 5 of the cache-cleared measurements are interesting, as they do not seem to be caused by network issues. Remember that the while the microb and Opera browsers were run interleaved, Safari was not. As the S60OM features a similar outlier on iteration 4 and microb another on iteration 8, our hypothesis about their cause is server congestion.

4.4.4 jQuery

John Resig has been in charge of the jQuery project [jQu07] since its beginning in January 2005. The programming style is as terse as Prototype's. Notably jQuery uses the same character, $, as a function name. This makes incorporating both toolkits challenging, although it is difficult to conceive reasons for wanting to do so. One of jQuery's strengths is the *chaining* of function calls, which allows designers to manipulate the same element multiple times in just one statement.

The demos on jQuery's web site are links to external servers and developed by project volunteers. Here, jQuery is represented by Jack Born's tutorial site, *15 days of jQuery* [Bor06]. The site contains multiple demos, and for this task we have selected an “edit-in-place” (EIP) interface. EIP allows (text) paragraphs to be modified just by clicking the editable area. This is a very small modification, but with significant usability repercussions, as it spares the user from searching for the same text in a separate edit field.

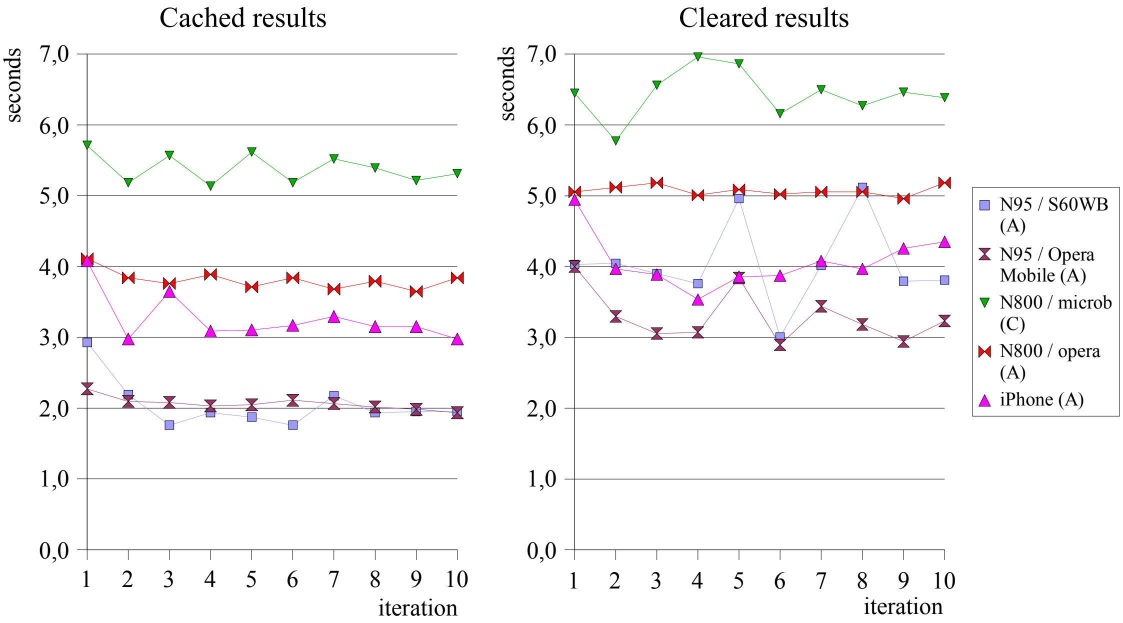


Figure 3: 15 days of jQuery, EIP #2. http://15daysofjquery.com/examples/jqueryEditInPlace/demo.php.

The results shown in Figure 3 are encouraging for Ajax, since four of the five browsers pass the capability test with no defects detected. This time the exception is the microb, as it has difficulties displaying the unedited paragraphs after a successfully performed edit. This issue has been classified as a major one in the scope of this test, although it might be quite easy to fix. This classification thus serves as an example of the grading criteria: the level of debugging necessary is not speculated upon evaluation.

In the benchmarks, microb is the slowest browser by a clear margin, followed by N800's Opera. Safari on the iPhone does not show the seesaw effect, possibly again due to the time scale. The highest variation is shown by the S60WB in the cache-cleared measurement, speculatively due to network conditions. Although the time taken might seem like plenty for only a single application component, it is necessary to keep in mind that object instantiation and other setup tasks might easily dominate the overall measurement.

4.4.5 Yahoo! User Interface Library

The Yahoo! User Interface Library (YUI) incorporates both DHTML techniques and Ajax quite seamlessly [Yah07b]. It is definitely on the larger side of the toolkits, as YUI is constantly developed by Yahoo!'s dedicated staff. YUI has been used by Yahoo!'s applications since 2005, and it was released as open source in February 2006. Despite its open license, no formal method for submitting patches exists.

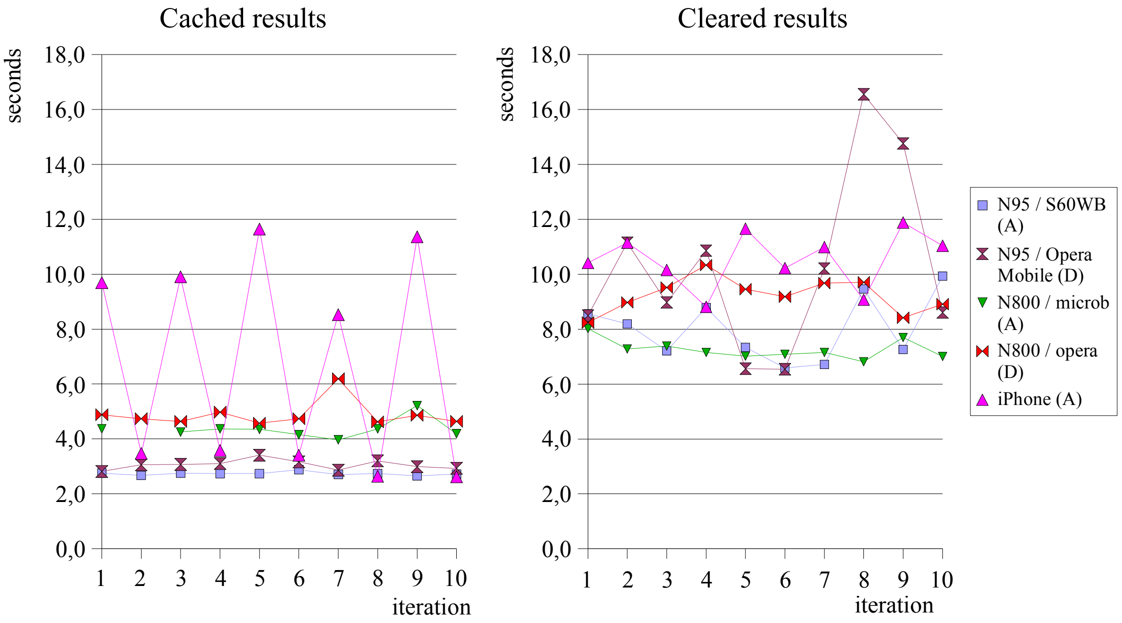


Figure 4: YUI dynamically loading tree view, http://developer.yahoo.com/yui/examples/treeview/dynamic_tree_clean.html.

YUI is represented in this measurement by a dynamically loading tree view picked from the official YUI library examples. The tree view is quite similar to earlier DHTML constructs, as its purpose is simply to display a set of data in a hierarchical data structure. The key difference with an Ajax-enabled tree is that the data does not have to be preloaded to the browser with the initial view. In this case, the tree contains a large amount of data, making it too large to completely fit in the memory of a constrained device. Additionally, the freshness of the data is guaranteed by dynamically loading the information from Yahoo!'s search database.

The cached results displayed in Figure 4 again prominently display the iPhone's signature seesaw graph. The application seems to be quite well supported on three of the browsers. The S60WB scores a perfect grading while simultaneously being the fastest browser in the cached benchmark. Both the Operas fail to display the tree, neglecting both tree lines and the display of any results when links are clicked.

In this test, the microb browser exhibited its first indeterministic crash. It happened during iteration two of the cached benchmark. The second iteration is performed consecutively after the first, but also after all of the capability testing preceding both. Therefore, it is conceivable that the missing value signals a memory leak, wherein some object references have carried over from the earlier testing.

Without cache, the performance results are quite similar in overall. Interestingly, the S60OM shows the largest variation despite its obvious flaws in the presentation. Figure 4 shows that sometimes it may take longer to display a faulty presentation.

4.4.6 Dojo

Like YUI, the Dojo Toolkit [Doj07a] is on the heavier side of the toolkits. Its origins run back to a DHTML project called NetWindows by Alex Russell. Russell serves as the president for the Dojo Foundation, while NetWindows has been incorporated into the Dojo toolkit along with a number of other projects. Due to its size, Dojo features dynamic loading of specified toolkit parts only.

The demo email application presented by the Dojo project was one of the first candidates selected for these measurements. Unfortunately, said application was also removed just days before the testing phase begun. The reason for the application's removal was noted as conflicts with nightly builds of the development tree. Therefore, these results have been run from our computer science department's main web server, against the application provided by version 0.9.0 of Dojo's installation package. The package contains a host of synthetic benchmarks and capability tests, which will unfortunately not be further described herein due to reasons of brevity.

Figure 5 shows that most of the browsers had problems with this application. Both the N95 browsers failed to display the message header list, or indeed much of the layout at all. In addition, all of the buttons remained unresponsive. The relatively reliable microb

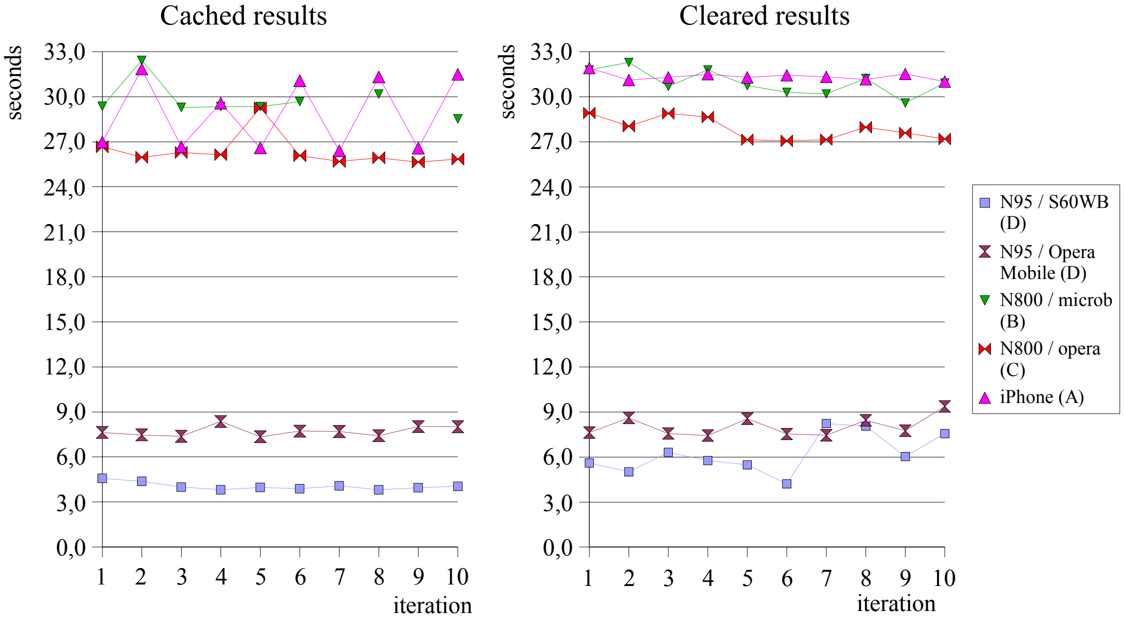


Figure 5: Dojo email application, ../dojo-release-0.9.0/dijit/demos/mail.html

could not perform automatic completion on form fields and it felt quite sluggish in use. microb also crashed twice during the cached measurements, during iterations 7 and 9. This is quite probably indicative of leaking memory, as no crashes were detected during the cache-cleared measurement. microb also had problems just loading the application, as four iterations had to be redone by refreshing or clearing the cache and then retrying twice, in order to repopulate the cache.

The N800 Opera displayed more severe problems with the message header lists, bordering on an overall failure. In contrast to it, the Apple iPhone rose to the challenge and presented the application correctly after corrupting the layout once. Despite rigorous retrying, the problem could not be repeated. It might have been caused either by the Safari browser or by corrupted output from a server-side program.

Dojo's demo email application is quite heavy on the rendering engine, visible by the minuscule difference between the cached and cleared measurements. The effect of caching was less than three seconds in all cases, and less than two seconds with both of the N800 browsers. In practice, it might be questionable if loading times of over 25 seconds can be tolerated by any user.

4.4.7 Ext JS

The Ext JavaScript library (Ext JS) begun as an extension to YUI written by Jack Slocum. The toolkit was initially named *yui-ext*, but as its popularity grew, the project split from YUI and continued as a separate toolkit. At the end of 2006, the name was shortened to Ext, and version 1.0 was officially released April 1st, 2007. Ext was dependent on a set of other toolkits until version 1.1. Due to this history, Ext contains adapters that facilitate the use of extensions provided by other toolkits. Some compatibility issues remain, but this feature may become a strength of the Ext toolkit in the long run.

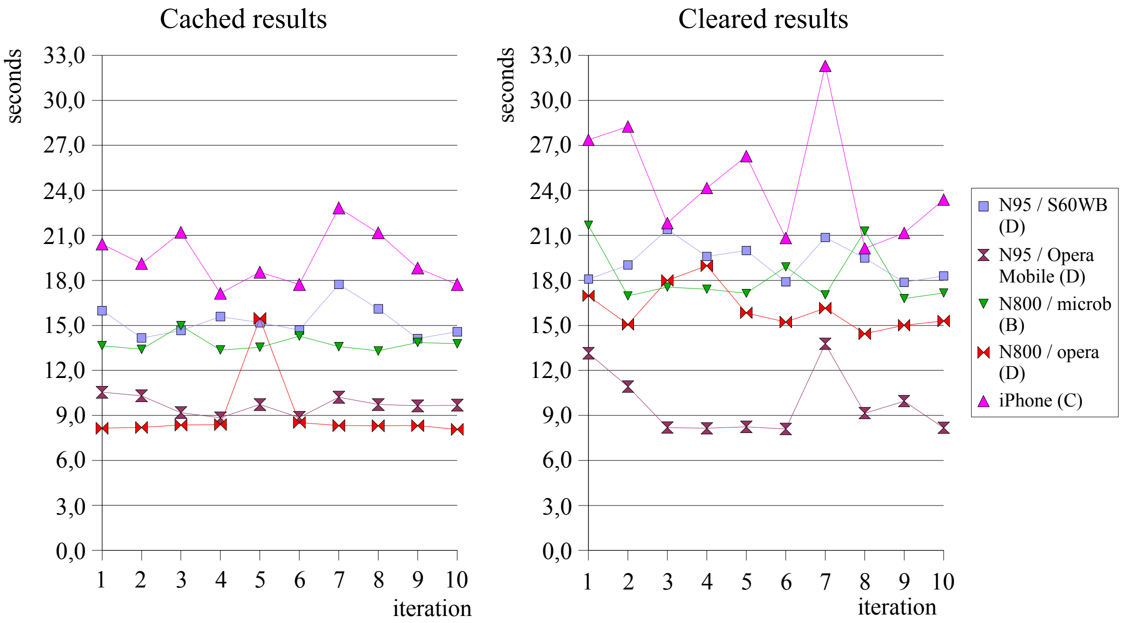


Figure 6: Ext JS dynamic form using XML, http://extjs.com/deploy/dev/examples/form/xml-form.html.

The Ext JS project hosts a number of examples, of which the dynamic XML form is chosen as a representative. This is the first application that specifically uses XML as the communication format. The purpose of the demo is to simply gather a few fields of input from the user, and then return an XML-formatted error message upon submit. To make things a bit more interesting, the form incorporates a small “date picker”, i.e., a calendar web widget. It is worth mentioning that for added pedagogic value, in this case the JavaScript code was left uncompressed, which will affect the results. The XML form is a practically reusable, but fairly basic application. As such, it might be expected that the browsers could easily support it.

Figure 6 shows that the reality is more grim. None of the browsers are able to present the application without defects. At the time of testing, the application was verified to work with the desktop browsers. It has since stopped working on the Firefox browser, so changes in development code may have caused some of the tested defects as well.

The culprit is the load button, which must be clicked before submitting the form to populate the data. Said button remained dysfunctional on both of the N95 browsers. Since the service requested by the user may not be executed, both browsers are marked as having failed the test. The S60WB performed better visually, supporting all of the other form elements, whereas the S60OM had trouble with both the pull-down menu and the date picker.

On the N800, microb has minor issues with the form. The date picker remains unavailable and the pull-down menu is perceivably slow. Opera fails this test, for the load button remains inaccessible, whereas the date picker works. Finally, Safari is quite close to hitting the mark, but the whole form is displayed with an inadequate height. Load and submit seem to work, but only the first input field is visible.

It takes extremely long to display this quite simple form. For microb, the browser with the best overall presentation, it takes in average over 13 seconds with cache and over 18 without. As similar forms might be expected to be found embedded in more complete applications, these loading times are definitely too long. However, it must be reiterated that a production-quality version would probably greatly benefit from a more compressed JavaScript format, as well as reusing the same JavaScript in general.

4.4.8 Google Gears and Web Toolkit

Google Gears [Goo07c] has recently received a great amount of interest from the developer community [Doj07b]. It is essentially a browser extension that widens the distributed nature of web (and Ajax) applications with additional offline functionality. At the time of testing, no version was yet available for the selected mobile platforms, so Gears was excluded from further study herein.

Google Web Toolkit (GWT) [Goo07d] works by compiling server-side Java into JavaScript interpretable by the client browser. GWT takes special focus in testing procedures and debugging tools available to Java but lacking from JavaScript. The compilation allows designers to borrow on Google's experience with browser inconsistencies, ideally circumventing them totally.

As a representative for GWT, the dynamic table application was selected from the official samples. Its purpose is to display planned schedules for university students and staff. The view changes asynchronously with user-selectable date criteria. The dynamic schedule is an interesting application from a mobile perspective, for its intended usage is easy to imagine. It also employs a flow-based layout with a very clear interface.

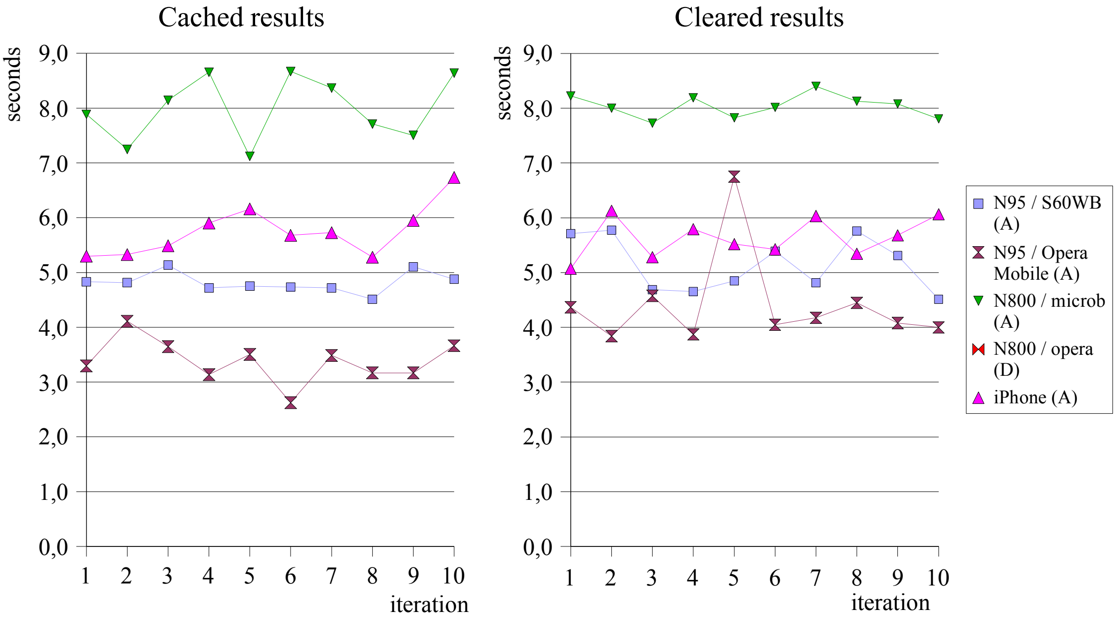


Figure 7: GWT dynamic table, http://gwt.google.com/samples/DynaTable/DynaTable.html.

GWT:s dynamic table is well supported by the browsers. Four of the browsers support it near-perfectly, although both N95 browsers require two back button presses to return from the application. This might caused by using IFrames as the communication channel. The only browser that fails the capability test is the default Opera on the N800. It fails to display either table or controls, so the results have been omitted as redundant.

Looking at the measured performance in Figure 7, microb distinguishes itself by taking over two seconds longer to display the application in both measurements. It also exhibits the largest variance (0,35) in the cached measurement, closely followed by the iPhone (0,21). S60OM is the fastest browser, with a visibly snappier presentation than the others.

4.4.9 Direct Web Remoting

Direct Web Remoting (DWR) is a toolkit designed as the client-side counterpart for server-side Java applications [Get07]. The DWR toolkit also focuses on *Reverse Ajax*, meaning the category of push techniques that includes Comet [Rus06]. Like GWT, DWR also transforms Java to JavaScript. The project is supported by TIBCO and lead by Joe Walker.

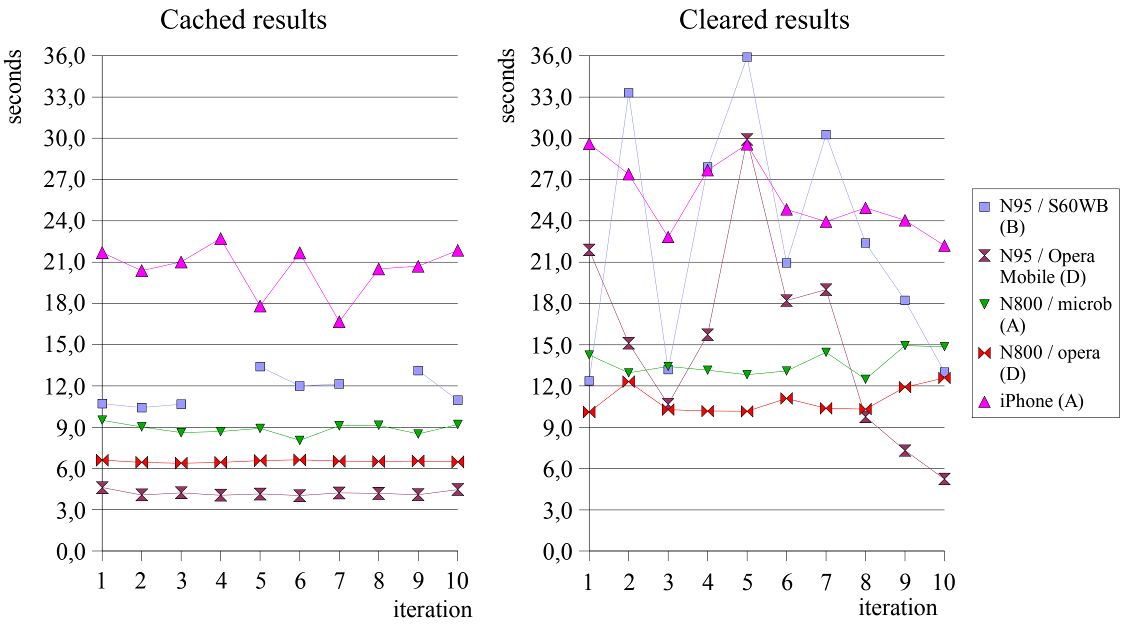


Figure 8: DWR / Jetty Hightide auction demonstration, http://www.webtide.com/auctiondemo/.

DWR is only partially represented in this test. The examples on the project web site were considered a bit too simple even as components in a larger Ajax application. Therefore, a hybrid application was selected for this test: the Acme Auctions demonstration site [Web07b] designed by Webtide [Web07c], the main developers of the open-sourced Java container Jetty. In addition to DWR and Jetty, the auction application contains features from Apache's ActiveMQ Ajax [Apa07]. The purpose of this amalgam is to demonstrate the use of a Comet-style push technique. Even though it is not immediately clear from Figure 8, Comet works spectacularly well in this application.

Both of the Operas have failed the capability test for the simple reason that the login button is inaccessible from their user interface. The cached measurements show that the Operas have also clocked the two fastest loading times for the application. It is thus credible that the errors are due to JS code getting ignored by the browsers.

More interestingly, Comet works in each of the tree other mobile browsers. The minor problem encountered with the S60WB is that after the server has pushed updates to the client, the S60WB has problems of displaying the results properly. Also, the S60WB crashed pseudo-regularly in the benchmarks. The S60WB crashed regularly every fourth time the application was loaded. After each crash, the application was loaded twice to ensure caching. The crashes look like a clear indication of a memory leak, since no crashes were detected during the cache-cleared measurement.

The screen saver (not the backlight) on the S60WB will disconnect the open TCP connection, so testing this application requires the user to activate the screen regularly. On the iPhone and the N800 microb, no such user operations are necessary. In the three functioning browsers, the passive HTTP connection was kept alive for 10 minutes. Additional testing (not included) on the N800 showed that connections remained open after 30 minutes. Both platforms might well be able to keep the TCP connection alive indefinitely. Further, it should be emphasized that these tests were run through a network address translation (NAT) device. The capability evaluation has also been repeated with a second N800 device and a separate, NAT-operated network.

Examining the performance results in Figure 8, it can be seen that Safari is slower than any other browser in the cached measurement. Safari is also slowest in the cleared measurements in average, although the S60WB displays a greater variance of results. There, variation is also displayed by the S60OM. Remembering that cache-cleared measurements on a platform were interleaved, it is possible that the variances are due to network conditions or server congestion. First-hand observation of the loading process, as noted in the notes in the appendix, supports this possibility. The client-server communication seemed to vary between iterations, whereas the rendering phase was similar.

4.4.10 MooTools

The MooTools toolkit has a threefold focus: being compact, modular, and object-oriented [Pro07]. It is an independent open source project led by Valerio Proietti. As mentioned, the architecture division of MooTools and moo.fx resemble Prototype and script.aculo.us somewhat. The difference is that MooTools v1.00 was released as late as 28.1.2007, making this project much younger and possibly more aggressively motivated in their development. Hypertext Preprocessor (PHP) [PHP07] programmers might consider the programming style familiar.

The HistoryManager [Kir06] developed by Harald Kirschner is designed as an invisible web widget. It is meant to be used as a modular fix for the back button problem, wherein partial updates do not necessary cause changes to the browser's page history. Using the HistoryManager, the user should be able to undo each interface click registered to the widget. This is a quite simple application, and it could thus be expected to work universally with the browsers. Yet browser support remains lacking.

The S60WB manages to support the lists and the accordion pane as intended, but click-

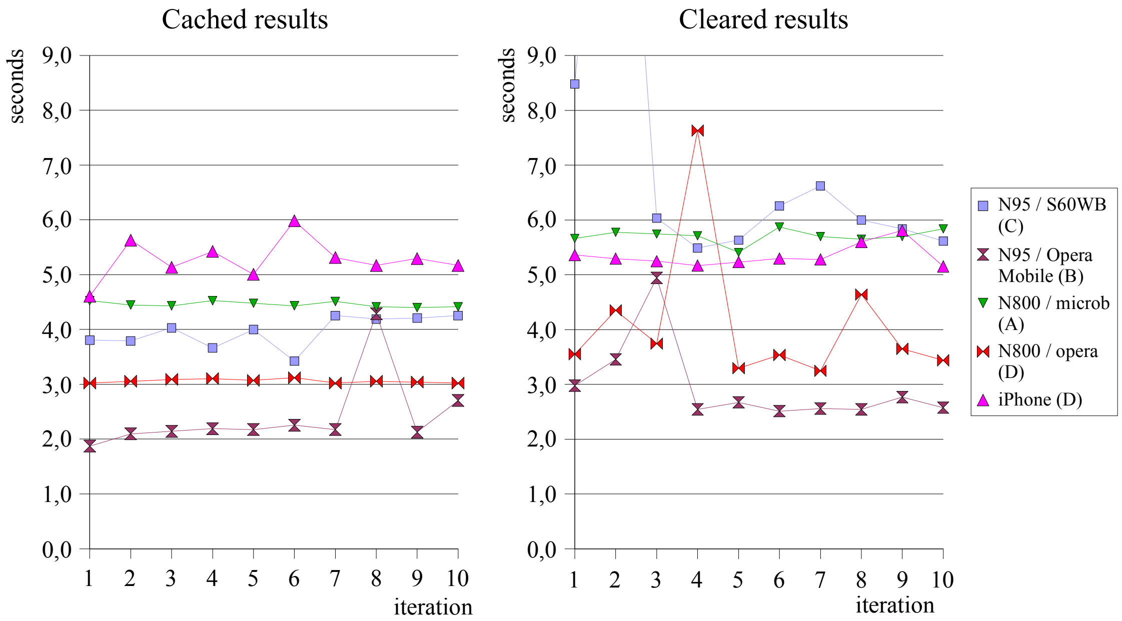


Figure 9: MooTools HistoryManager, http://digitarald.de/playground/history.html.

ing on any of the numbered content links make the browser to consistently crash. Invoking the back button causes a complete screen redraw. Opera on the N95 manages a bit better, supporting both the lists and content links, whereas the animation effect of the accordion is not displayed. The S60OM also performs complete redraws each time back is invoked.

Figure 9 shows a single working presentation of the application, the one displayed by the microb browser on the N800. Opera on the same device performs complete redraws upon each invocation of the back button, but this time the UI remains unchanged. Although most of the UI remains visible, this classifies as a failure in the capability test. Finally, Safari supports all links, but each click yields a complete redraw. The back button does the same.

Looking at the cached measurements of the browsers, the results are curious. The Safari browser has the worst support for the application, but manages to clock the slowest load time as well. Conversely, the S60OM suffers from only minor problems, but its load time is the fastest. Second slowest is the microb, offering the best overall support for the application. Turning over to the cache-cleared measurements, S60OM is still the fastest, but here the iPhone outpaces both the microb or the S60WB. A single outlier for the S60WB is drawn outside the scale of the graph. Its value has been kept, although no explanation for it can be offered. This emphasizes that only outliers caused purely by user input errors have been purified from the results.

4.4.11 moo.fx

moo.fx [mad07] is possibly the smallest of all the toolkits presented in this chapter, constituting only 3 kilobytes in total. In addition to MooTools, moo.fx may complement Prototype as well. It is notably easier to install the toolkit through MooTools, as moo.fx may be included in the download.

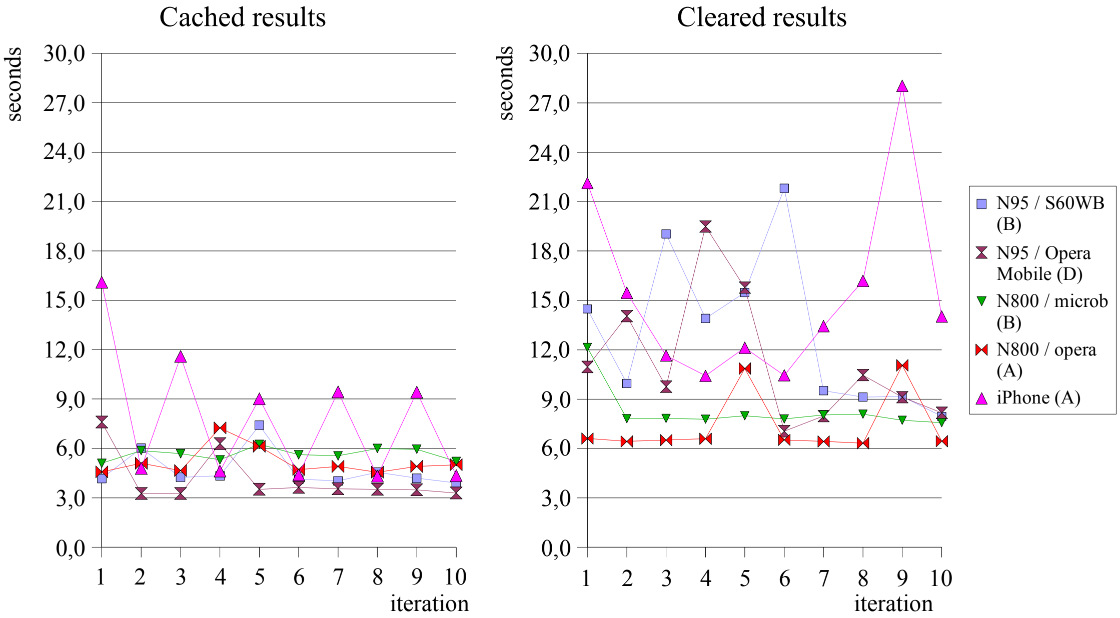


Figure 10: MooTools / moo.fx Asset.images, http://demos.mootools.net/Asset.images.

Asset.images demonstrates a project built using moo.fx's visual effects. The demo consists of a very basic image gallery that asynchronously loads five pictures and separates them with a fading transition effect. It is available through MooTools demos [Pro07]. In addition to the self-running demo, the application's JavaScript, HTML, and CSS program code tabs were also tested.

Two of the browsers support the application without incidents, two with minor remarks and one fails completely. With the S60WB, the pictures were displayed outside the designated container with the black background. S60OM could not display the gallery reliably: sometimes the asynchronous transfers failed and the cache had to be completely cleared to fix this. On the microb, a curious and undeterministic glitch was encountered. Sometimes the last picture failed to load, while all the others were presented correctly. This problem is designated as a minor one, since it was encountered with the desktop Firefox as well. Opera on the N800 and Safari on the iPhone support the application without issues.

Three of the five browsers in Figure 10 display great variance of their results in the cache-cleared measurement. The N800 browsers perform more constantly than the others. It is possible that the variance is caused by network conditions. But this analysis may also be duly criticized, as it would require said conditions to disappear during the interleaved N800 measurements, only to reappear when the iPhone was benchmarked. Please note that the measurement targets the initial loading time, and not the overall time to complete the picture slide show.

4.4.12 ASP.NET AJAX

ASP.NET AJAX [Mic07c] is Microsoft's product, formerly code named Atlas[7] [Smi06]. As the newer title implies, ASP.NET AJAX operates close to Microsoft's other products, e.g., .NET and Visual Studio 2008. The toolkit consists of multiple components that can be chosen depending on the desired level of interoperability with ASP.NET. When complemented with all of the server-side technologies, the toolkit distinctly manifests all the characteristics of a framework.

The ASP.NET showcase contains an ample amount of real-world examples, but as such they are more suitable for Chapter 5. It is harder to find the ASP.NET AJAX Control Toolkit samples [Mic07d], which are also fairly basic when compared to the previous demos. The selected application is a rich text editor developed by Eric Williams. Willi-

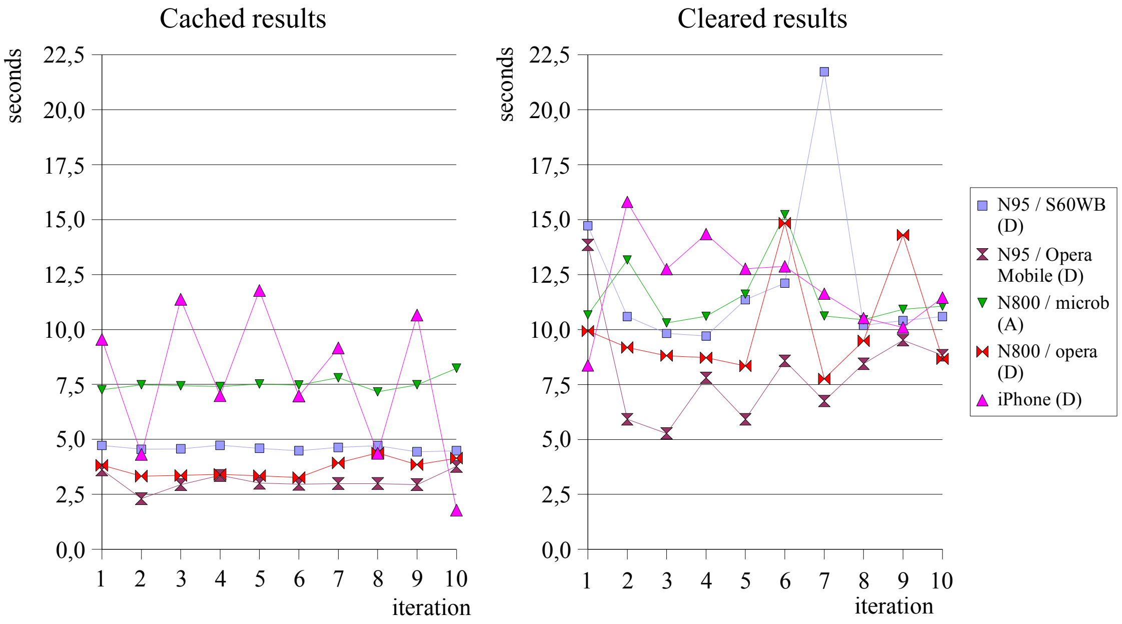


Figure 11: ASP.NET AJAX HTML editor, http://winthusiasm.com/Pages/HtmlEditor.aspx.

[7] It is still common to see Atlas in use, conceivably because the official naming is considered difficult.

ams has also designed the Colorado Geographic application featured in the showcase, and analyzed in Section 5.6. The editor allows an user to employ an interface resembling a word processor to format HTML code. Rich text editors like this one are currently popular as Ajax widgets, if only for their pedagogic value.

Unfortunately, this specific application is poorly supported by the browsers. All but one fail the presentation completely. Conversely, the single browser capable of presenting the application does so without glitches. These failures are curious, since the other application by the same author presents fairly well. The best presentation is offered by the microb browser on the N800. All the others fail to display any content in the design or preview tabs. Additionally, clicking on preview causes a full page redraw on both Operas and the Safari browser.

Before looking at the performance, it must be mentioned that the cached measurements of both of the N95 browsers were performed in a separate wireless network due to time constraints on the selected devices. The results were later verified using the regular network connection. There is little new to be gained from the charts in Figure 11: microb offers the best presentation, but also the second slowest time. Safari is marginally slower in average, but only because of its distinctive seesaw variation.

The cache-cleared measurements are performed in the same network as all other cases. There is significant variation in all results, possibly due to server congestion or network conditions. The application seems to contain a lot of code, visible by the difference in scale between the two measurement methods.

4.4.13 Frost Ajax Library

Frost [Geo07a] is an upcoming toolkit based on the work by Rocco Georgi of PavingWays. This toolkit will be specifically targeted towards mobile devices. The project web site is yet sparse on the details, but it contains a very usable capability test. As of writing, no Frost demos or release candidates have been released, so the following measurement may be seen as misleadingly named.

However, the conjecture is that the upcoming toolkit will build on the experience of its developers, which means their earlier projects. It is easy to consider the XML 2006 event schedule [Pav06] as an interesting target, for it can demonstrate the effects of good testing on a mobile Ajax application. The schedule is also arguably the first “real” application, as it has been used in a real environment. Therefore, this event schedule

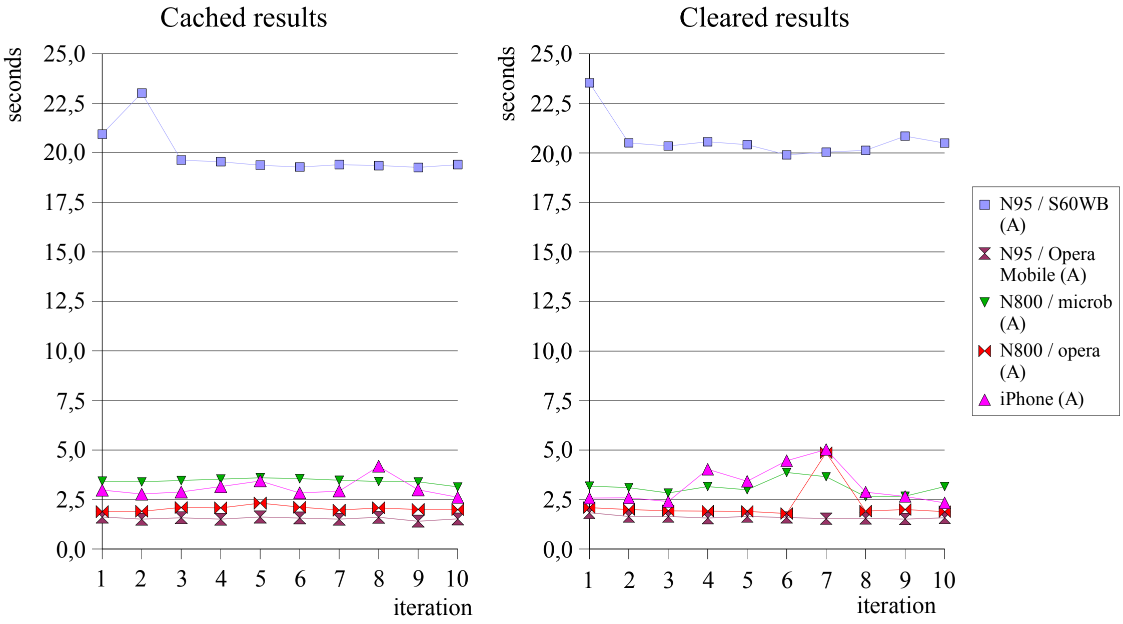


Figure 12: Pre-Frost XML 2006 event schedule, http://pwmwa.com/xml06/.

serves as a gateway between this chapter and the next, which concerns production-quality web applications.

In overall, it is easy to feel that this application is useful in practice. The event schedule allows users to list currently active program items, combined with color-coded information on upcoming and already passed items. Users may enter comments on the program items, and the written text is saved using XHR calls. By looking at the overall capability grades in Figure 12, it is clear that mobile testing has been done well. Every single browser is able to support the application with no defects detected.

While uniformly excellent support is encouraging from a designer's perspective, it leaves little to be analyzed. Fortunately, the measured values provide additional insight. As may be deduced from its name, the XML 2006 event schedule employs XML as the browser-server message format. All of the browsers are able to congest the data without incidents, save one: the S60WB. Both the cached and cache-cleared measurements show without doubt that the browser has serious performance problems parsing XML. Further, the virtually nonexistent difference between the two measurement methods shows that S60WB's poor performance has nothing to do with caching. The exhibited slowness is fortunately not totally disruptive for the user. While the page loads, the user may continue viewing the already parsed event information. In fact, the loading of new items is visible by continuously scrolling the page downwards.

5 Surfing Web Sites with Ajax

In theory, the applications selected in the following sections have gone through much more testing than the samples and demos presented in the previous chapter. This theory will subsequently be put to testing, as checking for compatibility on *all* browsers has already been noted as nontrivial. If desktop browser support has already been verified, the question remains if the mobile browser version is able to present the same level of capability and performance. The initial hypothesis is that the selected mobile browsers will show comparable levels of capability, but with significantly reduced performance. Evaluating the capability and measuring the performance of mobile devices should be indicative of potential issues concerning the support of Ajax applications.

There are already far more Ajax applications than could be ever measured in a work of any single author. New applications seem to be developed constantly, whereas old ones are re-released as new versions. Therefore, the following sections present a snapshot of the situation as it was, hopefully informative to application designers and device manufacturers.

As the following cases are analyzed, we will see that graceful degradation becomes a problem when determining mobile browser support. Many times it seems that the designers have erred on the side of caution, prohibiting mobile users from even trying to access the full version of the application.

5.1 How to Read the Results

For those readers that already know the background information and have skipped straight to this section: it is **strongly** recommended to read Section 3.3.1 concerning the capability grading before trying to interpret the following visualizations. Due to space constraints, the graphs include information for all browsers, although their presentations differ significantly. The capability grades are listed in the legends next to both graphs.

The graphs are presented as pairs of cached and cache-cleared measurements. In each measurement, the X-axis shows the 10 sequential measurement iterations. Each Y-axis shows time in seconds. Measured results are marked by the icons presented in the legend, and plotted lines are added to enhance visibility. The legend also lists the grades given to the browsers during each evaluation. As outlined in Section 3.1, these grades are the key element of an unbiased comparison between the measured loading times. If the grades are ignored, the charts can be easily misread. This is a known limitation of the chosen presentation format.

5.2 Google

Google's products are in many ways the pathfinders of current web technology. It is uncommon to find a source describing Ajax that does not mention Google Maps [Goo07b] as well as Garrett's original text [Gar05]. Maps is a service that undeniably has benefits for mobile users, if the browsers are able to present it adequately. There are similar benefits with other Google applications: the N800 features no calendar application, making Google Calendar a very attractive choice. However, Google Calendar was excluded from the test due to its similar behavior to Google Mail, tested below.

5.2.1 Google Maps

Google Maps [Goo07b] uses graceful degradation actively, perhaps even too eagerly. Previous evaluations show that both of the tested S60 browsers are able to present Ajax applications, provided that they have gone through adequate testing for differing interpretations. Maps' take is more conservative, offering a very reduced version of the application for the S60WB, while allowing the others to proceed to the full version.

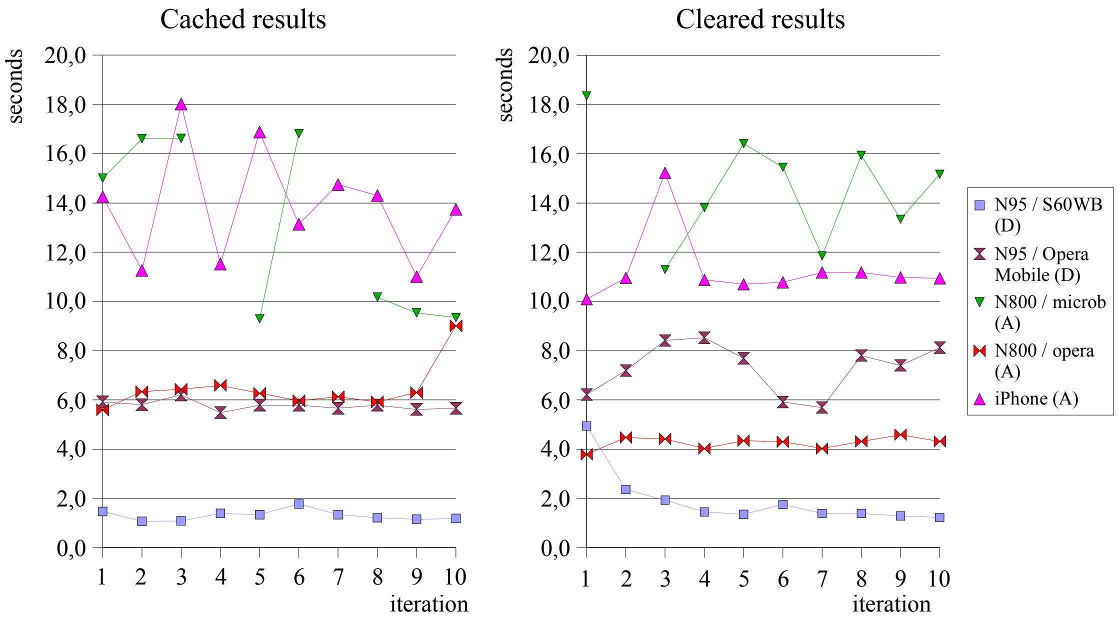


Figure 13: Google Maps, http://maps.google.com/.

Inconsistently, the degraded version also returns very dissimilar service. The limited version includes no Ajax functionality, but more importantly it also returns different result sets. It would seem that the mobile version yields only those services that have participated in Google's advertisement campaigns. *Summa summarum*, the capability test can only be marked as a failure for the S60WB, as none of the local street addresses entered yielded results closer than several tens of kilometers. Opera on the N95 is able to access the full version of the Google Maps, but unable to present it correctly. Complete results are fetched and displayed, but the map never centers on them, and neither can it be manually scrolled due to Opera's input limitations.

Turning over to the N800, the microb and Opera browsers offer a vastly improved presentation. Opera's only problem was sometimes showing the dialogue bubbles as blank. The microb can be criticized for a somewhat slow responsiveness, giving cause to re-evaluate more specific functionality measurements in this case. Because of time constraints, such measurements have been left as a subject for a further study. In addition to the slowness, microb detected the address input field only occasionally, correctly bringing up the virtual keyboard. During the ten iterations, the keyboard was presented thrice. Again, the major problem with microb was randomly crashing. Figure 13 shows missing values during iterations four and seven of the cached measurement, and during iteration two of the cache-cleared measurement. Crashing despite a cleared cache is highly unusual and therefore worth noting. Additional testing was performed to check for repeatability; the results are available in the appendix.

Last but not least, Google Maps on the iPhone is able to interact with the locally installed widget. Search results are passed on to the widget and visualized there instead of the Safari browser window. This feature is currently quite unique.

Moving on to the performance analysis, an extremely strange phenomena may be observed by comparing the two measurement methods. Opera and microb on the N800, as well as Safari on the iPhone, sometimes perform *worse* when operating with already cached content. The visual observation carries over to the calculated averages for both the Opera and Safari, whereas microb's variation is too great for a straight-forward comparison. Ten more iterations were performed for the microb and Safari. Although the values observed were now closer to the cache-cleared measurements, both the average and the variance remained greater. See the appendix for details.

5.2.2 Google Suggest

Google Suggest [Goo07e] offers, in theory, extreme benefits for mobile browsers. The applications' ability to predict search terms by suggesting strings matching the already entered portion can greatly enhance the overall input speed of the user. Because the sheer physical size of the mobile devices forces the user to use less optimal input methods, Suggest is a very interesting choice for an often-used functionality, searching for web sites. The application's program logic has been dissected by Justus [Jus04].

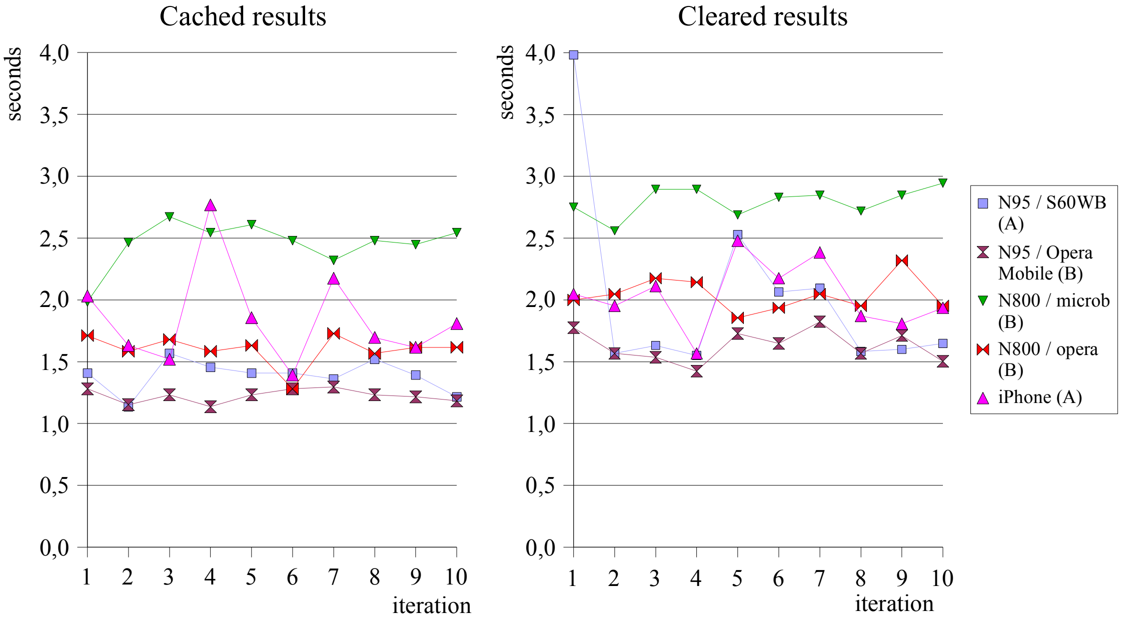


Figure 14: Google Suggest, http://www.google.com/webhp?complete=1&hl=en.

Figure 14 shows that Suggest is a very fast application to load. Despite its simplicity, the browsers had some trouble presenting all of Suggest's functionality. With the S60WB, browser detection techniques caused a graceful degradation into normal Google Search. By the definition of the capability grades, this must be labeled as passing without defects, since the user receives the requested functionality. Opera Mobile on the N95 shows that the degradation may be warranted. With this browser, the application seems to be fully functional. Yet the major problem with the S60OM is that only the first of the suggested results may be selected.

Both of the N800 browsers present Google Suggest in a similar vein. At first, the distinctive functionality of this application seems to be completely missing. After some testing, the lack of any and all suggestions was discovered to depend on the events generated by the browsers. Once the user presses the backspace key or equivalent, a suitable event is generated and the suggested results are presented. This somewhat hampers the usability of the application, but the suitably informed user may still benefit from the general input speedup. Finally, Safari degrades into the normal version of the search, yielding the same overall grade as the N95 browsers despite no improvements in input speed.

Moving over to the performance analysis, Safari displays an uncharacteristically high variance in the cached measurement, even despite its distinctive seesaw effect. A second run of ten iterations was performed, leading to a somewhat reduced variation. Nevertheless, even the second calculation returned a standard deviance more than double that of the second highest browser.

This time, microb's significantly slower performance in both measurements does not yield a better overall presentation. Opera on the same device is almost twice as fast, but with the exact same problems in presentation, and a slower variance to boot. In the cache-cleared measurement, the S60WB show the highest variation. The variation is not only due to the single outlier.

5.2.3 Google Mail

Google Mail [Goo07f] is probably the second best known Ajax application, straight after Google Maps. The target for this measurement was the previous version of the mail application, not the recently-released update [Pup07a]. Due the new version's gradual release, it had not yet reached the author's user account.

In contrast with some other Google applications, Google Mail has a built-in function that allows bypassing the browser detection techniques usually involved with graceful degradation. Initial capability testing was performed using this optional feature, but it was soon discovered that the browser check defends its position well. Both the S60WB and S60OM returned horribly broken presentations of the application, whereas Safari refused to even try displaying the full version. In all three cases, the degraded, mobile version works well. Its responsiveness is beneath what can be reliably measured using the selected method, and thus no benchmark results have been recorded.

In contrast, both of the N800 browsers are capable of displaying the full application, and they even manage quite well. However, Figure 15 shows that their performance leaves something to be desired. Whereas an average loading time of 13-14 seconds for Opera users might be acceptable, microb's 24-26 seconds will quite probably give cause not to use the application twice. The overall impression while using the application was that microb was definitely too slow. All of the functions were eventually performed, but only after noticeable response times. Despite this critique, it should be emphasized that

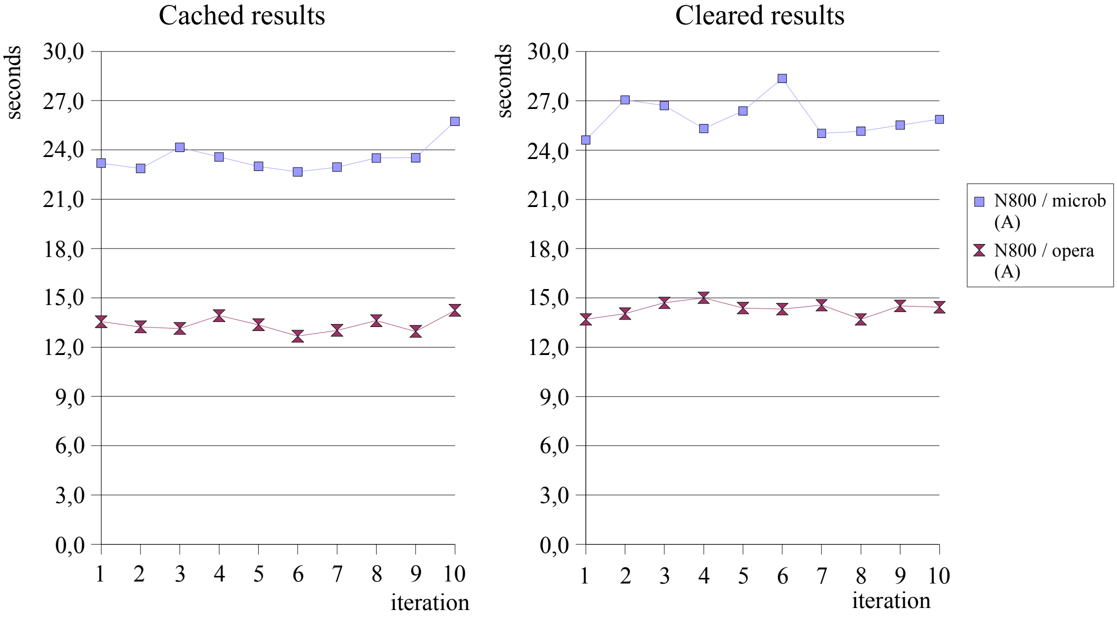


Figure 15: Google Mail, http://mail.google.com/mail?nocheckbrowser/.

Google Mail is a very complex application, and being able to present it at all is a herculean task for a mobile browser.

5.3 Yahoo! Mail

Yahoo! has been aggressively developing DHTML and Ajax techniques, as evident by the increasing popularity of their YUI toolkit (Section 4.4). The company develops multiple products with YUI. Since the development of YUI, Yahoo! has also acquired the Flickr service [Yah07c]. As Flickr has previously been developed with their own code base, the application is handled separately in the next section. Here, we will concentrate on Yahoo! Mail [Yah07f], a web e-mail application not totally unlike Google Mail.

Unfortunately, this evaluation can only be categorized as a total failure, in the scope that no figure is presented here. Yahoo! Mail uses browser detection techniques and degrades into “Mail Classic”. The designers have been thoughtful and left a selectable option to turn over to the full version, but it remains unusable in each one of the tested browsers. Presentation failures include exhausting the memory space of the browser. Nondeterministically, the application may also detect faults in the loading process and degrade into Mail Classic despite the opposite selection.

Mail Classic is fully usable with both microb and Safari. Opera on the N800 has major issues with this presentation as well, whereas Opera on the N95 performs better visually. Navigation on it is tricky because of the lack of a cursor, but manageable after some training. The S60WB fails to load Mail Classic due to some text popups that quickly exhaust the available memory and cause the browser to crash.

5.4 Flickr

Flickr [Yah07c] is one of the best known online image services currently available. It offers both free and subscription-based services for its users. Having started out with its own code base, Flickr is now being (partially) developed with the YUI toolkit. An interesting detail concerning Flickr is that the service now provides widget applications for some mobile platforms, like the product series that the N95 is part of [Yah07d].

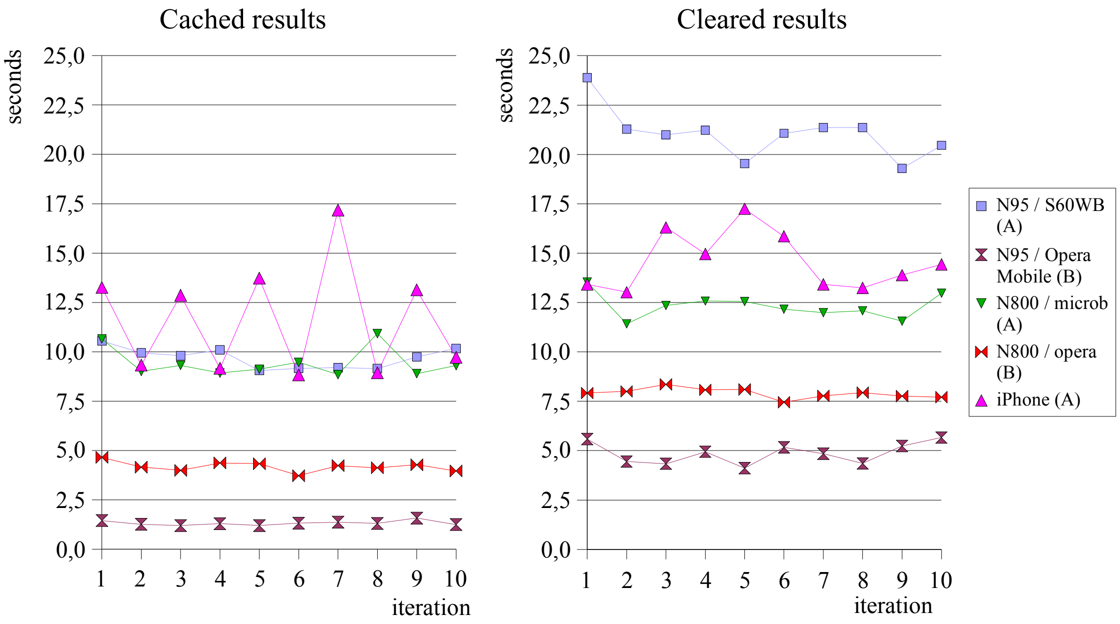


Figure 16: Flickr image gallery, http://www.Flickr.com/.

The target for this benchmark is the loading of a single picture, together with the manipulation interface provided by the image gallery. Thus, one important factor concerns the size of the picture that is downloaded, for it linearly affects the initial loading time of the gallery application. Because we are interested in the performance of the interface, and not the actual image presented, these measurements target a dummy image consisting of a single black pixel on a white background, with a size of 817 bytes.

Proceeding to the capability evaluation, all of the browsers present Flickr quite well. Specially the S60WB surprises positively. The only defect detected with it relates to the automated slide show function, which refuses to work. However, as the slide show is actually a Flash application, it has been excluded from the overall grading. Opera on the N95 has more problems, failing to present the edit-in-place text fields, the photo stream selector arrows, and the slide show.

Turning over to the N800 browsers, microb manages to present the Flickr application without detected defects. However, microb performs a bit slower than Opera. Conversely, Opera has minor issues here as well. It fails to present the same selector arrows as its N95 sibling, but in contrast to it, the N800 version almost fully manages the slide show. Only the “back to your photos” link is broken. It must be mentioned that Opera crashed once during the initial capability testing, while displaying the slide show.

Again, Safari on the iPhone performs quite well. The slide show remains broken, but everything else works. Moving over to the performance charts, iPhone seems to have trouble loading Flickr – Safari is the slowest browser in the cached measurement, while being second slowest with a cleared cache. Interestingly, the slowest browser starting is not microb in this cache-cleared measurement. With Flickr, the S60WB clocks the slowest result, combined with a variance just below Safari's. Both Operas are incredibly fast in both measurements, but they also have the most problems with the presentation. This gives reason to doubt that the browsers may have completely ignored some of the application code while loading.

5.5 Journey Planner for Cycling

The Journey Planner for Cycling [YTV07] provides a pathfinder algorithm that plans near-optimal bicycling routes between locations in the greater Helsinki area. The functionality is somewhat complex, as both selectable way points and a dynamically generated, scalable route image are served to the user. The service is an extremely interesting application for mobile cyclists, and as the analysis proceeds, it becomes clear that the application has been well tested with the predicted field of client browsers.

Safari makes the exception to this rule, since it crashes consistently each time the Journey Planner is accessed. Despite these problems, Safari must be noted for its overall stability, as this is the only application that did cause Safari to crash. Simultaneously, it must also be repeated to the application's benefit that the iPhone had not yet been released in Europe at the time of testing. Therefore, the Journey Planner had probably not yet been tested with the Safari browser.

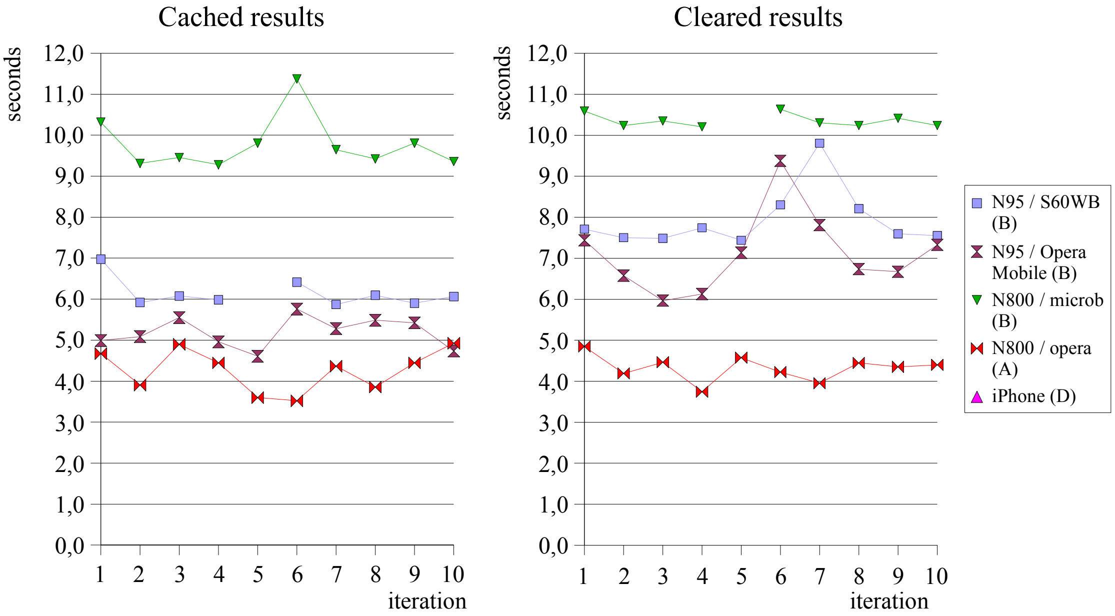


Figure 17: YTV / Journey Planner for Cycling, http://kevytliikenne.ytv.fi/.

In addition to Safari, the S60WB crashed once during iteration five of the cached measurement. Outside of the crash, the S60WB's presentation is quite good. The arrow images used to move the map can not be accessed, but clicking on the way points centers the map correctly. Not all map levels seem to display quite as intended, though. Automatic text completion seems to work, but only if focus leaves the text field, i.e., the user clicks outside the input field containing the partial search term. Opera on the N95 behaves similarly, although it has more problems with the text completion. Due to the way Opera uses its selector field, the browser skips to the end of the list when it is scrolled. This means that only the last seven addresses are selectable. Using the Journey Planner on either of the N95 browsers requires some zooming, as the design is based on a fixed layout.

Opera on the N800 resembles its N95 version, but all the list elements are now selectable thanks to the stylus-based input. Strangely, the arrow keys still refuse to work. While benchmarking microb, a singular network glitch was encountered during iteration five in the cache-cleared measurement, reported by the browser as a communication error. Ignoring this fault, microb managed a fully functional presentation. The cost is high, though: microb was approximately a third slower than the N95 browsers, and more than twice as slow as Opera on the same device. The S60OM was the fastest browser in both of the measurements, but its selector field is certainly problematic in some applications. Responsiveness is noticeably improved by presenting the user with the input fields as early as possible. The benchmarks in Figure 17 reflect this – the measurement stops when the user may start to input data.

5.6 Colorado Geographic

Microsoft provides a showcase of applications built using its ASP.NET AJAX tools on the product home pages [Mic07c]. The candidates are well presented, but a closer inspection reveals that many of the applications make use of Ajax in ways that are difficult to isolate for a repeatable benchmark. Colorado Geographic [Col07] is an exception as its content is quite static, yet informative.

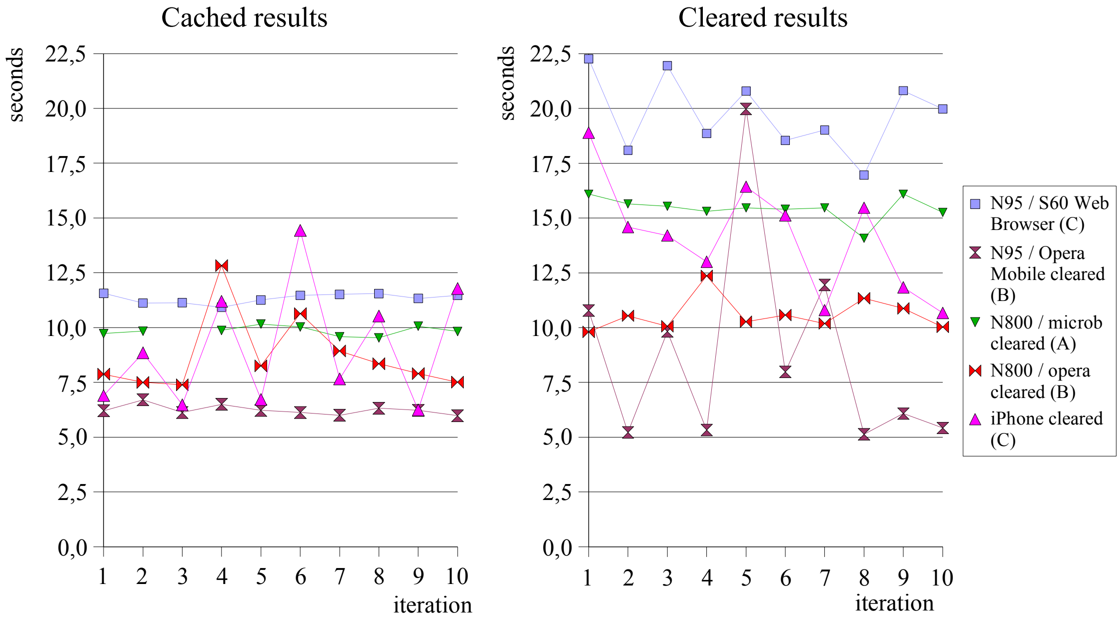


Figure 18: ASP.NET AJAX, Colorado Geographic, http://www.coloradogeographic.com/.

The target application is mobile by nature, being a touring site of photographic locations complete with driving instructions. During testing, it was somewhat difficult not to take a liking to the content, as it is well chosen. The interface uses asynchronous calls in abundance, both in browsing through the images and also in transitions between the categories. Colorado Geographic was presented quite uniformly by the browsers, but the details caused significant hindrances for the application's usability.

The S60WB had major problems with the asynchronous requests. Every link access seemed to cause a redraw, but the target was also retrieved correctly. On the right hand side of the interface, highlighted links were not recognized at all. It may be worth noting that a desktop Firefox also had problems with the links in question, causing a very recognizable “link flicker” when the cursor was positioned above a link. Also, the presentation text flowed over the bottom edge of the container, making it impossible to scroll or read said text. S60OM did a bit better, retrieving the highlighted links asynchronously, but both the left hand navigation and the navigation arrows caused full page redraws.

Following almost typical behavior for it by now, microb crashed once but also offered the best presentation. The only defect detected was a broken back button. This flaw has been excused, as the button does not work on a desktop Firefox either. The detected crash occurred after the capability test, during iteration three of the cached measurement. Ten extra iterations were performed in order to check for a memory leak, but microb remained solid after the singular crash. However, the server did throw four exceptions (during all of the iterations) to the client as JavaScript alerts. The details are in the appendix. Opera had the same problems as on the N95, although this time the presentation text remained visible despite overflowing.

Interestingly, the application was quite difficult to present for Safari. Responsiveness was hampered by a general slowness during use, and also it took visibly longer to load the last stages of the application. As it is difficult to judge when the interface has completed loading on some of the browsers, the measurements presented in Figure 18 end when the default picture starts to become visible. This seemed to be the only fair alternative, as some of the browsers render the interface piece by piece, while others seem to display the results only after everything has been loaded.

Deduced from the difference between the measurement methods charts in Figure 18, Colorado Geographic seems like a graphics intensive application. Conversely, the average loading time of the S60OM changes by less than three seconds in average, despite the outlier in the cache-cleared measurement. This quickness carries over to Opera on the N800 as well – the average increase is just under two seconds. Other browser averages are more obviously affected: the S60WB almost doubles the loading time, whereas loading times for microb and Safari increase by over a half.

5.7 myAOL

AOL LLC[8] is a very large internet service provider operating in the USA. Their myAOL service [AOL07] can be described as a personal portal page that allows users to design

[8] Formerly America Online, Inc. (AOL).

their own home pages. The piecemeal content is served in the form of small widget windows. AOL develops applications using the Dojo toolkit (Section 4.5.5); myAOL used version 0.4.3 at the time of testing.

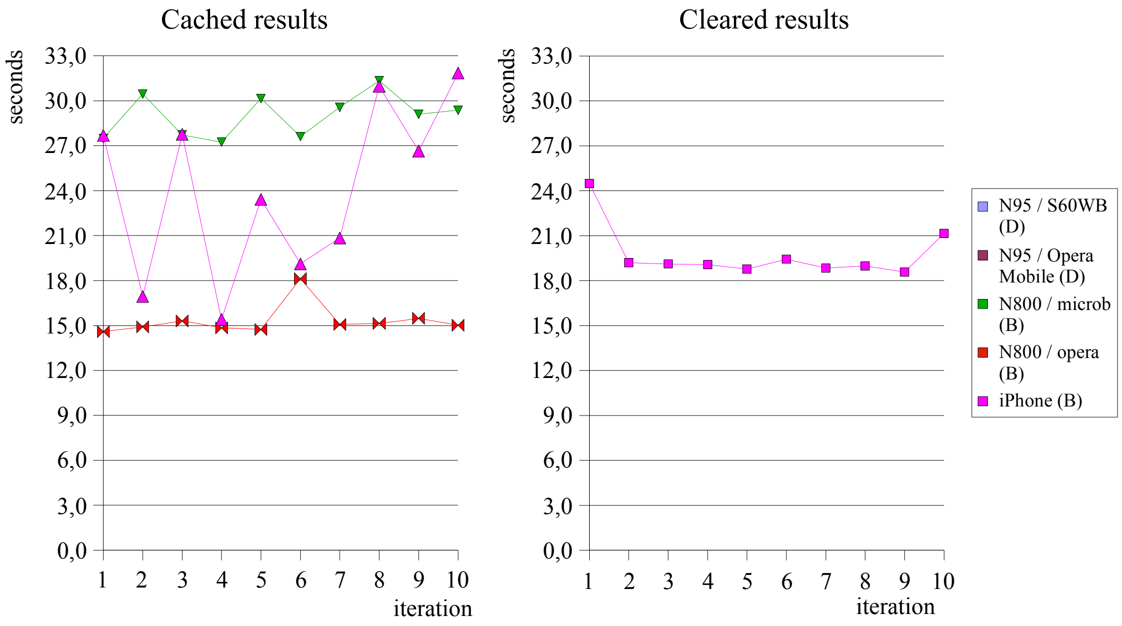


Figure 19: myAOL, http://my.aol.com/.

Unfortunately, myAOL as a target was an almost complete failure in every sense of the capability test. Coincidentally, it was also the very last target in the planned suite. From the capability perspective, both of the N95 browsers failed to present the application as intended. The S60WB was caught by the applications browser detection and redirected to AOL Mobile, which is only accessible by registered users. No method for activating the full application could be found. S60OM displayed almost all of the application content on the second try, but none of the window controls worked, and many windows were altogether devoid of content.

Opera and microb on the N800 managed a bit better, although usability was severely injured by a splash screen that must be closed before accessing the application. The screen could only be closed by activating the browsers' full screen mode. Opera loaded the interface correctly and activated the window controls, but many of the window contents were left blank. The window tabs worked, and the *Mgnet* section loaded its image assortment asynchronously, but it also crashed the browser consistently. microb managed to display seemingly every piece of the interface, both the windows and their content. When accessing tabs, some uninformative (see appendix) errors were returned to the user as JavaScript alerts.

The above-mentioned splash screen was worse for Safari, though. For the first nine iterations the splash screen controls were inaccessible, so that it could not be closed. After the tenth iteration the splash screen disappeared. Weirdly, clearing the cache and reloading the application did not reinstate the splash screen on the first time, but it did do so on the second attempt.

The performance measurements in Figure 19 are visibly lacking. The reason for this is the great variance of presentation errors. These disparities have made it impossible to reliably select a common “loading target”, upon which the measurement would depend. Moreover, there were differences in the presentation between iterations on the **same** browser. All of these facts together have given sufficient cause to leave the measurement as incomplete. The iPhone's results are the only ones present in the cache-cleared measurement, and they exist only to verify the iPhone's distinguishing seesaw variation.

6 Considerations

In the two years that have passed since the coining of the name Ajax [Gar05], it has been criticized for a plethora of flaws, both real and imagined. This chapter is a review of those flaws that have passed the test of time. By now, these issues can and should be addressed as real risks and disadvantages of using Ajax techniques. Many of the issues are independent of the type of client browser, mobile or otherwise, but they may be further exacerbated by the constrained properties of mobile devices.

In no way should this chapter be taken as the conclusive list of issues that must be addressed by every web designer. The purpose is only to demonstrate that problems still remain. Some of the matters described herein, e.g., external references, might have no optimal solutions. Instead, their solutions consist of finding the right balance for the task at hand.

6.1 Cross-Site Scripting Attacks

Cross-site scripting (XSS[9]) attacks [CER00, Cgi02] employ security flaws in the client browsers to execute code with the access privileges granted to the non-compromised application. In essence, during a successful XSS attack the client's (necessary) trust on an application is extended to all additional content generated by other users of the applica-

[9] Formerly CSS, but then renamed to avoid naming issues with Cascading Style Sheets.

tion. XSS attacks are common in the sense that if users are allowed to store data, and that data can later be retrieved (by *any* user), all web applications must take some countermeasures [Oll07] to ensure the presentational safety of said data. If the application does not properly validate the input/output format, stored information meant to be represented as data may be instead evaluated by the client browser as program code, enabling an attack vector. The attack vectors can be divided into three categories based on the methods taken to bypass the browser security mechanisms [Kle05, Wik07b].

XSS attacks attacks are in no way uniquely bound to the use of Ajax. Earlier attacks have abused JavaScript security flaws, causing part of the language's infamy, but all scripting languages are potentially vulnerable [Mic05]. In addition to scripting languages, XSS attacks may be enabled by vulnerabilities in other HTML entities as well, e.g., applets, objects, and forms [Cer00]. As Ajax is fundamentally based on existing technologies, vulnerabilities from the components carry over to Ajax.

XSS attacks are mitigated by the *same origin* policy [Rud01, LiE07] implemented by the browsers. This policy prohibits scripts originating from a specific host to request or access content provided by other hosts. By itself, the policy does not protect users from attacks launched by other users on the same site [CER07]. With vast distributed applications being served from singular host names, the same origin policy becomes inadequate protection against all XSS attack vectors. The policy may also be circumvented by several normally useful techniques, like hidden IFrames and delayed references to external script files. When the policy is circumvented, the attack type changes to a cross-site request forgery.

6.2 Cross-Site Request Forgery

Cross-site request forgery (CSRF[10]) [Har88, Aug07] shares common properties with XSS attacks, but when an attacker executes a CSRF successfully, the goal is to gain access to a secondary host where the compromised client has already been authenticated. An attacker may not require initial information on what hosts the target client has access to [Wal07]. For example, many users advertise their web e-mail accounts by default, and the attacker can launch an attack against all potential users of the e-mail service.

The reduced input methods set by the constrained physical size of mobile devices have been mentioned earlier. As the requirements for password complexity rise in accordance

[10] CSRF is also abbreviated XSRF.

with the available processing power, longer passwords are required to offer reasonable security. Security-conscious users have little reason to trust multiple web applications by reusing the same password, which multiplies the mental capacity required to remember all the keys to commonly accessed services. Entering multiple lengthy pass phrases with reduced input quickly becomes a tedious task and allowing the browser to save the passwords is an alluring alternative. If a malicious entity is then able to bypass the security mechanism of the browser, saved passwords greatly enhance the severity of a CSRF attack. Even if the attacker is unable to download the credentials (XSS), the compromised browser may allow further requests without additional authorization.

6.3 External References

External references are a nontrivial topic from the mobile communications perspective because their use may both enhance and hamper performance. HTML files may include JavaScript, CSS, and other elements either inline or through an external reference. In beginner's tutorials, external references are praised by default, for they enable code reuse and also subtly direct the would-be web designer into current best practices, i.e., the layering of web content [Yan07]. Through the expertise of more advanced engineers, we find out that inline code has its own merits. A recent book by Souders [Sou07] presents a checklist that high performance web site designers should be aware of. The same suggestions are also published by Yahoo! [Yah07e]. In this ordered list, the pole position is taken by the suggestion to minimize HTTP requests. Conversely, the eight list item describes reasons for making JavaScript and CSS external references.

As external references target file names as uniform resource locators (URI), each uncached reference causes a HTTP request for content. In complex applications, multiple host names may be used to bypass the two-connection limit set by the HTTP [Fie99] and increase parallelism [The07a]. If these new host names are initially unknown by the client, each host name will yield an additional DNS query. After the initial requests, caching will reduce further request delays. However, constrained memory properties may effectively limit the amount of entries in the cache.

Although these delays normally cumulate only in complex applications with multiple references, their lengths depend entirely on the round-trip time (RTT) exhibited by the communications network. Mobile devices may very well encounter RTT:s that are orders of magnitude higher than those tested by the designers.

The balance question between inline elements and external references is influenced by the predicted usage profile of the application [Yah07e]. If the typical user visits only a single page and then leaves, aggregating content into a smaller number of files may be the optimal solution. Ultimately, all content may be combined into a single file. This may also be the case if the additional file sizes are very small. The alternative behavior, a single user visiting multiple pages, should be met with separating the content and ensuring that it will be cached through the use of HTTP:s expiration headers. When external references are used, care should be taken to position CSS and JavaScript segments correctly. Depending on the order of rendering phases implemented by the browser, waiting for external requests may block rendering [Nok07e, Yah07e]. Thus, it is recommended to reference CSS in the meta headers and JavaScript as late as possible.

Finally, the approaches may be combined by allowing the user to quickly load the predicted page, and then use delayed loading to request the external references. Advanced techniques combine this approach with image clipping [Gra07b] or CSS sprites [She04]. Asynchronous requests may benefit the loading phase, while very long default expiration periods combined with file name versioning can enhance caching.

6.4 Caching Problems

The beneficial effects of successful caching systems have been long since known [Abr95, Bae97, Jia99], and they carry over to HTTP transfers quite elegantly. Observations made by Yahoo!'s performance team have given cause to adapt the Pareto principle, also known as the 80/20 rule, into browser-server communications [The06]. In this case, the principle tells us that 80% of the presentation delay can be affected by the *front-end engineering* of the user interface, i.e., how the browser fetches external references to page elements. These observations also form the rationale for the high ranking of the HTTP minimization suggestion, described in the previous section. Again, these measurements were done in near-optimal conditions; there is little reason to disbelieve that the dominant delay should be enforced when the RTT increases.

Exactly how the browsers implement caching has become a mire of surprises for inexperienced designers [Not06]. The detailed workings of the caching mechanisms defined in the HTTP are somewhat complex by necessity; the subject gets cloudier when actual implementation differences are considered. In some cases, the encountered behavior may differ due to (transparent) proxies or server-side redirects administered on legacy applications. For example, if a XHR request should silently handle such redirects or expect them to be processed by a separate function is an open question.

Fortunately, it seems that at least the major browsers handle caching for the XHR requests similar to HTTP caching in general. Still, disparities do exist. As the multiple benchmarks in Chapters 4 and 5 demonstrate, the cached performance varies significantly between browsers. There is a risk that the strange behavior exhibited by the Safari browser steals part of the show. Two speculated causes for its dualistic cached benchmark results are presented in Section 4.4.2. Nevertheless, Safari is by no means the only browser that has exhibited paranormal caching behavior [Dav06]. Here, it is sometimes difficult indeed to separate bugs from features.

On mobile devices, the possibility to employ caching has been constrained by the available storage space. The situation is changing, as all of the devices presented in this thesis feature storage capacities in the gigabyte range. With such capacities, comparable client-side permanent caches depend only on the selected storage algorithms [GaT05], as care must be taken to avoid unnecessary wear out of the flash memories. Another interesting development is the release of Google Gears [Goo07c], as the project aims to enable offline use of internet applications, including Ajax.

But caching is not the silver bullet for HTTP performance. An empty cache must be initially filled by requesting every content element. The same behavior is also encountered when the client determines that the contents of its cache are stale. Studies concerning the amount of clients with either empty or stale caches have predicted [Abr95, Bae97] or measured [The07b] that the hit maximum hit rate achievable is between 30% and 60%. Other techniques must be implemented to ensure adequate performance for the remaining portion of the clients.

6.5 Memory Leaks

A major factor affecting application crashes are the occurrence of memory leaks [Eri02], and web browsers are no exception. Partially caused by caching techniques, leaks occur when the browser fails to free unused memory reservations through garbage collection techniques [Par07, Pup07b]. Such techniques may be confused by repeated DOM manipulations that cause removed elements to remain referenced by data structures or event handlers [Cro07]. The culprit need not always be a bug in the browser engine, for certain programming techniques can also result in situations where the garbage collection process may not justly tear down memory objects [Gur07, Lec07]. Also, a memory leak may be hard to distinguish from main memory caching [Moz07] if the leak does not progress into a full blown browser crash.

Multiple crashes were encountered during the capability tests of Chapters 4 and 5. The crashing applications were further tested for memory leaks by executing additional iterations. Results were somewhat inconclusive, for while at least one clear memory leak was detected (Section 4.4.9), many times the repeated iterations were insufficient to cause further crashes. One possible reason is that simply loading the application does not trigger the conditions causing the memory leak – they are only met after prolonged use, such as the initial capability evaluation at the beginning of each measurement.

If the memory leak does not cause the browser to crash, increasing amounts of memory are hogged by the process. On mobile devices, the amount of main memory has not increased at the same pace as the amount of storage capacity. Battery drain is one of the factors limiting development [VeF05]. Therefore, a memory leak may exhaust the full memory space far more quickly than on a desktop computer. If the operating system is inadequately protected against such incidents, the user may be left without access to the other features offered by the converged device, e.g., voice calls [Mol07]. During the performance measurements, several situations were encountered when power cycling was left as the only option available. These situations were extremely frustrating, although individual reactions are naturally subjective.

6.6 Accessibility

Accessibility is a hard property to quantify in the web environment, where clients may present vastly different capabilities concerning nearly every property, e.g., input type, display resolution, and network connection. This includes screen readers, i.e., client browsers with minimal or non existing visual elements, replaced by audio feedback. Screen readers' additional limitations may include notably reduced scripting support [Edw06]. Examining the least capable clients has more than pedagogic value, for political decisions have been made to ensure that user interfaces remain accessible for all potential visitors. Regulations like the U.S. Section 508 law [Sec07] must be acknowledged if the designed application is a public service. This is important, since public services have already embraced web technologies like electronic forms, and one of the main benefits of Ajax is the improved responsiveness of such forms.

The question if Ajax can be made accessible has been met with heated debate . Progressive enhancement techniques may make the designing of complex applications a heavily iterative process, where the same application is written more than once to ensure accessibility. In a worst case scenario, the same application would have to be now designed once more for screen readers, a task which requires very specialized quality assurance skills. No doubt should exist that the additional amount of work increases development costs.

Fortunately, this issue has not been neglected by the designers. As web development is by nature fast-paced, multiple people have remarked on the accessibility problems, and proceeded to develop workarounds for them [Fea05, Koc05, Kra05]. Thiessen and Chen [ThC07] recently demonstrated a chat application that uses Ajax and specific markup called live regions to inform assistive technologies, like screen readers, of updated elements.

6.7 A Side Note on Battery Life

Battery lifetime has been repeatedly mentioned when considering the limitations of mobile devices. This amount of attention may not be unwarranted, as there is reason to believe that mobile batteries will remain one of the bottlenecks in the near future. Moore's law says nothing about the electrical capacities of batteries, and comparing the current properties with those of the 1990's, we learn that capacities have only tripled [CaC05]. As all of the measurements were executed with no attached chargers, the following observations concerning charging periods may be made.

N95 was charged five times during the testing period of one week. This is about twice (or less) the normal charging count for an equivalent usage period with no web browsing, but “normal” voice calls and SMS messaging. The N95 certainly has issues with its battery life in normal usage. This might be indicative of parasitic drain [VeF05] caused by the number of functional components present in the converged device.

N800 was charged four times. This is about thrice the normal charging count for the equivalent period with some, but not continuous, use of the device. The N800 has a great battery lifetime, probably enhanced by the large physical size of its battery.

The iPhone was charged three times. Please note that the iPhone was also powered down at the end of each benchmark workday. As noted in Section 3.6, the iPhone was not connected to a carrier network during the tests. I have no experience with the daily use or charging pattern of an iPhone, and can thus offer no further analysis concerning it.

Based on the N95 and the N800, even extreme Ajax usage does not seem to increase the battery drain by a factor greater than four. Note that despite the heavy usage, the devices still remained on standby and idle for most of the testing period. It seems reasonable to confirm that the parasitic drain currently dominates over the active drain.

7 Conclusion

While beginning the initial tests, it seemed that the component technologies of Ajax were adequately supported by the chosen mobile browsers. After testing the browsers' support for the toolkit applications, it becomes clearer that testing is the key for success for full-blown mobile Ajax applications. One can not yet assume that applications supported by the desktop browsers would be consequently supported by the mobile browsers [Raa07]. Browser fragmentation seems to flow over to the mobile devices with the shared code bases of the mobile and desktop user agents.

	A	B	C	D
S60 Web Browser	8	4	2	5
S60 Opera Mobile	4	5	1	9
microb on N800	10	7	1	1
Opera on N800	6	5	2	6
iPhone	11	2	3	3

Table 2: Frequencies of overall capability grades for the selected browsers. Range from A (best) to D (worst).

Table 2 shows the frequencies of different capability grades listed by browser. Note that these results ignore caching, as it affects only the performance. In the table, Safari on the iPhone and microb on the N800 have the most perfect (A) grades, signaling capability evaluations with no defects detected. The S60WB is not far behind, though. Combining the number of perfect and near-perfect (B) grades, microb is the better browser in overall. It also enjoys the fewest number of total failures. Safari is very close to microb, while the S60WB shows its disadvantage: significantly more failures in total. In the evaluations, Opera Mobile on the N95 had the most problems. These problems can not be explained by the age of the browser, as the evaluated version 8.65 was released as late as October 1st, 2007 [Ope07b].

Table 3 shows the benchmark results listed by speed, calculated by the average timings of each measurement respectively. Failed measurements are indicated by the 'x' column. Measurement failures are typically caused by significant deviations in the visual presentations of the applications, causing the lack of a common measurement target. Detailed reasons for not measuring a browser are listed in the analyses.

	1	2	3	4	5	x
S60 Web Browser	6	8	6	7	7	1
S60 Opera Mobile	22	8	3	1	0	1
microb on N800	1	4	9	14	7	2
Opera on N800	8	16	7	4	0	2
iPhone	0	1	10	8	14	2

Table 3: Rankings by speed, represented by the result averages of each measurement, scale from 1 (fastest) to 5 (slowest). A rank of 'x' indicates a failed measurement, caused by unacceptable differences in the presentation..

By far, the fastest browser is Opera Mobile on the N95. This seems to be well in line with the overall worst capability in the capability evaluations. This combination seems to be indicative of ignored program directives, meaning that the browser gains speed by not executing some parts of the application code. Safari's high number (14) of slow results is caused by the browser's distinctive performance variation, specifically of pairwise high and low values. This phenomena has not yet been satisfactorily explained.

The Opera-based browser on the N800 defends its place as the default browser for the OS2007 well. Opera is decidedly faster than microb, but the cost is paid in reduced capability. Just by looking at the table, microb seems to be just a little faster than Safari. This conclusion may be a bit misleading, as the visual representations of the graphs better shows the stability of microb's measurements. Safari may be faster than microb on every second attempt, but it remains invisible here.

	1	2	3	4	5	0
S60 Web Browser	2	6	1	2	3	21
S60 Opera Mobile	5	1	0	0	0	29
microb on N800	1	3	6	9	1	17
Opera on N800	5	3	2	2	0	25
iPhone	0	0	6	5	9	15

Table 4: Rankings of perfect presentations by speed, represented by the result averages of each measurement, scale from 1 (fastest) to 5 (slowest). A rank of 0 indicates a capability grade below A, meaning that the presentation contained errors.

Last but not least, Table 4 aggregates the results of the capability grades and the performance results. The relative speeds are now calculated by only considering perfect presentations, i.e., those with capability grades of 'A'. Both Operas dominate the number one positions, but at a high cost: these browsers also exhibit the most defects.

Now that only perfect grades are considered, microb is no longer the slowest browser. This honor falls to Safari, although it is a fair trade-off. Safari may never be the fastest, but it gets the job done most often. In the measurements, it also displayed the fewest crashes. But please note that its main competitor, microb, was still considered a development version at the time of testing. On the other hand, browser development is definitely a continuous process.

And it well should continue, for there is certainly room for improvements. While initial loading times of over 20 seconds are perhaps endurable by those users who enjoy the newest in mobile gadgetry, almost half a minute is an unacceptable delay if the application is to be used regularly. The benefits for heightened responsiveness that Ajax brings are lost if the user never executes the application at all. Likewise, improved network connections and successful caching prove insufficient when the browser becomes the bottleneck. Fortunately, the same benchmarks also convey glimmers of hope. As we have seen with the Journey Planner for cycling (Section 5.5), the waiting period may be diminished by cleverly reordering the content so that users may begin to input their query while loading continues. The XML 2006 event schedule and GWT:s dynamic table demonstrate that mobile Ajax application not only exist, but are also be presented in a (nearly) uniform vein. Ajax toolkits will eventually feature support for mobile browsers, although much work yet remains undone.

7.1 Further Research

The measurements in this thesis should be further extended by more in-depth performance analysis of the mobile browsers' rendering phases. As noted, parsing errors of the different code types, e.g., HTML, CSS and JavaScript, may cause the relevant execution or presentation phases to be aborted, which heavily enhances performance at the cost of capability. Such an analysis will probably require a switch from the current method of black-box testing into a white-box setup, where the browsers are injected with timing hooks. Otherwise, transitions from one phase to another may remain invisible for measurement purposes. Additionally, multiple synthetic Ajax benchmarks have recently been developed. A review of such applications could be the subject of a further study, as some have shown initial promise and acceptance by the development community.

8 Acknowledgment

The following people must be acknowledged for their aid in the effort of writing this thesis:

Kimmo Raatikainen for the original innovation and the name for the thesis. Professor Raatikainen managed to instruct this thesis well by sparking the right ideas early on, and then by gently nudging towards the possible answers. I sincerely hope that he has the possibility to inspire many more students.

Jussi Kangasharju for making time during the original instructor's absence and taking special interest in this thesis. The alternative analysis of Safari's benchmark behavior came from professor Kangasharju.

Kimmo Vehkalahti for consultation concerning the visualization of the results. His expertise formed the basis of the presentation of the graphs, whereas all flaws are naturally mine (the graphs have proven resistant to further improvements). It is hard to think of a more approachable lecturer.

Guido Grassel and Mikko Honkala for asking a great many intelligent questions. Trying to answer them helped me clarify the benchmarking procedure and what could be expected from the results. The idea of dividing the measurements into separate phases in a future work came from Mikko Honkala.

Niko Välimäki, Samppa Kytömäki, and Janne-Pekka Ahvo for several discussions on how to best present the analysis and make the results as easy to approach as possible. Through our discussions it was easier to focus on the relevance of the conclusions. The spark of the idea of what Safari might be doing in the benchmarks came from Samppa.

Finally, my grandmother, Seija Latonummi for teaching me how to read, my grandfather, Pentti Savolainen for urging me to study, and my mother for making it possible.

References

Abr95 Abrams, M., *et al.*, Caching Proxies: Limitations and Potentials. *Electr. Proc. 4th International WWW Conference '95: The Web Revolution* (Boston, MA, 11.-14.12.1995).

Ado07a Adobe Systems Incorporated, ActionScript Technology Center, http://www.adobe.com/devnet/actionscript/. [1.1.2008]

Ado07b Adobe Systems Incorporated, Adobe Flash CS3 Professional, http://www.adobe.com/products/flash/. [1.1.2008]

Aja07a AjaxPatterns.org, On-Demand JavaScript. AjaxPatterns wiki, http://ajaxpatterns.org/On-Demand_Javascript. [1.1.2008]

Aja07b AjaxPatterns.org, HTTP Streaming. AjaxPatterns wiki, http://ajaxpatterns.org/HTTP_Streaming. [1.1.2008]

Aja07c AjaxPatterns.org, Frameworks. AjaxPatterns wiki, http://ajaxpatterns.org/Frameworks. [1.1.2008]

Alm05 Almaer, D., Ajax Survey Results Are In, 20.10.2005, http://ajaxian.com/archives/ajax-survey-results-are-in. [1.1.2008]

Alm07 Almaer, D., Yahoo! Search Conceptual Precaching, 16.11.2007, http://ajaxian.com/archives/yahoo-search-contextual-precaching. [1.1.2008]

And98 Andreessen, M., Innovators of the Net: Brendan Eich and JavaScript. Netscape TechVision columns, 24.6.1998, http://wp.netscape.com/columns/techvision/innovators_be.html. [1.1.2008]

Ant07 Anttila, E., Ajax-työkalupakit. Master's Thesis for the Department of Computer Science, University of Helsinki, 23.5.2007 (in Finnish). [Also available from http://www.cs.helsinki.fi/u/eeanttil/cv/Ajax-tyokalupakit-Eero_Anttila.pdf. [1.1.2008]

AOL07 AOL LLC, myAOL, http://my.aol.com/. [1.1.2008]

Apa07 The Apache Software Foundation, Apache ActiveMQ, http://activemq.apache.org/. [20.12.2007]

App07a Apple Inc., Apple iPhone Technical Specifications, http://www.apple.com/iphone/specs.html. [1.1.2008]

App07b Apple Inc., iPhone Dev Center, http://developer.apple.com/iphone/devcenter/designingcontent.html. [1.1.2008]

Arc07 Arcand, J-F., Introducing Grizzlet! Building Ajax/Comet based application with POJO, Jean-Francis Arcand's Blog, 21.9.2007, http://weblogs.java.net/blog/jfarcand/archive/2007/09/introducing_gri_1.html. [1.1.2008]

Ash00 Ashley, B., Remote Scripting. Ashley IT Remote Scripting Resources ...and home of JSRS, http://www.ashleyit.com/rs/. [1.1.2008]

Aug07 Auger, R., The Cross-Site Request Forgery (CSRF/XSRF) FAQ v1.58. Cgisecurity.net, Articles, 28.1.2007, http://www.cgisecurity.com/articles/csrf-faq.shtml. [1.1.2008]

Bae97 Baentsch, M., *et al.*, Enhancing the Web's Infrastructure: From Caching to Replication. *Internet Computing, IEEE*, 1,2(March/April 1997), pages 18-27.

Blu07 Bluetooth Special Interest Group (SIG), the Official Bluetooth Technology Info Site, http://www.bluetooth.com/bluetooth/, [1.1.2008]

Bor06 Born, J., 15 Days of jQuery, http://15daysofjquery.com/. [1.1.2008]

Bra06 Bray, T., *et al.*, Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation 29.9.2006, http://www.w3.org/TR/2006/REC-xml-20060816/. [1.1.2008]

Bur07 Burckart, E., The allure of Comet. IBM, developerWorks, WebSphere, Comment lines, 7.11.2007, http://www.ibm.com/developerworks/websphere/techjournal/0711_col_burckart/0711_col_burckart.html. [1.1.2008]

ByH07 Byers, P., Hendry, K., Aplix's AJAX platform, http://wiki.webvm.net/pp2007/. [1.1.2008]

Car07 Carter, M., HTTP Streaming and Internet Explorer. CometDaily, Articles & News, 25.10.2007, http://cometdaily.com/2007/10/25/http-streaming-and-internet-explorer/. [1.1.2008]

CaC05 Casas, R., Casas, O., Battery Sensing for Energy-Aware System Design. *Computer*, 38,11(November 2005), pages 48-54.

Cat05 Catalan, M., *et al.*, TCP/IP analysis and optimization over a precommercial live UMTS network. *Wireless Communications and Networking Conference*, 2005 IEEE, vol. 3, 13.-17.3.2005, pages 1503-1508.

CER00 CERT, CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests, 3.2.2000, http://www.cert.org/advisories/CA-2000-02.html. [1.1.2008]

CER07 CERT, Understanding Malicious Content Mitigation for Web Developers. CERT Coordination Center, http://www.cert.org/tech_tips/malicious_code_mitigation.html. [1.1.2008]

Cgi02 Cgisecurity.com, The Cross Site Scripting (XSS) FAQ, May 2002, http://www.cgisecurity.com/articles/xss-faq.shtml. [1.1.2008]

CCP02 Chakravorty, R., Cartwright, J., Pratt, I., Practical Experience with TCP over GPRS. *Global Telecommunications Conference, 2002* (GLOBECOM '02, IEEE, 17.-21. November, 2002, Taipei, Taiwan), vol. 2, pages 1678-1682.

Cha03 Champeon, S., Progressive Enhancement and the Future of Web Design. Webmonkey, 18.6.2003, http://www.webmonkey.com/webmonkey/03/21/index3a.html?tw=design. [1.1.2008]

Col07 Colorado Geographic, http://www.coloradogeographic.com/. [1.1.2008]

Cos02 Costello, E., Remote Scripting with IFrame. O'Reilly Network, 8.2.2002, http://www.oreillynet.com/pub/a/javascript/2002/02/08/iframe.html. [Also available from http://developer.apple.com/internet/webcontent/iframe.html 1.1.2008].

CPJ05 Crane, D., Pascarello, E., James, D., *Ajax in Action*. Manning Publications Co., Greenwich, October 2005.

Cro01 Crockford, D., JavaScript: The World's Most Misunderstood Programming Language, 2001, http://www.crockford.com/javascript/javascript.html. [1.1.2008]

Cro07 Crockford, D., JScript Memory Leaks, http://javascript.crockford.com/memory/leak.html. [1.1.2008]

Dav06 Davison, B. D., Caching-related browser bugs. web-caching.com, Browser Bugs, 21.9.2006, http://www.web-caching.com/. [1.1.2008]

Dav07 Davis, F., Mobile browsing proxies. S60 Browser Blog, 17.8.2007, http://blogs.s60.com/browser/2007/08/mobile_browsing_proxies.html. [1.1.2008]

Dis06 Disabato, N. J., The effects of Ajax web technologies on user expectations: a workflow approach. Master's Thesis for the School of Information and Library Science, University of North Carolina, Chapel Hill, 10.4.2006, http://etd.ils.unc.edu:8080/dspace/handle/1901/269. [1.1.2008]

Doj07a The Dojo Foundation, The Dojo Toolkit, http://www.dojotoolkit.org/. [1.1.2008]

Doj07b The Dojo Foundation, The Dojo Offline Toolkit, http://dojotoolkit.org/offline. [1.1.2008]

Ecm99 Standard ECMA-262, ECMAScript Language Specification, 3rd edition. Ecma International, December 1999, http://www.ecma-international.org/ publications/standards/Ecma-262.htm. [1.1.2008]

Edw06 Edwards, J., AJAX and Screenreaders: When Can it Work? SitePoint, JavaScript & Ajax Tutorials, 5.5.2006, http://www.sitepoint.com/article/ajax-screenreaders-work. [1.1.2008]

Eer06 Eernisse, M., Build Your Own AJAX Web Applications. SitePoint, JavaScript & Ajax Tutorials, 28.6.2006, http://www.sitepoint.com/article/build-your-own-ajax-web-apps. [1.1.2008]

Eri02 Erickson, C., Memory leak detection in embedded systems. *Linux Journal*, 2002,101(September 2002), page 9. [Also available from http://www.linuxjournal.com/article/6059 1.1.2008].

Esp00 Esposito, D., Exchanging Data Over the Internet Using XML. *MSDN Magazine*, 15,4(April 2000). [Also available from http://msdn.microsoft.com/ msdnmag/issues/0400/cutting/default.aspx 1.1.2008].

Fea05 Featherstone, D., JavaScript and Accessibility. Box of chocolates, 12.5.2005, http://www.boxofchocolates.ca/archives/2005/06/12/javascript-and-accessibility. [1.1.2008]

FFJ03 Ferraiolo, J., 藤沢 淳 (FUJISAWA Jun), Jackson, D., Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation 14.1.2003, http://www.w3.org/TR/SVG11/. [1.1.2008]

Fie99 Fielding, R., *et al.*, Hypertext Transfer Protocol – HTTP/1.1. Request for Comments 2616, 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html [1.1.2008]

Gal06 Galbraith, B., Ajaxian.com 2006 Survey Results, 23.9.2007, http://ajaxian.com/archives/ajaxiancom-2006-survey-results. [1.1.2008

Gal07a Galbraith, B., Ajaxian.com 2007 Survey Results, 12.10.2007, http://ajaxian.com/archives/ajaxian-2007-survey-results. [1.1.2008]

Gal07b Galbraith, B., Moo on Us: Survey Update, 16.10.2007, http://ajaxian.com/archives/moo-on-us-survey-update. [1.1.2008]

Gam95 Gamma, E., *et al.*, Design Patterns: *Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, December 1995.

Gar05 Garrett, J. J., Ajax: A New Approach to Web Applications. Adaptive Path essay archives, Feb. 2005, http://www.adaptivepath.com/publications/essays/ archives/000385.php. [1.1.2008]

GaT05 Gal, E., Toledo S., Algorithms and data structures for flash memories. *ACM Computing Surveys* (CSUR), 37,2(June 2005), pages 138-163.

Geo07a Georgi, R., Frost Ajax Library. PavingWays, 2007, http://www.pavingways.com/frost-ajax-library/. [1.1.2008]

Geo07b Georgi, R., Mobile Widgets: the ubiquitous mobile web. PavingWays, 6.5.2007, http://www.pavingways.com/mobile-widgets-the-ubiquitous-mobile-web_84.html. [1.1.2008]

Geo07c Georgi, R., Ajax on mobile devices – making mobile web apps ubiquitous. XTech 2007 (15.-18.5.2007, Paris, France), http://2007.xtech.org/public/schedule/paper/58. [1.1.2008]

GEO07d GEOTEK IT Consulting, Web Rendering Services, http://ipinfo.info/html/rendering_services.php. [1.1.2008]

Get07 Getahead, DWR – Easy Ajax for Java, http://getahead.org/dwr. [1.1.2008]

GoL05 Goldman, O., Lenkov, D., XML Binary Characterization. W3C Working Group Note 31.3.2005, http://www.w3.org/TR/xbc-characterization/. [1.1.2008]

Goo02 Goodman, D., *Dynamic HTML: The Definitive Reference*, 2nd edition. O'Reilly Media Inc., September 2002.

Goo07a Google, Google Talk, http://www.google.com/talk/. [1.1.2008]

Goo07b Google, Google Maps, http://maps.google.com/. [1.1.2008]

Goo07c Google, Google Gears Beta, http://gears.google.com/. [1.1.2008]

Goo07d Google, Google Web Toolkit, http://code.google.com/webtoolkit/. [1.1.2008]

Goo07e Google, Google Suggest, http://www.google.com/webhp?complete=1&hl=en. [1.1.2008]

Goo07f Google, Google Mail, http://mail.google.com/. [1.1.2008]

Gra07a Grassel, G., AJAX in Widgets and Web UIs. Position paper, W3C/Open-Ajax Alliance Workshop on Mobile Ajax (Microsoft Silicon Valley campus, Mountain View, California, USA, 28.9.2007), http://www.w3.org/2007/06/mobile-ajax/papers/nokia.grassel.pdf. [1.1.2008]

Gra07b Graef, A., pzImageCombine. dev.portalZINE, 25.11.2007, http://dev.portalzine.de/index?/pzImageCombine. [1.1.2008]

Gri04 Griffits, P., Elastic Design. A List Apart, no. 167(9.1.2004), http://www.alistapart.com/articles/elastic/. [1.1.2008]

Gru07 Gruhier, S., Prototype Window Class (PWC) version 1.3, 24.4.2007, http://prototype-window.xilinus.com/. [1.1.2008]

Gul07 Gully, S., How to implement COMET with PHP. Zeitoun.net, 22.7.2007, http://www.zeitoun.net/index.php?2007/06/22/46-how-to-implement-comet-with-php. [1.1.2008]

Gur07 Gurevich, P., IE+JScript Performance Recommendations Part 3: JavaScript Code Inefficiencies. MSDN, The Microsoft Internet Explorer Weblog, 4.1.2007, http://blogs.msdn.com/ie/archive/2007/01/04/ie-jscript-performance-recommendations-part-3-javascript-code-inefficiencies.aspx. [1.1.2008]

Han07 Hansson, D. H., Ruby on Rails, http://www.rubyonrails.org/. [1.1.2008]

Har88 Hardy, N., The Confused Deputy. *ACM SIGOPS Operating Systems Review*, 22,4(October 1988), pages 36-38. [Also available from http://www.cis.upenn.edu/~KeyKOS/ConfusedDeputy.html 1.1.2008].

Hed07 Hedges, A., Speed test: innerHTML versus DOM manipulation, 5.7.2007, http://andrew.hedges.name/experiments/innerhtml/. [1.1.2008]

Hég04 Le Hégaret, P., Document Object Model (DOM) Techical Reports. W3C technical reports, http://www.w3.org/DOM/DOMTR. [1.1.2008]

Hol99 Holzschlag, M., Dynamic vs. Fixed: A Proposal for Peace at the Table. webtechniques, 10(October 1999). [Also available from http://molly.com/articles/markupandcss/1999-10-dynamic.php 1.1.2008].

IBM07 IBM, WebSphere Application Server Feature Pack for Web 2.0 Beta Program, https://www14.software.ibm.com/iwm/web/cc/earlyprograms/websphere/ibmajaxw/. [1.1.2008]

IDC07 IDC, Converged Mobile Device Market Surges Ahead on 42% Growth in 2006, According to IDC. IDC Press Release, 26.2.2007, http://www.idc.com/getdoc.jsp?containerId=prUS20578607. [1.1.2008]

Jaa06 Jaaksi, A., Building consumer products with open source. LinuxDevices.com, 11.12.2006, http://www.linuxdevices.com/articles/AT7621761066.html. [1.1.2008]

Jao06 Jaokar, A., Mobile web 2.0: AJAX for mobile devices – why mobile AJAX will replace both J2ME and XHTML as the preferred platform for mobile applications development. Open Gardens, futuretext, 1.1.2006, http://opengardensblog.futuretext.com/archives/2006/01/mobile_web_20_a.html. [1.1.2008]

JGR07 Jaokar, A., Georgi, R., Rieger, B., Mobile Ajax FAQ. Horizon Channel, 1.7.2007, http://www.horizonchannel.com/archives/26. [3.12.2007]

Jia99 Jia, W., A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communications Review*, 29,5(October 1999), pages 36-46.

jQu07 jQuery, official site, http://jquery.com/. [1.1.2008]

Jus04 Justus, C., Google Suggest Dissected..., Server Side Guy blog, http://serversideguy.blogspot.com/2004/12/google-suggest-dissected.html. [1.1.2008]

Kal04 Kalat, J. W., *Biological Psychology*. Wadsworth Pub. Co., July 2003.

Kan07 Kangasharju, J., An XML Messaging Service for Mobile Devices. Licentiate Thesis for the Department of Computer Science, Helsinki University, 4.2.2006, http://urn.fi/URN:NBN:fi-fe20061478. [1.1.2008]

Kes07 van Kesteren, A., The XMLHttpRequest Object. W3C Working Draft 10-June-2007, http://www.w3.org/TR/XMLHttpRequest/. [1.1.2008]

Kin06 Kinnunen, K., GTK+ WebCore Project Home Page. SourceForge.net, 2006, http://gtk-webcore.sourceforge.net/. [1.1.2008]

Kin07 Kingman, H., Nokia N800 gains a Mozilla-based browser. Linuxdevices.com, 19.7.2007, http://www.linuxdevices.com/news/NS8360022837.html. [1.1.2008]

Kir06 Kirschner, H., HistoryManager, http://digitarald.de/playground/history.html. [1.1.2008]

Kle05 Klein, A., DOM Based Cross Site Scripting or XSS of the Third Kind. Web Application Security Consortium, Articles, 4.7.2005, http://www.webappsec.org/projects/articles/071105.shtml. [1.1.2008]

KLT05 Kangasharju, J., Lindholm, T., Tarkoma, S., Requirements and design for XML messaging in the mobile environment. *Proc. 2nd International Workshop on Next Generation Networking Middleware* (6.5.2005, Waterloo, ON, Canada), 2005, pages 29-36. [Also available from http://www.cs.helsinki.fi/u/jkangash/xml-messaging-mobile.pdf, 1.1.2008].

Kne06 Kneschke, J., COMET meets mod_mailbox. lighty's life, lighttpd's blog, 27.11.2006, http://blog.lighttpd.net/articles/2006/11/27/comet-meets-mod_mailbox. [1.1.2008]

Koc04 Koch, P-P., A history of browsers. QuirksMode.org, 2004, http://www.quirksmode.org/browsers/history.html. [1.1.2008]

Koc05 Koch, P-P., You should've been @media - part 2. QuirksMode.org, 14.6.2005, http://www.quirksmode.org/blog/archives/2005/06/you_shouldve_be_1.html. [1.1.2008]

Koc06 Koch, P-P., Benchmark – W3C DOM vs. innerHTML. QuirksMode.org, August 2006, http://www.quirksmode.org/dom/innerhtml.html. [1.1.2008]

Kra05 Krantz, P., AJAX and Accessibility. Standards Schmandards, 1.3.2005, http://www.standards-schmandards.com/2005/ajax-and-accessibility/. [1.1.2008]

LaH03 Laukkanen, M., Helin, H., Web Services in Wireless Networks – What Happened to the Performance? *Proc. International Conference on Web Services* (ICWS'03), Las Vegas, Nevada, USA, June 2003, pages 278-284.

Law01 Lawton, G., Browsing the Mobile Internet. *Computer*, 34,12(December 2001), pages 18-21.

Leb06	Lebedev, G., DHTML JavaScript Benchmark (DOM and innerHTML), 15.10.2007, http://www.gloo.ru/blogs/gloom.dhtml_javascript_benchmark._1_en.wiki.aspx. [1.1.2008]
Lec07	Lecomte, J., The Problem With innerHTML. Julien Lecomte's Blog, 12.12.2007, http://www.julienlecomte.net/blog/2007/12/38/. [1.1.2008]
Leh06	Lehtonen T., *et al.*, Towards user-friendly mobile browsing. *Proc. 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, Pisa, Italy, 2006, article no. 6.
Ler06	Lerner, R., JavaScript. At the Forge series, *Linux Journal*, 2006,149(September 2006), page 10. [Also available from http://www.linuxjournal.com/article/9131 17.12.2007].
LiB99	Lie, H.W., Bos, B., Cascading Style Sheets, level 1. W3C Recommendation 17.12.1996, revised 11.1.1999, http://www.w3.org/TR/CSS1. [1.1.2008]
LiE07	Livshits, B., Erlingsson, U., Using Web Application Construction Frameworks To Protect Against Code Injection Attacks. *Proc. 2007 workshop on Programming languages and analysis for security* (PLAS '07, 14.6.2007, San Diego, California, USA), ACM, New York, NY, USA, 2007, pages 95-104. [Also available from http://research.microsoft.com/~livshits/ 1.1.2008].
Lil00	Lilja, D. J., *Measuring computer performance: A practitioner's guide*. Cambridge University Press, Cambridge, U.K., 2000.
Lun02	Lundqvist, D., Remote scripting with javascript. http://www.dotvoid.com/view.php?id=13. [1.1.2008]
mad07	mad4milk, moo.fx – super lightweight javascript effects library, http://moofx.mad4milk.net/. [1.1.2008]
Mae07	Maemo 4.0 Architecture, Maemo documentation, http://maemo.org/development/documentation/how-tos/4-x/maemo_architecture.html. [1.1.2008]
Mar07	Martin, B., Ajax Timelines and the Semantic Web. *Linux Journal*, 2007,153(January 2007), page 8. [Also available from http://www.linuxjournal.com/article/9301 1.1.2008].

Mic05 Microsoft Corporation, Secure Code. Managing a Secure IIS 6.0 Solution, IIS 6.0 Technical Reference, Microsoft Windows Server TechCenter, Microsoft TechNet, 22.8.2005, http://technet2.microsoft.com/windowsserver/en/library/d07fa94f-2b2c-4b31-9545-c97f4ec845b01033.mspx?mfr=true. [1.1.2008]

Mic07a Microsoft Corporation, VBScript User's Guide. MSDN technical document, http://msdn2.microsoft.com/en-us/library/sx7b3k7y.aspx. [1.1.2008]

Mic07b Microsoft Corporation, JavaScript (JScript/ECMAScript). MSDN technical document, http://msdn2.microsoft.com/en-us/office/aa905433.aspx. [1.1.2008]

Mic07c Microsoft Corporation, AJAX: The Official Microsoft ASP.NET Site, http://asp.net/ajax/. [1.1.2008]

Mic07d Microsoft Corporation, ASP.NET AJAX Control Toolkit samples, http://www.asp.net/ajax/ajaxcontroltoolkit/samples/. [1.1.2008]

Mol07 Moltzen, E., Mozilla Still Flummoxed by Firefox' Appetite for Memory. ChannelWeb network, blogs, 11.11.2007, http://www.crn.com/software/202804814. [1.1.2008]

Moz07 MozillaZine, Reducing memory usage – Firefox, http://kb.mozillazine.org/Reducing_memory_usage_-_Firefox. [1.1.2008]

Nie96 Nielsen, J., Why Frames Suck (Most of the Time). Jakob Nielsen's Alertbox, December 1996, http://www.useit.com/alertbox/9612.html. [1.1.2008]

Nie05 Nielsen, J., Forms vs. Applications. Jakob Nielsen's Alertbox, September 19th 2005, http://www.useit.com/alertbox/forms.html. [1.1.2008]

Nok06 Nokia Plc., Overview of AJAX Support in Nokia Web Browser v1.0 (English). Forum Nokia, 2006, http://www.forum.nokia.com/info/sw.nokia.com/id/1e66bc62-0a3e-4e36-b8c9-4e20e8b8cdd8/Overview_of_AJAX_Support_in_Nokia_Web_Browser_v1_0_en.pdf.html. [1.1.2008]

Nok07a Nokia Plc., Featured devices. The maemo platform, Forum Nokia, Platforms, http://www.forum.nokia.com/main/platforms/maemo/. [1.1.2008]

Nok07b Nokia Plc., S60WebKit. Projects, Open Source at Nokia, http://opensource.nokia.com/. [1.1.2008]

Nok07c Nokia Plc., Nokia N800 Internet Tablet Technical Specifications. Nokia Phones, http://www.nokiausa.com/A4779086. [1.1.2008]

Nok07d Nokia Plc., Widgets. Forum Nokia, Web Technologies, http://www.forum.nokia.com/main/resources/technologies/browsing/ widgets.html. [1.1.2008]

Nok07e Nokia Plc, S60WebKit FAQ. Open Source at Nokia, http://opensource.nokia.com/projects/S60browser/s60-oss-browser-faq.html. [1.1.2008]

Not06 Nottingham, M., The State of Browser Caching. mnot's Web log, 3.6.2006, http://www.mnot.net/blog/2006/05/11/browser_caching. [1.1.2008]

NVD07 National Vulnerability Database, CVE and CCE Vulnerability Database Advanced Search. National Institute of Standards and Technology, 2007, http://nvd.nist.gov/nvd.cfm?advancedsearch. [1.1.2008]

Oll07 Ollman, G., HTML Code Injection and Cross-site scripting - Understanding the cause and effect of CSS (XSS) Vulnerabilities. Technical Info, whitepapers, http://www.technicalinfo.net/papers/CSS.html. [1.1.2008]

Ols07 Olsson, T., Graceful Degradation & Progressive Enhancement. Accessites.org, 6.2.2007, http://accessites.org/site/2007/02/graceful-degradation-progressive-enhancement/1/. [1.1.2008]

Ope07a Opera Software ASA, Opera Mobile. Opera's web site, http://www.opera.com/products/mobile/. [1.1.2008]

Ope07b Opera Software ASA, Opera Mobile™ 8.65 for Windows Mobile now available for Smartphone and Pocket PC enthusiasts, handset OEMs and mobile carriers. Opera's web site, press release, http://www.opera.com/pressreleases/en/2007/10/01/. [1.1.2008]

Par07 Parmenter, S., Memory fragmentation. Pavlov.net, 10.11.2007, http://blog.pavlov.net/2007/11/10/memory-fragmentation/. [1.1.2008]

Pav06 PavingWays, XML 2006 Events, Mobile Event Finder, http://pwmwa.com/xml06/. [1.1.2008]

Pay07 Payne, L., What is a Toolkit? Dojo Toolkit Documentation, The Dojo Book 0.4, Introduction, 1.9.2007, http://dojotoolkit.org/book/dojo-book-0-4/part-1-introduction/what-toolkit. [1.1.2008]

PCT07 Prototype Core Team, Prototype JavaScript framework, http://www.prototypejs.org/. [1.1.2008]

PHP07 The PHP Group, PHP: Hypertext Preprocessor, http://www.php.net/. [1.1.2008]

Pro07 Proietti, V., MooTools – the compact javascript framework, http://www.mootools.net/. [1.1.2008]

Pup07a Pupius, D., Code changes to prepare Gmail for the future. The Official Gmail blog, 29.10.2007, http://gmailblog.blogspot.com/2007/10/code-changes-to-prepare-gmail-for.html. [1.1.2008]

Pup07b Pupius, D., Garbage Collection in IE6, 7.3.2007, http://pupius.co.uk/log/2007-03-07/. [1.1.2008]

Raa07 Raatikainen, K., Mobile Ajax. Position paper, W3C/OpenAjax Alliance Workshop on Mobile Ajax (Microsoft Silicon Valley campus, Mountain View, California, USA, 28.9.2007), http://www.w3.org/2007/06/mobile-ajax/papers/hiit.raatikainen.MobileAjaxPositionPaper.pdf. [1.1.2008]

Rah06 Rahman, R., innerHTML vs createElement. Cute solutions and something else..., 4.7.2006, http://cute-solutions.blogspot.com/2006/07/innerhtml-vs-createelement.html. [1.1.2008]

RHJ99 Raggett, D., Le Hors, A., Jacobs. I., HTML 4.01 Specification. W3C Recommendation 24.12.1999, http://www.w3.org/TR/html401/. [1.1.2008]

Rob07 Roberts, E., Resurrecting the Applet Paradigm. *Proc. 38th SIGCSE technical symposium on Computer science education* (SIGCSE'07, 7.-10.3.2007, Covington, Kentucky, USA), ACM Press, New York, NY, USA, pages 521-525.

Rud01 Ruderman, J., The Same Origin Policy. JavaScript Security, mozilla.org, 24.8.2001, http://www.mozilla.org/projects/security/components/same-origin.html. [1.1.2008]

Rus06 Russell, A., Comet: Low Latency Data for the Browser. Continuing Intermittent Incoherency, 23.3.2006, http://alex.dojotoolkit.org/?p=545. [1.1.2008]

Rus07a Russell, A., *et al.*, Bayeux Protocol – Bayeux 1.0draft0. Dojo Foundation, 2007, http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html. [1.1.2008]

Rus07b Russell, A., The Browser.Next list. Continuing Intermittent Incoherency, 12.9.2007, http://alex.dojotoolkit.org/?p=623. [1.1.2008]

RSS07 RSS Advisory Board, RSS 2.0 Specification (version 2.0.10), http://www.rssboard.org/rss-specification. [1.1.2008]

S6007 S60 3rd Edition, S60.com, http://www.s60.com/business/productinfo/ softwareversions/3rdedition. [1.1.2008]

Sco06a Scott, L., Ajax: Home Was Never Like This! Musings from Mars, 9.2.2006, http://www.musingsfrommars.org/2006/02/ajax-home-was-never-like-this.html. [1.1.2008]

Sco06b Scott, L., Ajax/DHTML Library Scorecard: How Cross Platform Are They? Musings from Mars, 4.3.2006, http://www.musingsfrommars.org/2006/03/ ajax-dhtml-library-scorecard.html. [1.1.2008]

scr07 script.aculo.us, http://script.aculo.us/. [1.1.2008]

Sec07 Section 508, The Road to Accessibility, http://www.Section508.gov/. [1.1.2008]

She04 Shea, D., CSS Sprites: Image Slicing's Kiss of Death. A list apart, number 173, 5.3.2004, http://alistapart.com/articles/sprites. [1.1.2008]

ShM97 Shubin, H., Meehan, M., Navigation in Web Applications. *Interactions*, vol. 4, issue 6 (Nov./Dec. 1997), pages 13-17.

SIM07 SIMILE project, Timeline, http://simile.mit.edu/timeline/. [1.1.2008]

Sky07 Skype Ltd., Skype official website, http://www.skype.com/intl/en/. [1.1.2008]

Smi06 Smith, K., Simplifying Ajax-Style Web Development. *Computer*, 39,5(May 2006), pages 98-101.

SmS07 Smullen, C. W., Smullen, S. A., AJAX application server performance. *SoutheastCon, 2007. Proc. IEEE*, March 2007, pages 154-158.

Sou07 Souders, S., High Performance Web Sites, http://stevesouders.com/hpws/. [1.1.2008]

StH04 Stenback, J., Heninger, A., DOM Level 3 Load and Save Specification 1.0. W3C Recommendation 07-April-2004, http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407. [1.1.2008]

Sri01 Srinivan, R. N., Java Web Start to the rescue. JavaWorld.com, 7.6.2001, http://www.javaworld.com/javaworld/jw-07-2001/jw-0706-webstart.html. [1.1.2008]

StS02 Stein, L. D., Stewart, J. N., The World Wide Web Security FAQ, version 3.1.2. W3C, 4.2.2002, http://www.w3.org/Security/Faq/. [1.1.2008]

Sun07 Sun Microsystems, Mobile Information Device Profile (MIDP); JSR 37, JSR118, http://java.sun.com/products/midp/. [1.1.2008]

Thc07 Thiessen, P., Chen, C., Ajax live regions: chat as a case example. *Proc. 2007 international cross-disciplinary conference on Web accessibility* (W4A'07, 7.-8.3.2007, Banff, Canada), ACM, New York, NY, USA, pages 7-14.

The06 Theurer, T., Performance Research, Part 1: What the 80/20 Rule Tells Us about Reducing HTTP Requests. Yahoo! User Interface Blog, 28.11.2006, http://yuiblog.com/blog/2006/11/28/performance-research-part-1/. [1.1.2008]

The07a Theurer, T., Performance Research, Part 4: Maximizing Parallel Downloads in the Carpool Lane. Yahoo! User Interface Blog, 11.4.2007, http://yuiblog.com/blog/2007/04/11/performance-research-part-4/. [1.1.2008]

The07b Theurer, T., Performance Research, Part 2: Browser Cache Usage – Exposed! Yahoo! User Interface Blog, 4.1.2007, http://yuiblog.com/blog/2007/01/04/performance-research-part-2/. [1.1.2008]

Tok07 Toker, A., WebKit Maemo port, N800 and the EAL. Alp Toker's blog, 7.8.2007, http://www.atoker.com/blog/2007/08/07/webkit-maemo-port-n800-and-the-eal/. [1.1.2008]

UMT07 UMTSWorld.com, 3G and UMTS Frequently Asked Questions, http://www.umtsworld.com/umts/faq.htm. [1.1.2008]

VeF05 Venkatachalam, V., Franz, M., Power reduction techniques for microprocessor systems. *ACM Computing Surveys* (CSUR), 37,3(September 2005), pages 195-237.

VRP05 Vacirca, F., Ricciato, F., Pilz, R., Large-scale RTT measurements from an operational UMTS/GPRS network. *Proc. First International Conference on Wireless Internet, 2005* (WICON'05), 10.-14.7.2005, pages 190-197.

W3C03 World Wide Web Consortium, Some early ideas for HTML. http://www.w3.org/MarkUp/historical. [1.1.2008]

Wal07 Walker, J., CSRF Attacks or How to avoid explosing your GMail contacts. Joe Walker's Blog, 1.1.2007, http://getahead.org/blog/joe/2007/01/01/csrf_attacks_or_how_to_avoid_exposing_your_gmail_contacts.html. [1.1.2008]

Web05a Webber, J., Mapping Google. as simple as possible, but no simpler blog, 9.2.2005, http://jgwebber.blogspot.com/2005/02/mapping-google.html. [1.1.2008]

Web05b Webber, J., Making the Back Button Dance. as simple as possible, but no simpler blog, 12.2.2005, http://jgwebber.blogspot.com/2005/02/making-back-button-dance.html. [1.1.2008]

Web06 Webb, D., The JavaScript Library World Cup. SitePoint, JavaScript & Ajax Tutorials, 14.6.2006, http://www.sitepoint.com/article/javascript-library/. [1.1.2008]

Web07a The WebKit Open Source Project, http://webkit.org/. [1.1.2008]

Web07b Webtide, Acme Auctions auction demonstration site, http://www.webtide.com/auctiondemo/. [1.1.2008]

Web07c Webtide, the Ajax and Comet Server, http://www.webtide.com/. [1.1.2008]

Wei06 Weiss, A., The Web Designer's Dilemma: When Standards and Practice Diverge. *netWorker* 10,1(March 2006), pages 18-25.

Whi05 White, A., Measuring the benefits of Ajax. Developer.com, XML Articles, October 2005, http://www.developer.com/xml/article.php/10929_3554271_1. [1.1.2008]

Wik07a Wikipedia, ECMAScript, http://en.wikipedia.org/wiki/ECMAScript. [1.1.2008]

Wik07b Wikipedia, Cross-site scripting, http://en.wikipedia.org/wiki/XSS. [1.1.2008]

Wil03 Wilson, B., Browser Timelines, 2003, http://www.blooberry.com/indexdot/history/browsers.htm. [1.1.2008]

W-J07a Wilton-Jones, M., The history of JavaScript. http://www.howtocreate.co.uk/jshistory.html. [1.1.2008]

W-J07b Wilton-Jones, M., XML importing script. http://www.howtocreate.co.uk/tutorials/jsexamples/importingXML.html. [1.1.2008]

W-J07c Wilton-Jones, M., Communicating with the server. http://www.howtocreate.co.uk/loadingExternalData.html. [1.1.2008]

W-J07d Wilton-Jones, M., Safari and page load timing. http://www.howtocreate.co.uk/safaribenchmarks.html. [1.1.2008]

W-J07e Wilton-Jones, M., Browser speed comparisons. http://www.howtocreate.co.uk/browserSpeed.html. [1.1.2008]

Wri07 Wright, G., WebKit and the N800. George Wright's Blog, 7.8.2007, http://blog.gwright.org.uk/articles/2007/08/07/webkit-and-the-n800. [1.1.2008]

Yan07 Yank, K., Simply JavaScript: The Three Layers of the Web. SitePoint, JavaScript & Ajax Tutorials, 27.7.2007, http://www.sitepoint.com/article/simply-javascript/. [1.1.2008]

Yah07a Yahoo! Inc., Graded Browser Support. Yahoo! Developer Network, http://developer.yahoo.com/yui/articles/gbs/index.html. [1.1.2008]

Yah07b Yahoo! Inc., The Yahoo! User Interface Library (YUI). Yahoo! Developer Network, http://developer.yahoo.com/yui/. [1.1.2008]

Yah07c Yahoo! Inc., Flickr, http://flickr.com/. [1.1.2008]

Yah07d Yahoo! Inc., Flickr: Mobile Tools, http://www.flickr.com/tools/mobile/. [1.1.2008]

Yah07e Yahoo! Inc., Best Practices for Speeding Up Your Web Site. Yahoo! Developer Network, http://developer.yahoo.com/performance/rules.html. [1.1.2008]

Yah07f Yahoo! Inc., Yahoo! Mail, http://mail.yahoo.com/. [1.1.2008]

You07 YouTube, http://www.youtube.com/. [1.1.2008]

YTV07 Pääkaupunkiseudun yhteistyövaltuuskunta YTV, Journey Planner for Cycling, http://kevytliikenne.ytv.fi/?lang=en. [1.1.2008]

Yu06 Yu, J., *et al.*, OpenXUP: an Alternative Approach to Developing Highly Interactive Web Applications. *Proc. 6th International conference on Web engineering* (11.-14.7.2006, Palo Alto, California, USA), ACM, New York, NY, USA 2006, pages 289-296. [Also available from http://www.cse.unsw.edu.au/~jyu/icwe06/fp697-yu.pdf, 1.1.2008].

Appendix 1. Test notes

Appendix 1 contains my notes from the capability tests and performance measurements. The notes have been gone through only marginal editing, as the intent is only to provide the numerical results for additional analysis. Each case contains a time stamp in order to make it possible for developers to verify the actual versions of the target applications. Toolkit versions have been recorded if they were available in the source code. Results marked with a red color signal that the browser's capability test was rated as a failure. Additional iterations have been performed during selected tests. Reasons for the extra observations are recorded in the notes. The usual reason is a search for memory leaks.

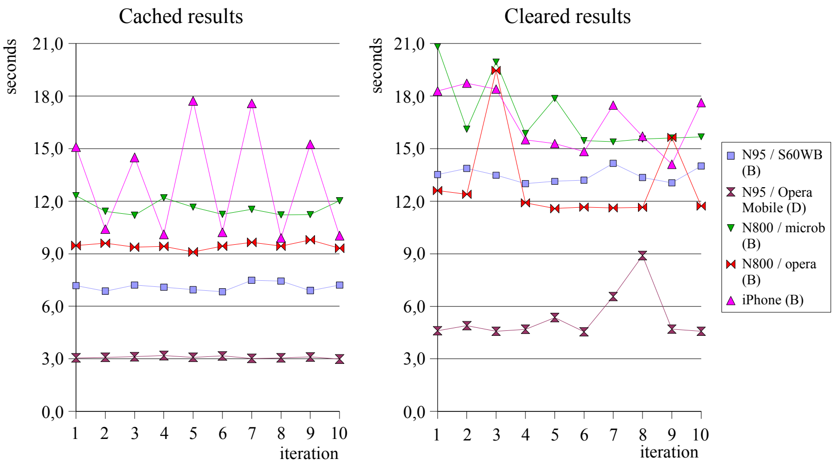
Name	PWC-OS demo application
Category	prototype + window extension
Version(s)	prototype 1.5.1_rc3, script.aculo.us effects.js v1.7.1_beta1, window v1.3, script.aculo.us dragdrop.js v1.7.1_beta1
URL	http://prototype-window.xilinus.com/PWC-OS
Test description	loading time, measurement ends when windows and background are visible, border shadows drawn last.
test begun	29.10.07 18:13
test ended	29.10.07 20:44

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
0	N95 / S60 Web Browser cached	7,2	6,9	7,2	7,1	6,9	6,8	7,5	7,4	6,9	7,2	7,12	7,14	0,23
1	N95 / Opera Mobile cached	3,1	3,1	3,1	3,2	3,1	3,2	3,0	3,1	3,1	3,0	3,09	3,09	0,06
2	N800 / microb cached	12,3	11,4	11,2	12,2	11,7	11,2	11,5	11,2	11,2	12,0	11,6	11,48	0,43
3	N800 / opera cached	9,5	9,6	9,4	9,4	9,1	9,4	9,6	9,4	9,8	9,3	9,46	9,44	0,19
4	iPhone cached	15,1	10,4	14,5	10,1	17,7	10,2	17,6	9,9	15,2	10,0	13,08	12,45	3,27
5	N95 / S60WB (B)	13,5	13,9	13,5	13,0	13,1	13,2	14,2	13,4	13,1	14,0	13,48	13,42	0,41
6	N95 / Opera Mobile (D)	4,6	4,9	4,6	4,7	5,4	4,5	6,6	8,9	4,7	4,6	5,34	4,7	1,39
7	N800 / microb (B)	20,8	16,1	19,9	15,9	17,9	15,5	15,4	15,6	15,6	15,7	16,83	15,77	2,01
8	N800 / opera (B)	12,6	12,4	19,5	11,9	11,6	11,7	11,6	11,6	15,6	11,7	13,03	11,81	2,58
9	iPhone (B)	18,3	18,7	18,4	15,5	15,3	14,8	17,5	15,7	14,1	17,6	16,6	16,6	1,68

Notes

ref id nota bene

all at test start, browser cache is cleared. after initial loading, app is tested for defects.
0 Task bar is positioned in the middle of the screen. D&D and all else works. (MINOR)
1 Does not render the windows at all. (FAILURE)
2 Everything works, but extremely slowly. (MINOR)
3 No D&D available, buttons work. (MINOR)
4 No D&D available, buttons work, desktop dimensions too large. (MINOR)
4 Note the variance in results

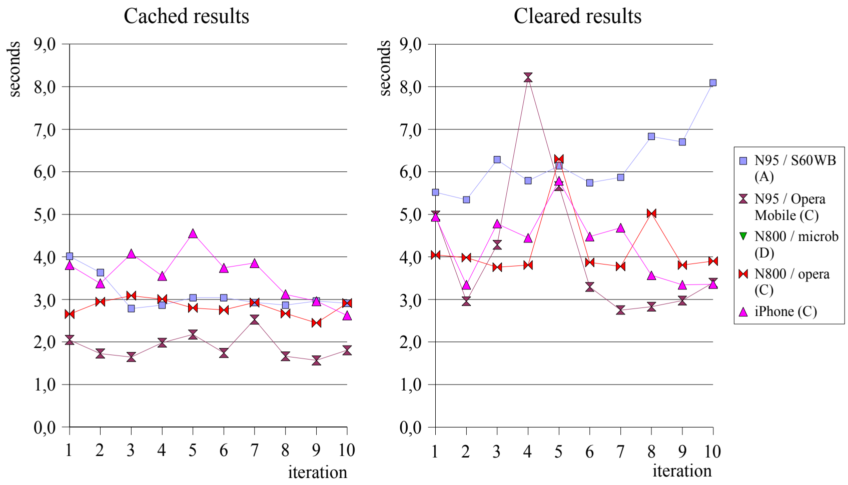


Name	script.aculo.us shopping cart
Category	prototype + script.aculo.us
Version(s)	prototype 1.3.0
URL	http://demo.script.aculo.us/shop
Test description	loading time, test ends when all elements have loaded.
test begun	29.10.07 21:42
test ended	29.10.07 22:51

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
10	N95 / S60 Web Browser cached	4,0	3,6	2,8	2,9	3,0	3,0	2,9	2,9	3,0	2,9	3,1	2,94	0,4
11	N95 / Opera Mobile cached	2,0	1,7	1,6	2,0	2,2	1,7	2,5	1,7	1,6	1,8	1,89	1,78	0,3
12	N800 / microb cached													
13	N800 / opera cached	2,7	2,9	3,1	3,0	2,8	2,8	2,9	2,7	2,4	2,9	2,82	2,86	0,19
14	iPhone cached	3,8	3,4	4,1	3,6	4,6	3,7	3,9	3,1	3,0	2,6	3,57	3,65	0,57
15	N95 / S60WB (A)	5,5	5,3	6,3	5,8	6,1	5,7	5,9	6,8	6,7	8,1	6,23	6,01	0,81
16	N95 / Opera Mobile (C)	5,0	3,0	4,3	8,2	5,7	3,3	2,8	2,8	3,0	3,4	4,14	3,34	1,74
17	N800 / microb (D)													
18	N800 / opera (C)	4,0	4,0	3,8	3,8	6,3	3,9	3,8	5,0	3,8	3,9	4,23	3,89	0,82
19	iPhone (C)	4,9	3,3	4,8	4,4	5,8	4,5	4,7	3,6	3,3	3,4	4,27	4,46	0,84

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
10	renders correctly. d&d possible, but tricky due to 4-way nav button with selector in middle (PASSED)
11	d&d not possible, some layout bugs (MAJOR)
12	browser crashes, measurement impossible (FAILURE)
13	d&d not possible, some layout bugs (MAJOR)
14	d&d not possible. (MAJOR)

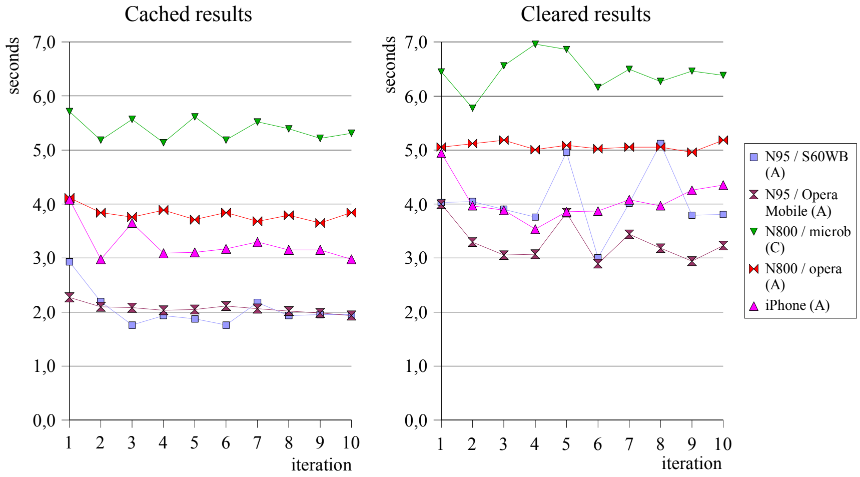


Name	15days of jquery Edit-in-place 2
Category	jQuery
Version(s)	
URL	http://15daysofjquery.com/examples/jqueryEditInPlace/demo.php
Test description	loading time, test ends when text has stopped flowing
test started	30.10.07 09:13
test finished	30.10.07 10:45

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
20	N95 / S60 Web Browser cached	2,9	2,2	1,8	1,9	1,9	1,8	2,2	1,9	2,0	2,0	2,05	1,94	0,34
21	N95 / Opera Mobile cached	2,3	2,1	2,1	2,0	2,0	2,1	2,1	2,0	2,0	1,9	2,06	2,06	0,09
22	N800 / microb cached	5,7	5,2	5,6	5,1	5,6	5,2	5,5	5,4	5,2	5,3	5,38	5,35	0,21
23	N800 / opera cached	4,1	3,8	3,8	3,9	3,7	3,8	3,7	3,8	3,6	3,8	3,81	3,82	0,13
24	iPhone cached	4,1	3,0	3,6	3,1	3,1	3,2	3,3	3,2	3,2	3,0	3,26	3,15	0,34
25	N95 / S60WB (A)	4,0	4,0	3,9	3,8	5,0	3,0	4,0	5,1	3,8	3,8	4,04	3,96	0,6
26	N95 / Opera Mobile (A)	4,0	3,3	3,1	3,1	3,8	2,9	3,4	3,2	2,9	3,2	3,3	3,21	0,37
27	N800 / microb (C)	6,4	5,8	6,6	7,0	6,9	6,2	6,5	6,3	6,5	6,4	6,44	6,46	0,34
28	N800 / opera (A)	5,1	5,1	5,2	5,0	5,1	5,0	5,1	5,1	5,0	5,2	5,07	5,06	0,07
29	iPhone (A)	4,9	4,0	3,9	3,5	3,9	3,9	4,1	4,0	4,3	4,4	4,07	3,97	0,38

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
20	No defects found (PASSED)
21	No defects found (PASSED)
22	After saving and receiving the alert, only the saved paragraph is shown. (MAJOR)
23	No defects found (PASSED)
24	No defects found (PASSED)

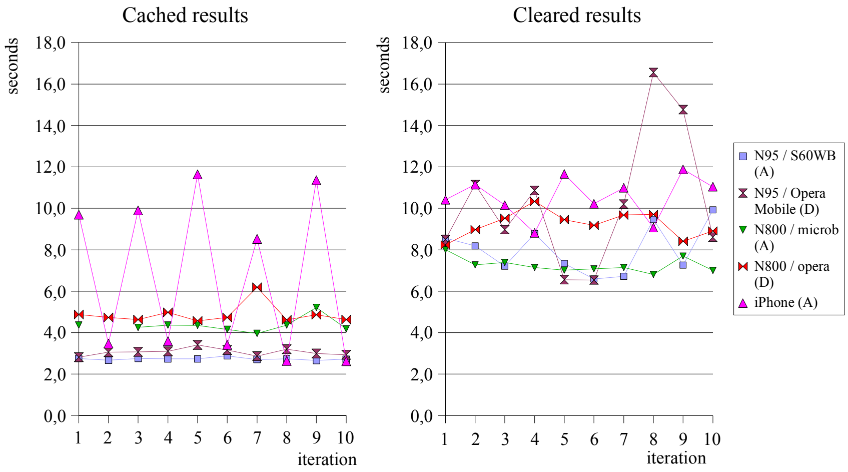


Name	Dynamically loading TreeView
Category	YUI
Version(s)	YUI, Event, Connection, Treeview, 2.3.1
URL	http://developer.yahoo.com/yui/examples/treeview/dynamic_tree_clean.html
Test description	loading time, test ends whentext has stopped flowing or throbber stops (whichever last)
test started	30.10.07 11:15
test finished	30.10.07 12:36

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
30	N95 / S60 Web Browser cached	2,8	2,7	2,8	2,7	2,7	2,9	2,7	2,7	2,7	2,7	2,73	2,74	0,06
31	N95 / Opera Mobile cached	2,8	3,1	3,1	3,1	3,4	3,2	2,9	3,2	3,0	2,9	3,06	3,06	0,17
32	N800 / microb cached	4,4		4,3	4,4	4,4	4,2	4,0	4,4	5,2	4,19	4,36	4,35	0,35
33	N800 / opera cached	4,9	4,7	4,6	5,0	4,6	4,7	6,2	4,6	4,9	4,6	4,89	4,74	0,48
34	iPhone cached	9,7	3,5	9,9	3,6	11,6	3,4	8,5	2,6	11,4	2,6	6,69	6,06	3,84
35	N95 / S60WB (A)	8,6	8,2	7,2	8,8	7,3	6,6	6,7	9,5	7,3	9,9	8,01	7,77	1,16
36	N95 / Opera Mobile (D)	8,5	11,2	9,0	10,9	6,6	6,5	10,2	16,6	14,8	8,6	10,27	9,59	3,27
37	N800 / microb (A)	8,0	7,3	7,4	7,2	7,0	7,1	7,2	6,8	7,7	7,0	7,26	7,15	0,36
38	N800 / opera (D)	8,2	9,0	9,5	10,3	9,5	9,2	9,7	9,7	8,4	8,9	9,24	9,32	0,63
39	iPhone (A)	10,4	11,2	10,2	8,8	11,7	10,2	11,0	9,1	11,9	11,0	10,54	10,7	1,02

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
30	No defects detected (PASSED)
31	Does not render tree lines, clicking on visible links engages pacifier but no results (FAILURE)
33	Does not render tree lines, clicking on visible links engages pacifier but no results (FAILURE)
33	Throbber starts twice, items remaining shows 1-10 of 10 and then 11-13 of 13.
32	No defects detected (PASSED)
32	On second iteration, browser crashed. Time was as follows 7,2
30&31	Noticed that N95 rebooted by itself after running these and test 33, during test 32. Probably unrelated to testing.
34	No defects detected (PASSED)
34	Note the variance.
33&38	Does not always render even the first-level links. Seems to be timing-oriented, sometimes loads on second (cached) try.

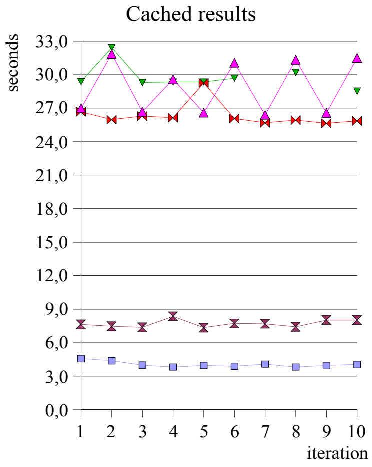


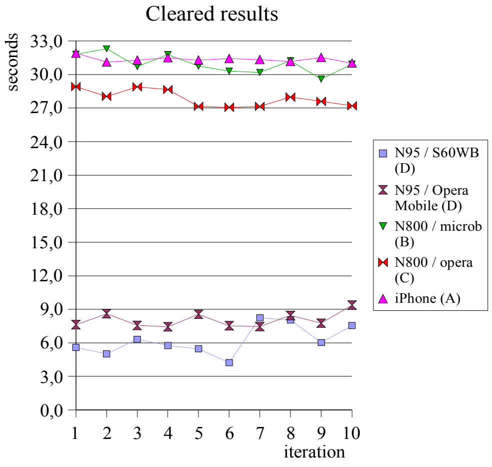
Name	Demo Mail Application
Category	dojo toolkit
Version(s)	dojo 0.9.0
URL	http://www.cs.helsinki.fi/u/pervila/dojo-release-0.9.0/dijit/demos/mail.html
Test description	loading time, test ends when layout has stopped flowing or throbber stops (whichever last)
test started	30.10.07 14:43
test finished	30.10.07 17:25

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
40	N95 / S60 Web Browser cached	4,6	4,4	4,0	3,8	4,0	3,9	4,1	3,8	4,0	4,0	4,1	4,0	0,24
41	N95 / Opera Mobile cached	7,6	7,5	7,4	8,4	7,3	7,7	7,7	7,4	8,0	8,0	7,7	7,7	0,34
42	N800 / microb cached	29,4	32,4	29,3	29,3	29,3	29,7		30,2		28,5	29,8	29,4	1,17
43	N800 / opera cached	26,7	26,0	26,3	26,1	29,3	26,1	25,7	25,9	25,6	25,9	26,4	26,0	1,06
44	iPhone cached	27,0	31,9	26,7	29,6	26,6	31,1	26,4	31,3	26,6	31,5	28,9	28,3	2,41
45	N95 / S60WB (D)	5,6	5,0	6,3	5,8	5,5	4,2	8,2	8,0	6,0	7,6	6,2	5,9	1,33
46	N95 / Opera Mobile (D)	7,6	8,6	7,6	7,4	8,6	7,5	7,5	8,4	7,8	9,4	8,0	7,7	0,66
47	N800 / microb (B)	31,8	32,3	30,7	31,8	30,7	30,3	30,2	31,2	29,6	30,9	31,0	30,8	0,83
48	N800 / opera (C)	28,9	28,0	28,9	28,7	27,1	27,1	27,1	28,0	27,6	27,2	27,9	27,8	0,75
49	iPhone (A)	31,9	31,1	31,3	31,5	31,3	31,4	31,3	31,2	31,5	31,0	31,4	31,3	0,26

Notes

ref id	nota bene	avg cache diff
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.	2,17
all	the dojo email online test was recently broken, decided to run from the local web server	0,33
40	No message list, no layout, buttons dead. Application unusable (FAILURE)	1,19
41	No message list, no layout, buttons dead. Application unusable (FAILURE)	1,51
41	Throbber continues way past layout flow, no changes	2,51
42	Almost no defects. New message slow, autocomplete in to- or subject fields do not work. (MINOR)	
42	Fails to load application 4 times, refresh or cache clearing helps. (MAJOR)	
42	Browser crashed during 7. iteration, failed to load app during 9.	
43	Fails to display msg header list in any folder, New message broken (MAJOR)	
44	Initially: Options menu works, new message button visible, get mail works, everything else broken. (FAILURE)	
44	Subsequently, everything seems to work (PASSED)	
47&48	There doesn't seem to be any difference on when the cache is cleared, as long as its cleared outside of the app.	
42&43&44	Loading a message with the color picker is noticeably slow , 3~4 seconds	
all	This app requires some rendering: effect of caching the files diminishes into very small.	



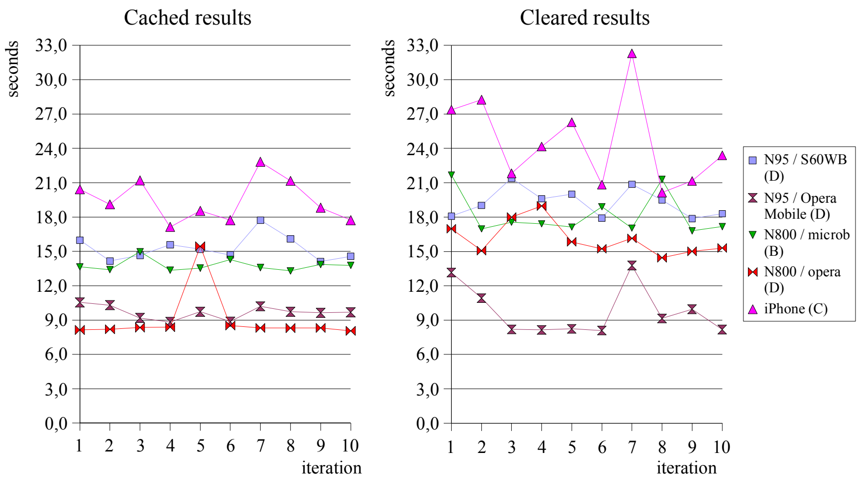


Name	XML Form
Category	Ext JS
Version(s)	Library 2.0 beta 1
URL	http://extjs.com/deploy/dev/examples/form/xml-form.html
Test description	Loading time. Test ends when layout has stopped flowing or throbber stops (whichever latest)
test started	30.10.07 18:38
test finished	30.10.07 21:20

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
50	N95 / S60 Web Browser cached	16,0	14,2	14,7	15,6	15,2	14,7	17,7	16,1	14,1	14,6	15,3	14,9	1,11
51	N95 / Opera Mobile cached	10,5	10,3	9,2	8,8	9,7	8,9	10,2	9,7	9,6	9,7	9,7	9,7	0,58
52	N800 / microb cached	13,6	13,4	15,0	13,4	13,5	14,3	13,6	13,3	13,9	13,8	13,8	13,6	0,51
53	N800 / opera cached	8,1	8,2	8,4	8,4	15,5	8,5	8,3	8,3	8,3	8,1	9,0	8,3	2,27
54	iPhone cached	20,4	19,1	21,2	17,1	18,5	17,7	22,8	21,2	18,8	17,7	19,5	19,0	1,86
55	N95 / S60WB (D)	18,1	19,0	21,4	19,6	20,0	17,9	20,9	19,5	17,9	18,3	19,3	19,3	1,24
56	N95 / Opera Mobile (D)	13,2	10,9	8,2	8,1	8,2	8,1	13,8	9,2	10,0	8,2	9,8	8,7	2,16
57	N800 / microb (B)	21,7	17,0	17,5	17,4	17,1	18,9	17,0	21,3	16,8	17,2	18,2	17,3	1,83
58	N800 / opera (D)	17,0	15,1	18,0	19,0	15,8	15,2	16,1	14,4	15,0	15,3	16,1	15,6	1,46
59	iPhone (C)	27,4	28,3	21,8	24,2	26,3	20,8	32,3	20,1	21,2	23,4	24,6	23,8	3,92

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
all	The JS is not “minified”, i.e., not compressed. This increases loading time.
50	Load button does not work. Consequently, form may not be submitted. Everything else works. (FAILURE)
51	Form was not displayed on two first attempts. Load button, pull-down menu, date picker do not work. (FAILURE)
52	Date picker is broken, state selector slow. (MINOR)
53	Load button does not work. Date picker works, but cannot be minimized. (FAILURE)
53	No explanation for the outlier.
54	Form is drawn too small. Both load and submit seem to work. (MAJOR)

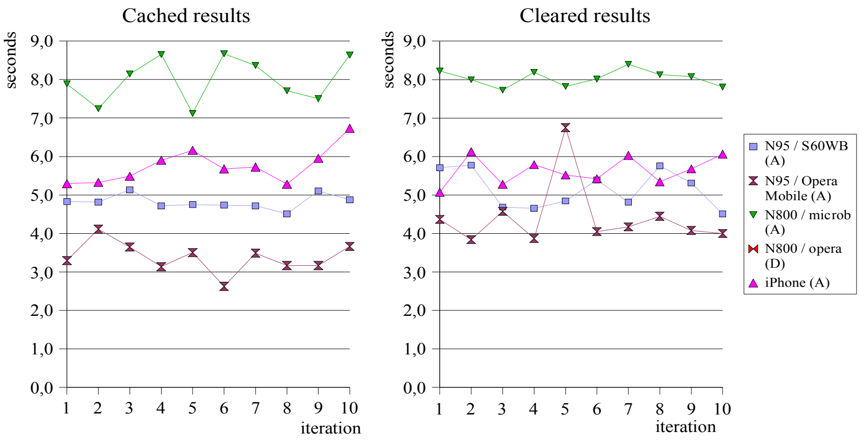


Name	Dynamic Table: School Schedule
Category	GWT
Version(s)	
URL	http://gwt.google.com/samples/DynaTable/DynaTable.html
Test description	Loading time. Test ends when layout has stopped flowing
test started	30.10.07 21:43
test finished	30.10.07 22:45

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
60	N95 / S60 Web Browser cached	4,8	4,8	5,1	4,7	4,8	4,7	4,7	4,5	5,1	4,9	4,8	4,8	0,19
61	N95 / Opera Mobile cached	3,3	4,1	3,6	3,1	3,5	2,6	3,5	3,2	3,2	3,7	3,4	3,4	0,4
62	N800 / microb cached	7,9	7,2	8,1	8,7	7,1	8,7	8,4	7,7	7,5	8,6	8,0	8,0	0,59
63	N800 / opera cached													
64	iPhone cached	5,3	5,3	5,5	5,9	6,2	5,7	5,7	5,3	6,0	6,7	5,8	5,7	0,46
65	N95 / S60WB (A)	5,7	5,8	4,7	4,7	4,8	5,4	4,8	5,8	5,3	4,5	5,1	5,1	0,5
66	N95 / Opera Mobile (A)	4,4	3,8	4,6	3,9	6,8	4,0	4,2	4,4	4,1	4,0	4,4	4,1	0,86
67	N800 / microb (A)	8,2	8,0	7,7	8,2	7,8	8,0	8,4	8,1	8,1	7,8	8,0	8,0	0,21
68	N800 / opera (D)													
69	iPhone (A)	5,1	6,1	5,3	5,8	5,5	5,4	6,0	5,3	5,7	6,1	5,6	5,6	0,36

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
60	throbber get stuck at 78 KB, clears when arrows are pressed, no other defects found (PASSED)
61	no defects found (PASSED)
60&61	requires double press of back button. Due to IFrame?
62	no defects found (PASSED)
63&68	displays no table or controls (FAILURE)
64	no defects found (PASSED)

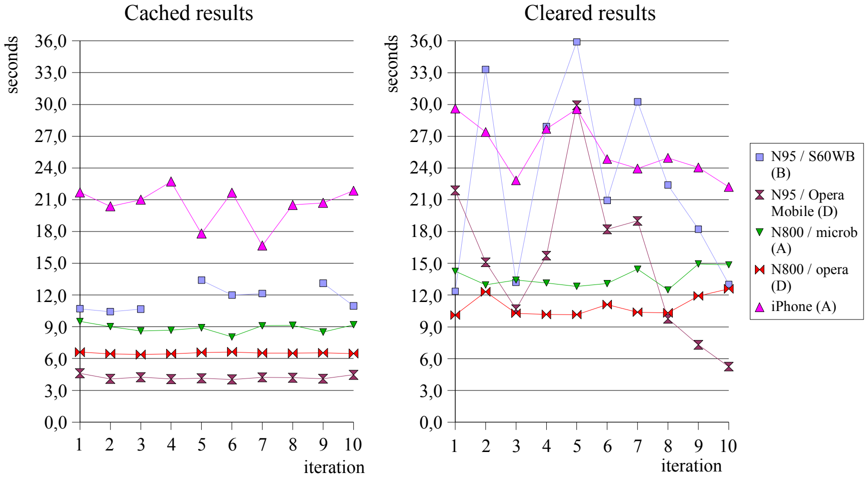


Name	Webtide Acme Auctions
Category	DWR + reverse ajax / comet
Version(s)	DWR, scriptaculous, prototype, dojo
URL	http://www.webtide.com/auctiondemo/
Test description	loading time, ends when text has stopped flowing (throbber sometimes shows 0 KB)
test started	31.10.07 10:06
test finished	31.10.07 14:18

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
70	N95 / S60 Web Browser cached	10,7	10,4	10,7		13,4	12,0	12,1		13,1	11,0	11,7	11,5	1,16
71	N95 / Opera Mobile cached	4,6	4,1	4,2	4,1	4,1	4,0	4,2	4,2	4,1	4,5	4,2	4,2	0,19
72	N800 / microb cached	9,5	9,0	8,6	8,7	8,9	8,0	9,1	9,1	8,5	9,2	8,9	9,0	0,42
73	N800 / opera cached	6,6	6,4	6,4	6,4	6,6	6,6	6,5	6,5	6,5	6,5	6,5	6,5	0,08
74	iPhone cached	21,7	20,4	21,0	22,7	17,8	21,7	16,7	20,5	20,7	21,9	20,5	20,9	1,88
75	N95 / S60WB (B)	12,4	33,3	13,2	27,9	35,9	20,9	30,3	22,4	18,2	13,0	22,8	21,7	8,73
76	N95 / Opera Mobile (D)	21,9	15,1	10,7	15,7	29,9	18,2	19,0	9,8	7,3	5,2	15,3	15,4	7,43
77	N800 / microb (A)	14,3	13,0	13,4	13,2	12,8	13,1	14,4	12,5	14,9	14,9	13,6	13,3	0,9
78	N800 / opera (D)	10,1	12,3	10,3	10,2	10,2	11,1	10,4	10,3	11,9	12,6	10,9	10,4	0,98
79	iPhone (A)	29,6	27,4	22,8	27,7	29,6	24,8	23,9	25,0	24,0	22,2	25,7	24,9	2,68

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
70	with continuous bids, connection stays open for at least 120 seconds
70	without, 30 seconds ok, 60 seconds ok, 90 seconds ok IF the browser is “watched”
70	if not, screen saver kills the connection. If screen saver is avoided by regular clicks, comet works after 10 minutes.
70	if connection is killed, subsequent updates, even from the client, are unable to touch the amount field (MINOR)
70	browser crashed after iteration 4 and iteration 8 and iteration 12. Seems to be a pattern: 1 uncached + 4 cached.
70	results for 4 and 8 are as follows 10,6 10,6
71	Note that 5 and 9 are cached results.
72	login button broken (FAILURE)
73	App was left running by itself for >10 minutes, comet still works after that. (PASSED)
74	login button broken (FAILURE)
75	App was left running by itself for >10 minutes, comet still works after that. (PASSED)
75&76	browser crashes consistently if back button is pressed before doing app login
	high variation probably due to network conditions (deduced from loader progress)

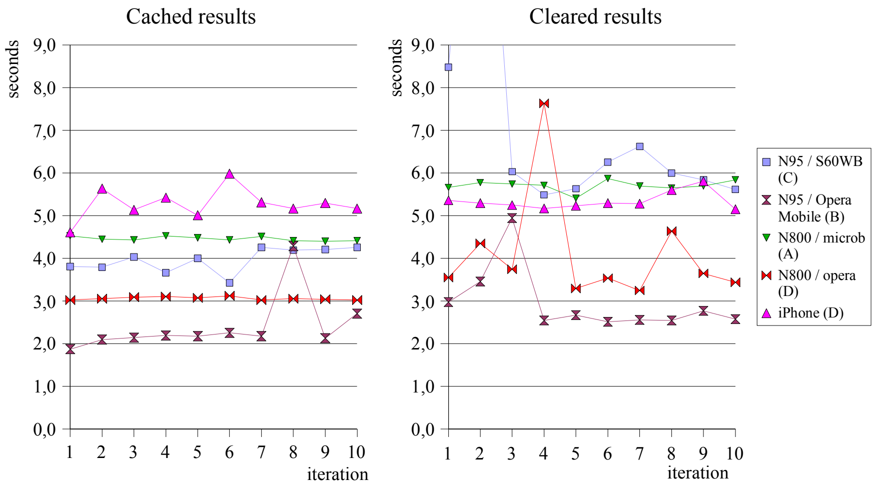


Name	Harald's HistoryManager 1.0rc2
Category	Mootools / moo.fx
Version(s)	mootools “1.2/dev” (modified 2.8.2007)
URL	http://digitarald.de/playground/history.html
Test description	loading time, ends when text has stopped flowing, “start” visible
test started	31.10.07 21:52
test finished	31.10.07 23:17

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
90	N95 / S60 Web Browser cached	3,8	3,8	4,0	3,7	4,0	3,4	4,3	4,2	4,2	4,3	4,0	4,0	0,28
91	N95 / Opera Mobile cached	1,9	2,1	2,1	2,2	2,2	2,3	2,2	4,3	2,1	2,7	2,4	2,2	0,69
92	N800 / microb cached	4,5	4,4	4,4	4,5	4,5	4,4	4,5	4,4	4,4	4,4	4,5	4,4	0,05
93	N800 / opera cached	3,0	3,1	3,1	3,1	3,1	3,1	3,0	3,1	3,0	3,0	3,1	3,1	0,03
94	iPhone cached	4,6	5,6	5,1	5,4	5,0	6,0	5,3	5,2	5,3	5,2	5,3	5,2	0,37
95	N95 / S60WB (C)	8,5	16,4	6,0	5,5	5,6	6,3	6,6	6,0	5,8	5,6	7,2	6,0	3,34
96	N95 / Opera Mobile (B)	3,0	3,5	4,9	2,5	2,7	2,5	2,6	2,5	2,8	2,6	3,0	2,6	0,76
97	N800 / microb (A)	5,7	5,8	5,7	5,7	5,4	5,9	5,7	5,6	5,7	5,8	5,7	5,7	0,13
98	N800 / opera (D)	3,6	4,4	3,7	7,6	3,3	3,5	3,2	4,6	3,6	3,4	4,1	3,6	1,32
99	iPhone (D)	5,4	5,3	5,2	5,2	5,2	5,3	5,3	5,6	5,8	5,2	5,3	5,3	0,21

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
90	lists and accordion work, content 1,2,3,4 -links consistently crash browser, back causes screen redraw (MAJOR)
91	lists work, accordion does not compress but correct text shows, content-links work, back causes redraw (MINOR)
92	No defects detected (PASSED)
93	Contents visible, accordion compresses. All links cause redraw, back button causes redraw but no change. (FAILURE)
94	All links work, but also cause screen redraws. Back causes redraw (FAILURE)

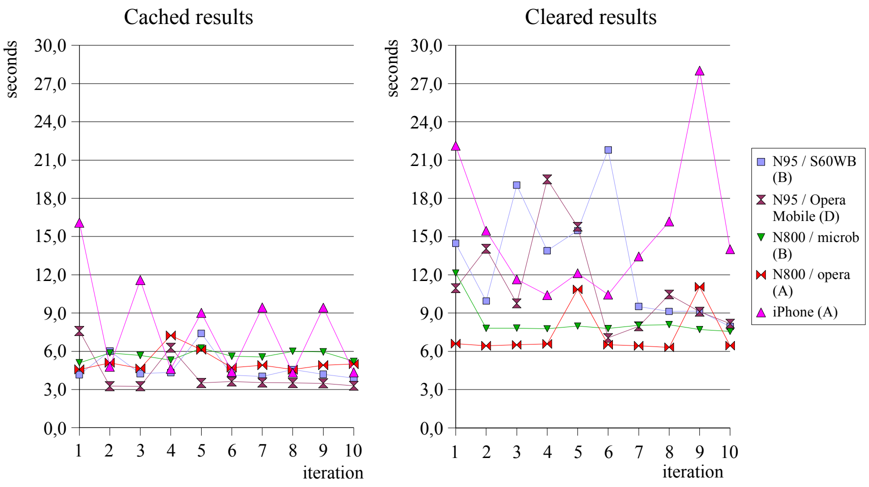


Name	Asset.images
Category	Mootools / moo.fx
Version(s)	mootools v1.11/SVN,
URL	http://demos.mootools.net/Asset.images
Test description	loading time, ends when text has stopped flowing, “start” visible
test started	31.10.07 15:48
test finished	31.10.07 17:39

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
80	N95 / S60 Web Browser cached	4,2	6,0	4,3	4,3	7,4	4,1	4,0	4,6	4,2	3,9	4,7	5,6	1,12
81	N95 / Opera Mobile cached	7,6	3,3	3,3	6,3	3,5	3,6	3,6	3,5	3,5	3,3	4,1	3,5	1,51
82	N800 / microb cached	5,1	5,9	5,7	5,3	6,3	5,6	5,6	6,0	6,0	5,2	5,7	5,6	0,37
83	N800 / opera cached	4,6	5,1	4,7	7,2	6,1	4,7	4,9	4,6	4,9	5,0	5,2	4,9	0,86
84	iPhone cached	16,1	4,8	11,6	4,6	9,0	4,4	9,4	4,3	9,4	4,4	7,8	6,9	4,01
85	N95 / S60WB (B)	14,5	10,0	19,0	13,9	15,5	21,8	9,5	9,1	9,2	8,0	13,0	11,9	4,71
86	N95 / Opera Mobile (D)	11,0	14,0	9,8	19,5	15,8	7,1	8,0	10,5	9,1	8,2	11,3	10,1	3,96
87	N800 / microb (B)	12,1	7,8	7,8	7,8	8,0	7,8	8,0	8,1	7,7	7,6	8,3	7,8	1,37
88	N800 / opera (A)	6,6	6,4	6,5	6,6	10,9	6,5	6,4	6,3	11,1	6,4	7,4	6,5	1,89
89	iPhone (A)	22,1	15,5	11,6	10,4	12,1	10,4	13,4	16,2	28,0	14,0	15,4	13,7	5,63

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
all	doc ref and js/html/css “code divs” are also tested with each browser
80	Pictures outside picture frame, sometimes the same “last cow problem” (MINOR)
81	Unreliable: sometimes works with cache cleared. Problems with always visible code divs. (FAILURE)
82	On mozillas (desktop too) last cow pic sometimes refuses to load. (MINOR)
83	No defects detected (PASSED)
84	No defects detected (PASSED)
85,86,87	Outliers possibly due to network conditions?



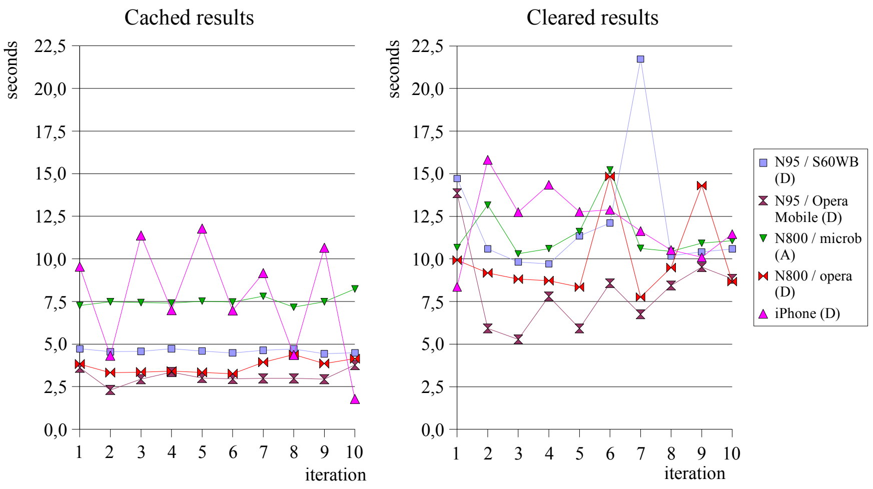
Name	HTML Editor
Category	ASP.NET AJAX
Version(s)	ASP.NET 2.0, AJAX Extensions 1.0
URL	http://winthusiasm.com/Downloads/HtmlEditor/Demo.aspx
Test description	loading time, starts when link is accessed, ends when text in edit field is visible or reflow stops

test started	01.11.07 22:19	02.11.07 12:20
test finished	01.11.07 23:00	02.11.07 14:36

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
110	N95 / S60 Web Browser cached	4,7	4,6	4,6	4,7	4,6	4,5	4,6	4,7	4,4	4,5	4,6	4,6	0,11
111	N95 / Opera Mobile cached	3,6	2,3	2,9	3,4	3,0	3,0	3,0	3,0	2,9	3,8	3,1	3,0	0,41
112	N800 / microb cached	7,3	7,5	7,4	7,4	7,5	7,5	7,8	7,2	7,5	8,2	7,5	7,5	0,3
113	N800 / opera cached	3,8	3,3	3,4	3,4	3,3	3,3	3,9	4,4	3,9	4,1	3,7	3,6	0,4
114	iPhone cached	9,6	4,3	11,4	7,0	11,8	7,0	9,2	4,4	10,7	1,8	7,7	8,1	3,38
115	N95 / S60WB (D)	14,7	10,6	9,8	9,7	11,4	12,1	21,7	10,2	10,4	10,6	12,1	10,6	3,68
116	N95 / Opera Mobile (D)	13,9	5,9	5,3	7,8	5,9	8,6	6,8	8,4	9,52	8,8	8,1	8,1	2,48
117	N800 / microb (A)	10,7	13,2	10,3	10,6	11,6	15,2	10,6	10,4	10,9	11,1	11,5	10,8	1,56
118	N800 / opera (D)	9,9	9,2	8,8	8,7	8,4	14,8	7,8	9,5	14,3	8,7	10,0	9,0	2,48
119	iPhone (D)	8,4	15,8	12,8	14,3	12,8	12,9	11,6	10,5	10,1	11,5	12,1	12,2	2,14

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
110&111	these measurements were performed using a separate 10/10 Mbit/s LAN without other users.
110&111	results were verified to be consistent with the cs wlan
110	GUI fully visible, but no content in Design or Html tabs. (FAILURE)
111	GUI fully visible, no content in Design or Html tabs, preview works but causes refresh (FAILURE)
112	No additional defects detected. There are some bugs with desktop browsers as well. (PASSED)
113	GUI fully visible, no content in Design or Html tabs, preview works but causes refresh (FAILURE)
114	GUI fully visible, no content in Design or Html tabs, preview works but causes refresh (FAILURE)

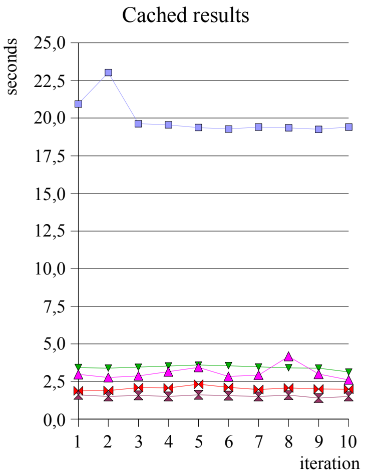


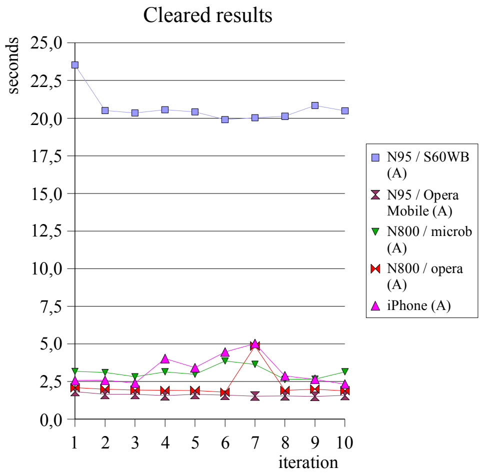
Name	XML 2006 event schedule
Category	Frost
Version(s)	Frost “predecessor”, app from 2006
URL	http://www.pwmwa.com/xml06/
Test description	loading time, starts when “show all” is clicked, ends when text stops reflowing
test started	01.11.07 11:30
test finished	01.11.07 13:00

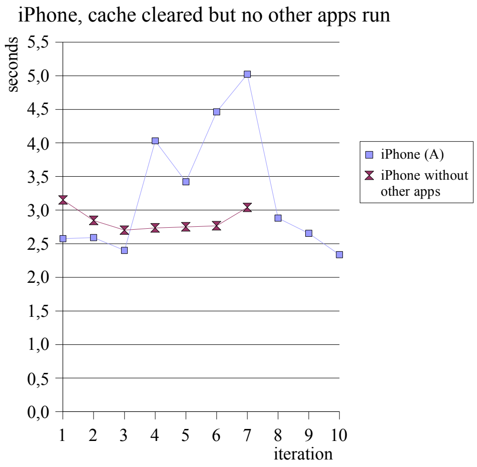
		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	std dev
100	N95 / S60 Web Browser cached	20,9	23,0	19,6	19,5	19,4	19,3	19,4	19,4	19,3	19,4	19,9	19,4	1,2
101	N95 / Opera Mobile cached	1,6	1,5	1,6	1,5	1,6	1,6	1,5	1,6	1,4	1,5	1,5	1,5	0,07
102	N800 / microb cached	3,4	3,4	3,5	3,5	3,6	3,6	3,5	3,4	3,4	3,1	3,4	3,4	0,13
103	N800 / opera cached	1,9	1,9	2,1	2,1	2,3	2,1	2,0	2,1	2,0	2,0	2,0	2,0	0,13
104	iPhone cached	3,0	2,8	2,9	3,2	3,4	2,8	2,9	4,2	3,0	2,6	3,1	3,0	0,45
105	N95 / S60WB (A)	23,5	20,5	20,3	20,6	20,4	19,9	20,0	20,1	20,8	20,5	20,7	20,5	1,04
106	N95 / Opera Mobile (A)	1,8	1,6	1,6	1,6	1,6	1,6	1,5	1,6	1,5	1,6	1,6	1,6	0,09
107	N800 / microb (A)	3,2	3,1	2,8	3,2	3,0	3,9	3,6	2,6	2,7	3,2	3,1	3,1	0,39
108	N800 / opera (A)	2,1	2,0	1,9	1,9	1,9	1,8	4,9	1,9	2,0	1,9	2,2	1,9	0,93
109	iPhone (A)	2,6	2,6	2,4	4,0	3,4	4,5	5,0	2,9	2,7	2,3	3,2	2,8	0,95
109b	iPhone without other apps	3,15	2,85	2,7	2,74	2,75	2,77	3,04				2,9	2,8	0,17

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
all	the XML file loads in <1 s on desktop browser in same LAN
100	No defects detected, back button works. (PASSED)
100	User may begin browsing after ~3 seconds, while XML loads. The positioning of new text is actually visible!
101	No defects detected, back button works, user may browse while loading. (PASSED)
102	No defects detected, back button works, user may browse while loading. (PASSED)
103	No defects detected, back button works, user may browse while loading. (PASSED)
104	No defects detected, back button works, user may browse while loading. (PASSED)
106	Crash after clearing cache, exiting S60, starting opera at the beginning of iteration 8. Could not force quit, so power cycled
109b	Is without running other apps in between iterations. This should be indicative of the memory cache?





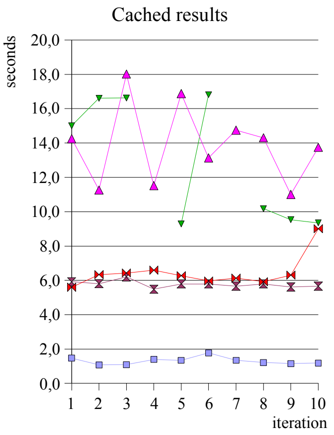


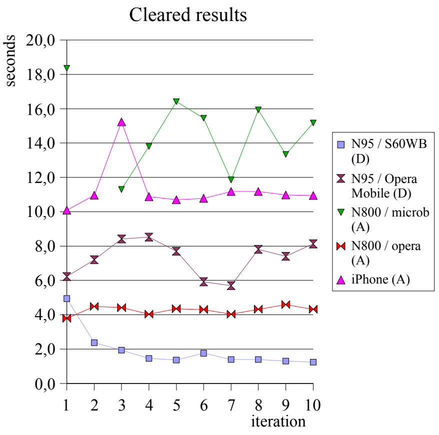
Name	Google Maps
Category	Google
Version(s)	
URL	http://maps.google.com/
Test description	loading time, starts when link is accessed, ends when search form accepts input
test started	02.11.07 14:40
test finished	02.11.07 18:49

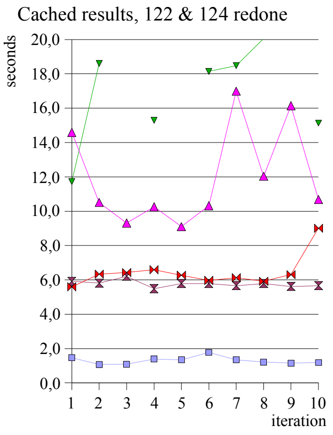
		iteration													
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	c. avg	med	stdev
120	N95 / S60 Web Browser cached	1,5	1,1	1,1	1,4	1,3	1,8	1,3	1,2	1,2	1,2	1,3		1,3	0,21
121	N95 / Opera Mobile cached	5,9	5,8	6,2	5,5	5,8	5,8	5,7	5,8	5,6	5,7	5,8		5,8	0,2
122	N800 / microb cached	15,0	16,6	16,6		9,3	16,8		10,2	9,5	9,3	12,9		12,6	3,62
122b	N800 / microb cached	11,7	18,6		15,3		18,1	18,5	20,1		15,1		16,8	18,1	2,87
	122 & 122b combined												14,7	15,3	3,75
123	N800 / opera cached	5,6	6,3	6,4	6,6	6,3	6,0	6,1	5,9	6,3	9,0	6,5		6,3	0,94
124	iPhone cached	14,3	11,3	18,0	11,5	16,9	13,1	14,8	14,3	11,0	13,7	13,9		14,0	2,32
124b	iPhone cached	14,6	10,5	9,3	10,3	9,1	10,3	17,0	12,0	16,1	10,7		12,0	10,6	2,87
	124 & 124b combined												12,9	12,6	2,72
125	N95 / S60WB (D)	4,9	2,4	1,9	1,5	1,4	1,8	1,4	1,4	1,3	1,2	1,9		1,4	1,12
126	N95 / Opera Mobile (D)	6,2	7,2	8,4	8,5	7,7	5,9	5,7	7,81	7,4	8,1	7,3		7,6	1,03
127	N800 / microb (A)	18,3		11,3	13,8	16,4	15,5	11,9	15,9	13,3	15,2	14,6		15,2	2,26
128	N800 / opera (A)	3,8	4,5	4,4	4,0	4,4	4,3	4,0	4,3	4,6	4,3	4,3		4,3	0,24
129	iPhone (A)	10,1	11,0	15,2	10,9	10,7	10,8	11,2	11,2	11,0	10,9	11,3		11,0	1,42

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
120	Content is degraded into non-ajax, only mobile web sites available? (MAJOR/FAILURE)
120	Could greatly benefit from form autocomplete.
121 onwards	Maps seems to implement delayed loading?
121	Submitting an address yields results, but map will not center on target. (FAILURE)
121	GUI almost fully visible at minimal zoom, but not scrollable. Map partially visible. Back button works.
122	Closing dialogue bubbles is slow, as is drag and drop. Back button works. No further defects detected (PASSED)
122	Loading app sometimes (5,9,10) brings up virtual keyboard, sometimes not (1,2,3,6,8).
122	One crash detected while testing functionality, outside timing iterations
122	crashed after iteration 4,7, before showing application
122b	crashed after iteration 3,5,9, before showing application
123	Dialogue bubbles show up blank, no d&d, otherwise as good as 122. (PASSED)
124	Some addresses can not be found, even though they are in 122&123. (PASSED)
124	Maps seems to be able to interact with phone widget: results are displayed there.
127	microb crashed after iteration 2, time as follows 11,5
122&123	full screen mode was activated
129&127	Noncached results were lower than cached, so I redid 124,122 straight afterwards. Aggregate results in column N


Cached results


Cleared results

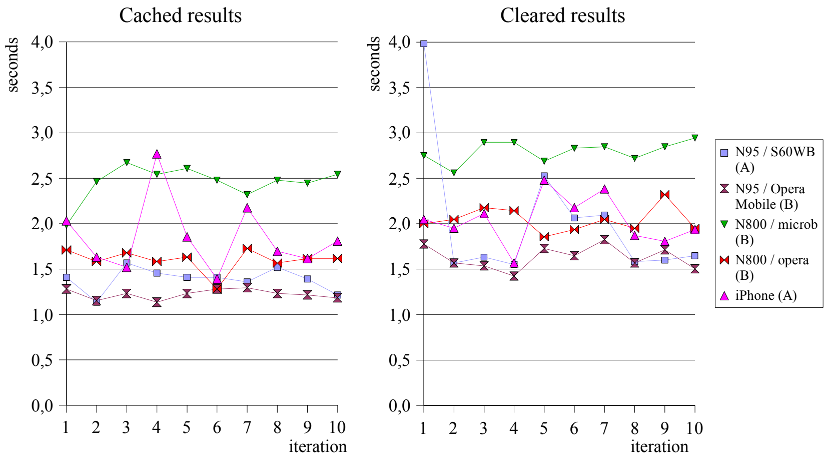

Cached results, 122 & 124 redone

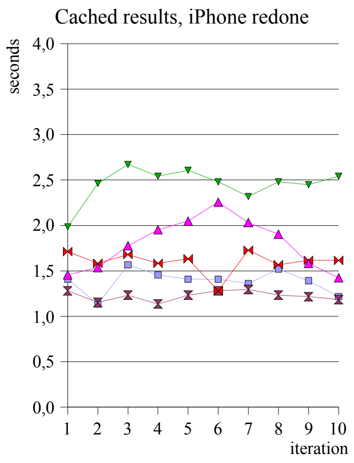
Name	Google Suggest
Category	Google
Version(s)	
URL	http://www.google.com/webhp?complete=1&hl=en
Test description	loading time, starts when link is accessed, ends when UI fully visible.
test started	02.11.07 19:00
test finished	02.11.07 20:16

		iteration													
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	c. avg	med	stdev
130	N95 / S60 Web Browser cached	1,4	1,1	1,6	1,5	1,4	1,4	1,4	1,5	1,4	1,2	1,4		1,4	0,13
131	N95 / Opera Mobile cached	1,3	1,2	1,2	1,1	1,2	1,3	1,3	1,2	1,2	1,2	1,2		1,2	0,05
132	N800 / microb cached	2,0	2,5	2,7	2,5	2,6	2,5	2,3	2,5	2,4	2,5	2,5		2,5	0,19
133	N800 / opera cached	1,7	1,6	1,7	1,6	1,6	1,3	1,7	1,6	1,6	1,6	1,6		1,6	0,13
134	iPhone cached	2,0	1,6	1,5	2,8	1,9	1,4	2,2	1,7	1,6	1,8	1,8		1,8	0,4
134b	iPhone cached	1,5	1,5	1,8	2,0	2,0	2,3	2,0	1,9	1,6	1,4		1,8	1,8	0,29
	134 & 134 b combined												1,8	1,8	0,34
135	N95 / S60WB (A)	4,0	1,6	1,6	1,6	2,5	2,1	2,1	1,6	1,6	1,6	2,0		1,6	0,76
136	N95 / Opera Mobile (B)	1,8	1,6	1,5	1,4	1,7	1,6	1,8	1,57	1,7	1,5	1,6		1,6	0,13
137	N800 / microb (B)	2,8	2,6	2,9	2,9	2,7	2,8	2,8	2,7	2,8	2,9	2,8		2,8	0,12
138	N800 / opera (B)	2,0	2,0	2,2	2,1	1,9	1,9	2,0	2,0	2,3	2,0	2,0		2,0	0,14
139	iPhone (A)	2,0	2,0	2,1	1,6	2,5	2,2	2,4	1,9	1,8	1,9	2,0		2,0	0,27

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
130	Degrades gracefully into normal google search. (PASSED)
131	Almost full functionality. Can only select the first of the results shown (MINOR)
131	Browser crashed while loading at the beginning of iteration 1, could not force quit, so power cycled the device.
132	Full functionality, but only when erasing text with backspace (MINOR)
133	Full functionality, but only when erasing text with backspace (MINOR)
134	Degrades gracefully into normal google search. (PASSED)
134b	Extra iterations due to high variance, again.



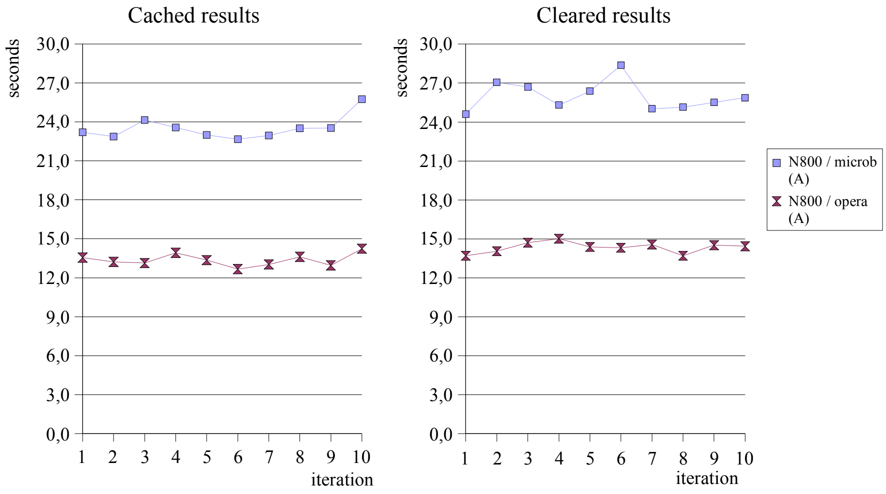


Name	GMail
Category	Google
Version(s)	beta
URL	http://mail.google.com/mail?nocheckbrowser/
Test description	Loading time. Ends when message header list is visible.
test started	03.11.07 21:50
test finished	03.11.07 23:14

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	stdev
180	N95 / S60 Web Browser cached	PASSED												
181	N95 / Opera Mobile cached	PASSED												
182	N800 / microb cached	23,2	22,9	24,2	23,6	23,0	22,7	23,0	23,5	23,5	25,7	23,5	23,4	0,89
183	N800 / opera cached	13,6	13,2	13,1	13,9	13,4	12,7	13,0	13,6	13,0	14,2	13,4	13,3	0,47
184	iPhone cached	PASSED												
185	N95 / S60WB (A)													
186	N95 / Opera Mobile (A)													
187	N800 / microb (A)	24,6	27,1	26,7	25,3	26,4	28,4	25,0	25,1	25,5	25,9	26,0	25,7	1,14
188	N800 / opera (A)	13,7	14,0	14,7	15,0	14,4	14,3	14,6	13,7	14,5	14,4	14,3	14,4	0,42
189	iPhone (A)													

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
180	Gracefully degrades to mobile, basic HTML can be selected, but no Ajax. Faster than measurement method. (PASSED)
181	Without browser check, the UI is very broken, inbox contents invisible, most links do not work.
181	With browser check, gracefully degrades into basic HTML . Faster than measurement method. (PASSED)
182&183	Full screen mode activated.
182	Address autocomplete requires leaving form field and returning to it.
182	Most functions take 1~2s, noticeable for user. (PASSED)
183	As 182. Backspace seems to trigger autocomplete as well. (PASSED)
184	As 180, Ajax version inaccessible. Selecting “standard” yields “mobile”.(PASSED)



Name	Yahoo! Mail
Category	YUI
Version(s)	
URL	http://mail.yahoo.com/
Test description	
test started	03.11.07 15:00
test finished	03.11.07 16:25

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	stdev
150	N95 / S60 Web Browser cached	FAILURE												
151	N95 / Opera Mobile cached	MINOR												
152	N800 / microb cached	PASSED												
153	N800 / opera cached	MAJOR												
154	iPhone cached	PASSED												
155	N95 / S60 Web Browser (D)													
156	N95 / Opera Mobile (B)													
157	N800 / microb cleared (A)													
158	N800 / opera cleared (C)													
159	iPhone cleared (A)													

Notes

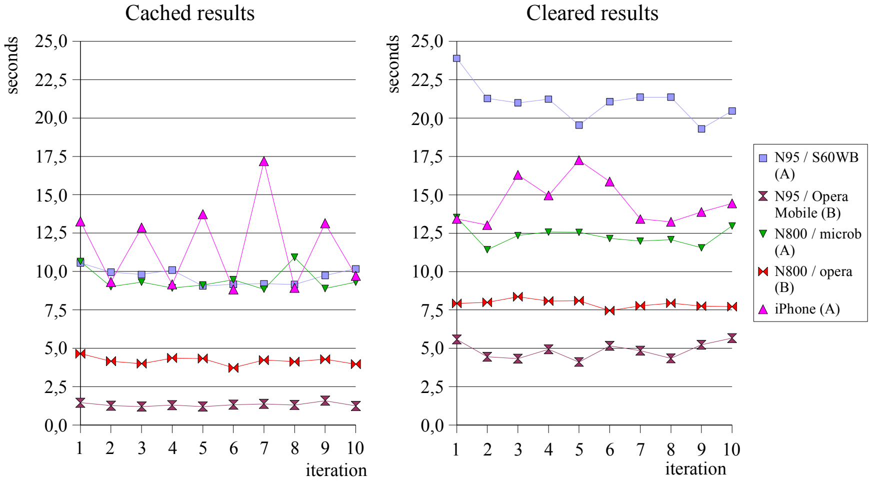
ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
150	User is offered choice between full site and mail classic
150	Ran out of memory while accessing the full application, after loading 1,29 MB but did so in a controlled manner.
150	However, choosing the mail classic after this is difficult.
150	Mail classic seems to include some text popups that quickly fill S60 memory and cause browser crash.
152	Yahoo mail experienced a login error: name:NS_ERROR_DOM_WRONG_DOCUMENT_ERR message:Node cannot be used in a document other than the one in which it was created lineNumber: 512 App noticed that "loading was taking longer than usual" and offered the choice to use mail classic instead.
152	app noticed that the screen resolution was beneath the minimum (1024x768) and offered mail classic.
152	no defects found from mail classic. There is some DHTML functionality, but in overall every fetch causes redraw.
152	some links send the user back into new mail and then the degradation phase
151	App is switched to mail classic automatically upon login. Navigation tricky without a cursor.
151	clicking on "all-new mail" link yields multiple JS errors on load, and a broken app.
153	As 151, mail classic automatically. The right edge is cropped off messages, despite zooming.
153	Additionally, browser detection is used to prohibit the loading of the full site. Browser upgrade is required.
154	Browser is detected and app selects yahoo mobile mail automatically. App recommends internal widget

Name	flickr
Category	internal development + YUI?
Version(s)	internal development
URL	http://www.flickr.com/photos/9092161@N07/1843554028/
Test description	**loading time. measured page includes a test photo of 817 bytes, test ends when picture visible**
test started	03.11.07 16:30
test finished	03.11.07 19:22

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	stdev
160	N95 / S60 Web Browser cached	10,6	10,0	9,8	10,1	9,1	9,2	9,2	9,2	9,7	10,2	9,7	9,8	0,52
161	N95 / Opera Mobile cached	1,5	1,3	1,2	1,3	1,2	1,3	1,4	1,3	1,6	1,2	1,3	1,3	0,12
162	N800 / microb cached	10,7	9,0	9,3	8,9	9,1	9,5	8,8	10,9	8,9	9,3	9,4	9,2	0,74
163	N800 / opera cached	4,7	4,2	4,0	4,4	4,3	3,7	4,2	4,1	4,3	4,0	4,2	4,2	0,26
164	iPhone cached	13,3	9,3	12,9	9,2	13,7	8,8	17,2	8,9	13,1	9,7	11,6	11,3	2,82
165	N95 / S60WB (A)	23,9	21,3	21,0	21,2	19,5	21,1	21,4	21,4	19,3	20,5	21,0	21,1	1,25
166	N95 / Opera Mobile (B)	5,6	4,4	4,3	4,9	4,1	5,2	4,8	4,4	5,2	5,7	4,9	4,9	0,54
167	N800 / microb (A)	13,5	11,4	12,4	12,6	12,6	12,2	12,0	12,1	11,6	13,0	12,3	12,3	0,63
168	N800 / opera (B)	7,9	8,0	8,4	8,1	8,1	7,5	7,8	7,9	7,8	7,7	7,9	7,9	0,25
169	iPhone (A)	13,4	13,0	16,3	15,0	17,2	15,9	13,4	13,2	13,9	14,4	14,6	14,2	1,46

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
160&161	offers mobile version upon login, full version accessible. Browsing easier at 50% zoom
160	EIP works (1-2 s), photostream works, slideshow is broken, overall very good (PASSED)
161	Photostream selector broken, arrows can not be accessed, slide show broken
161	EIP does not work (no “click here”) in main view, unclickable in photo view. (MINOR)
162	No defects detected. Slide show works, hovered captions work (PASSED)
163	Photostream selector broken, arrows can not be accessed, slide show works but “back to your photos” does not (MINOR)
163	crashed during initial testing, while using slide show
164	Everything works except slide show (PASSED)
166	with opera, select “view full version” from mobile flickr, then clear cache, then click on link to repeat measurement.
165&166	note that flickr defaults to mobile if on n95. Thus, only disk, not all cached data was cleared during these tests.

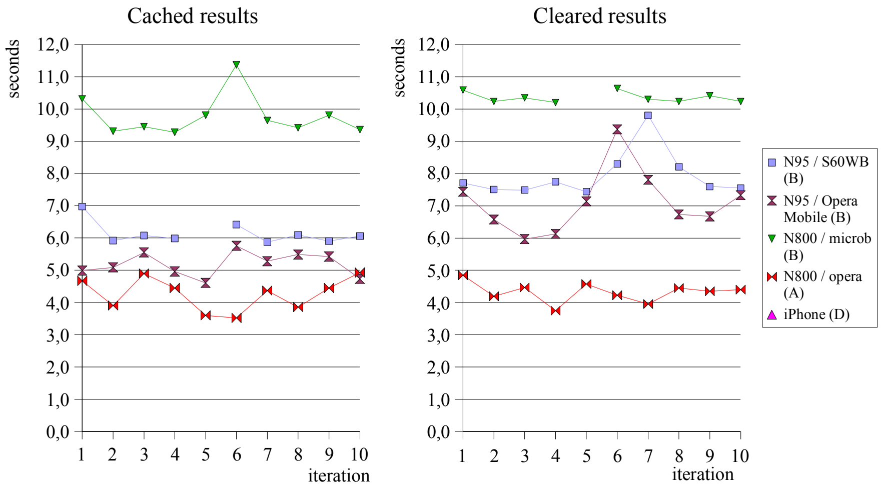


Name	Journey Planner for Cycling
Category	YTV
Version(s)	NaviciAjaxApi.js, klero_pack.js
URL	http://kevytliikenne.ytv.fi/
Test description	loading time, starts when link is accessed, ends when text can be entered into search
test started	02.11.07 20:45
test finished	02.11.07 22:17

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	stdev
140	N95 / S60 Web Browser cached	7,0	5,9	6,1	6,0		6,4	5,9	6,1	5,9	6,1	6,1	6,1	0,35
141	N95 / Opera Mobile cached	5,0	5,1	5,6	5,0	4,6	5,8	5,3	5,5	5,4	4,7	5,2	5,2	0,37
142	N800 / microb cached	10,3	9,3	9,5	9,3	9,8	11,4	9,6	9,4	9,8	9,4	9,8	9,6	0,64
143	N800 / opera cached	4,7	3,9	4,9	4,4	3,6	3,5	4,4	3,9	4,4	4,9	4,3	4,4	0,51
144	iPhone cached													
145	N95 / S60WB (B)	7,7	7,5	7,5	7,7	7,4	8,3	9,8	8,2	7,6	7,6	7,9	7,7	0,72
146	N95 / Opera Mobile (B)	7,4	6,6	6,0	6,1	7,1	9,4	7,8	6,7	6,7	7,3	7,1	6,9	0,98
147	N800 / microb (B)	10,6	10,2	10,4	10,2		10,6	10,3	10,2	10,4	10,2	10,4	10,3	0,16
148	N800 / opera (A)	4,8	4,2	4,5	3,7	4,6	4,2	4,0	4,4	4,4	4,4	4,3	4,4	0,31
149	iPhone (D)													

Notes

ref id	nota bene	
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.	
140	Arrow keys inaccessible, but clicking on waypoint links centers map correctly. Zooming is a bit slow. (MINOR)	
140	Not all zoom levels work perfectly.	
140	Crash after extensive testing + iteration 5. Time as follows	5,9
141	Crash before iteration 1, possibly due to 140. Force quit did not work, so N95 was power cycled.	
141	Autocomplete scrolls to bottom 7 of list. If there are fewer items, list is usable.	
141	Zoom and arrow keys unselectable, but map scrolls from waypoint links. Back button does not work. (MINOR)	
141	Since Opera does not use a cursor, it is harder to select the input field.	
140&141	Viewing map and entering text require use of zooming functions.	
142&143	full screen mode	
142	Arrow keys inaccessible, but clicking on waypoint links centers map correctly. Back button does not work. (MINOR)	
143	Back button does not work. Everything else does. (PASSED)	
144	Crash while loading (FAILURE)	
147	communication error in iteration 5. Time as follows	10,5

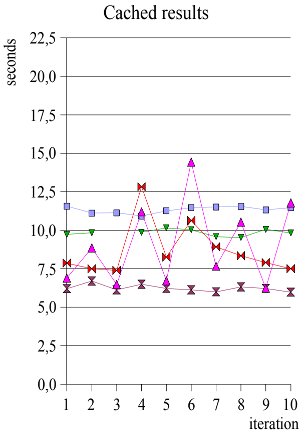


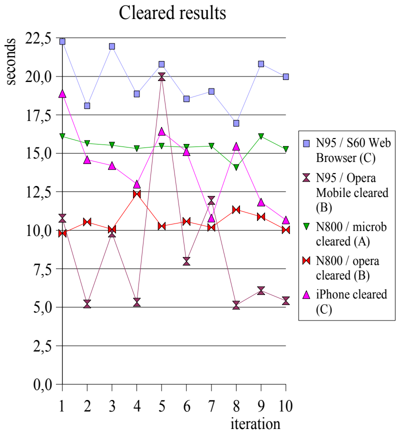
Name	Colorado Geographic
Category	Microsoft
Version(s)	ASP.NET AJAX
URL	http://www.coloradogeographic.com/
Test description	Loading time. Ends when I can see the blue sky above Kebler pass in the main window.
test started	03.11.07 19:37
test finished	03.11.07 21:21

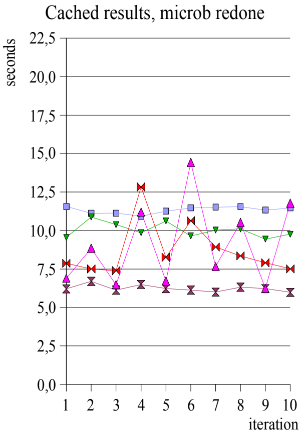
		iteration													
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	c. avg	med	stdev
170	N95 / S60 Web Browser cached	11,6	11,1	11,1	10,9	11,3	11,5	11,5	11,6	11,3	11,5	11,3		11,4	0,22
171	N95 / Opera Mobile cached	6,2	6,7	6,1	6,5	6,2	6,1	6,0	6,3	6,2	6,0	6,2		6,2	0,22
172	N800 / microb cached	9,7	9,8		9,9	10,2	10,0	9,6	9,5	10,1	9,8	9,8		9,8	0,21
172b	N800 / microb cached	9,6	10,9	10,4	9,9	10,6	9,6	10,0	10,1	9,4	9,8		10,0	9,9	0,48
	172 & 172b combined												9,9	9,9	0,38
173	N800 / opera cached	7,9	7,5	7,4	12,8	8,3	10,6	8,9	8,4	7,9	7,5	8,7		8,1	1,73
174	iPhone cached	6,9	8,8	6,5	11,2	6,7	14,4	7,7	10,5	6,2	11,8	9,1		8,3	2,78
175	N95 / S60 Web Browser (C)	22,3	18,1	21,9	18,9	20,8	18,5	19,0	17,0	20,8	20,0	19,7		19,5	1,72
176	N95 / Opera Mobile cleared (B)	10,8	5,2	9,8	5,3	20,0	8,0	12,0	5,1	6,1	5,4	8,8		7,0	4,69
177	N800 / microb cleared (A)	16,1	15,6	15,5	15,3	15,5	15,4	15,5	14,1	16,1	15,3	15,4		15,5	0,56
178	N800 / opera cleared (B)	9,8	10,5	10,1	12,4	10,3	10,6	10,2	11,3	10,9	10,0	10,6		10,4	0,77
179	iPhone cleared (C)	18,9	14,6	14,2	13,0	16,4	15,1	10,8	15,5	11,8	10,7	14,1		14,4	2,59

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
170	UI loads correctly. Every transition causes a redraw. “Highlights” links are not recognized (MAJOR)
170	Desktop mozilla also has problems with the “Highlights”, causing the infamous “link flicker”
170	Right hand text flows beneath div, not all of the text is scrollable or readable.
171	UI loads correctly, left hand set pic and upper right arrows cause redraws. “Highlights” load correctly with async call.
171	Right hand text flows beneath div, all of it is readable. Back button works. (MINOR)
172&173	Full screen mode activated
172	Back button does not work. (PASSED)
172	Crash after initial testing AND iteration 3, when clicking back button. Time as follows 10,1
172b	Additional iterations in order to detect a possible memory leak as the cause of the crash above.
172b	Iterations 9,16,18,19 caused the following while exiting with back button: JS alert: Sys.WebForms.PageRequestManagerServerErrorException: An unknown error occurred while processing the request on the server. The status code returned from the server was: 2147746065
173	As 171, but full text visible beyond border. Back works here too. (MINOR)
174	As 170, but red arrows are broken. Additionally, site is extremely sluggish. (MAJOR)
174	Last stage of loading takes 5~10s more than on the others. But that's okay, delayed loading is accepted.
174	Here, again, the differing speeds are clearly visible as different loading types on the Safari. everything at once = slow
176	The outlier is possibly either due to 1) network error 2) erraneous cache clearing phase, causing including of connection setup
177&178	windowed mode


Cached results


Cleared results

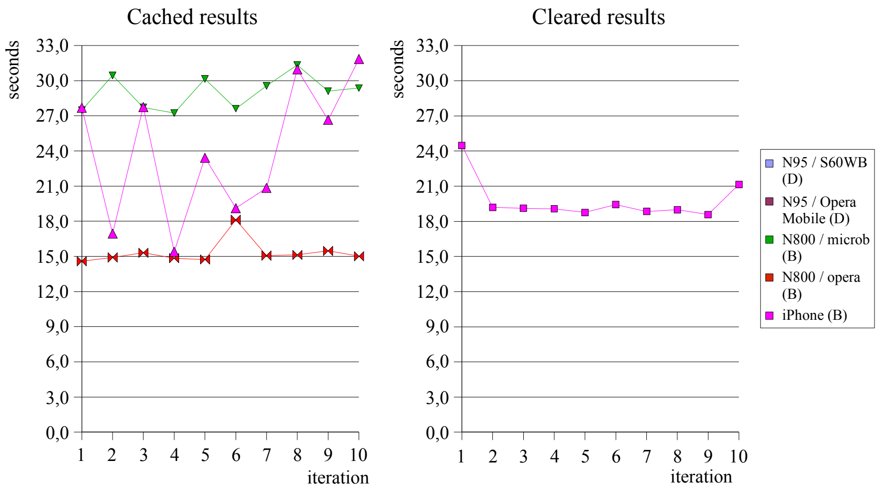

Cached results, microb redone

Name	myAOL (beta)
Category	AOL
Version(s)	dojo 0.4.3
URL	http://my.aol.com/
Test description	Loading time. Ends when content of "Right Now on AOL" is visible.
test started	05.11.07 10:00
test finished	05.11.07 11:41

		iteration												
test identifier	name	1	2	3	4	5	6	7	8	9	10	avg	med	stdev
190	N95 / S60 Web Browser cached													
191	N95 / Opera Mobile cached													
192	N800 / microb cached	27,5	30,5	27,7	27,2	30,2	27,6	29,6	31,3	29,1	29,4	29,0	29,2	1,43
193	N800 / opera cached	14,6	14,9	15,3	14,8	14,7	18,1	15,1	15,1	15,5	15,0	15,3	15,0	1,02
194	iPhone cached	27,7	16,9	27,8	15,4	23,4	19,1	20,8	31,0	26,7	31,9	24,1	25,0	5,8
195	N95 / S60WB (D)													
196	N95 / Opera Mobile (D)													
197	N800 / microb (B)													
198	N800 / opera (B)													
199	iPhone (B)	24,5	19,2	19,1	19,1	18,8	19,4	18,8	19,0	18,6	21,1	19,8	19,1	1,8

Notes

ref id	nota bene
all	at test start, browser cache is cleared. after initial loading, app is tested for defects.
all	myAOL heavily employs delayed loading
190	degrades into AOL mobile, server content only for registered users. no link to full content (FAILURE)
191	seems to load full version, content may not be navigated
191	if fit to window is activated, most content seems to be readable. many links do not work
191	on second iteration, loads most of UI. window controls unaccessible, many windows blank. (FAILURE)
191	tabs seem to work
192	splash window can be closed only through full screen mode
192&193	full screen mode activated
192	back button does not work at all, chokes on "Connecting"
192	JavaScript alerts when accessing tabs: "Sorry, the first time could not be finished. Please try again later"
192	loading tabs notably slow (MINOR)
193	UI loads correctly, splash screen can be closed. Window controls work. Most windows without content.
193	blank windows: AOL public galleries, AOL mail, local news, weather, most of AOL video search, right now on AOL
193	**this is time to load app + advertisement window, since attempted target (test description) does not load**
193	tabs work, mgnet loaded pictures asynchronously but froze browser, back button works.
194	splash window's controls inaccessible, as are all links in it – it remains on top of UI. back button works.
194	the variance is not due to 1) how long I spend time on the previous page 2) if the target is allowed to load completely
194	after iteration 9, the splash screen was disabled. was not able to repeat this. (MINOR)
194	clearing cache and cookies did not reinstate loading of the splash screen on the first time. Second time did it.
194	clearing the cache **twice** might have an effect (iteration 10)


Cached results
seconds
iteration
Cleared results
seconds
iteration
N95 / S60WB (D)
N95 / Opera Mobile (D)
N800 / microb (B)
N800 / opera (B)
iPhone (B)