Visualising Social Network Activity on Mobile Browsers

Christoffer Björkskog

Helsinki June 4, 2008

Master's Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Christoffer Björkskog

Työn nimi — Arbetets titel — Title

Visualising Social Network Activity on Mobile Browsers

Oppiaine — Läroämne — Subject

Työn laji — Arbetets art — Level Master's Thesis	Aika — Datum — Month and year June 4, 2008	Sivumäärä — Sidoantal — Number of pages 65 pages + 5 appendix pages

Tiivistelmä — Referat — Abstract

Online social communication and contribution is very common today. There are many different online social networks and it is difficult to follow the activities in all. In this thesis we study whether online social activities can be aggregated and visualised in a graspable way on mobile Web browses using AJAX technologies.

We have created Funnelry, a social media mashup application, designed to fetch the online activities of the user's friends from different services and present them in a filtered list view on mobile web browsers.

This thesis comes to the conclusion that it is currently possible to visualise these activities on certain mobile browsers using AJAX technologies. It is possible that in a near future most mobile Web browsers will support these technologies. Clustering and filtering out data that is not interesting to the user is needed when there is a great deal of online activities going on.

ACM Computing Classification System (CCS):
H.3.5 [Online Information Services],
H.4.3 [Communications Applications],
H.5.1 [Multimedia Information Systems],
H.5.2 [User Interfaces]

Avainsanat — Nyckelord — Keywords
Web 2.0, mobile media, visualisation, user interfaces, service aggregation, mobile AJAX

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Human beings are social creatures with an ability to work together in groups, creating value that is greater than the sum of its parts [WM08]. Many online services exist where users can contribute with content and communicate with each other. For example, many people have an online diary where they publish their thoughts. According to a questionnaire targeted at American Internet users, social and professional networking sites have increased in popularity, people remix online content, categorise, or "tag", online content and upload photos, videos and content they have produced [Pew08]. A great deal of content is being viewed and produced by online users, and communication over the Internet in different forms is happening all the time [WM08]. People use different social networking services to keep up with their friends. However, it becomes difficult to follow the activities in all of them when there are a large number of services that a person's friends are active in.

Mobile phones are closely connected to persons as well as to a global network [CtHS06]. Social networking on mobile phones may enhance social activities when the context of the users is taken into account. However, displaying a great deal of content on mobile devices may be difficult because of the limited screen size of the phone [Noi05].

The aim of this thesis is to see whether online social activities can be aggregated and viewed in a user friendly way on mobile web browsers. Funnelry is a prototype developed for the Helsinki Institute for Information Technology (HIIT) designed to aggregate and display social activity on mobile browsers, and is the software part of this thesis. Funnelry visualises online activity using Extensible HyperText Markup Language (XHTML), Cascading Style Sheets (CSS), Document Object Model (DOM) manipulation and client side scripting (JavaScript), fetching data as JavaScript Object Notation (JSON) feeds using XMLHttpRequests (XHR).

The structure of the thesis is as follows:

In Chapter 2, a background on social media services and Web 2.0 is described. These are phenomena that have emerged during recent years and have changed the way the Internet is used. Social networking on mobile devices is introduced.

In Chapter 3, the limitations and strengths of mobile devices as a platform for social networking are discussed, and the capabilities of modern mobile web browsers needed to run dynamically changing Rich Internet Applications (RIA) are examined.

In Chapter 4, principles for usability, visualisation and models for perception are introduced. Two examples of visualising large amounts of data on mobile devices with small and very small displays are presented.

Chapter 5 introduces service aggregation, which involves gathering information from different sources into one. Syndication, a method of providing a computer readable version of content through RSS (Really Simple Syndication), is introduced. Mashups, combining elements from different services into new services are presented. Techniques enabling mashups, such as RSS and AJAX, are presented.

Chapter 6 describes Funnelry, a mobile social media mashup that combines and filters online activities from user's online social networks on a mobile web browser.

Chapter 7 summarises the results of this thesis and takes a glimpse at what to expect in the near future of the mobile web.

2 Social Media Services and Web 2.0

2.1 Social Media Services

During the last ten years, the web has matured both technologically and in usage [WM08]. The focus is on user contribution, collaboration, communication and social networking [O'R05, WM08]. Web applications that enable these activities are called social software [CtHS06].

The web and social software enable social networking in ways that were previously unattainable, providing opportunities for collaboration and communication between people online [WM08]. A social network is defined as a group of three or more persons sharing information with each other, such as a scout troop, church, university or any other socially structured relationship [WM08]. Social networking is influencing how the web is used and is causing a major shift in the function and design of the Internet.

The use of social software has increased tremendously in the past few years [CtHS06]. People contribute and communicate using blogs, social networking and tagging systems, while they continue to use existing socially oriented software such as newsgroups, bulletin boards and collaborative filtering systems.

Many of the popular social networks grow through word of mouth, which is the natural behaviour of people telling others about products or services that are either good or bad [WM08]. A survey showed that from 2005 to 2006 the number of daily online social network users, based on American Internet users aged between 18 and 29, increased from 4% to 31% [Pew08]. Data from this survey, summarised in Table 1, shows statistics of American Internet users' activities on the web.

There are different kinds of online social media services and networks. The following is a brief introduction to a few of them.

Facebook (facebook.com) connects users with friends, fellow students and peers enabling them to keep in touch, upload photos and to share links and videos.

Orkut (orkut.com) is similar to Facebook and is popular in Brazil, India and the USA.

MySpace (myspace.com) is an online community and a meeting place for friends similar to Orkut and Facebook.

YouTube (youtube.com) allows people to upload videos and share them across websites, mobile devices and e-mail. Users can rate and comment videos and reply to them with a video of their own. Users can also create channels and playlists. In the previously mentioned survey, 70% of American Internet users in ages 18 to 29 answered that they watch videos on video sharing sites, and 30% of the users had done it the previous day [Pew08]. Within the same age group, 36% said that they have shared online content they have made themselves such as videos, photos, artwork or stories.

Flickr (flickr.com) is a service where users can upload their digital photos. It allows users to share these photos with the world, their friends or family. Images can be uploaded from various photo managing software on the users' home computers, the web or mobile devices. The survey of American Internet users found that, in ages 18 to 29, 51% have uploaded images to the Internet for others to see [Pew08]. In ages 30 and older the percentage was around 30.

Digg (digg.com) is a place where people discover and share web content recommendations. If users like the content they can "digg" it, increasing the number of diggs for that content. The greater the number of diggs the content receives, the more visible it becomes and is therefore likely to receive more traffic. Users can also "bury" content, thereby decreasing the number of diggs. In ages 18 to 29, 78% of American Internet users said that they can go online for no particular reason, and almost 40% had gone online for no particular reason

American Internet users:	**Asked**	**18-29**	**30-49**	**50-64**	**65+**
Read or send e-mail	Dec 07	91%	94%	91%	90%
	Yesterday	*62%*	*65%*	*57%*	*45%*
Get news online	Dec 07	75%	73%	68%	55%
	Yesterday	*37%*	*41%*	*32%*	*26%*
Look for info about a hobby or an interest	Feb-Mar 07	86%	88%	77%	62%
	Yesterday	*34%*	*30%*	*26%*	*17%*
Go online for no particular reason	Feb-Apr 06	78%	64%	53%	43%
	Yesterday	*39%*	*28%*	*21%*	*15%*
Send instant messages tom someone who is online at the same time	Aug 06	54%*	36%*	36%*	28%*
	Yesterday	*20%**	*9%**	*5%**	*1%**
Take part in chat rooms or online discussions	Sept 05	40%	20%	14%	9%
	Yesterday	*9%*	*4%*	*1%*	**%*
Play online games	Aug 06	50%**	34%**	26%**	23%**
	Yesterday	*13%***	*8%***	*9%***	*8%***
Create content for the Internet such as helping build a website, create online diaries, post on bulletin boards or participate in online communities	Mar 04	25%	21%	11%	4%
Search info about a person you know or might meet	Sept 05	33%	28%	26%	24%
	Yesterday	*7%*	*5%*	*6%*	*3%*
Create or work on your online journal or blog	Dec 07	24%	11%	4%	2%
Read someone else's online journal or blog	Jan 06	52%	37%	34%	23%
Rated a product/service using an online rating system	Sept 07	37%	34%	29%	20%
	Yesterday	*5%*	*3%*	*2%*	*4%*
Use online social or professional networking sites	Aug 07	49%*	8%*	4%*	1%*
	Yesterday	*31%**	*4%**	*2%**	*0%**
Remix online material into your own creation	Feb-Mar 07	25%	16%	14%	6%
	Yesterday	*5%*	*2%*	*3%*	*1%*
Download a podcast	Aug 06	14%*	12%*	12%*	4%*
Categorize or tag content like photos, news, stories or blog posts	Dec 06	32%	31%	23%	18%
	Yesterday	*10%*	*6%*	*4%*	*5%*
Watch a video on a video sharing site	Dec 07	70%	51%	30%	16%
	Yesterday	*30%*	*14%*	*7%*	*4%*
Look for information on Wikipedia	Feb-Mar 07	44%	38%	31%	26%
	Yesterday	*11%*	*8%*	*5%*	*6%*
Create an avatar or online representation of yourself	Feb-Mar 07	19%	9%	2%	2%
Upload photos to a website so you can share them online	Aug 06	51%**	34%**	30%**	35%**
Post review about a product you bought or service you received	Sep 07	32%	34%	27%	18%
Share something online that you created yourself such as your own artwork, photos, stories or videos	Dec 07	36%	18%	15%	15%
Post comments to an online newsgroup, web page, blog or photo site	Dec 07	35%	21%	14%	12%

Table 1: Table over the online activities of American Internet users [Pew08]. Users where asked whether they have performed certain Internet activities. For some activities, the users were asked whether they had performed the activity the previous day.

the previous day [Pew08]. Services similar to Digg can be useful for finding interesting web content when surfing the net for no particular reason.

del.icio.us (del.icio.us) is a collection of users' favourite links. Users can share favourites with friends, family, co-workers and the del.icio.us community. The del.icio.us service can be used to find new content that others have added as favourites. Typical favourites are about technology, entertainment and information users find useful. Users can see how many have bookmarked a certain link, and the principle is similar to Digg in that the more people bookmark it, the more likely it is that people check it out.

Updating to a network of friends using the Internet is evolving into a human method of social interaction [WM08]. Immediately after the killings at Virgina Tech in April 2007 many students used online social networks such as Facebook to let their friends know that they were not harmed. Virginia Tech students updated their status and put messages on their personal pages saying "I'm all right", "I'm safe" or "I'm coming home". The communications during and after the incident showed that social networking is an important part of communication.

Some people are active in many different networks because their friends are active in different networks (such as Facebook, Myspace or Orkut) or there may be different networks providing different services and features to the user such as Flickr, YouTube and del.icio.us. Different social networking sites such as Facebook and Myspace are not interconnected, meaning that users may need to maintain several accounts in order to keep in touch with friends that are active in one or the other, which may become difficult and time consuming for the users [NKHS07].

As social software gets integrated into our mobile phones, the use of social software is likely to take a significant leap forward [CtHS06]. This is because our phones are closely connected to ourselves as well as to a global network. Programs can be created that enable users to undertake social computing when they are in a social context. Mobile phones have the potential to bring ubiquitous and social computing to everyday users because they are available to most people in developed countries.

Counts et al. [CtHS06], claims that mobile social software give birth to the following research questions:

- How will mobile social software change the way people interact and socialise with each other? A study of Japanese teenagers' use of text messages showed that they use text messages to accomplish social convergence, or micro-coordination [ID05], where an initial general meeting place and time is set, for example "Shibuya, noon", after which several status updates are received until the actual meeting occurs.

- How will the emerging availability of location services change social software? Examples where location information can be used include social networking and recommendation systems. The users can be notified when a friend is nearby. Alternatively, the system can notify, for instance, that people in your social network recommend a different Italian restaurant than the one you are entering. Location is one context attribute of the user. The major factor that differentiates mobile social software from its desktop counterpart is the increased use of context. Another question is how to incorporate context into mobile social software?
- How much information should be made known about a users context to his social network? Where is the border where the benefit of context information is outweighed by the users' loss of privacy? Finding a good way to determine this is an important question. How does this vary depending on the social context of the user?

Progress in mobile technologies has made mobile phones a convenient platform for Web 2.0 applications [NKHS07]. Not all social web applications can be used on mobile devices due to limitations in bandwidth, memory, screen size and keypad size. An average Myspace or Facebook page exceeds 1Mb in size, and according to Natchetoi et al. [NKHS07], not even the most advanced smart phone can render pages of that size. In order to test this last statement, we tried to view a profile page on Facebook, using a Nokia E61i mobile phone with an empty cache. The profile page designed for mobile phones rendered nicely, but when we tried to view the full page, an error message saying that the memory was full emerged (when 2.4MB had been loaded), and the page was not rendered completely.

2.2 Web 2.0

The new way in which the Internet is being used, with user contributed data, social networks and rich user interfaces, is called Web 2.0. It consists of features found in services that survived the dot-com bubble crash in 2001, and is focused around users, communication and synergy [O'R05].

One of the most popular Web 2.0 services are blogs. A blog is a popular medium for ordinary people to publish content. Blogs are personal websites that contain diary-like entries presented in a reverse chronological order where people publish and share their ideas, stories or any other content they want to share [Ham05]. Blogs are increasing rapidly in popularity. The Technorati website [Tec08], which currently tracks activities in over 112 million blogs, report that every day there are over 175,000 new blogs created and that bloggers generate over 1.6 million blog entries a day. One of the things that has made blogs popular is RSS (Really Simple Syndication) [O'R05]. RSS enables users to subscribe to the content of a site by represents it in a machine readable format. Another reason for the popularity of blogs is that they enable visitors to comment on the blog posts, which creates a sense of community and encourages discussion. Visitors may return to the same blog post several times in order to see what other people have commented. The survey mentioned in Section 2.1 showed that 35% of American Internet users aged 18 to 29 have posted comments in online newsgroups, web pages, blogs or photo sites [Pew08].

In the Web 2.0 era, software is treated as a service and not as a product, which means that they need to be maintained frequently [O'R05]. For instance, Google must constantly crawl the web for new pages and updates. They must continuously filter out spam links and any attempts to manipulate their results. At the same time their service must respond to hundreds of millions of asynchronous queries while simultaneously matching them with appropriate advertisements. Web 2.0 products are developed in the open, where new features emerge monthly, weekly and sometimes daily. Services such as Gmail, Google Maps, Flickr, and del.icio.us are likely to bear a "Beta" logo for long periods of time. Sometimes Flickr deploys new builds every half hour. This shows that Web 2.0 applications use a different release cycle compared to traditional software [O'R05].

Web 2.0 applications rely heavily on user-provided data. However, the protection of intellectual property limits data re-use and prevents experimentation. O'Reilly suggests using licences with few restrictions so that data can be re-used, "hacked" and "remixed" [O'R05]. According to the user study mentioned in Section 2.1, 25% of American Internet users aged between 18 and 39 said that they have remixed online content into a creation of their own [Pew08].

3 Capabilities of Mobile Web Browsers

Mobile web browsers, as well as the devices they reside on, can differ in functionality, performance, maturity and accessibility [NKHS07]. If content is not very readable it causes a negative user experience. Content designed for desktop browsers may be distorted, or otherwise unsuitable for viewing on mobile devices.

Many modern mobile phones are equipped with useful features for social networking, such as a built in camera, GPS (Global Positioning System), Bluetooth connected devices and sensors. The problem is that current browsers do not support these kinds of input. Natchetoi et al. [NKHS07] suggest that a new type of mobile browser should be developed that could take advantage of these additional input methods such as image upload via the built in camera and utilisation of voice input. They also suggest that the browser should automatically scale down the complexity of web pages in order to present the content in a suitable way for the mobile device.

According to Natchetoi et al. [NKHS07], it takes time for mobile web browsers to mature. They usually support HTML (Hypertext Markup Language), XHTML, Links in Image-Maps, SSL (Secure Sockets Layer) and UTF8/UTF16 (Unicode Transformation Format). However, technology like Javascript 2.0 or CSS 2 (Cascading Style Sheets) are not usually supported and only few browsers support iFrames, RSS 1.0, Cookies or DOM manipulations.

The Acid tests (`http://www.acidtests.org/`), published by The Web Standards Project (WaSP), are designed to expose limitations in the implementation of web standards in web browsers. The Acid2 test was designed to aid web browser creators in ensuring their products support standards that web designers desire. However, this has not been possible, because major browsers did not follow those industry standards [WSP08a]. Browser creators have started to use the tests to ensure that their software interpret web pages correctly [WSP08b, Hic08]. Acid2 tested whether browsers followed certain industry standards for HTML and CSS specifications, but also if the browsers supported features such as transparent PNGs (Portable Network Graphics).

The Acid2 test was released on April 12, 2005 and presented the results of a successful test graphically as a smiling face [Aci08b]. The test renders a smiling face, if the browser follows the tested features of W3C (World Wide Web Consortium) HTML and CSS 2.0 specifications. On October 31, 2005, 6 months after the release of the test, Safari 2.0.2 was the first browser to pass the test. Other browsers such as

Opera and Konqueror followed after. Major stable browsers that do not yet support Acid2 are Internet Explorer and Firefox, but Acid2-supporting versions are under development; see IE 8 and Firefox 3 beta 4 in Appendix A. More details about the Acid2 test can be found in [WSP08a].

An internal version of Opera Mobile 9 on Nokia Series 60 was the first mobile browser to pass the test [Gol07]. A new version of WebKit, an open source web engine that powers Safari, Dashboard and the mobile browser for Series 60, passes the test, and is what the mobile browsers in Series 60 are built upon [WJ07, Siv07]. Obigo, another mobile web browser, has also passed the test. The current browser in the Nokia Series 60 is based on WebKit, but does not render the Acid2 test correctly [Siv07].

In order to test how well current desktop browsers support the standards tested by Acid2, I visited the test page `http://acid2.acidtests.org/` using certain major browsers on both a Mac OS X 10.5.2 and Windows XP Professional Service Pack 2. The results can be seen in Appendix A along with results from external sources. Different versions of Opera and Safari passed the test on both Mac and PC. Internet Explorer 7 on a Windows machine did not pass the test. Firefox 2 did not pass the test in either Mac or PC, but Firefox 3 beta 4 on the PC did. According to [IEB07], Internet Explorer 8 will support the features tested in Acid2. It seems as though major browsers support, or are striving to support, the standards tested in Acid2.

In 3 March 2008, the Web Standards Project, released their third test, Acid3 (`http://www.webstandards.org/action/acid3/`). The Acid3 test was designed for testing standard specifications that are used in Web 2.0 [WSP08b]. Flaws in the implementation of standards related to DOM scripting, such as ECMAScript and DOM 2, XHTML 1.0, parts of CSS2 and CSS3 and SVG (Scalable Vector Graphics), were exposed. The test comprised 100 mini tests and exposed flaws in all tested browsers including Internet Explorer, Firefox, Opera and Safari. I tested various web browsers against the Acid3 test and none of the installed browsers obtained a full score on the test. The results from the Acid3 test can be found in Appendix B. The browsers that obtained the best result was Safari 3.1 on Mac OS X 10.5.2 which scored 74/100 points. Firefox seems to improve its result on the test from version to version.

No web browser achieved the full score (100 out of 100) when the test was released [Sto08]. However, browser creators worked hard to develop the first version to pass the test.

On Wednesday 26 of March 2008, developer versions of both Opera and WebKit got a full score on the Acid3 test and became the first two browsers to pass the test [Alt08, Sta08]. A downloadable version of WebKit that passes the Acid3 test was available shortly afterward.

Since the mobile browsers in Nokia Series 60 phones and iPhone/iPod Touch are based on WebKit and Opera develops popular browsers for mobile phones it seems that advanced Web 2.0 browser features will soon come to mobile web browsers as well. I believe mobile browsers will become very capable within a short period of time and that both basic and advanced web standards will be supported on both mobile and desktop web browsers.

4 Web Usability

This section introduces usability, aspects of web usability, mobile web usability and visualisation aspects related to usability.

4.1 Usability

There are two major quality attributes of user interface design [Nie03]. One is usability, and the other is utility. Utility refers to the functionality of the design. If the functionality of a program is not what the user wants it has no bearing it is easy to use. On the other hand if a program can do what the user wants, but is too difficult to use, it is no good either.

According to [Lew95], usability depends on the product in use and the context it is used in. Jakob Nielsen [Nie03] defines usability as a quality feature that consists of five quality components: learnability, efficiency, memorability, errors and satisfaction. These components will be looked at below in further detail.

Learnability defines how easy is it for a user to accomplish his tasks when he is first faced with the design [Nie03]. According to [Har01], humans have a tendency to learn quickly, remember well and to overcome difficulties in clever ways. These attributes can be taken advantage of when designing material that is to be learned, such as user interfaces. According to [Har01], learnable material should be memorable, logical, reconstructible, consistent and visual.

Efficiency defines how quickly the user can start to perform tasks once he has learnt the design [Nie03]. There is often a conflict between learnability and efficiency [LH01]. The rule of thumb is to design user interfaces that are efficient, later adding learnability [LLK06]. A user interface with bad efficiency often breaks the whole user interface. If, however, the interface has first been made efficient, then learnable, both attributes can be preserved.

Web pages that require much of the users' working memory cause efficiency problems because human working memory is very limited [LLK06]. Working memory is a bottleneck for human problem solving and information processing capacities for two reasons. Firstly, the storing period is short, only a few seconds, and secondly it has a limited capacity of only 4-7 items called chunks of information that can be kept in the working memory at the same time. Large chunks of data can be reduced to smaller sets by reorganising them into logical groups that are more comprehensible.

Memorability defines how easily a user can regain his skills when he has not used the design for a period of time [Nie03]. Memory is closely connected to association and there are different types of memory [Har01]. There is:

- Short term memory which stores a short shopping list, for example, which will be forgotten after a while.
- Medium term memory is where things one should remember for next week, for example, are stored. Reminder notes and association are used to remember things in the medium term memory.
- Stored in our long-term memory are things that we will remember for life. Some things we learn by brute force methods to be able to recall them rapidly, letters of the alphabet or the multiplication table, for instance. Other things may be learned by frequent use, such as playing an instrument or driving a car.

Errors define how many errors the user makes, how severe they are and how easily the user can recover from these errors.

Satisfaction is defined by how pleasant the design is to the user.

4.2 Web Usability

Nielsen [Nie99] argues that content is the reason web users go online. However, as the use of the Internet has shifted from being an information repository into a social meeting place, I claim that this it is not the only reason why users go online. I also claim that users contribute content as well as consume it as they communicate with others or publish content online. Nielsen continues to argue that content is the first thing that a user looks for when they enter a web page, and that it is one of the two most important features of a web page. The other is how easily users can find the information they desire, and that web users are goal-driven and very impatient. That is why content must be strongly oriented towards providing fast answers and being useful. I agree with Nielsen in these last statements. I would like to add that in addition to providing the information users desire, developers should seek to identify the users' goals and provide easy ways of achieving them.

From the very first moment users enter a web site they experience the usability of that site [Nie99]. That is one of the reasons why usability is so important for web pages. Users tend to leave the site if the information is difficult to read, the web page does not state what the users can do, they get lost, or if the website does not answer their key questions. Users do not read a manual of a website, and they do not spend much time trying to learn an interface. Since there are many other websites, according to [Nie99], leaving the site is the first defence mechanism when a user encounters difficulties.

One web usability issue that affects how pleasant a web application is to use is response time [Nie99]. Research shows that in order for the user to freely navigate through an information space, the response time needs to be under one second, and that users cannot keep their focus for longer than 10 seconds while waiting for a page to load. If the response time is below one tenth of a second, users feel that the system is reacting instantaneously. If this occurs then no feedback is necessary, other than displaying the result. When the response time is less than one second, the users flow of thought will remain uninterrupted, even though they may experience a delay. According to Nielsen [Nie99], feedback is not necessary if the response time is between 0.1 and 1.0 seconds, but the user can loose the feeling of operating directly on the data. If the response time is longer than 10 seconds the users will start doing something else while the page loads.

4.3 Mobile Web Usability

Mobile web browsing is different from desktop web browsing [KR03]. Text input is much slower on a mobile device, there might be no cursor for interacting with objects and some devices may not support horizontal scrolling. The amount of cookie data that can be stored on the devices is usually limited and the user's mobile Internet account may be such that the user has to pay for the amount of data that is transferred. In addition to these restrictions, there is a great variation among mobile devices in terms of screen size, browser capabilities, input methods and processing power. These attributes are necessary to consider when designing web applications for mobile devices. According to user studies, users value different aspects when browsing on a mobile device than on a desktop computer [VRM03]. Because of the differences in platform capabilities, and user expectations, not all usability guidelines intended for desktop web browsing can be applied to mobile browsing.

Large content does not fit into tiny screens. In order to view content in its original format, paging, scrolling or zooming is required. It has been found that scrolling in mobile web pages has a negative effect on user's task performance efficiency and it makes comparing and reading data difficult [CSA07]. User performance increases when large content is divided into several pages. The optimal page length depends on whether the page is meant to be informative or interactive. An intuitive approach for viewing large content on small displays is to show a minimised view of the page where users can zoom in on the desired content using a dedicated controller on the device [Rot06]. However, most mobile phones do not have a dedicated zooming controller, so other methods need to be found.

Some mobile browsers provide two modes for presenting web content, Narrow Layout and Original Layout [Rot06]. Narrow Layout eliminates horizontal scrolling by reformatting content so that it fits into a single column and only vertical scrolling is necessary. However, not all web content is comprehensive in this format, wide data tables or large maps for example. Therefore browsers provide a second mode, Original Layout, for presenting content in the same way as on a PC. There are, however, problems with these approaches. It is the user's responsibility to change between the modes, but user studies have found that most users do not know about these modes. Large data tables require Original Layout, but wide text columns have only good readability in Narrow Layout.

Baudish et al. [BXWM04] address the issue of visualising large content on small displays through a method called Collapse-to-zoom. Collapse-to-zoom allows users to zoom into relevant areas, but also allows users to collapse areas that they find irrelevant, such as menu-columns, archive material or advertising. When content is collapsed, all remaining content expand in size, revealing more detail, which increase the user's chance of finding content of relevance.

Lam and Baudisch [LB05] address the problem with thumbnail views, where texts become unreadable. They present an enhanced thumbnail view, called Summary Thumbnails, with additional readable text fragments. The method helps users to identify viewed material and to distinguish between areas that are similar. The method proved to be 41% faster, with 71% less error rates than interfaces with Narrow Layout.

Roto et al. [Rot06], present a mode-less solution for displaying web pages on small displays called MiniMap. Using MiniMap, all pages are viewable in the same mode. The solution is to modify web pages so that they are similar to the original layout but more suitable for small screens, making text columns the same width as the screen for instance, and showing a page overview when the user scrolls continuously. These approaches are intended for viewing standard web pages on mobile devices. However, when web applications are designed for mobile devices, the aspects of mobile browsing can be taken into consideration in the application design.

According to user studies, mobile web users prefer to navigate content that is presented in a list format so that detailed information can be acquired via items in the list [KR03]. Scrolling has been found to be suitable for skimming and scanning tasks in order to get an overview of the content [CSA07], but task performance increases when page hierarchies are shallow [KR03, CSA07].

A user study of mobile XHMTL navigation conducted by Kaikkonen and Roto [KR03] found that unique page titles were important for communicating the user's current location. For menu selections, less errors occur if interaction feedback is provided [CSA07]. Keyword search was found to be popular for finding content, and is often utilised by users of small screen devices [KR03, CSA07]. The browsers back button was found to be the most important way of going back when navigating a mobile XHTML page [KR03]. This may become a problem when developing web user interfaces that are dynamically updated without page reloads, because history handling of current mobile or desktop browsers is not intended to track different states in a dynamically changing document. For users, however, it is intuitive to think that the back button takes them one step backwards in the history of changes that have occurred.

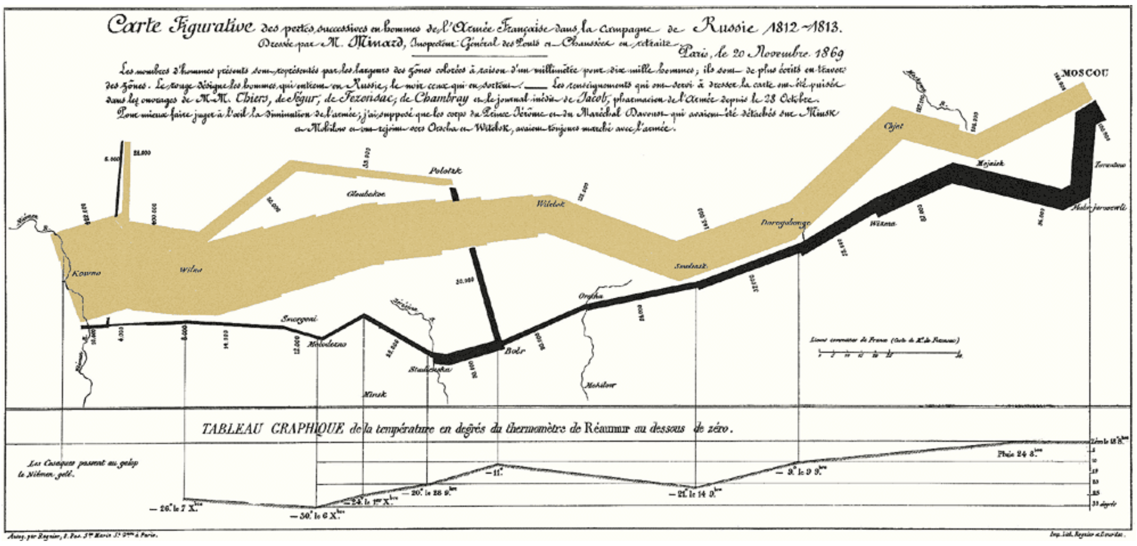


Figure 1: Portrait visualising the losses of Napoleon's army during the Russian campaign of 1812 [Tuf01]. From the Polish-Russian border goes a thick band that shows the size of the army at each position. The retreat from Moscow during the cold winter is visualised by the dark lower band and is connected to time scales and temperature levels.

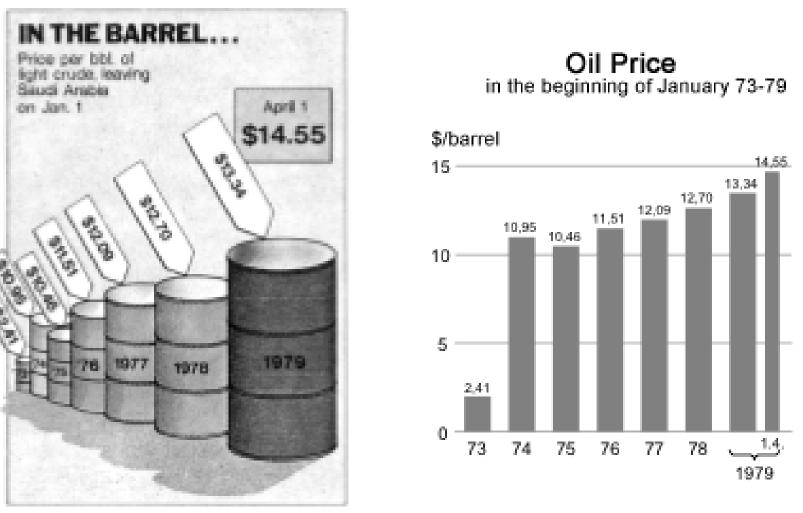
4.4 Usability and Data Visualisation

Due to the limited mobile web usability and visualisation guidelines for modern browsers, this section introduces the principles behind visualisation and certain aspects regarding human perception in order to allow user friendly interfaces and intuitive data visualisations on mobile web browsers.

Statistical graphics

According to Tufte [Tuf01], the best statistical graphics are able to communicate complex ideas with clarity, precision and efficiency. One example of this, and in Tufte's opinion probably the best statistical graphics ever drawn, is Charles Joseph Minard's portrait of the losses suffered by Napleon's army during the Russian campaign of 1812; see Fig. 1.

Statistical graphics should not emphasise or have the user focus on anything else than the data. The graphics should direct the user's attention towards the substance that is presented. The graphics should not distort the data by exaggerating or decorating the data; see Fig. 2. Instead the graphics should be able to make a



(a) (b)

Figure 2: Graph (a) is an example of a statistical graphic with too much coding. Graph (b) shows the same data without extra coding [Tuf01, LLK06].

logical representation of the data, encouraging the viewer to compare different parts of the data. Tufte recommends representing the data at different levels of detail, ranging from a general overview to finer details.

Visual Perception

When we experience a number of stimuli, they are not perceived as "a number" of separate things, but they are organised and grouped together according to certain principles. These principles are called Gestalt laws [Wer38].

Consider the shapes A-C in Fig. 3. They consist of circles, all of the same shape, with equal distance from neighbouring circles. However, the same circles seem to make up columns in shape B and rows in shape C [Wer38]. Our mind arranges them into natural groups. This phenomena is called the *Factor of Similarity*, implying that similar objects group together.

In shape D, the elements appear to form columns because of the differently shaped elements. In shape E the same rule is applied and the elements appear to form rows. However, in shape F, the red elements appear to make up one group whereas the blue elements appear to make up another. According to Healey [Hea07], similar colour is a stronger grouping factor than similar shape.

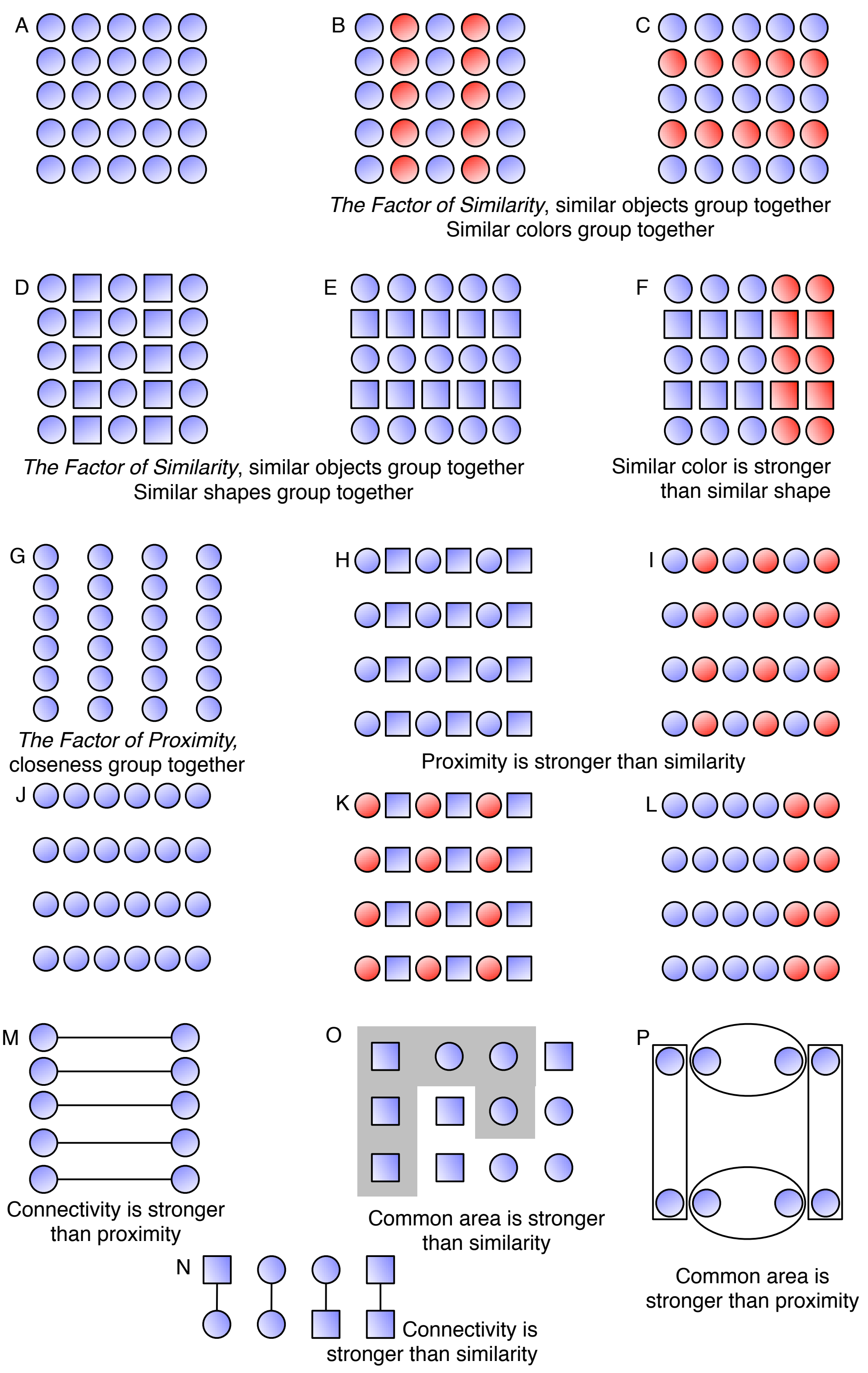

A
B
C
The Factor of Similarity, similar objects group together
Similar colors group together
D
E
F
The Factor of Similarity, similar objects group together
Similar shapes group together
Similar color is stronger than similar shape
G
H
I
The Factor of Proximity, closeness group together
Proximity is stronger than similarity
J
K
L
M
O
P
Connectivity is stronger than proximity
Common area is stronger than similarity
Common area is stronger than proximity
N
Connectivity is stronger than similarity

Figure 3: Gestalt laws. [LLK06, Wer38, Hea07].

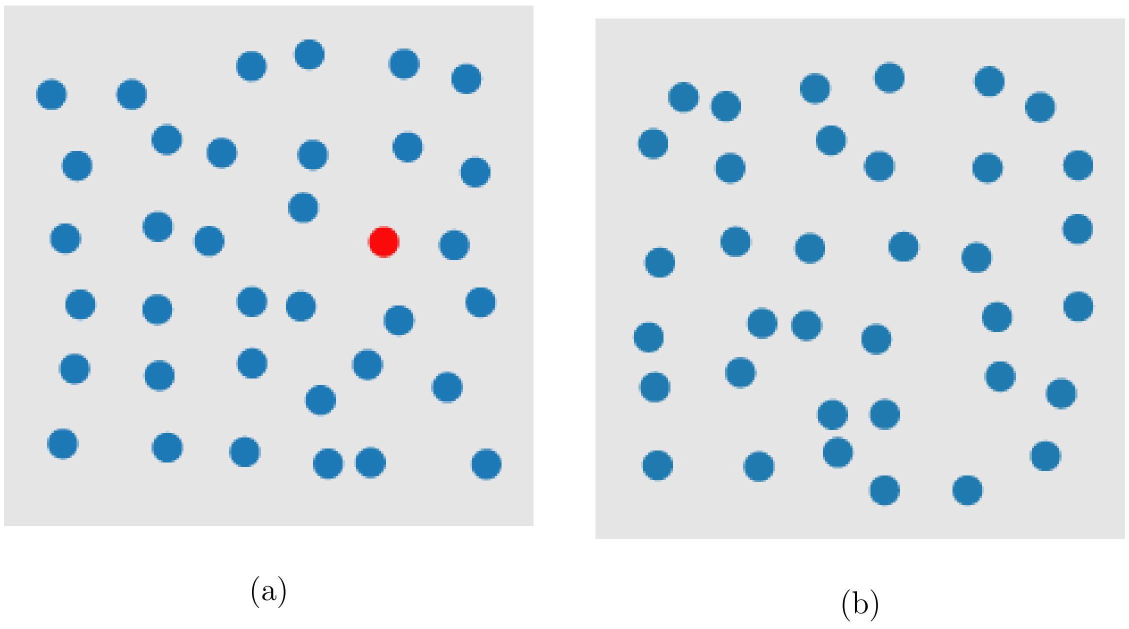


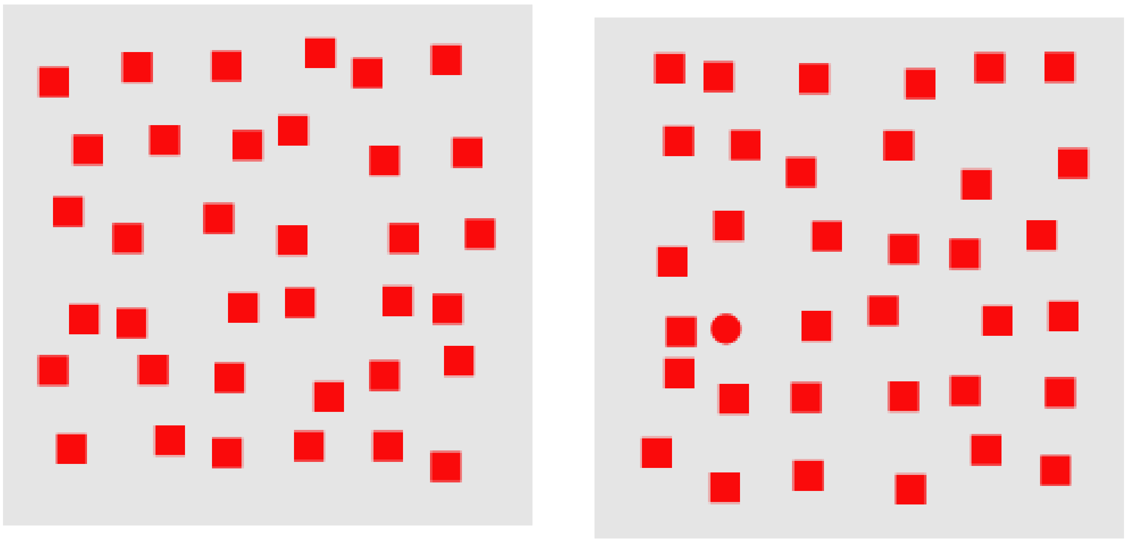
Figure 4: Determining whether a red circle is present or absent in a sea of blue circle distractors: (a) target is present; (b) target is absent [Hea07].

In shapes G-L, the *Factor of Proximity* can be observed [Wer38]. In shape G the circles are perceived to be organised in columns, whereas in shape J the circles appear to be organised in rows. Even though the *Factor of similarity* is applied to the circles, which can be seen in shapes H, I, K and L, the *Factor of Proximity* is stronger and the elements appear to be organised in rows [LLK06].

In shape M, the adjacent columns are joined together with a line. This connection is stronger than proximity, even though the circles are closer together in the columns, the circles appear to be grouped together according to the connections [LLK06]. Connectivity is stronger than proximity.

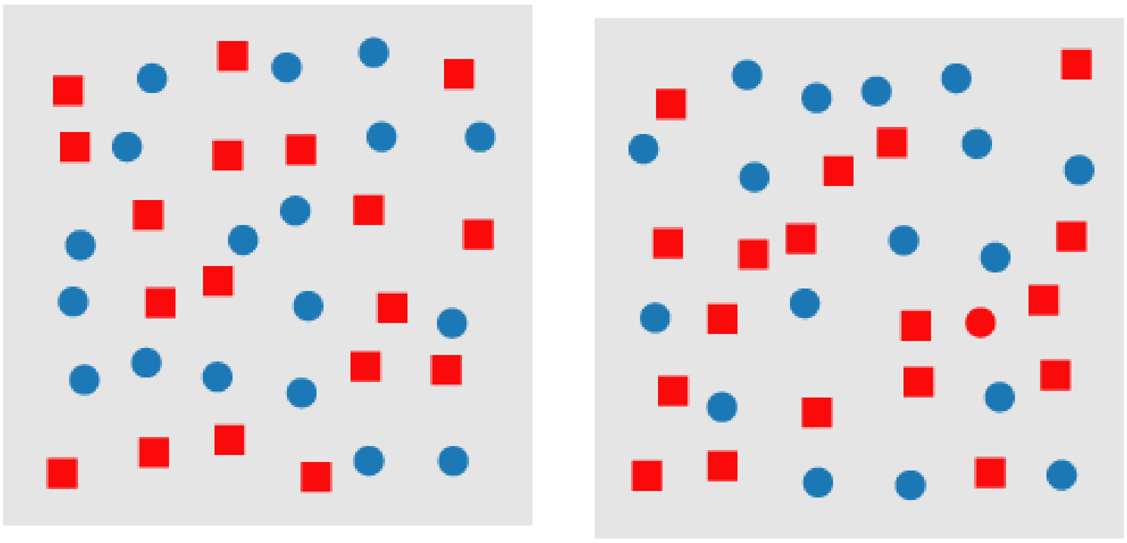
In shape N it can be seen that line connectivity is also stronger than similarity [LLK06]. The elements group together according to the connections rather than objects of the same shape. In shapes O and P it can be seen that a common area is a stronger grouping factor than both similarity and proximity [LLK06].

Our low-level visual system has the ability to recognise and extract certain features from what we see immediately, even before we focus on the image. This is called preattentive processing [Hea07]. Observation tasks that can be performed between 200-250 milliseconds on a large display are called preattentive. It takes at least 200 milliseconds before the eyes begin to move, meaning that the user has not focused on any particular part of the image. Even so, it has been observed that viewers can



(a) (b)

Figure 5: Determining whether a red circle is present or absent based on difference in shape: (a) target is absent; (b) target is present [Hea07].



(a) (b)

Figure 6: A combined search for a target red circle: (a) target is absent; (b) target is present [Hea07].

perform these tasks with little effort.

An example of a preattentive task is finding a red circle among blue circles; see Fig. 4. Similarly, in Fig. 5, our low-level visual system recognises differences in shape and therefore finding a circle among squares can be performed preattentively. However, when variations both in shape and colour appear and there are no unique features that the visual-system can extract the search is no longer preattentive.

Fig. 6 is an example of a combined search when trying to find a red circle among red squares and blue circles. The visual system has no unique visual property to search for when the target is to be located. When a circle is searched for, the visual system returns blue circles and when a red item is searched for, the visual system always returns red squares. Studies have shown that the target cannot be found preattentively [Hea07]. The viewer must conduct a time consuming serial search in order to determine the presence of the red circle.

Based on the understanding that the low-level visual system can rapidly identify certain visual properties, in the design of multidimensional visualisations, the designer can assign different visual features to different data attributes [Hea07]. By doing this, multiple data can be shown simultaneously in a single view. One requirement to facilitate search is that data features are mapped so that the visualisation does not produce interference, i.e., different visual features do not hide or mask information; see Fig. 6. In order to allow rapid search for combinations of data attributes, one must ensure that the resulting combinations contain at least one unique visual feature, so that identifying the elements can be performed preattentively [Hea07].

Colour and information

Humans are sensitive to colour variations and trained colourists can distinguish among one million colour variations under ideal circumstances [Tuf90]. An average person can distinguish approximately 20 000 different colours, where constraints are due to visual memory rather than the ability to differ between similar colour tints. Can colour be used to code data? When it comes to coding abstract information using colours, it has been shown that using more than 20 or 30 colours gives a negative effect, making data difficult to distinguish [Tuf90]. It is a difficult and complex task to connect the right colour to the right place when using colours in data coding. The fundamental uses of colours in data visualisation are (1) to label where colour is used as a noun, (2) to measure where colour represents quantity, and (3) to represent reality and enliven or decorate where colour is used to bring beauty.

Colours used in visualisation of data have a tendency to generate graphical puzzles since the mind does dot give a visual ordering to colours, even though we are familiar with the colour spectrum from rainbows and science textbooks [Tuf01].

The use of different shades of grey can show quantity better than using colour, because they have a visual hierarchy [Tuf01]. Multiple layers of information can be created by providing multiple viewing depths and multiple viewing angles. Graphics can have three viewing depths where one represents what is seen from a distance, and is an overall structure aggregated from the underlying microstructure. The second depth is what is seen close up, showing finer detail. The third depth shows the underlying data. Other examples of colour scales in use are the rainbow spectrum, red-blue or red-green scales and grey-red saturation values [Hea07].

Approximately 8% of caucasian males have some kind of colour sight deficiency [Wol99]. This affects which colours are to be chosen when choosing a palette. In [Wol99], a lookup table is provided for web designers with normal colour sight so they can see how persons with different colour deficiencies see the colours. The lookup table is available from `http://www.internettg.org/newsletter/mar99/color_challenged_table.html` [retrieved 14-05-2008].

Other attempts in producing colour scales include techniques where the colours are chosen on the basis of how viewers perceive them rather than their separation from each other in the RGB (Red-Green-Blue) colour scale [Hea07]. According to [Hea07] these techniques are improvements that allow balance among colours so that each step in a colour scale produces the same amount of colour change, each colour is equally distinguishable from all the other colours and no specific colour is easier or harder to identify. The improvements also allow for flexibility where the colours can be selected from any part of the colour space, and is thus not restricted only to greens, reds and blues.

Data visualisation on mobile devices

It is now possible to access large amounts of data using mobile devices [Noi05]. Visualising large amounts of data is more difficult on mobile devices than on personal computers because of the limited memory, processing power and screen size of mobile devices. Even if memory, processing power and screen quality improve, the size of mobile phone screens will remain small for it to be possible to carry the devices around. According to [Noi05], it is not possible to just adapt representation meant for the personal computers onto mobile devices. New solutions need to be found

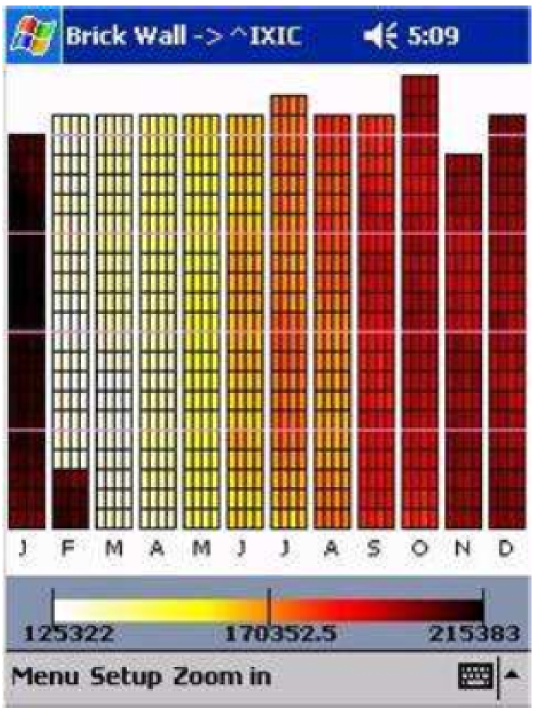
When there is much information to be visualised, everything may not fit on one screen due to the limited screen size [Noi05]. One solution is to split the information into several pages. This, however, results in problems with navigation and orientation. Considering the display of searching tasks, it is suggested that the amount of page-to-page navigation needed to view search should be kept to a minimum. It has been observed that the additional effort arising when the user needs to scroll vertically has a lesser effect on user performance than that of page-to-page navigation. It is recommended that the user interface should adapt for vertical scrolling, since users are more comfortable with scrolling vertically than horizontally [Noi05]. When users need to select items from a list, and the items are familiar to the user, 15 items are still acceptable. When the items are unknown there should be no more than eight [Noi05].

Problems appear differently for devices with small screens (pocket PC's) and very small screens (mobile phones) [Noi05]. On a pocket PC the information can be summarised on the screen where more details can be viewed by opening additional windows. On a mobile phone, however, it is recommended that the visualisation should be simplified. When there is much data, it is recommended that analytical methods are used to reduce the amount of information before visualising it [Noi05].

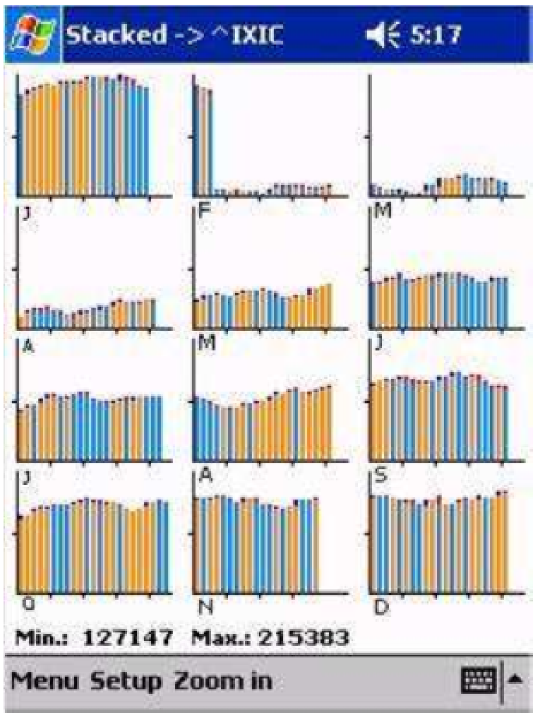
The size and layout of pointing devices and keyboards on mobile devices vary from model to model and have different effects on the usability of the device for certain tasks. A case study of presenting time series data was made by [Noi05]. Stock visualisations intended for large screens are usually not very readable on a mobile device. In order to save space when visualising on a small display, [Noi05] recommends that precise and concise information should be given, using as few words as possible and using a vocabulary that the user understands.

Often graphical representations on PDA's are miniatures of the visualisations used on PC's. However, it is not enough, if an overview of the data can be shown, since it cannot provide enough details on a small screen. Noirhomme-Fraiture et al. [Noi05] presents two methods for visualising time series data. One utilises bar chart visualisation and the other uses colours to indicate change.

The first, called a *Brick Wall Chart*, represents the data values using scales of colour; see Fig. 7. The colour range is white, yellow, red and black. White represents the minimum value and black represents the maximum value. There are 12 columns representing each month. Each column is divided into rows that represent the days in each moth and each row is divided into 4 sections representing 4 different attributes,



(a)

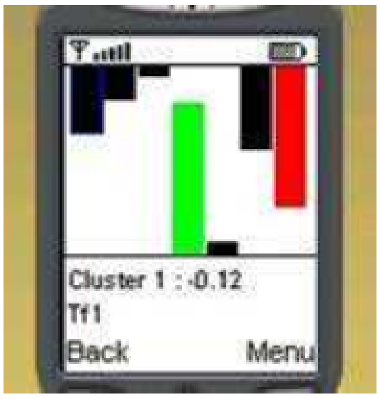


(b)

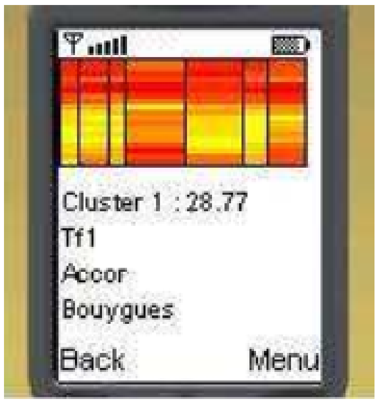
Figure 7: Two different ways of displaying time depended data on a PDA, (a) Brick Chart Wall and (b) Stacked Bar Chart. Both were just as effective but 17 out of 20 test users chose the Brick Chart Wall as their favourite [Noi05].

colour coded according to their values. The user can get the precise values of the attributes for each day by tapping on the screen over the desired day. A box then opens showing the values of the four attributes. The other, called a *Stacked Bar Chart*, divides the months into 12 parts using the metaphor of a calendar where each part contains its own bar chart showing time on the x-axis and value on the y-axis; see Fig. 7. The bar chart represents the values of the attributes for the different days by showing them as elements with different height dependent on the value. This visualisation can include two different attributes that are coded in blue and orange. The attribute with the smaller value is placed in front of the attribute with larger value. This visualisation is not as rich as the Brick Wall Chart, but it uses elements that are familiar to the users.

These two methods of visualising time series data were evaluated in order to determine the efficiency of the visualisations and user preference [Noi05]. The measure attribute for the efficiency was performance, based upon how long it took the users to complete a task and how well the days and the values were represented. The



(a)



(b)

Figure 8: Two different ways of displaying time depended data on mobile phones. (a) Alternate Bar Chart and (b) Pixel Bar Chart [Noi05].

results were inconclusive as to which was better, but 17 out of 20 users chose the Brick Wall Chart as their favourite visualisation. The Brick Wall Chat was thus more appreciated than the Stacked Bar Chart.

On a very small screen, however, detailed graphics are not readable and it is not advised to solve the problem by adding pop-up windows [Noi05]. Outlines are to be avoided and pictures replaced with coloured or greyed surfaces. Some Gestalt laws have shown that contrast boundaries are better than line boundaries for making shapes stand out. If there are too many values, [Noi05] recommends to summarise the values through clustering. If there exists, for example, a portfolio with many different stocks, one solution is to display a clustered result of these stocks. Many stocks have common behaviour so they can be assembled into classes with similar behaviour. In the following visualisation examples that were designed for mobile phones instead of PDA's, [Noi05] merged 40 stocks into 7 clusters. Clustering was also performed to reduce the number of days into fewer periods.

The first example, *Alternate Bar Chart*, displays a visualisation close to that which workers in the stock exchange are familiar with; see Fig. 8 (a). The bar heights are proportional to the value being represented. In this case, the maximum variation of stock value during a period. This value can be both positive and negative. If the value is positive, it is represented from bottom to top, and if negative, the value is represented from top to bottom. Each bar represents a cluster of stocks. For more details about a certain stock cluster, the user can point on a specific bar. The screen space usage is optimised and the important negative vs. positive changes are quickly visualised.

The second example, *Pixel Bar Chart*, visualises the evolution of values for different clusters of stocks (or individual stocks); see Fig. 8 (b). Each cluster has its own column where the width of the column is proportional to the number of stocks in the clusters. Each day is represented from top to bottom as a line in the column, and a colour scale is used to map daily changes. The colours are the same as for the Brick Wall Chart example. This visualisation also uses maximum available screen space. People with weak sight might have problems with this visualization, although the authors hope that improved screen quality will reduce the problem.

These two methods of visualising stock data on mobile phones were tested on users. Both visualisations had rather good performance, but according to the authors, the tasks to be executed were simple. The preference of the users showed that the Alternate Bar Chart was easier to learn since it was close to the standard representation, whilst the Pixel Bar Chart required some time for training.

5 Service Aggregation

An aggregate is a whole formed by several elements [Oxf01]. Combining content from different web pages into so called aggregated documents is a common phenomena in the Internet today [SDJ+08]. Millions of blogs and a huge amount of online newspapers and websites provide their content in a machine readable format. This is accomplished by providing feeds that enable users and applications to access data. The goal of the thesis is visualising online activity on mobile devices. In order to visualise online activity, data, such as contributions, events, comments and updates, need to be gathered from different online services where the users' friends are active. This section examines the means of acquiring, combining and integrating such data.

5.1 Techniques

RSS and Atom

RSS is a collection of XML (eXtensible Markup Language) formats that summarises the contents of a webpage [BGS06]. The concept emerged in 1997 and the first version of RSS emerged in 1999 [GMR07]. Today the term Really Simple Syndication is used most commonly for RSS. Several versions of RSS exist due to forks in the

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 732-1234",
        "646 123-4567"
    ]
}
```

Figure 9: Data represented in JSON format.

development of the standard. The common versions in use are RSS 1.0, RSS 2.0 and Atom. Each has its specific uses as well as advantages and disadvantages for feed publishers. RSS is especially used by sites where content is added regularly, for instance news sites and weblogs, and the amount of sites that support RSS is increasing rapidly [BGS06].

RSS and Atom are widely established within the blogging community [Ham05]. There are millions of weblogs, personal online diaries, being written worldwide and most of these produce a syndication feed such as RSS or Atom.

We use the term RSS to refer to RSS 1.0, RSS 2.0 and Atom unless explicitly stated.

JSON

JSON (JavaScript Object Notation) is a data-interchange format that is lightweight, easy for humans to read and write and also easy for machines to parse and generate [Jso08]. JSON is based on a subset of the JavaScript Programming Language. JSON is an ideal format for exchanging data because it is entirely language independent, but it uses elements that are familiar with programming languages such as C, C#, C++, Java, JavaScript, Perl, Python, PHP or other

JSON is built of two structures, a collection of name/value pairs and an ordered list of values. Fig. 9 shows an example of data represented in JSON format.

JSON is convenient as a data-interchange language for JavaScript because JSON in itself is JavaScript. That makes it easy to parse because the text can be evaluated producing a JavaScript object. The following piece of code is one way of creating a JavaScript object out of a JSON feed. We assume here that the feed is stored as a string in the JSON_text variable.

```
var p = eval("(" + JSON_text + ")");
```

However, if an application is using JSON in an untrusted environment, it is not advisable to evaluate the JSON code directly but rather to parse it, since malicious code could be injected into the feed causing harm and security vulnerabilities.

AJAX

AJAX (Asynchronous JavaScript and XML) consists of a set of standardised web technologies developed in the nineties that are used today for making rich, responsive and interactive web applications [Pau05].

In [Gar05], AJAX is described as follows:

> "AJAX is not a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. AJAX incorporates:
>
> - standards-based presentation using XHTML and CSS
> - dynamic display and interaction using the Document Object Model
> - data interchange and manipulation using XML and XSLT
> - asynchronous data retrieval using XMLHttpRequest
> - and JavaScript binding everything together."

An example of an AJAX site is Google Maps (http://maps.google.com). When new data is needed the application fetches that data asynchronously in the background and integrates the new data with a map. Some web applications containing maps require the whole page to be reloaded when new parts of a map needs to be fetched; see Fig. 10.

Most user actions in the interface of a classic web application trigger an HTTP request to a web server [Gar05]; see Fig. 11 (a). The server processes the request and returns an HTML page to the client. The original use of the web was as a hypertext medium, and for this the classic model works well. The classic model

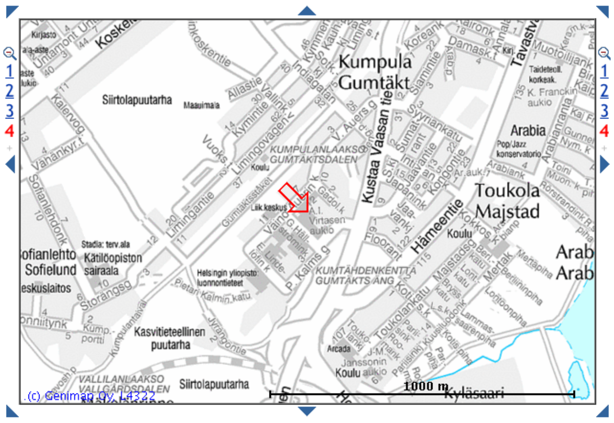
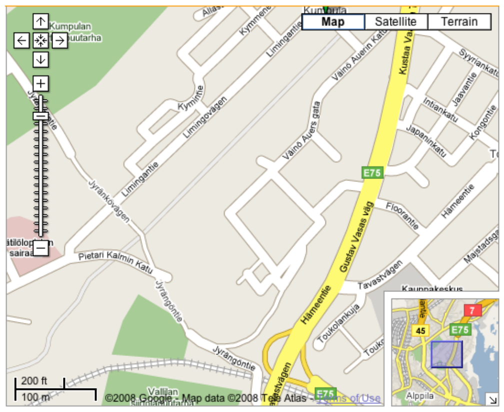



Figure 10: Two maps from different web applications, Reittiopas [YTV08], to the left, and Google Maps [Goo08a], to the right. When the user needs to see content that is just outside the visible part of the map, on the map to the left the user has to click on the navigation arrows which causes the whole page to reload, while on the map to the right the user can drag the hidden parts of the map into view.

that is suitable for the web as a hypertext medium may not be suitable for rich web applications. When the browser is waiting for a page to load, which happens whenever new data is requested, the only thing that the user can do is to wait for the page to finish loading. As usability guidelines suggest, response time needs to be under one second in order for the user to freely navigate through an information space, and in order for the user to feel that the system is reacting instantaneously, the response time needs to be below one tenth of a second [Nie99].

The nature of starts and stops that synchronous interaction causes can be eliminated using AJAX [Gar05]. Instead of loading a web page, an AJAX engine, written in JavaScript, is loaded between the user and the server. The AJAX engine communicates with the server, renders the user interface and deals with user interaction; see Fig. 11 (b). Requests go to the AJAX engine via JavaScript calls. If communication with the server is needed, e.g., fetching additional interface code, new data or submitting data for processing, the engine makes the requests asynchronously without interrupting or delaying the user's interaction.

In Appendix C, there is the source code for an AJAX call. The following is a description of the relevant parts of the AJAX call and is a simplified version of the code in Appendix C. First an XHR object is created for managing calls.

```
xmlHttp = new XMLHttpRequest();
```

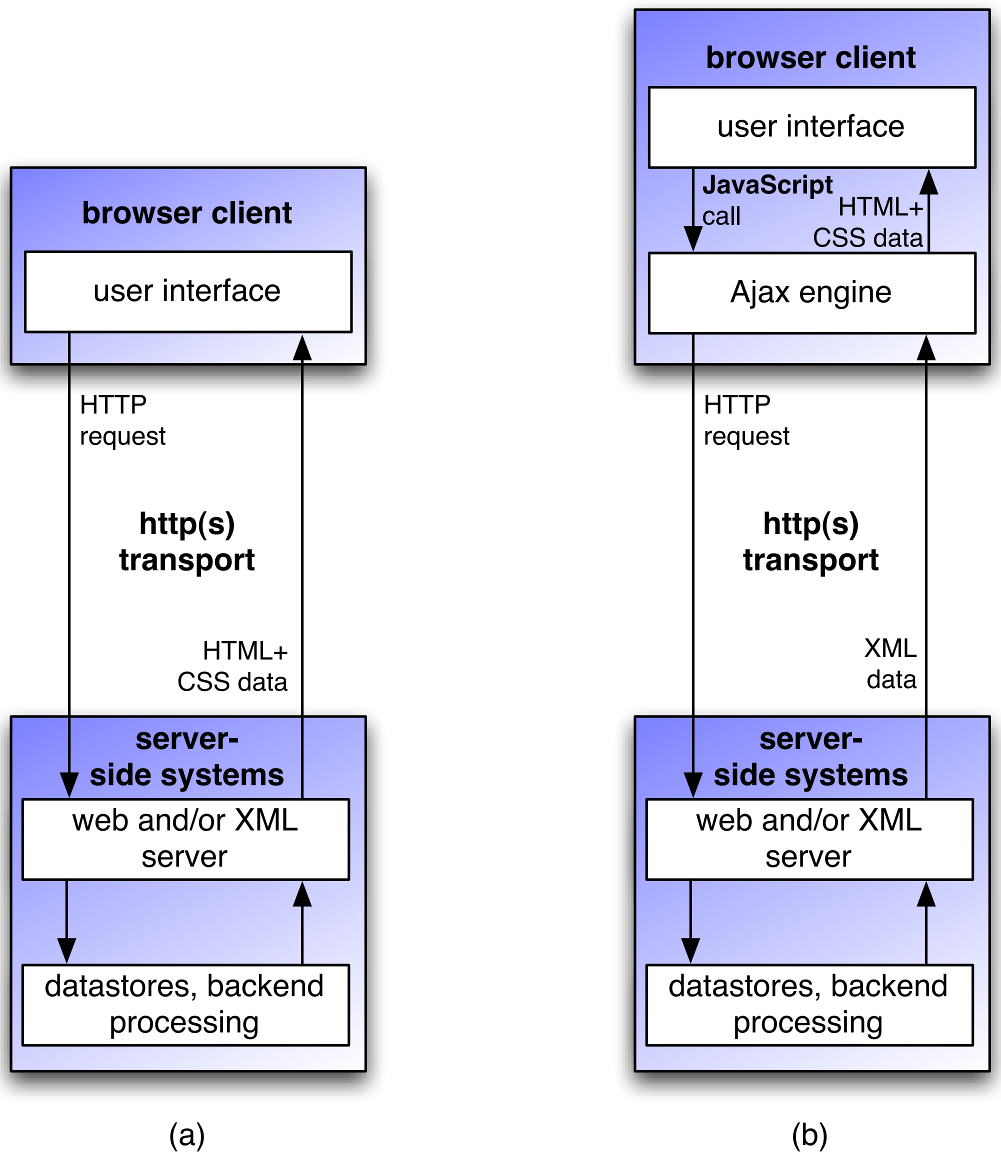


Figure 11: Classic web application model (a) and AJAX web application model (b) [Gar05].

To the xmlHttp object, a callback function is defined that will manage the content after a call has been made.

```
xmlHttp.onreadystatechange = function() {
  document.getElementById("ajax_output").innerHTML
  = xmlHttp.responseText;
}
```

After that a connection is opened calling for the contents of the "pages/index.html" file to be fetched using the GET method.

```
xmlHttp.open("get","pages/index.html");
```

After the call has been made, the `onreadstatechange` function of `xmlHttp` will be called once the content has been loaded. That function takes care of updating the web page with the new content.

JSON, XML or XHTML are the types of content that are usually fetched using AJAX calls. However, new JavaScript libraries, CSS style sheets or even images and videos may also be retrieved so that the application may fetch, for instance, needed handlers of interface elements.

Several JavaScript toolkits exist that facilitate creating web applications using AJAX. Examples of these are Prototype (`http://www.prototypejs.org`), Dojo (`http://dojotoolkit.org`), jQuery (`http://www.jquery.org`) and Frost (`http://frostlib.org`). Frost is intended for AJAX development on limited browsers such as mobile phone browsers, micro browsers and gaming consoles.

Mobile AJAX

Georgi [Geo07] defines Mobile AJAX as:

> “The use of existing web standards and technologies (XHTML, DOM, CSS, JavaScript, XHR) to create more responsive, mobile web applications that reduce bandwidth usage by avoiding full page refreshes and providing a more ’desktop application’ user experience.”

According to Georgi [Geo07], mobile AJAX is not much different from desktop AJAX and the core technology is the same. Mobile AJAX is just one of the technologies that can contribute to make the user experience better and AJAX should not be used just for the sake of using it [Geo07]. Rather, one should see Mobile AJAX as a helper technology that can enhance and build on top of traditional web applications and sites.

One problem with mobile AJAX is that not all mobile web browser currently support the necessary technologies and standards [Geo07]. However, Opera Mobile, IE Mobile, Nokia S60, Minimo, Safari on the iPhone and iPod Touch, and likely most future mobile web browsers support the needed standards. Another problem is that currently mobile devices have limited capacities in CPU performance, memory, battery lifetime and bandwidth [Geo07]. Data transmission and JavaScript operations cause heavy CPU load, which may result in two things: slow performance and battery drainage. Another thing is that currently DOM manipulation also cause a heavy load on CPU and memory. That is why Georgi [Geo07] recommends the AHAH (Asynchronous HTML and HTTP) method, which means that it should be avoided to access web services directly, but rather to route the traffic via a server-side script that outputs pre-generated HTML [Wik08].

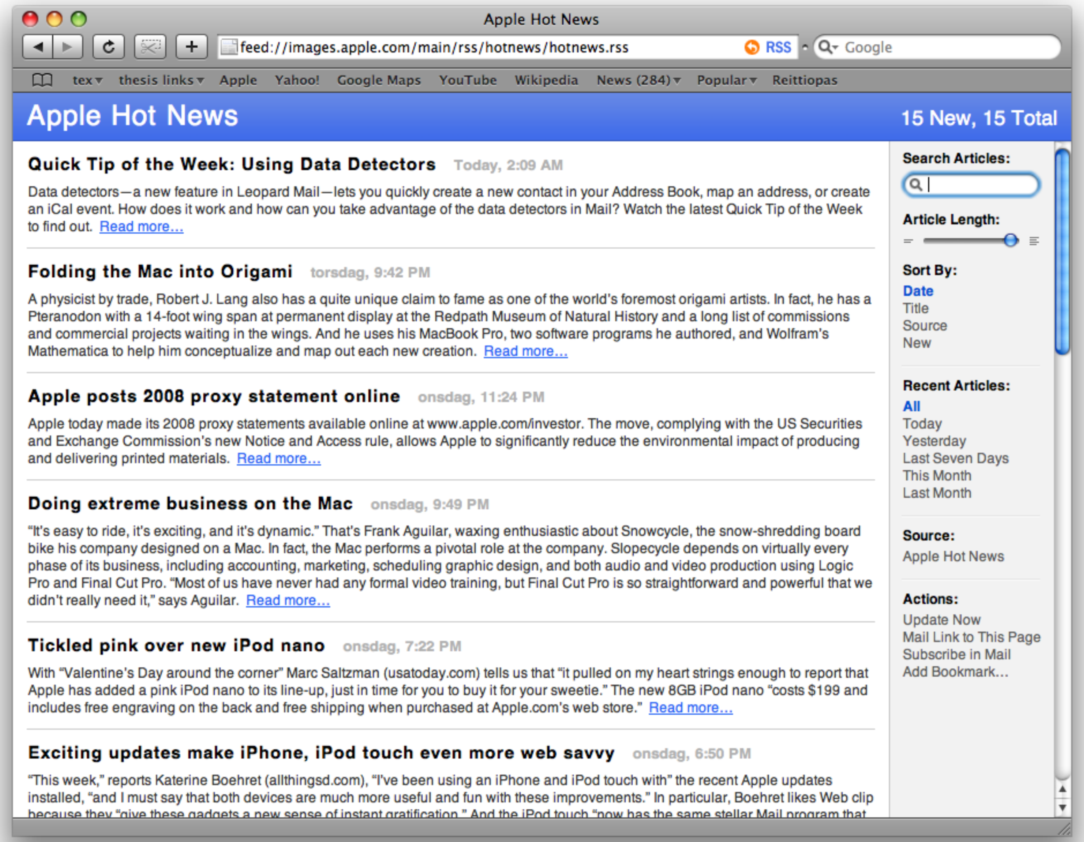
Georgi [Geo07] suggests that one approach to the fact that there are many different display sizes is to design the application using the lowest possible divisor of the display capabilities and implement the application for that. Another approach is to identify a target browser and develop the application for that.

5.2 Syndication and Aggregation

Syndication means supplying feeds that contain content in a machine readable format so that others can subscribe to the content [GMR07]. An example of this is a blog that syndicates its content as RSS feeds. Other people can subscribe to the feed using a RSS reader; see Fig. 12. When the author has written a new blog entry the subscribers are notified of the change. Syndication increases traffic to the web page and it helps to build brand-awareness for the webpage. Syndication can also enhance the web page's ranking in search engines [Ham05]. Syndication helps to maintain relationships between sites in a community and improves the relationship between the site and the users. Additional technology that may enhance syndicated services is for instance an application that observes a syndicated feed and notifies a subscriber via instant messaging.

Syndicated content can be aggregated and brought together into one place using for instance RSS readers. Different kinds of RSS readers exist enabling an overview of syndicated content. The earliest ways of aggregating and reading RSS feeds were through web-based readers. Web based RSS readers provide a convenient way for staying up to date with the content of web pages of interest without having to sit at the own personal computer. One popular example of a web-based aggregator is bloglines [Blo08], and the recently introduced Google reader[1].

[1]`http://www.google.com/reader/`



Apple Hot News

Quick Tip of the Week: Using Data Detectors Today, 2:09 AM

Data detectors—a new feature in Leopard Mail—lets you quickly create a new contact in your Address Book, map an address, or create an iCal event. How does it work and how can you take advantage of the data detectors in Mail? Watch the latest Quick Tip of the Week to find out. Read more...

Folding the Mac into Origami torsdag, 9:42 PM

A physicist by trade, Robert J. Lang also has a quite unique claim to fame as one of the world's foremost origami artists. In fact, he has a Pteranodon with a 14-foot wing span at permanent display at the Redpath Museum of Natural History and a long list of commissions and commercial projects waiting in the wings. And he uses his MacBook Pro, two software programs he authored, and Wolfram's Mathematica to help him conceptualize and map out each new creation. Read more...

Apple posts 2008 proxy statement online onsdag, 11:24 PM

Apple today made its 2008 proxy statements available online at www.apple.com/investor. The move, complying with the US Securities and Exchange Commission's new Notice and Access rule, allows Apple to significantly reduce the environmental impact of producing and delivering printed materials. Read more...

Doing extreme business on the Mac onsdag, 9:49 PM

"It's easy to ride, it's exciting, and it's dynamic." That's Frank Aguilar, waxing enthusiastic about Snowcycle, the snow-shredding board bike his company designed on a Mac. In fact, the Mac performs a pivotal role at the company. Slopecycle depends on virtually every phase of its business, including accounting, marketing, scheduling graphic design, and both audio and video production using Logic Pro and Final Cut Pro. "Most of us have never had any formal video training, but Final Cut Pro is so straightforward and powerful that we didn't really need it," says Aguilar. Read more...

Tickled pink over new iPod nano onsdag, 7:22 PM

With "Valentine's Day around the corner" Marc Saltzman (usatoday.com) tells us that "it pulled on my heart strings enough to report that Apple has added a pink iPod nano to its line-up, just in time for you to buy it for your sweetie." The new 8GB iPod nano "costs $199 and includes free engraving on the back and free shipping when purchased at Apple.com's web store." Read more...

Exciting updates make iPhone, iPod touch even more web savvy onsdag, 6:50 PM

"This week," reports Katerine Boehret (allthingsd.com), "I've been using an iPhone and iPod touch with" the recent Apple updates installed, "and I must say that both devices are much more useful and fun with these improvements." In particular, Boehret likes Web clip because they "give these gadgets a new sense of instant gratification." And the iPod touch "now has the same stellar Mail program that

Figure 12: Some web browsers have a built in RSS reader.

5.3 Mashups

Web users today have transformed from being content consumers into content providers [TSK08]. Today you can create professional looking websites and blogs without knowing HTML. That has not always been the case. The newest kinds of web tools, mashups, enable users to combine content, services and functionality from different web sources into unique services. An aggregate is a combination of data, whereas a mashup can combine data, as well as functionality, into a single tool. Mashups have become popular because of the emphasis on interactive user experience and the combination of data provided by different services [Mer06]. There are many different kind of handcrafted mashup solutions and a few different classes of popular mashups being developed [AKTV07, Mer06]. An example of mashups is AdSense where ads related to the content of the page are embedded into the webpage [AKTV07]; see Fig. 13.

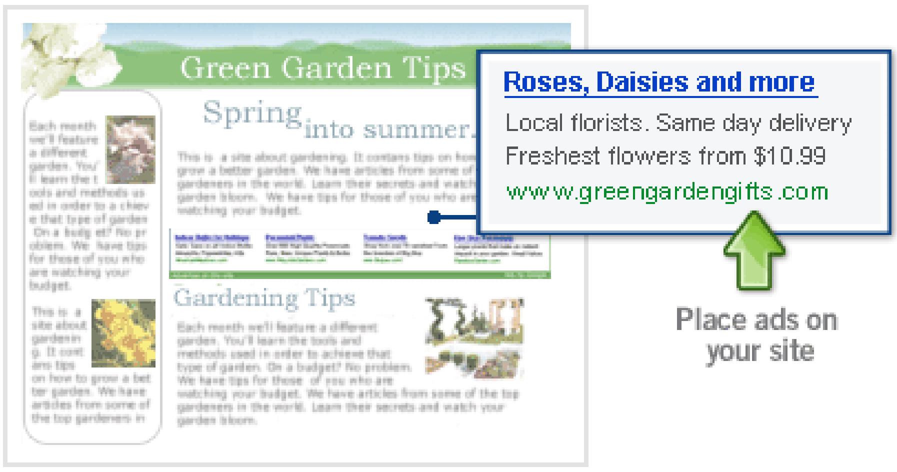


Figure 13: Ads related to the text in a page can be embedded into a webpage via Google AdSense service.

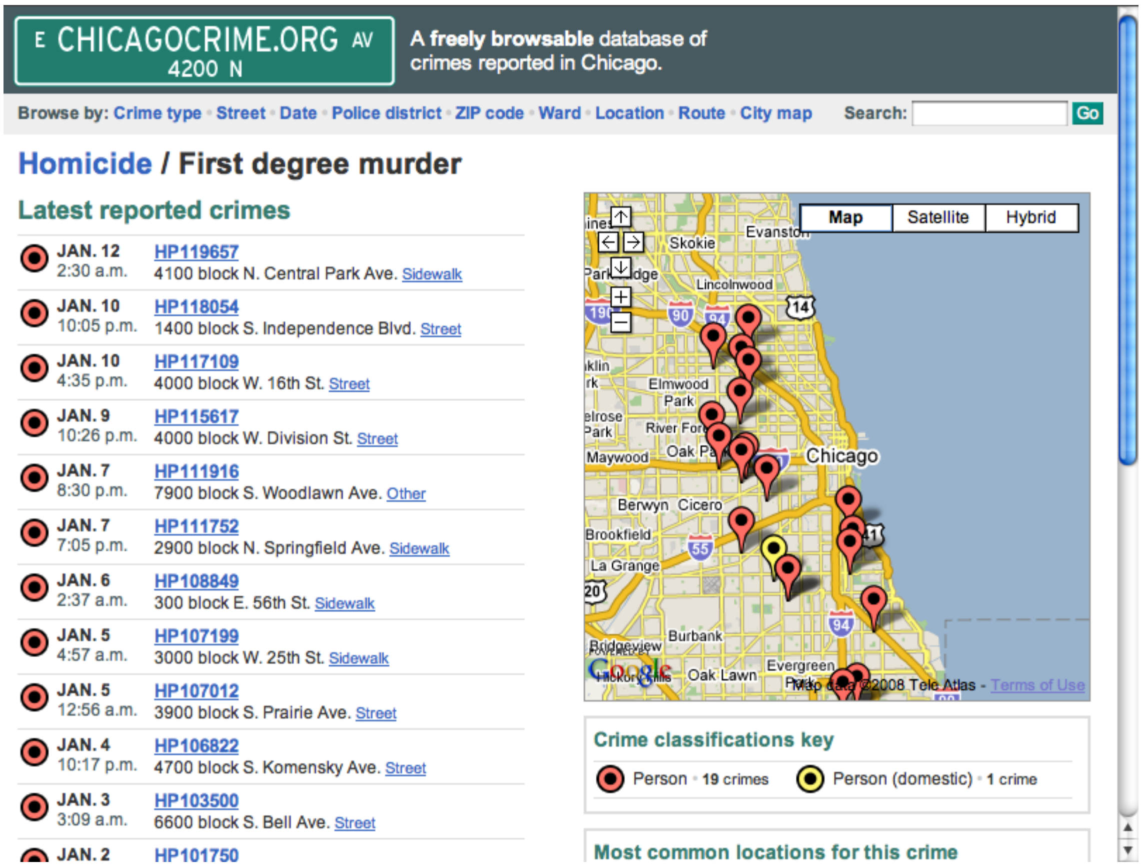
Mashups aggregate and combine third-party data, and take advantage of data sources that usually come from another site or service [Mer06]. Popular types of mashups include mapping mashups, video and photo mashups, search and shopping mashups, and news mashups [Mer06].

Mapping mashups

One factor why mapping mashups, where data are presented graphically using maps, became popular was the release of public Application Programming Interfaces (API) for cartographic imagery such as Google Maps[2], Yahoo Maps[3] and Virtual Earth[4]. People are collecting and creating vast amounts of data, and much of this data has got some location associated with it. This means that the information can be displayed on a map. Several mapping mashups exist; some are collected to `http://googlemapsmania.blogspot.com/`.

An example of a mapping mashup is the ChicagoCrime.org website which displays Chicago crime activity on Google Maps. Users can interact with the mashup site, having it for instance to display a map of south Chicago with the details of recent murders shown graphically on the map; see Fig. 14. Even though the concept and the representation are simple, the combination of crime and location information can become visually powerful. The Google Maps API describes how the developer can use Google Maps as part of a mashup [Goo08b]. The code in Appendix D

[2]`http://maps.google.com/`
[3]`http://maps.yahoo.com/`
[4]`http://www.microsoft.com/virtualearth/`



E CHICAGOCRIME.ORG AV 4200 N

A freely browsable database of crimes reported in Chicago.

Browse by: Crime type · Street · Date · Police district · ZIP code · Ward · Location · Route · City map Search: Go

Homicide / First degree murder

Latest reported crimes

JAN. 12 2:30 a.m.	HP119657	4100 block N. Central Park Ave. Sidewalk
JAN. 10 10:05 p.m.	HP118054	1400 block S. Independence Blvd. Street
JAN. 10 4:35 p.m.	HP117109	4000 block W. 16th St. Street
JAN. 9 10:26 p.m.	HP115617	4000 block W. Division St. Street
JAN. 7 8:30 p.m.	HP111916	7900 block S. Woodlawn Ave. Other
JAN. 7 7:05 p.m.	HP111752	2900 block N. Springfield Ave. Sidewalk
JAN. 6 2:37 a.m.	HP108849	300 block E. 56th St. Sidewalk
JAN. 5 4:57 a.m.	HP107199	3000 block W. 25th St. Sidewalk
JAN. 5 12:56 a.m.	HP107012	3900 block S. Prairie Ave. Street
JAN. 4 10:17 p.m.	HP106822	4700 block S. Komensky Ave. Street
JAN. 3 3:09 a.m.	HP103500	6600 block S. Bell Ave. Street
JAN. 2	HP101750	

Crime classifications key

● Person · 19 crimes ● Person (domestic) · 1 crime

Most common locations for this crime

Figure 14: ChicagoCrime.org presents crime data on a map.

outputs a map, which is zoomed to the level 15, around the location on longitude 60.205796 and latitude 24.962525 (Which is the Kumpula campus of the University of Helsinki) and puts a marker there; see Fig. 15.

Video and photo mashups

Some photo hosting services, such as Flickr[5], provide APIs to access shared photos [Mer06]. This has led to the emergence of many interesting mashups. The content providers have metadata associated with the images that are hosted. This metadata can contain information, e.g., about who took he picture, what the picture is about, when and where it was taken. Mashup developers can use the photos along with other information and match the metadata of the photos with other content. An example is a combination of the lyrics of a song with images related to the words in the song.

Search and shopping mashups

[5]http://www.flickr.com/

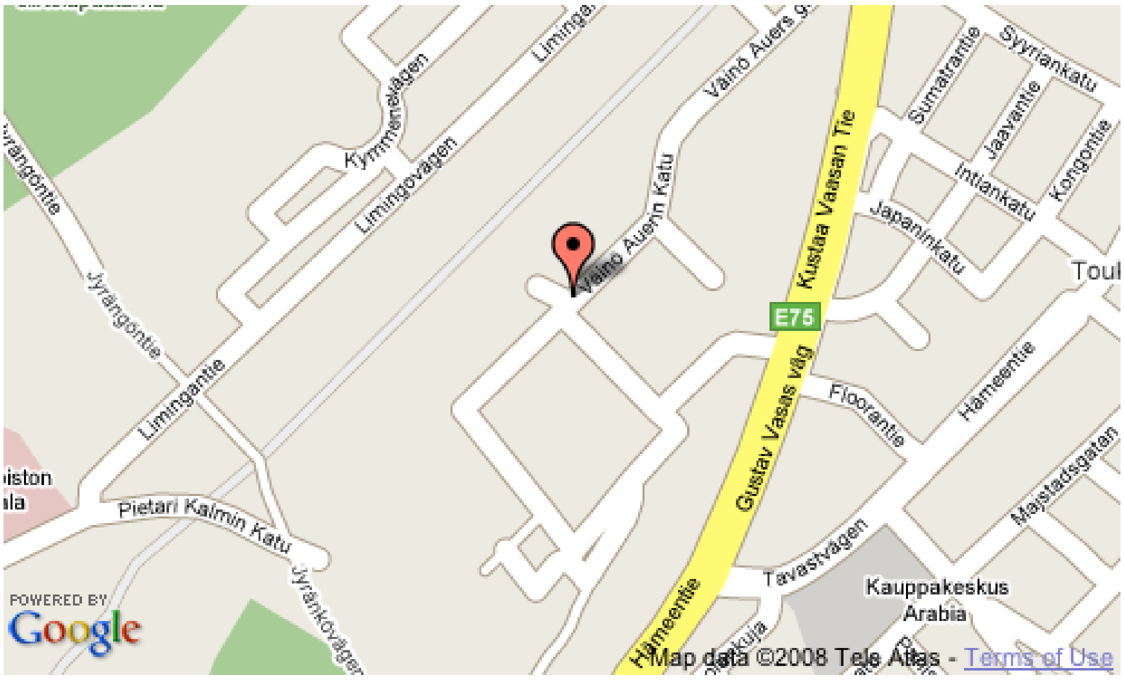


Figure 15: The example code in Appendix D outputs this map

Even before web APIs were common, and the term mashup was coined for combining web content, several search and shopping mashups existed. Services such as MySimon, BizRate, PriceGrabber and Google's Froogle aggregate price data so that it can be compared. Consumer marketplaces such as eBay and Amazon provide APIs to make it easier to create mashups and interesting web services. For instance Amazon enables mashup creators to get revenues from their mashups. A mashup developer might use the Amazons API to enable users to create wish lists of things they would like to buy or receive as gifts. If then someone via the site buys an item from Amazon, the mashup developer gets a certain percentage of the purchase in revenue from Amazon. Another example of shopping mashups is Google's AdSense, where ads that are related to the content are placed on a dedicated place on the webpage; see Fig. 13. A reseller or site owner that wants to draw customers or visitors to their site can purchase certain keywords from Google using their AdWords program. Their ads are then placed on sites that have content that relates to their ads. This enables ads to be targeted to people interested in similar topics as the ads.

News mashups

Certain news sites have been syndicating their content through RSS-feeds according to topics since 2002. Theses are for instance BBC, New York Times and Reuters.

Aggregating mashups of syndicated feeds can create personalised newspapers from the users' preferred feeds according to interests. One example of a news feed mashup is Diggdot.us that mash up feeds from Digg.com, Slashdot.org and del.icio.us.

The architecture of a mashup site

The architecture of a mashup site consists three major parts; the service providers, the mashup site and the client's browser. These will be looked at below in further detail.

- **API/service providers** The content providers may be unaware of the fact that their content is used to create web mashups. However, the providers can also provide their content for this purpose among others by providing APIs or feeds to access the content. The content may be provided through web protocols, for instance JSON, web services or RSS. Many potential data providers do not yet provide APIs for accessing the data. Mashup developers acquire information from these web sites through screen scraping, which will be introduced later in this section. The web page ChicageCrime.org uses Google respectively the Chicago Police Department as API and Service providers.
- **The mashup site** The mashup site is where the service is hosted and the logic of the mashup resides. Though, it is not necessarily where the mashup is executed. The page can be executed through dynamic content generation on the server, as well as through client side scripting such as JavaScript, or a combination of these, which is most common. Mashup applications often use data that is provided from their own user base, which implies that at least one set of data is local. Complex queries may require processing that is not suitable on the client side.
- **The client's browser** The application is rendered graphically in the client's web browser and some processing takes place there. For instance the Google Maps API is accessed via the browser using JavaScript.

The process of building a mashup

For complex mashups certain problems need to be solved [TSK08]. The data source for mashups may vary in format, structure and location [MMD06]. The source of the data could be a web page, a feed or even a PDF on a local file system. The

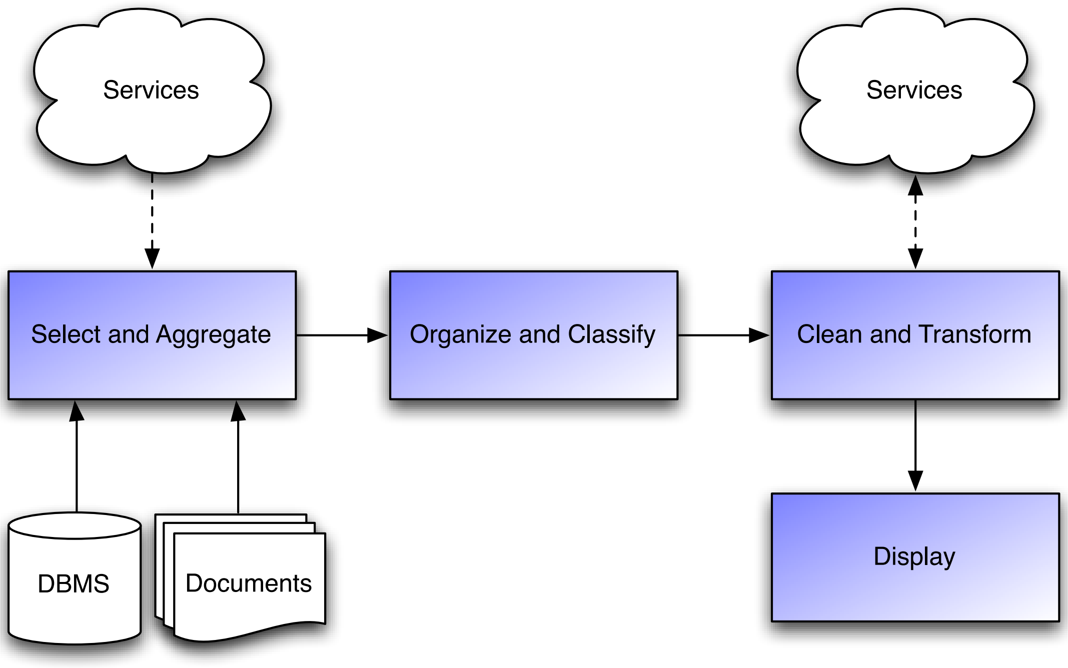


Figure 16: The process of producing mashups [MMD06].

data needs to be extracted into a manageable form [TSK08]; see Fig. 16. If the data is dynamically extracted, the rules for how to extract the data need to be solved. Data might span different locations and sources making it more difficult to manage. The data needs to be organised and classified to make it manageable and to find relationships between existing data sources and new ones [MMD06, TSK08]. The data may need to be transformed and cleaned in order to fix misspellings or to transform the data into the desired format. If the data retrieved has for instance the artist name Norah Jones in the format Jones, Norah, and another data set has got the artists' names in the former manner, a transformation is needed to integrate the new data set so that it matches our existing one, or an address needs to be transformed into geographical position. The transformation process may use other services for these transformations as well. In [TSK08], a data integration step is presented in the process of creating mashups. If for instance, one would build a mashup that would list all the movies performed by this year's Oscar award winners, one would need to retrieve a list of this year's Oscar award winners and merge that using database join operations on the winners names with a movie database. Data visualisation takes the final data and represents it to the user in a certain way.

Screen scraping

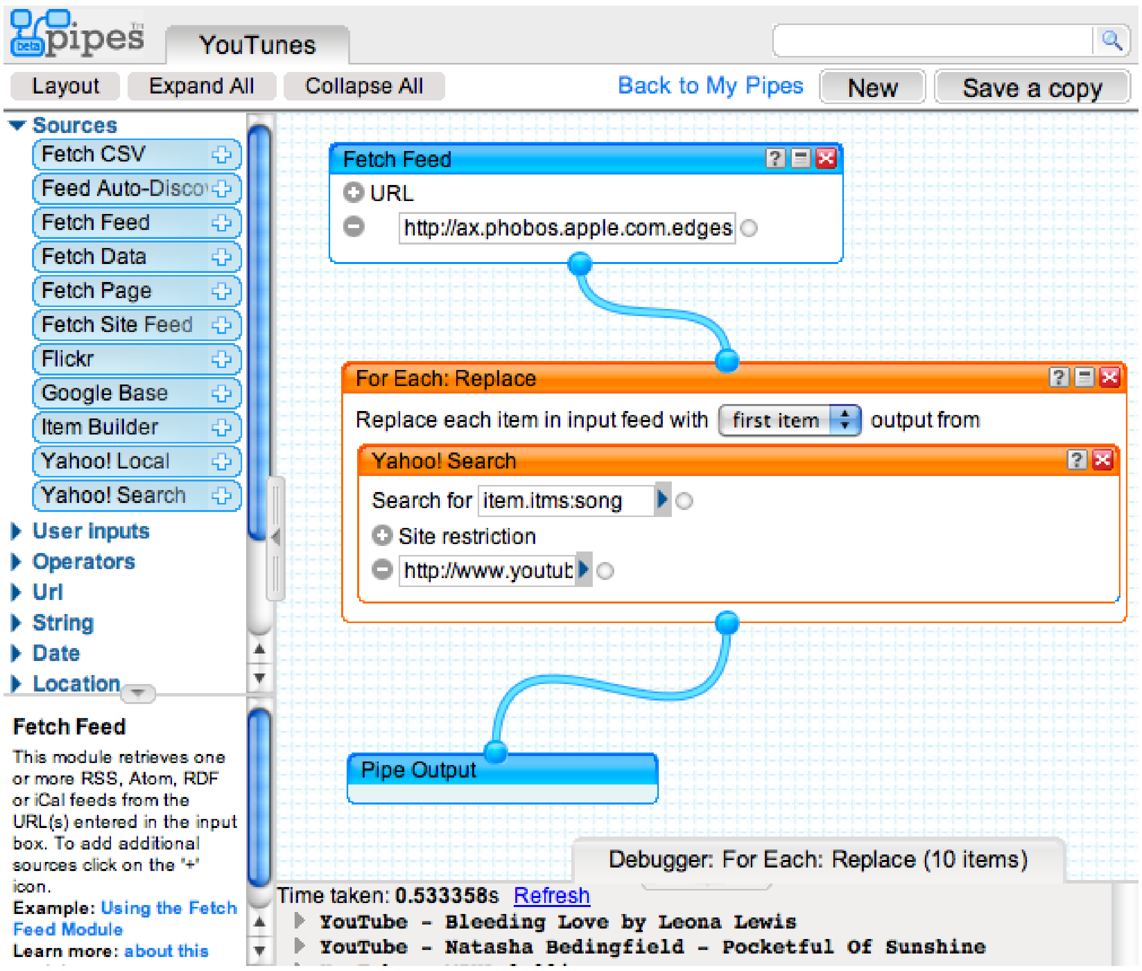
Screen scraping is defined as gathering data from web pages originally intended for human consumption by parsing and analysing the content so that semantic data structures and information can be acquired [Mer06]. If the content providers do not provide an API or a feed to access the data, mashup developers may need to resort to screen scraping in order to retrieve certain information [Mer06]. The information and data structures acquired can then be used to create a mashup. Screen scraping as data acquisition is used by a handful of mashups, especially when retrieving information from the public sector. An example of a mashup project that uses screen scraping is XMLTV[6], which is a set of tools that aggregates TV program listings from all over the world.

Screen scraping has some significant drawbacks. Unlike APIs which are interfaces of how to access data, there is no specific programmatic contract between the content-provider and the content-consumer for screen scraping. The scrapers have to analyse the content model of the site to be scraped and design their tools to comply with that. If the provider changes the way content is represented, the scraping tools are likely to stop working. Web sites periodically update their look-and-feel in order to remain fresh and stylish and this causes maintenance headaches to developers using screen scraping. Another issue is that there does not exist any screen-scraping toolkit software (scrAPI), that is sophisticated and reusable enough. Why these APIs and toolkits does not exist is mainly because each scraping has got extremely application-specific needs which affects the tools. Designers are forced to reverse-engineer content, parse and aggregate raw data and create data models of the content. Some scrAPI prototypes are currently being developed. One approach is using CSS selectors to get the desired content [Rub06]. It is probable that the target content is styled and given a class name. In this way it can be possible to get the right content even if some parts of the site structure might change.

User built mashups

There are different approaches that allow users to create mashups [TSK08]. One is based on widgets where the user can manage different services and elements in the mashup through enclosed elements called widgets. The other is based on data extracting through DOM analyses.

[6]`http://xmltv.org`



Fetch Feed

This module retrieves one or more RSS, Atom, RDF or iCal feeds from the URL(s) entered in the input box. To add additional sources click on the '+' icon.

Example: Using the Fetch Feed Module

Learn more: about this

URL: http://ax.phobos.apple.com.edges

Replace each item in input feed with first item output from

Search for item.itms:song

http://www.youtub

```
Time taken: 0.533358s Refresh
▸ YouTube - Bleeding Love by Leona Lewis
▸ YouTube - Natasha Bedingfield - Pocketful Of Sunshine
```

Figure 17: Users can create mashups using Yahoo! Pipes, a mashup building tool that use widgets to allow users without programming knowledge to create their own custom mashups [AKTV07]. The example shows a user built mashup that provides a music video feed of the current day's top ten purchased songs from an online music store. For each item in the feed, the mashup searches for the song title from the site `youtube.com` and outputs the first result for each song. The pipe can be found from `http://pipes.yahoo.com/pipes/pipe.info?_id=uGJSZWC42xGQdxGFJZhxuA`

In [TSK08] a framework is presented that allows users without programming skills to build customised mashups by providing examples that the framework can interpret and gather similar data. The DOM of the page is analysed and similar items to the suggested one are extracted from the document using XPath analysis. The data is cleaned in a way where the user modifies an example data into the desired form and the system interprets the change and comes up with a transformation how to change all the similar data into the desired format.

Another way to create mashups is to use widgets. For example, Yahoo! Pipes (`http://pipes.yahoo.com`) is a tool that allows users to build mashups via widgets; see Fig. 17. These widgets produce an output that can either be entered into another widget as input or directed to the output of the mashup. Even though no programming is required, it may still be difficult for an average user to create a mashup since you need to have an underlying knowledge of certain programming terms and structure to utilise the features.

6 Implementation of Funnelry

It was noted in Chapter 2 that social networking sites have increased in popularity, with large numbers of people contributing content in these sites. We stated that this content is mainly text, photos, video, remixes of other content from other sites, or content users have produced themselves. We argued that a large amount of content is being produced and viewed online, and that communication in various forms is constantly occurring on the Internet. In this thesis we refer to online contribution, viewing, communication and user actions as online activities.

Funnelry is an application prototype developed at HIIT (Helsinki Institute for Information Technology) with Björkskog as the main coder. Its purpose is to allow users to follow the online activities of their friends in various social media using a mobile phone. Instead of requiring the user to use a computer and visit several sites, they can follow their friends' activities from one place, even when on the move.

6.1 Example Scenarios

We explain the use of Funnelry through two simple scenarios.

Keeping Up With Friends

Snehal is studying Computer Science at the University of Helsinki. He is originally from India, but lives, studies and works in Helsinki. Some of his Finnish friends are active in Facebook, but most of his Indian friends use another social network called Orkut. Each morning Snehal logs in to these services in order to see what has happened since he was last online. After he has checked his friends' status and possibly sent a greeting, he heads for school on the bus. With Funnelry, he could have stayed up to date with his online friends whilst riding the bus. He would need to use only one web application, which would gather all of his friends' relevant activities in one place, where he could gain a quick overview of what is happening.

New User

Gina is a high school student. She occasionally uploads some of her favourite photos to Flickr for her friends and family to see. Occasionally she uploads a video clip to YouTube, but, more often than not, she finds a funny or interesting video clip that she adds to her favourites for her friends to see when they visit her YouTube profile page. She has a blog in Blogger, and is active in Orkut and Facebook. She also adds bookmarks to del.icio.us occasionally. Gina has friends in all of these online services, and some of her friends use several of the same services as Gina; see Fig. 18.

Gina registers and logs into Funnelry. She registers the services for which she has accounts. Gina's friends are then extracted from these services. Gina chooses which friends she wants to follow in Funnelry. She notices that her friend Maria's Flickr, Youtube, Facebook and del.icio.us accounts are listed. Gina merges these accounts so that the system knows they belong to the same person. She chooses to follow all of Maria's account activities except her Flickr account. Gina can now follow the activities of her friends in her social networks. She can see the details of the activities and, for the activities that enable comments, she can comment directly from the Funnelry interface. If she only wants to see what Maria has been doing lately, she can also simply view only Maria's activities.

Gina recommends Funnelry to Maria. Maria registers as a new Funnelry user. Maria finds that her account information from Flickr, YouTube, Facebook and del.icio.us is already set by Gina, so she immediately sees the activities in those networks. She only needs to add her Twitter account so she can follow her friends there as well.

6.2 Specifications

The specifications for Funnelry are described below.

Different services Different services can be added as plug-ins to the program. These plug-ins behave similarly. To add a service you can write an appropriate plug-in and register it to the Funnelry database. These services follow the same interface and behaviour.

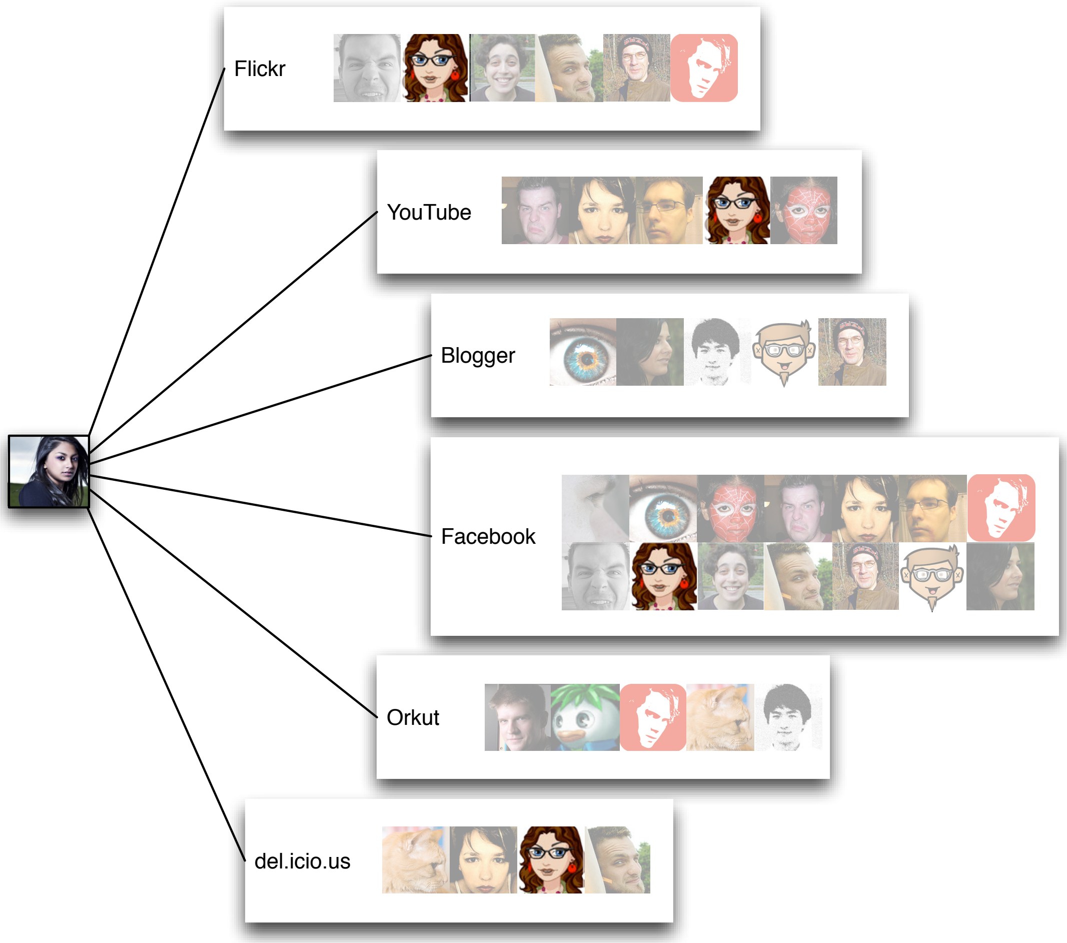


Figure 18: Gina has got many friends in different online social media. Some persons she is friends with in several networks.

Different visualisations The standard visualisation is a list of activities as they occur, where each activity can be viewed in its entirety. Additional visualisation handlers can be added in a similar way as adding service plug-ins.

Connecting additional services and filters A new visualisation filter should work without the necessity of rewriting anything for the service plug-ins. The client scripts communicate via an interface with the server, receiving and submitting the necessary content.

Contributing Some online services enable contributing via web services in the same way as receiving. The user must usually be logged into the specific services in order to submit content. Content submitted could be blog posts, status updates, wall posts, messages, comments etc.

Terminology		
ID Bundling	Associate different account id's to represent one "Person"	
Grouping	Group together several persons of interest	
Content Bundling	Inclusion	What we would like to see Selecting the media sources
	Aggregation	Group together Joining the media sources
	Filtering	Prioritise the bundled content by certain criteria Select the content to show
Visualisation	How to show the data	

Table 2: The terminology used.

List of friends Lists of friends can be created separately; one for family, one for work etc. Users can turn on or off different services for each friend dependent on their interests.

The terminology that is used can be found in Table 2.

6.3 Architecture

Funnelry has been implemented using a client-server architecture where the client interface is implemented through the mobile web browser; see Fig. 19. An AJAX engine builds the user interface and manages the data transfer between the server and the client. When the AJAX engine requests recent activities for the friends of a certain user, a pre-processed set of data is returned to the client in JSON format. The AJAX engine presents this data to the user via the user interface.

For enhanced user experience, the user can install a J2ME module (BeTelGeuse [NKL+07]) that gathers context data which is sent to the server. By using context data the activity information can be enriched. If, for instance, a user sees that a friend has updated his Facebook status to "I am bored", and the context data shows that the friend is nearby, the user can pay his friend a visit.

On the server side, data, such as activity data and user information, is fetched continuously from different services in order to provide short response time for the client. The data is organised and classified by the service-specific plug-ins, and eventually cleansed and transformed so that it can be integrated with the rest of the data. For example, a GPS position may be transformed into a logical location,

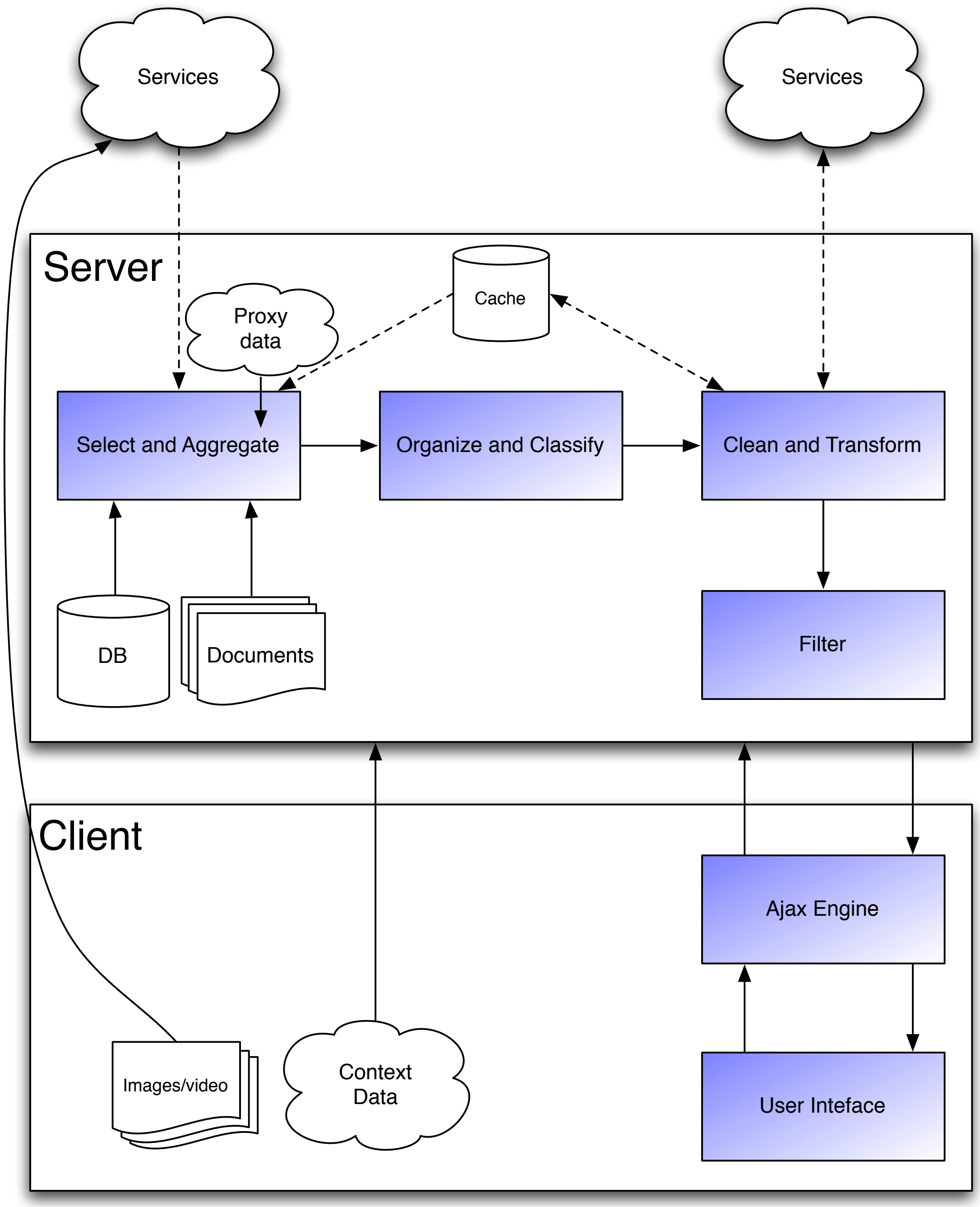


Figure 19: Funnelry system architecture.

such as home, or a specific address. The data is cached in order to be retrievable with only a short delay. When the user requests the data it can instantly be sent to them. Certain data may require that requests are sent to the services.

When there is too much information to display, the amount of data that is presented can be reduced through clustering [Noi05]. Our approach is to train the application to know which data the user finds interesting and relevant. Data can also be clustered. For example, when a user has uploaded several photos, it is summarised as a single event that the user uploaded 15 photos in Flickr, instead of having a new activity for each image.

When users navigate within the application and view the activities of their friends, a log is kept of what they are viewing in order to determine what kind of activities, based on tags, content type, content data, context and author, different users are interested in. When there is much data, the system can filter out irrelevant and uninteresting data.

A user can supply additional training data for the filter by installing a separate proxy on their desktop browser. What the user has watched is logged and analysed in order to find preferences for certain types of data and activities.

Data such as image and video can be uploaded to different services using third party applications such as ShoZu (`http://www.shozu.com`), which uploads images taken on the mobile phone to online services such as Flickr, Facebook, Blogger and YouTube.

Database Design

The database keeps track of different services, users, their specific service id's, the activities of their friends in the respective services and a log of which content the user has viewed using Funnelry; see Fig. 20. The following is a description of the specific database tables:

service An online social service such as Facebook, Flickr or YouTube.

user A user or a person added to the system. It may be a user of Funnelry, but it can also be a friend of a Funnelry user from one or many services. A user can be thought of as a container for service id's.

userService A person's account identifier for a specific service. A user can have a user id in several services.

userRelation Defines a relationship between different users.

relationType Different types of relationships, e.g., friend, peer, family or acquaintance.

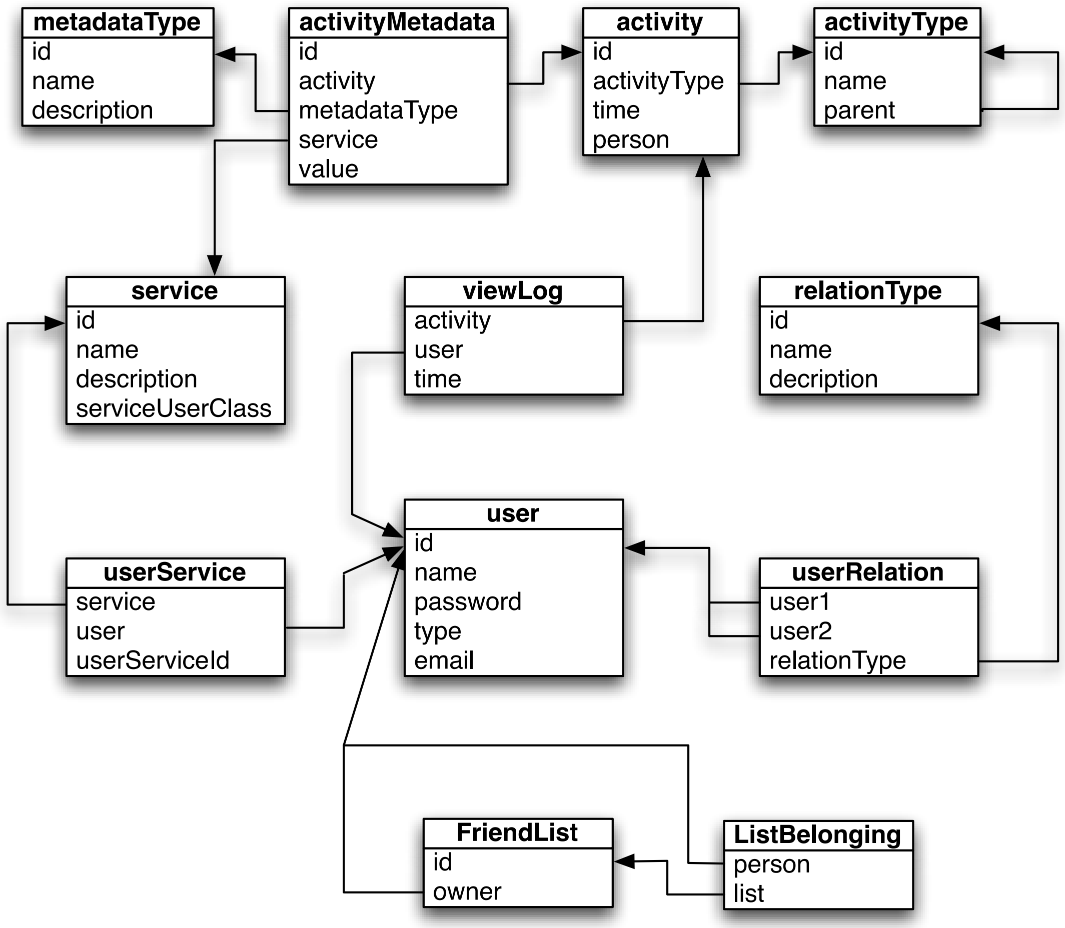


Figure 20: Database diagram.

activity An activity by a person in a service.

activityType Types of activity, including blogpost, uploaded photo, comment or status update, for example.

activityMetadata Metadata associated with an activity.

metadataType Types of metadata, e.g., tag, title, URL, description, content, image, video and publish time.

viewLog Stores what a user has viewed using Funnelry in order to personalise the filter so that only relevant content is presented to them.

friendList A list created by a user for grouping friends together.

listBelonging Persons belonging to specific lists.

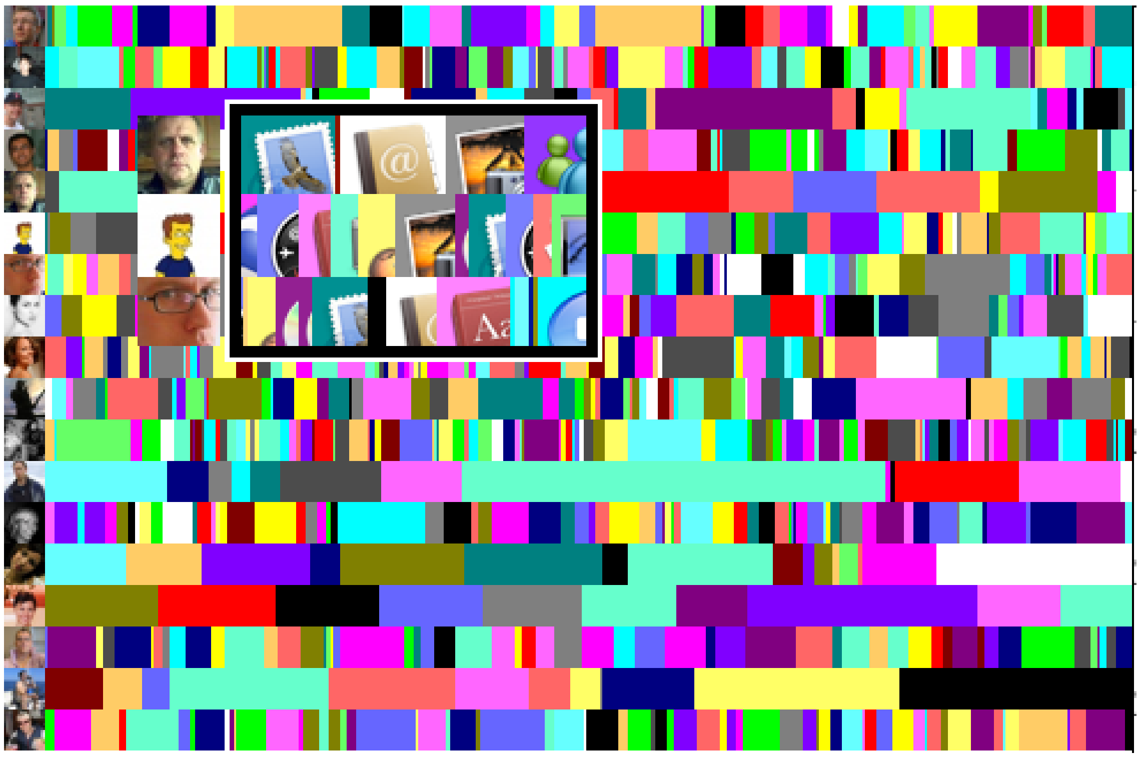


Figure 21: A prototype implementation of the initial user interface design. It was however rather slow to move the zoom-glass on a mobile device.

6.4 User Interface Prototype

In Fig. 21 an initial design of the user interface for Funnelry can be seen. It is one method of showing the activities in social media. Each row in the screen represents a friend. Each item along the x-axis represents different contributions in different social media by that person. The various services are represented by different colours. This is one way of seeing a large amount of data on a relatively small screen. When the user moves the cursor (or for this device, the pen) on the interface, a zoom-glass follows the cursor, showing more details about the activities. This was implemented as a test using CSS, Javascript and XHTML.

The interface consists of two div elements, one smaller and one larger, that contain the activities. The larger is clipped, however, so that only a small part is shown. This is the zoom glass. It is placed over the smaller div so that it appears to be zooming in on where the cursor is. When the cursor moves, a new clipping area of the larger div is calculated and the larger div is also moved in a relative way so that the correct content follows the cursor. In this way, an illusion is created of a magnifying glass hovering over the area. On a desktop browser it worked well, but

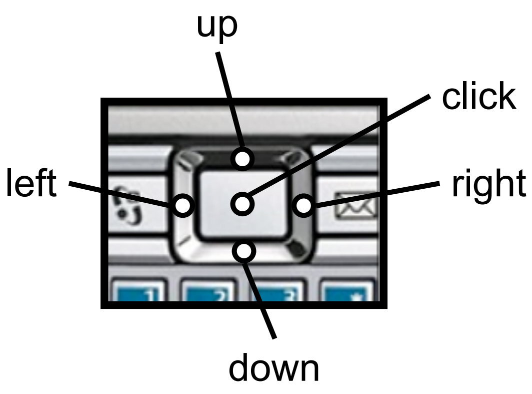


Figure 22: The pointing device is used as arrows for navigation. Here seen on the target device Nokia E61i.

on a mobile device it was too slow, since there was much work to be undertaken by the browser when the large area needed to be moved and re-clipped. One way of making this more effective might be to skip the larger div, and just logically create the content of the magnifying glass based on the data. This way, only as much as is shown needs to be manipulated, and the effect might be that it could run more smoothly on a mobile device.

6.5 Implementation

6.5.1 Navigation

We stated in Chapter 4 that mobile web users prefer to navigate content that is presented in a list format, so that detailed information can be acquired via items in the list. For this reason, we designed the next iteration of the user interface as a list-view. The built-in pointing device or arrow keys are used to navigate; see Fig. 22. By pressing up or down, different elements in a list are highlighted. By pressing right or clicking the activation button, the content determined by the selected item is shown.

One problem with this approach is that the use of the pointing device is interpreted as mouse movement and pressing the middle button is interpreted as a mouse click. So we could not gather those events by listening to key presses. We partially circumvented this problem by adding a mouse listener that registers changes in x- and y-axis. The top-left corner of the screen has the (x,y)-values (0,0), and the bottom-right corner has the x-value the same as the browser window's width and

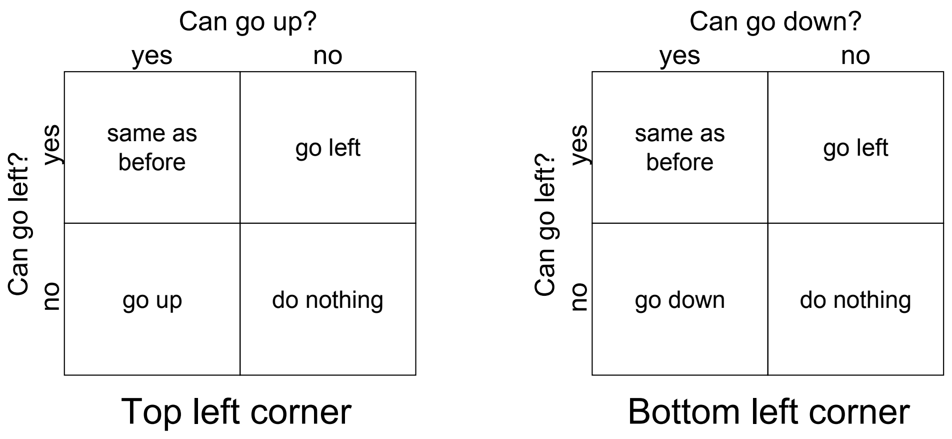


Figure 23: Interpreting whether up, down or left was pressed in the top left and bottom left corners.

the y-value the same as the browser window's height. When a change in the cursor position is greater along the y- than the x-axis, either up or down has been pressed. If the y-value is less than before the event occurred, the up button was pressed. If it is greater, down has been pressed. The same occurs for left and right regarding a change along the x-axis. However, this approach presents problem areas. When the cursor comes to the edges or the window corners, the mouse cannot travel further in those directions, and a coordinate position change will not occur. Four of these problem areas, when the cursor comes to the edges, but not in the corners, can be solved. If the cursor is, for instance, at the bottom edge and down is pressed, a mousemove event will be triggered, but there seems to be no change in either x- nor y-axis. We can then check what the x- and y-positions of the cursor are. If we find that the cursor is at the bottom edge, the cursor's y-coordinate is the same as the height of the viewport and its x-coordinate is greater than zero, but less than the width of the viewport. When no change in either x- nor y-direction has occurred, we know that down has been pressed. If left or right had been pressed, the x-value would have changed. If up had been pressed, the y-value would have shrunk. Similar rules can be applied to the other edges, but the corners remain a problem. If the cursor is, for instance, in the bottom left corner and a mousemove event is triggered but no change in cursor coordinates has occurred, we cannot know which of left or down has been pressed. We can specify a default case, but users are likely to get confused, if the application reacts in a different way than they commanded it to do.

One compromise for solving this would be to use the click (see Fig. 22) instead of right to activate a link, and left, as usual, to get back. Then the cursor would not

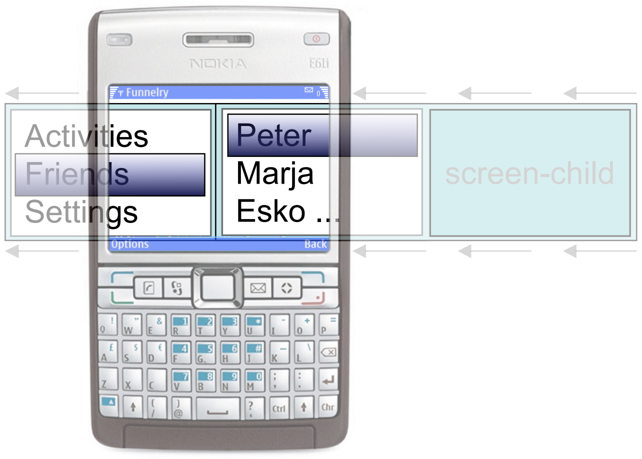


Figure 24: New content slides in from the right.

travel gradually rightwards and end up in one of the right corners. The only two problem areas would now be the top-left and bottom-left corners. We can diminish the impact of these problems by examining how the cursor got there. If the cursor ended up, for instance, in the top left corner and a mousemove event occurs with no change in cursor position occurring, we know that either up or left has been pressed, but not which one. If we look at the navigation event that happened before the cursor ended up in the corner, we can draw conclusions as to what the current navigation action is. A matrix for determining the actions to take when the cursor is in either top-left or bottom-left corners can be seen in Fig. 23. This compromise solution does not cover every case, but is better than always having one default action.

In Chapter 4 we further discovered that unique page titles are important for communicating the user's current location within the web structure. The navigation manager presents content in a list format and users can select an element from the list (by pressing up and/or down to highlight an element and activating a highlighted element by pressing the click button) opening a new page that slides in from the right, pushing the previous page out of view to the left; see Fig. 24. When the new page slides in, the title of the page changes so that it is the same as the text of the activated link. When the user presses left, the viewport slides back, so that the current page slides out of view to the right as the previous page slides in from the left and the title of the page changes to what it was previously.

The screen hierarchy is built up using screen nodes; see Fig. 25. Each screen node contains the content that is to be displayed when the node is selected. In addition, each screen node may contain a screen node as a child.

So that the reader should not be confused with the terms, we will specify what is meant here with the terms "list" and "link".

list is here used to describe an XHTML table or div element with the class name "list"

<table class="list"></table> or <div class="list"></div>

not to be confused with an ordered or unordered XHTML list object (<ul></ul> or <ol></ol>)

link is used to describe a table row or a div with the class name "link" or "link-selected"

<tr class="link"></tr>, <tr class="link-selected"></tr>,

<div class="link"></div> or <div class="link-selected"></div>

not to be confused with an XHTML anchor, or link (<a href="..."></a>) nor list items (<li></li>) in <ul></ul> or <ol></ol> XHMTL elements.

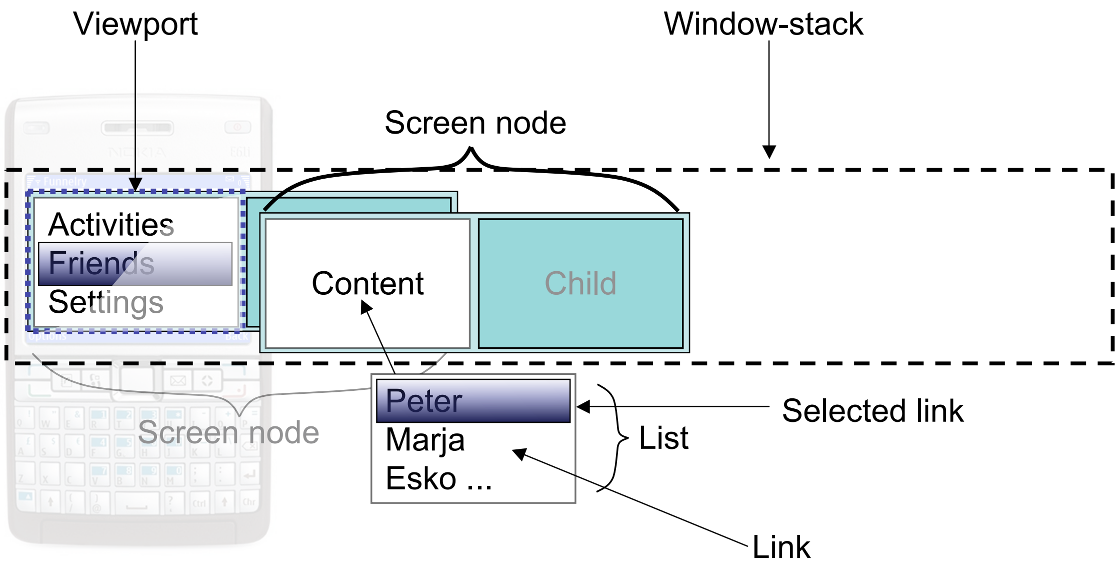


Figure 25: The user interface navigation system.

If there exists, in the content, a div or table with the class name "list", every tr respective div element with the class "link" can be iterated in the list using up or down. Each link can be given a function that returns the content that should be

```
<div id="horizontal-window-stack">
  <div id="win_1" class="screen-node">
    <div id="win_1_self" class="screen-content">
      <div class="list">
        <div id="win_1_link_1" class="link" rel="wiew_activities">Activities</div>
        <div id="win_1_link_2" class="link-selected" rel="wiew_friends">Friends</div>
        <div id="win_1_link_3" class="link" rel="wiew_settings">Settings</div>
        <div id="win_1_link_4" class="link" rel="logout">Logout</div>
      </div>
    </div>
    <div id="win_1_next" class="screen-child">
    </div>
  </div>
</div>
```

Figure 26: XHTML container that builds up the navigation structure.

```
<div id="horizontal-window-stack">
  <div id="node_1" class="screen-node">
    <div class="screen-content">
      <ul class="list">
        <li><a href="javascript:next(wiew_activities());">Activities</a></li>
        <li><a href="javascript:next(wiew_friends());" class="selected">Friends</li>
        <li><a href="javascript:next(wiew_settings());">Settings</li>
        <li><a href="logout.php">Logout</li>
      </ul>
    </div>
    <div class="screen-child">
    </div>
  </div>
</div>
```

Figure 27: Alternative XHTML container.

viewed if the logical link gets activated. The selected element in the logical link is given the class name "link-selected". Each up/down action changes which element is given the class name "link-selected". When a link is activated, the function that is associated with the selected link is executed by the navigation handler. The content that the function returns (XHTML) is inserted into a new screen node's content; see Fig. 25. The new screen node is added into the current screen-nodes' child. The container that holds the nodes, a div with the id "horizontal-window-stack", which may be larger than the viewport, is moved leftwards so that the invisible content to the right becomes visible and the currently visible content slides out of view to the left.

Fig. 26 shows an example of the XHTML container that builds up the navigation structure. Text inside the rel attributes are the name of the functions which are to be called when the links are activated. However, rel is not a valid attribute for a div in XHTML. This implementation works on mobile Series 60 devices. An alternative version can be seen in Fig. 27.

By implementing the navigation structure according to Fig. 27, it would support devices with a touch screen as well. The problem with the cursor ending up in a corner could also be solved, since the cursor snaps to the links in a page when the cursor is moved far enough and the page automatically scrolls when the cursor approaches the edges. However, care should be taken so that it is not the link (`<a>` tag) that the cursor points to that is triggered, but rather, the selected link. This can be achieved by having a script add an `onclick="doStuff();return false;"` to every `<a>` tag where a handler (here named doStuff) is responsible for determining what happens.

When left is pressed, the container slides rightwards exposing the parent node's content and the parent becomes the active screen. The node that was recently exited from is destroyed leaving the currently active node's child empty.

6.5.2 Visualising Activities

We stated in Chapter 4 that it is recommended, when visualising large amounts of data, to represent the data at different levels of detail ranging from a general overview to fine detail. We also argued that visualisation graphics should direct the user's attention to the presented data. We found that this can be achieved through preattentive processing, by taking advantage of the human low level visual system's ability to recognise and extract certain features immediately, without focusing on the picture. We discovered that in order to allow for preattentive processing of data combinations, one must assure that the resulting combination contains at least one unique visual feature.

The standard view is a list where the different activities are presented in a condensed list with some detail of the visible activities. The users can obtain more information about each activity by selecting and activating them. Each activity in the list is described with a photo of the person who performed the activity, a time indicator when the activity occurred, a small logo of the service and text describing what kind of activity it is. These are visual features allowing the user to immediately recognise an activity of interest. Each activity has a description field that differs depending on what type of content it is. A set of photos can show small thumbnails of the photos or, if someone wrote a blogpost, the title of the post is shown. This way users can immediately distinguish between certain types of activity in addition to the text that says what kind of activity it is.

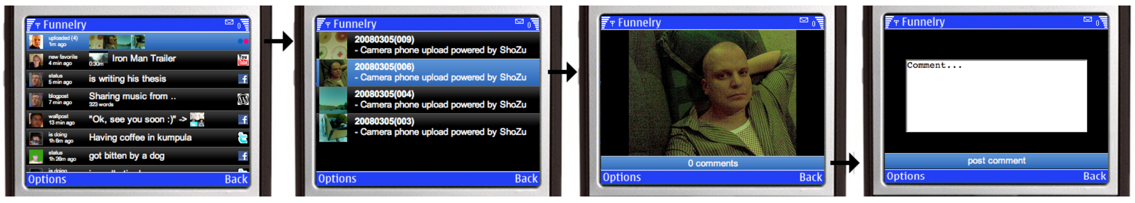
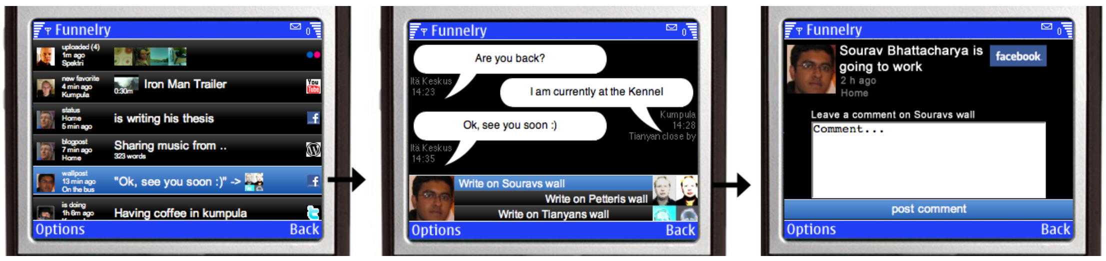

Funnelry
uploaded (4)
Iron Man Trailer
is writing his thesis
Sharing music from ..
"Ok, see you soon :)" ->
Having coffee in kumpula
got bitten by a dog
Options
Back
20080305(009) - Camera phone upload powered by ShoZu
20080305(006) - Camera phone upload powered by ShoZu
20080305(004) - Camera phone upload powered by ShoZu
20080305(003) - Camera phone upload powered by ShoZu
0 comments
Comment...
post comment

Figure 28: Visualising a set of uploaded photos. The photos can be viewed and commented on if the service provider supports that.



Are you back?

I am currently at the Kennel

Ok, see you soon :)

Sourav Bhattacharya is going to work
2 h ago
Home

Figure 29: Visualiation of wall post.

A set of photos that have been uploaded by a friend is visualised in the list view as tiny thumbnails; see Fig. 28. When opened, each photo is presented in a list with larger, titled thumbnails and a description. The user can select a photo to view in more detail, and is presented with a photo in appropriate size for a small display. The screen size, which is obtained using JavaScript, can be used in order to find a version of the photo in a similar size. Photos in different resolutions are generated by the server when photo activities are fetched. The user can also comment on the photo if the service supports comments. A comment is then submitted to the service from which the specific photo came.

Certain services, such as Facebook, have the possibility for users to write short messages on their friends' profile pages. These are referred to as wall posts. In the list view, wall posts are described using the message and pictures of the persons involved, and an arrow indicating the receiver of the message; see Fig. 29. When the link is opened, the user can see the history of the friend's messages to one other.

Status updates are common in many services and are used to describe what the user is currently doing. In the list view, a short representation of the status from a certain service is presented; see Fig. 30. If the text does not fit in the list view, the user can see a full text version by opening the link. On the detail page there is also a possibility to reply with a wall post or a message if the service allows it.

The title of a blog post is shown in the list view; see Fig. 31. When opened, the

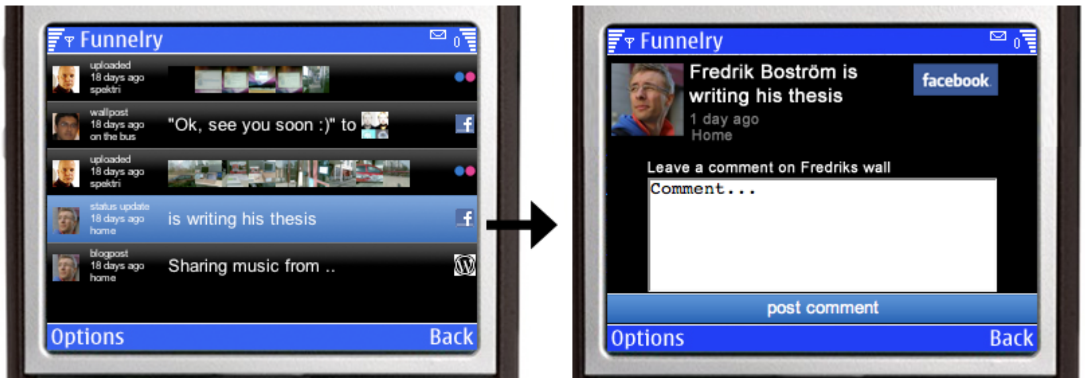
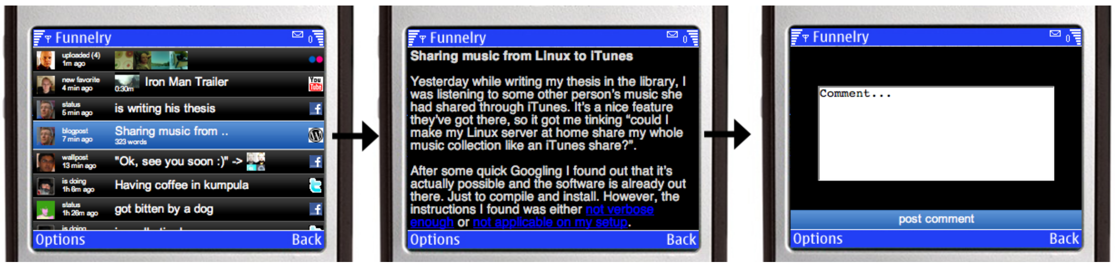

Funnelry
uploaded 18 days ago spektri
wallpost 18 days ago on the bus
"Ok, see you soon :)" to
uploaded 18 days ago spektri
status update 18 days ago home
is writing his thesis
blogpost 18 days ago home
Sharing music from ..
Options
Back
Fredrik Boström is writing his thesis
facebook
1 day ago
Home
Leave a comment on Fredriks wall
Comment...
post comment

Figure 30: Visualising a status update. When opened, the status update can be seen in full text and if the service supports wall posts or messages via APIs, the user can directly write a comment to the friend.



Sharing music from Linux to iTunes

Yesterday while writing my thesis in the library, I was listening to some other person's music she had shared through iTunes. It's a nice feature they've got there, so it got me tinking "could I make my Linux server at home share my whole music collection like an iTunes share?".

After some quick Googling I found out that it's actually possible and the software is already out there. Just to compile and install. However, the instructions I found was either not verbose enough or not applicable on my setup.

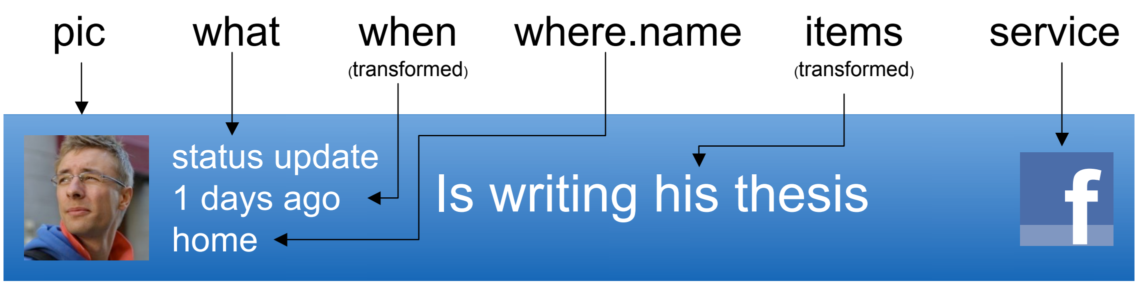
Figure 31: Visualization of text contribution such as blog posts.

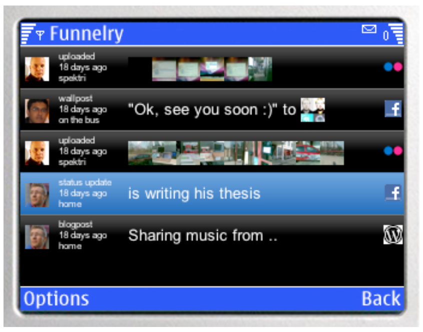
entire text can be read within Funnelry. If the service supports comments, the user can comment on the blogpost directly from Funnelry.

The interface and data visualisation was implemented on the client side with JavaScript, XHTML and CSS. Data was fetched asynchronously from the server using XHR and the data transferred was encoded in JSON format. Each activity was encoded in a similar way to this example using the JSON standard; see Fig. 32.

The only thing that differs in structure between activity types is the "items" attribute, which may change from content type to content type. The example in Fig. 32 shows how data representing a status update in Facebook was visualised.

If the content type was a set of images, the items attribute would contain an array of information about each image instead of a string, and it would be visualised as a row of image thumbnails. The other attributes were visualised in the same way for each activity type. Time was retrieved from the server as a timestamp, which is transformed by the visualisation filter and presented as a logical representation, e.g., "20 minutes ago".





```
{
  "id"   : "4",
  "name" : "Fredrik",
  "pic"  :   "http://someurl.com/small_picture_of_fredrik.jpg",
  ”type" : "status",
  "what" : "status update",
  "when" : "1206938076000",
  "where" : {
                  "ll" : "60.1889453302619,   24.804940223693848",
                  "name":"home”
                },
  "service" : "1",
  "nearby" : "",
  "items": "is writing his thesis"
}
```

Figure 32: The type of activity determines how the items attribute will be presented. The other attributes are presented in the same way independent of activity type.

When visualising the different activities, each content type has its own handler for adding the descriptive information based on the items attribute. Each content type also has its own handler for visualising the details of each activity. The handlers communicate with the server interface using XHR loading JSON feeds asynchronously.

To make JavaScript programming faster, a JavaScript library called jQuery (jquery.com) was used. This contains handlers for AJAX calls and animation from one CSS state to another. The sliding of the navigation was implemented using the animation feature of jQuery, which ran smoothly. The response time for interaction using the pointing device as arrows was not very fast, and did not appear to react immediately to input. The mouse event handler needs to be optimised in order to provide the feeling of instant interaction.

6.6 Loading Times

In order to determine the loading times of the site, and the activities, and to see whether it is necessary to create local resized copies of images to be shown, we measured how long it took to load certain elements of Funnelry. The results can be seen in Appendix E. The tests were performed on a Nokia E61i mobile phone, with a WLAN (Wireless Local Area Network) connection, using a stop watch. Each test was repeated ten times. The following loading times are average times calculated form the repeated tests. The time it took for the page to load (when it had been viewed before), including the Ajax engine, was 3.8 seconds. When the cache was cleared and the page reloaded, the loading time was 5.6 seconds. Opening an activity feed several times without reloading took 2 seconds. From a reloaded page it took 4.7 seconds to open an activity feed. The activities in the feed were similar to those seen in the visualisation examples in this section. The time was measured from when the activity feed was opened until the last image had been loaded. There were 15 images in the feed apart from the profile pictures of the users. The resolution of the pictures were 75x75 pixels, around 2.5 KB in size each, but they were displayed as 20x20 pixels. When these pictures were linked to their location on the Flickr server without being resized, the loading time from an empty cache was 8.2 seconds. When the feed was changed on the server side to use resized images, 20x20 pixels in dimension, around 0.6 KB in size, the loading time from an empty cache dropped to 7.5 seconds. There was no significant difference in loading time when the images were stored in the browser's cache. The activities represented with text were not included in this experiment since they became visible without any noticeable delay. The reason why there was so little change in loading time when the size of the images decreased may depend on the connection speed and the fact that there was no decrease in the amount of http-requests being made (there were just as many pictures), and the limited processing power of mobile devices.

Usability guidelines say that users will only stay focused for a very short period of time. Even though the loading times have been optimized, they are still too large. This indicates that providing feedback to the user about interface events is crucial for the user experience of Funnelry.

In order to further improve loading times, the following measures could be taken:

- Several small images that are to be presented next to each other are merged into one, which would reduce the amount of http-requests.
- Activity data presented in the sub page "Activities" is requested immediately when the user enters the front page and is kept in memory.
- Images are loaded before they are needed by creating JavaScript Image objects, which initialise http-requests once they have been assigned a URL.

It took 27 seconds to get the metadata for the 10 newest photos for two Flickr friends (20 photos), store the metadata in a database, download the thumbnails to the server and resize them from 75x75 pixels to 40x40 and 20x20 pixels. The communication with the services are bottlenecks. API calls can be slow, and if there is a great deal of users with a large pool of friends, a large amount of requests need to be made. If these channels of communication are too slow, the application suffers. It is necessary the data retrieval done in advance so that the user does not have to wait long. We will take these aspects into consideration by having the server automatically fetch activity data on a regular basis.

6.7 Lessons Learned

From the software process of this thesis some lessons can be learned. One is how much the standard input of mobile devices can be manipulated without loosing flexibility. If I were to redesign the application, I would choose to use the browsers built-in feature of jumping between highlighted link items. This could make the up-down iteration through the selectable links faster, since it would be managed by the browser itself, and not a script that the browser runs, and would use a standard way of navigating between links. For opening links I would choose that the links are to be clicked, and going backwards would be done through the browser's back button or via a back-link that would be placed so that it is easily accessed.

Another lesson comes from debugging. Some bugs were difficult to discover when a web page was dynamically manipulated using jQuery. When adding XHTML tags which did not have a separate closing element (e.g. <br/>), the web browser of the Nokia E61i could not add the XHTML element even though it was valid XHTML. Example of this are the <br/> and <img/> tags. If such a tag was entered as a HTML tag, e.g., <br> (not valid XHMTL), it worked. It also worked if the tags were entered with a close tag, e.g., <br></br>. The phenomenon caused problems when injecting XHTML from external sources. In order to prevent these kind of errors, the XHMTL to be injected should be checked and modified so that these kind of problems do not occur.

One lesson comes from competition with the market. A problem in the development process was that new services and competing applications emerged regularly. When we thought of new ideas, the same ideas could be found in a commercial application a few weeks later. It is difficult to be unique in a field that is very popular and constantly changing.

7 Conclusion

Social networking is a natural phenomenon in human communication. The evolution of the Internet has taken social networking to new levels with online social interaction, communication, collaboration and media sharing. Many socially oriented services, such as video and photo sharing services, social and professional networks, online recommendation systems, blogs, communities, collaborative editing, user reviews and status tracking enhance social networking. There are many different online social network services and it is difficult to follow the activities in all. In this thesis we have studied whether online social activities can be aggregated and visualised in a graspable way on mobile Web browses using AJAX technologies.

In order to visualise online activity in a social network of a user, activity data need to be gathered from different online services where the users' friends are active. Many services provide API's for developers to integrate, contribute, and interact with their data in various ways.

The software part of this thesis consisted of a mobile web application that aggregates online activities. The software fetches profile data and activities of the user's friends from online services and presents them to the user. The data is presented in a list-view, which is preferred by users when there is much content to be displayed.

A good way to visualise and manipulate data in web content is to use AJAX technologies. These consist of web standards that have existed for many years and are well known by developers. Recently several mobile browsers support these technologies, which enable mobile web applications to handle data and user interaction in new ways.

We conclude that it is possible to aggregate and visualise online activity on mobile devices, but in order for the application to remain scalable, server side data retrieval and pre-processing is necessary.

8 Acknowledgements

First of all, I would like to thank God for his, love, grace, inspiration, and wisdom. I would like to thank my wonderful wife Eva for all the support, encouragement and motivation she has given me, and for everything she has done for our family. I would like to thank our precious children Emma and David (and our unborn child) for being my inspiration and motivation, and source of joy. I would like to thank the PUPS project and the colleagues there, especially Petri Saarikko, who played a major role in the forming of Funnelry, provided the user interface graphics and had always ideas and a great hope for the application. I would like to thank Petteri Nurmi, my thesis supervisor, and Patrik Floréen, who took me onboard, for their guidance, directions, patience, feedback and time. They have read my thesis several times, found my mistakes and have helping me improve this thesis. I would also like to thank HIIT for giving me the opportunity to write this thesis, and my peers in the "Kennel" for their friendship, feedback, help and encouragement. I would like to thank Fredrik Boström for contributing with support, code, and help. I would also like to thank Peter Peltonen who read my thesis and gave feedback. I would like to thank Esko Kurvinen for his ideas and inspiration. I would like to thank my friend and brother in Christ, Dan Stephens for reading my thesis and helping me with my English. I would also like to thank everyone who has given me words of encouragement and advice along the way. May God bless you all.

References

Aci08a Acid2, 2008. URL `http://acid2.wikispaces.com`. [retrieved 14-04-2008].

Aci08b Acid2 - wikipedia, the free encyclopedia, 2008. URL `http://en.wikipedia.org/wiki/Acid2`. [retrieved 14-04-2008].

AKTV07 Ankolekar, A., Krötzsch, M., Tran, T. and Vrandecic, D., The two cultures: mashing up web 2.0 and the semantic web. *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, New York, NY, USA, 2007, ACM, pages 825–834.

Alt08 Altman, T., Opera and the acid3 test, March 2008. URL `http://my.opera.com/desktopteam/blog/2008/03/26/opera-and-the-acid3-test`. [retrieved 14-04-2008].

BGS06 Blekas, A., Garofalakis, J. and Stefanis, V., Use of RSS feeds for content adaptation in mobile web browsing. *W4A: Proceedings of the 2006 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, New York, NY, USA, 2006, ACM, pages 79–85.

Blo08 Bloglines, *About Bloglines.* IAC Search & Media, San Francisco Bay Area, 2008. URL `http://www.bloglines.com/about`. [retrieved 14-04-2008].

BXWM04 Baudisch, P., Xie, X., Wang, C. and Ma, W.-Y., Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content. *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 2004, ACM, pages 91–94.

CSA07 Costa, C. J., Silva, J. and Aparício, M., Evaluating web usability using small display devices. *SIGDOC '07: Proceedings of the 25th Annual ACM International Conference on Design of Communication*, New York, NY, USA, 2007, ACM, pages 263–268.

CtHS06 Counts, S., ter Hofte, H. and Smith, I., Mobile social software: realizing potential, managing risks. *CHI '06: CHI '06 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2006, ACM, pages 1703–1706.

Gar05 Garett, J. J., Ajax: A new approach to web applications, February 2005. URL `http://adaptivepath.com/ideas/essays/archives/000385.php`. [retrieved 21-04-2008].

Geo07 Georgi, R., Ajax on mobile devices - making mobile web apps ubiquitous, May 2007. URL `http://2007.xtech.org/public/asset/attachment/182`. XTech 2007.

GMR07 Glotzbach, R. J., Mohler, J. L. and Radwan, J. E., RSS as a course information delivery method. *SIGGRAPH '07: ACM SIGGRAPH 2007 Educators Program*, New York, NY, USA, 2007, ACM, page 16.

Gol07 Goldman, D., Opera for Windows Mobile passes Acid2 test; on pace to become first WM browser to pass test, April 2007. URL `http://operawatch.com/news/2007/04/opera-for-windows-mobile-passes-acid2-test-on-pace-to-become-first-wm-html`. [retrieved 14-04-2008].

Goo08a Google Inc, Google maps, 2008. URL `http://maps.google.com`. [retrieved 21-03-2008].

Goo08b Google Inc, Google maps api reference, 2008. URL `http://code.google.com/apis/maps/documentation/reference.html`. [retrieved 20-04-2008].

Ham05 Hammersley, B., *Developing Feeds with RSS and Atom.* O'Reilly & Associates, April 2005.

Har01 Haramundanis, K., Learnability in information design. *SIGDOC '01: Proceedings of the 19th Annual International Conference on Computer Documentation*, New York, NY, USA, 2001, ACM, pages 7–11.

Hea07 Healey, C. G., Perception in visualization, Department of Computer Science, North Carolina State University, January 2007. URL `http://www.csc.ncsu.edu/faculty/healey/PP`. [retrieved 14-04-2008].

Hic08 Hickson, I., The acid3 test, March 2008. URL `http://acid3.acidtests.org/`. [retrieved 14-04-2008].

ID05 Ito, M. and Daisuke, D., Mobile phones, Japanese youth, and the replacement of social contact. In *Mobile Communications: Re-negotiation of the Social Sphere*, Springer, London, UK, 2005, pages 131–148.

IEB07 IEBlog, Internet Explorer 8 and Acid2: A Milestone, december 2007. URL `http://blogs.msdn.com/ie/archive/2007/12/19/internet-explorer-8-and-acid2-a-milestone.aspx`. [retrieved 14-04-2008].

Jso08 Json.org, Json, 2008. URL `http://json.org`. [retrieved 20-03-2008].

KR03 Kaikkonen, A. and Roto, V., Navigating in a mobile xhtml application. *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2003, ACM, pages 329–336.

LB05 Lam, H. and Baudisch, P., Summary thumbnails: readable overviews for small screen web browsers. *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2005, ACM, pages 681–690.

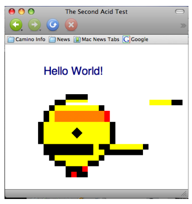
Lew95 Lewis, J. R., Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, 7,1(1995), pages 57–78. URL `http://portal.acm.org/citation.cfm?id=204774`.

LH01	Lauesen, S. and Harning, M. B., Virtual windows: Linking user tasks, data models, and interface design. *IEEE Softw.*, 18,4(2001), pages 67–75.
LLK06	Laakso, S. A. and Latva-Koivisto, A., Käyttöliittymät 2006, 2006. URL `http://www.cs.helsinki.fi/u/salaakso/papers/Kayttoliittymat-opetusmoniste-2006.pdf`. Lecturenotes.
Mer06	Merrill, D., Mashups: The new breed of web app, 2006. URL `http://www.ibm.com/developerworks/xml/library/x-mashups.html`. [retrieved 14-04-2008].
MMD06	Murthy, S., Maier, D. and Delcambre, L., Mash-o-matic. *DocEng '06: Proceedings of the 2006 ACM Symposium on Document Engineering*, New York, NY, USA, 2006, ACM, pages 205–214.
Nie99	Nielsen, J., *Designing Web Usability: The Practice of Simplicity.* New Riders Publishing, Thousand Oaks, CA, USA, 1999.
Nie03	Nielsen, J., Usability 101: Introduction to usability. URL `www.useit.com/alertbox/20030825.html`. [retrieved 20-04-2008].
NKHS07	Natchetoi, Y., Kaufman, V., Hamdi, L. and Shapiro, A., Mobile web 2.0 browser for collaborative social networking. *Position paper for European Conference on CSCW Workshop*, Limerick, Irland, September 2007.
NKL$^+$07	Nurmi, P., Kukkonen, J., Lagerspetz, E., Suomela, J. and Floréen, P., BeTelGeuse - A Tool for Bluetooth Data Gathering. *BodyNets '07: Proceedings of 2nd International Conference on Body Area Networks*, Florence, Italy, 2007.
Noi05	Noirhomme-Fraiture, M. et al., Data visualizations on small and very small screens. *Proceedings of the 11th International Symposium on Applied Stochastic Models and Data Analysis ASMDA-2005*, Brest, France, May 2005, ENST.
O'R05	O'Reilly, T., What is web 2.0: Design patterns and business models for the next generation of software. URL `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

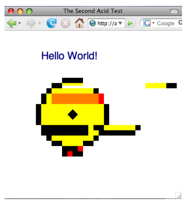
Oxf01	*The New Oxford American Dictionary.* Oxford University Press, USA, first edition, September 2001.
Pau05	Paulson, L. D., Building rich web applications with ajax. *IEEE Computer*, 38,10(2005), pages 14–17. URL `http://doi.ieeecomputersociety.org/10.1109/MC.2005.330`.
Pew08	Pew Internet & American Life Project, Usage over time, 2008. URL `http://www.pewinternet.org/trends/UsageOverTime.xls`. [retrieved 14-04-2008].
Rot06	Roto, V. et al., Minimap: a web page visualization method for mobile phones. *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2006, ACM, pages 35–44.
Rub06	RubyForge: scrAPI: Project Info, August 2006. URL `http://rubyforge.org/projects/scrapi`. [retrieved 14-04-2008].
SDJ$^+$08	Schrier, E., Dontcheva, M., Jacobs, C., Wade, G. and Salesin, D., Adaptive layout for dynamically aggregated documents. *IUI '08: Proceedings of the 13th International Conference on Intelligent User Interfaces*, New York, NY, USA, 2008, ACM, pages 99–108.
Siv07	Sivaraman, G., Web technologies driving innovations in mobile, 2007. URL `http://www.nokia.com/NOKIA_COM_1/Press/twwln/presentation_pdfs/Web_Technologies_TWWLN.pdf`. [retrieved 14-04-2008].
Sta08	Stachowiak, M., Webkit achieves acid3 100/100 in public build, March 2008. URL `http://webkit.org/blog/173/webkit-achieves-acid3-100100-in-public-build/`. [retrieved 14-04-2008].
Sto08	Storey, D., Opera on acid, March 2008. URL `http://my.opera.com/dstorey/blog/show.dml/1843336?cid=4964852`. [retrieved 14-04-2008].
Tec08	Technorati, Technorati: About us, 2008. URL `http://technorati.com/about/`. [retrieved 14-04-2008].

TSK08	Tuchinda, R., Szekely, P. and Knoblock, C., Building mashups by example. *IUI '08: Proceedings of the 13th International Conference on Intelligent User Interfaces*, New York, NY, USA, 2008, ACM, pages 139–148.
Tuf90	Tufte, E. R., *Envisioning Information.* Graphics Press, May 1990.
Tuf01	Tufte, E. R., *The Visual Display of Quantitative Information.* Graphics Press, May 2001.
VRM03	Venkatesh, V., Ramesh, V. and Massey, A. P., Understanding usability in mobile commerce. *Commun. ACM*, 46,12(2003), pages 53–56.
Wer38	Wertheimer, M., Laws of organization in perceptual forms. pages 71–88. available at `http://psy.ed.asu.edu/~classics/Wertheimer/Forms/forms.htm`.
Wik08	Wikipedia, Ahah - wikipedia, the free encyclopedia, 2008. URL `http://en.wikipedia.org/wiki/AHAH`. [retrieved 20-03-2008].
WJ07	Wilton-Jones, M., Acid 2 in major browsers, 2007. URL `http://www.howtocreate.co.uk/acid`. [retrieved 14-04-2008].
WM08	Weaver, A. C. and Morrison, B. B., Social networking. *Computer*, 41,2(2008), pages 97–100.
Wol99	Wolfmaier, T. G., Designing for the color-challenged: A challenge, March 1999. URL `http://www.internettg.org/newsletter/mar99/accessibility_color_challenged.html`.
WSP08a	The Web Standards Project, Acid2: The guided tour, 2008. URL `http://www.webstandards.org/action/acid2/guide/`. [retrieved 14-04-2008].
WSP08b	The Web Standards Project, Acid3: Putting browser makers on notice, again., March 2008. URL `http://www.webstandards.org/press/releases/20080303/`. [retrieved 14-04-2008].
YTV08	YTV, Reittiopas: Kartta, 2008. URL `http://aikataulut.ytv.fi/reittiopas/en/`. [retrieved 21-03-2008].

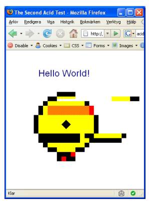
Appendix A. Acid2 Test



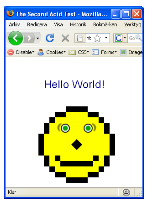
Camino 1.5.5
MacOS X 10.5.2



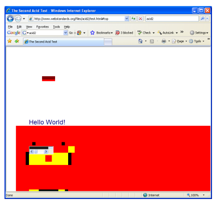
Firefox 2.0.0.13
Mac OS X 10.5.2



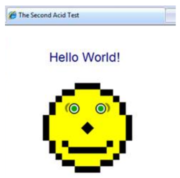
Firefox 2.0.0.12
Win XP SP 2



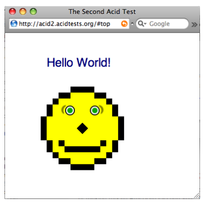
Firefox 3 *beta* 4
Win XP SP 2



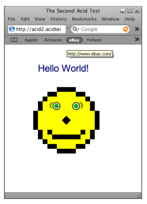
IE 7.0.7730.13
Win XP SP 2



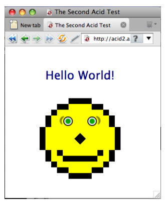
IE 8
[IEB07]



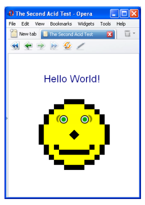
Safari 3.1
Mac OS X 10.5.2



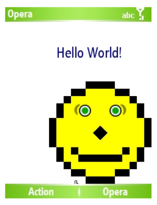
Safari 3.0.4 (523.15)
Win XP SP 2



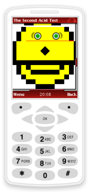
Opera 9.25
MacOS X 10.5.2



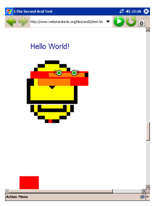
Opera 9.26
Win XP SP 2



Opera Mobile 9
Windows Mobile
[Gol07]

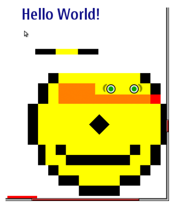


Opera Mini
MicroEmulator
[Aci08a]



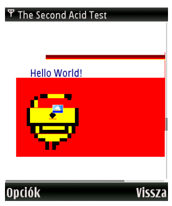
Opera Mobile (8.65)
PocketPC 2003

[Aci08a]



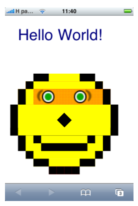
Opera Mini 4.0
Symbian 9.1

[Aci08a]



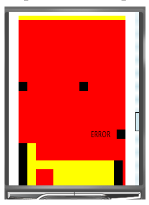
NokiaWebKit
Symbian 9.1

[Aci08a]

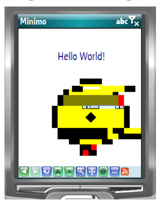


Safari
Mobile OSX

[Aci08a]

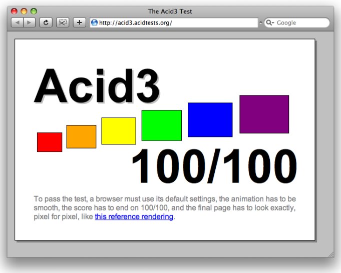


Internet Explorer 6
Windows Mobile 6
[Aci08a]



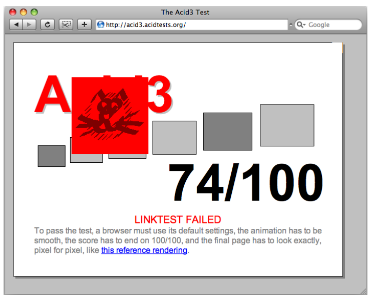
Minimo 2.0
Windows Mobile 6
[Aci08a]

Appendix B. Acid3 Test



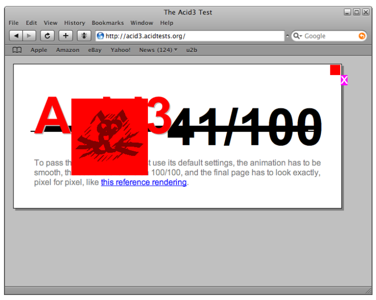
To pass the test, a browser must use its default settings, the animation has to be smooth, the score has to end on 100/100, and the final page has to look exactly, pixel for pixel, like this reference rendering.

Release r31342 of WebKit [Sta08]

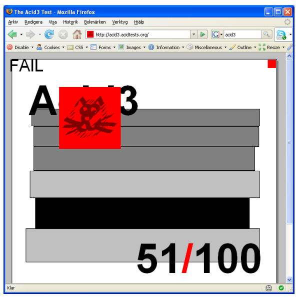


To pass the test, a browser must use its default settings, the animation has to be smooth, the score has to end on 100/100, and the final page has to look exactly, pixel for pixel, like this reference rendering.

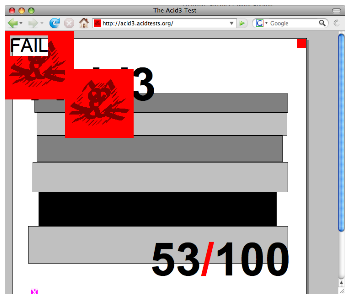
Safari 3.1
Mac OS X 10.5.2



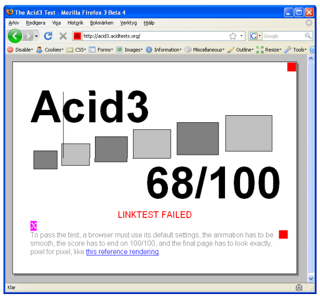
Safari 3.0.4 (523.15)
Win XP SP 2



Firefox 2.0.0.12
Win XP SP 2

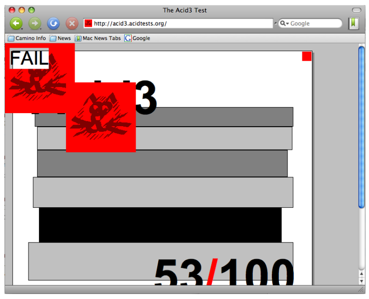


Firefox 2.0.0.13
Mac OS X 10.5.2

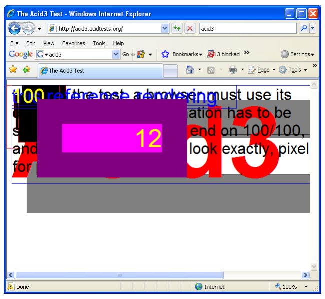


To pass the test, a browser must use its default settings, the animation has to be smooth, the score has to end on 100/100, and the final page has to look exactly, pixel for pixel, like this reference rendering.

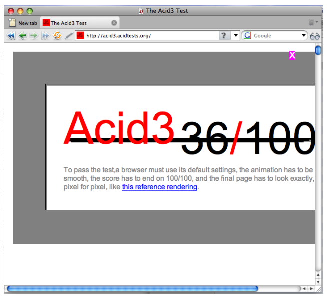
Firefox 3 beta 4
Win XP SP 2



Camino 1.5.5
Mac OS X 10.5.2

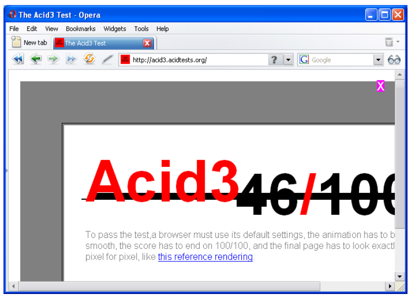


IE 7.0.5730.13
Win XP SP 2



To pass the test,a browser must use its default settings, the animation has to be smooth, the score has to end on 100/100, and the final page has to look exactly, pixel for pixel, like this reference rendering.

Opera 9.25
Mac OS X 10.5.2
Crashed after 36



Opera 9.26
Win XP SP 2
Crashed after 46

Appendix C. AJAX Code Sample

```
<div id="ajax_output">
Waiting to be replaced by Ajax Call
</div>

<script type="text/javascript">
<!-- // Required to be compliant with XHTML-->
 var xmlHttp=null; // Defines that xmlHttp is a new variable.
 // Try to get the right object for different browser
 try {
    // Firefox, Opera 8.0+, Safari, IE7+
    xmlHttp = new XMLHttpRequest(); // xmlHttp is now a XMLHttpRequest.
 } catch (e) {
    // Internet Explorer
    try {
       xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
       xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
 }
 xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4)
       try { // In some instances, status cannot be retrieved and will
               // produce an error (e.g. Port is not responsive)
          if (xmlHttp.status == 200) {
             //Set the main HTML of the body to the info provided by
             // the AJAX Request
             document.getElementById("ajax_output").innerHTML
                = xmlHttp.responseText;
          }
       } catch (e) {
          document.getElementById("ajax_output").innerHTML
             = "Error on Ajax return call : " + e.description;
       }

 }
 xmlHttp.open("get","pages/index.html"); // .open(RequestType, Source);
 xmlHttp.send(null); // Since there is no supplied form,
                        // null takes its place as a new form.
</script>
```

This example is taken from `http://en.wikipedia.org/wiki/Ajax_(programming)`

Appendix D. Google Maps API Code Sample

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps?
file=api&amp;v=2&amp;key=ABQIAAAAly60PJPTKn52cmHklOawphSDpq-8M7SD0oVCTvZ7J4UBlv1dIxQTX5hC
NBSieJ1510Q3vngcsZJ52A"
      type="text/javascript"></script>
    <script type="text/javascript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        // 60.205796, 24.962525 is the latitude and longitude of kumpula
        var location = new GLatLng(60.205796, 24.962525);
        var kumpula = new GMarker(location);
        map.setCenter(location, 15);
        map.addOverlay(kumpula);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

This example is a modification of the simple example found from `http://code.google.com/apis/maps/documentation/examples/index.html`

Appendix E. Loading Times

	1	2	3	4	5	6	7	8	9	10	Average
Load	4.1	3.6	3.9	3.9	3.5	3.8	3.9	3.6	3.7	3.6	**3.8**
Clear cache, reload	5.7	5.4	5.4	6.7	5.5	5.3	5.3	5.6	5.5	5.4	**5.6**
UNRESIZED PICTURES											
Open activities	2.1	2.1	2.0	2.1	2.1	2.0	2.0	2.1	1.9	1.8	**2.0**
Reload, open activities	5.4	4.3	5.0	4.4	4.4	4.6	5.1	4.5	4.9	4.4	**4.7**
Clear cache, reload, open activities	8.1	7.7	8.0	8.0	8.3	8.0	8.2	8.5	8.3	8.5	**8.2**
RESIZED PICTURES											
Open activities	2.1	1.9	1.8	1.8	1.8	1.8	1.9	1.8	1.9	1.9	**1.9**
Reload, open activities	4.7	4.6	4.9	4.3	4.9	4.6	4.8	4.8	4.5	4.7	**4.7**
Clear cache, reload, open activities	6.3	7.8	7.6	7.6	7.5	7.7	7.7	7.7	7.7	7.7	**7.5**

Table 3: Loading times, in seconds, for Funnelry, measured with a stop watch.