Social networks and mobile devices

The next generation mobile phonebook with social networking widgets

ERIK HEDIN

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-18

Social networks and mobile devices

The next generation mobile phonebook with social networking widgets

Erik Hedin

Examiner and academic supervisor:
Professor Gerald Q. Maguire Jr.

Industrial supervisor:
Per-Erik Brodin, Ericsson Research

15 September 2008

Abstract

Social networking services on the Internet are growing and increasing numbers of people are using these new ways to communicate and share information. Many users are communicating with both friends from outside the service as well as with people they have only been in contact with through a social networking service.

At the same time mobile phones are becoming more powerful and increasingly offer high speed Internet connectivity. Because of this people expect these social networking services to be available on their mobile device, as well as on their personal computer. Given the capabilities of today's mobile devices, it is possible to extend the existing phonebook with capabilities to support a variety of social networking services in addition to the existing communication options. By integrating the contacts gained from the social networking service into the mobile phonebook the user can reach these contacts easily.

Communication in online social networks via a mobile phone is expected to grow in popularity in the near future [1]. Several operators are simplifying access to these networks for their customers by offering different ways to connect to social networks [2][3]. However, access to these networks is often done via web sites or dedicated applications. These solutions are not optimal since browsing the web using a mobile browser can be bothersome and dedicated applications require installation of the application as well as any subsequent updates. Widgets on the other hand solve these problems in a convenient way. They can enable access to a device's full functionality, just like dedicated applications, and provide a platform for easy development using web technologies. Furthermore, widgets are highly flexible for example, updates do not require reinstallation, and when new social networks become popular new widgets that connect to them can easily be developed and distributed.

The goal of this thesis is to expand the mobile phonebook with functionality to enable communication on online social networks. To reach this goal I have created a new widget platform, where hybrid widgets run partly in a Java ME application and partly in the mobile web browser. This solution has the potential to significantly enhance the ways we communicate and interact with people. The users are therefore likely to have a larger number of people whom they interact with – rather than a smaller number; hence increasing the overall communication between people.

This thesis project was performed at Ericsson Research in Kista. Any opinions stated in the thesis are strictly my own. Similarly any technology selections made are my own and do not necessarily reflect any official position(s) of Ericsson.

Sammanfattning

Sociala nätverkstjänster på Internet växer och ett ökande antal personer använder dessa nya sätt att kommunicera och dela information. Många användare kommunicerar med både tidigare vänner och med personer de endast har haft kontakt med genom en social nätverkstjänst.

Samtidigt som de sociala nätverkstjänsterna har växt har mobiltelefoner fått betydligt bättre prestanda och erbjuder i allt högre grad höghastighetsanslutningar till Internet. På grund av detta förväntar sig människor att dessa sociala nätverkstjänster ska vara tillgängliga från deras mobiltelefon, precis som från deras dator. Givet dagens mobiltelefoners förmåga är det möjligt att utvidga den befintliga telefonboken med funktionalitet för att stödja en mängd olika sociala nätverkstjänster utöver de kommunikationsalternativ som redan finns. Genom att integrera kontakterna från de sociala nätverkstjänsterna i mobilens telefonbok kan användaren enkelt nå dessa kontakter

Kommunikationen på Internetbaserade sociala nätverkstjänster från mobiltelefonen förväntas växa i popularitet den närmaste tiden [1]. Flertalet operatörer håller för närvarande på att förenkla åtkomsten till dessa nätverk för deras kunder genom att erbjuda olika sätt att ansluta till social nätverk [2][3]. Åtkomst till dessa nätverks sker ofta genom hemsidor eller dedikerade applikationer. Dessa lösningar är dock inte optimala eftersom att använda mobiltelefonen för att surfa på webben kan vara ansträngande och dedikerade applikationer kräver installering av applikationen samt efterföljande uppdateringar. Widgets kan lösa dessa problem på ett smidigt sätt. De möjliggör åtkomst till enhetens fulla funktionalitet, som en dedikerad applikation, men tillhandahåller en plattform för enkel utveckling med hjälp av webb teknologier. Fortsättningsvis så är widgets mycket flexibla, uppdateringar kräver inte ominstallallation och när nya sociala nätverk blir populära kan nya widgets som ansluter till dem enkelt utvecklas.

Målet med denna uppsats är att utvidga mobiltelefonens adressbok med funktionalitet som möjliggör kommunikation på Internetbaserade sociala nätverkstjänster. För att förverkliga detta har jag skapat en ny widget platform där hybrida widgets körs delvis i en Java ME applikation och delvis i den mobila web läsaren. Denna lösning har potentialen att betydligt förhöja sätten som vi kommunicerar och interagerar med människor. Användarna kommer därför antagligen att ha ett större antal människor som de interagerar med – snarare än ett litet antal. Följaktligen kommer det att öka den totala kommunikationen mellan människor.

Detta examensarbete utfördes på Ericsson Research i Kista. Alla åsikter som uttrycks i uppsatsen är mina egna. Likaså är alla val av teknologier som gjorts mina egna och reflekterar inte nödvändigtvis Ericssons officiella position(er).

Table of contents

List of Figures

List of Tables

List of Abbreviations

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
CDC	Connected Device Configuration
CHAPI	Content Handler API
CLDC	Connected Limited Device Configuration
CSS	Cascading Style Sheets
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
FOAF	Friend Of A Friend
FQL	Facebook Query Language
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IM	Instant Messaging
JAD	Java Application Descriptor
JAR	Java Archive
Java ME	Java Platform, Micro Edition
Java SE	Java Platform, Standard Edition
JP-7	Java Platform 7
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MSA	Mobile Service Architecture
MWS	Mobile Web Server
OMA	Open Mobile Alliance

OTA	Over-The-Air
PIM	Personal Information Manager
PNG	Portable Network Graphics
RDF	Resource Description Framework
REST	Representational State Transfer
SMS	Short Message Service
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WRT	Web Runtime
WSL	Widsets Scripting Language
XFN	XHTML Friends Network
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XSS	Cross Site Scripting

1 Introduction

Over the last few years mobile communication devices have become increasingly powerful and today many of them support applications being installed and executed on the device. Simultaneously the expansion of the third generation wide area cellular networks and other high speed wireless data technologies have made the Internet more accessible for mobile users, at prices suitable for surfing the Internet and with sufficiently low delay that even interactive packet based services are feasible.

At the same time, the user experience of the web is expanding - facilitating increased collaboration and information sharing between users. Along with low cost high quality cameras, microphone, arrays etc. the web is supporting growing amounts of user generated content. This has lead to the evolution of online social networks[1] and other means of fostering interaction. Over the last several years, the way that people use the web has changed, today four of the ten most visited Internet sites are social networking services [4].

With the evolution in both high bandwidth wired & wireless access as well as the increasing impact of social networks, people expect these new ways of communication to be available in their mobile phone. Today, on-line social networks are becoming accessible via a mobile phone and both applications and websites specifically adapted to suit mobile devices exist. However, these applications and dedicated web sites are not the best method to access online resources from a mobile device. Magnus Wester, innovation director at Ericsson's Business Unit Multimedia, says "With significant user growth and clear mobile advantages, there is a compelling argument for network operators to introduce some kind of social networking service". But he adds that there are also risks of committing to one particular community since large, ego-centric communities have in some cases had short lifecycles.

A widget is a relatively new type kind of web application that is now becoming available for mobile devices. Widgets are single purpose web applications that greatly simplify access to online resources. Widgets do not require installation and can be easily be upgraded with little or no user participation. Both of these advantages make widgets very suitable for mobile devices and are two of the reasons why companies such as Nokia and Vodafone are developing their own widget engines [5] [6] (Widget details can be found in section 2.4).

Since your contacts are generally members of several different social networks and only some have entries in your phonebook, it is desirable to find a means to store and access them irrespective of how you first contacted them. A natural place for this on a mobile communications device is in the existing contact handler: the phonebook. This phonebook potentially has a central role since one of the main purposes of carrying a mobile communications device is to stay in contact with your friends, family, co-workers, etc. and the phonebook facilitates contacting all of these people. The phonebooks in mobile telephones has not evolved much over the years and functionality to contact people via your phonebook based upon contacts from online social networks, blogs, or instant messaging have not yet been implemented.

[1] Danah Boyd and Nicole Ellison define social network sites as "web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system".

The problems addressed in this master thesis concern how to simplify users' access to their social networks on the Internet when these users are using their mobile (phonebook equipped) device. The thesis also addresses how to match contacts from social networking services with contacts in the mobile device's phonebook.

Additionally a prototype solution will demonstrate this concept. This prototype will enhance the capabilities of a mobile phone by integrating existing social networks with the phonebook. This application is started from the phonebook so it gives the impression of being integrated with the phonebook, while at the same time provides fast access to contact information. Furthermore, the application provides a widget platform which enables users to create their own widgets, using web technologies, that can access device functionality such as persistent storage and phonebook information. One of the initial design goals of the project was to try to avoid using a remotely located server and put all the logic of the application in the mobile device. An application that requires a fixed server might need, potentially costly, updates of infrastructure if it rapidly becomes popular.

2 Background

Most mobile devices today provide Internet connectivity; as a result it is possible to access online social networks from the mobile device. There are already several social networking services that have realized this, thus they provide user interfaces specifically adapted for mobile devices. These interfaces are either applications that a user installs and runs on their device or websites that have been adapted for the small screens of mobile devices. These applications do not provide many features beyond simple access to the social network(s) that have been selected by their application provider. Furthermore, none of these applications can synchronize the device's phonebook with the social network's contact list for a given user.

There has been much talk of social network portability over the last year and that people want to be able to transfer their social network information from one service to another. This has lead to the development of several new technologies that specifically aim to facilitate interaction with social networking services [7] [8].

Additionally, there are many services that provide online backup of your device's contact list. Several standardization bodies have developed markup language standards for synchronization of different devices and to provide portability of phonebooks between devices, an example of such a standard is SyncML [9]. Unfortunately, the problems of moving the online social network information to your mobile device and how to integrate this information into the mobile domain have not yet received much discussion. This thesis will examine solutions to both problems.

Since the first desktop widgets was introduced a few years ago, the development of widgets has progressed quite far. Widgets are single purpose web applications that are easy to use and develop. These aspects of widgets make them particularly attractive on mobile devices and several experts in the mobile industry expect the breakthrough of mobile widgets to occur soon [10] [11].

2.1 Social networks

Social network graphs can be created using information found in different places, such as the contact lists from social networking sites, the phonebook from mobile devices, and blog rolls. By creating social network graphs it is possible to see how they overlap and use that information to synchronize the contacts on different social networks. This information can then be used to enable communication with the people found in the different social networks. Figure 1 tries to illustrate how different social networks overlap. Furthermore it shows that different contact lists, such as a mobile device's phonebook or a contact list on a social networking site, may contain contacts from several different social networks and therefore also overlaps.

There are several projects that try to facilitate the creation of social networks graphs and to make this information available to anyone. Friend of a Friend (FOAF) (see section 2.1.4) and XHTML Friends Network (XFN) (see section 2.1.3) are examples of these projects. Both attempt to create a network of web pages that are easily interpreted by computers. These web pages contain information about relationships between people. Thus a computer can automatically traverse the network and collect information about the relationships.

However, people use different aliases in different social networks. The OpenID project (see section 2.1.2) tries to solve this problem of each person having several aliases across the web by giving each person a unique URI as ID. Unfortunately, OpenID is used by web sites to authenticate users and not for identification. Furthermore, there are issues regarding what type of information is used for naming in the mobile device and on the web. SyncML (see section 2.1.5) is a standard for synchronizing mobile devices. This technology is used by ZYB (see section 3.3) to upload a mobile device's phonebook to a server. The uploaded information is then used to create an online social network in which the user can add information about contacts to enable communication with other social networking services, such as Facebook.

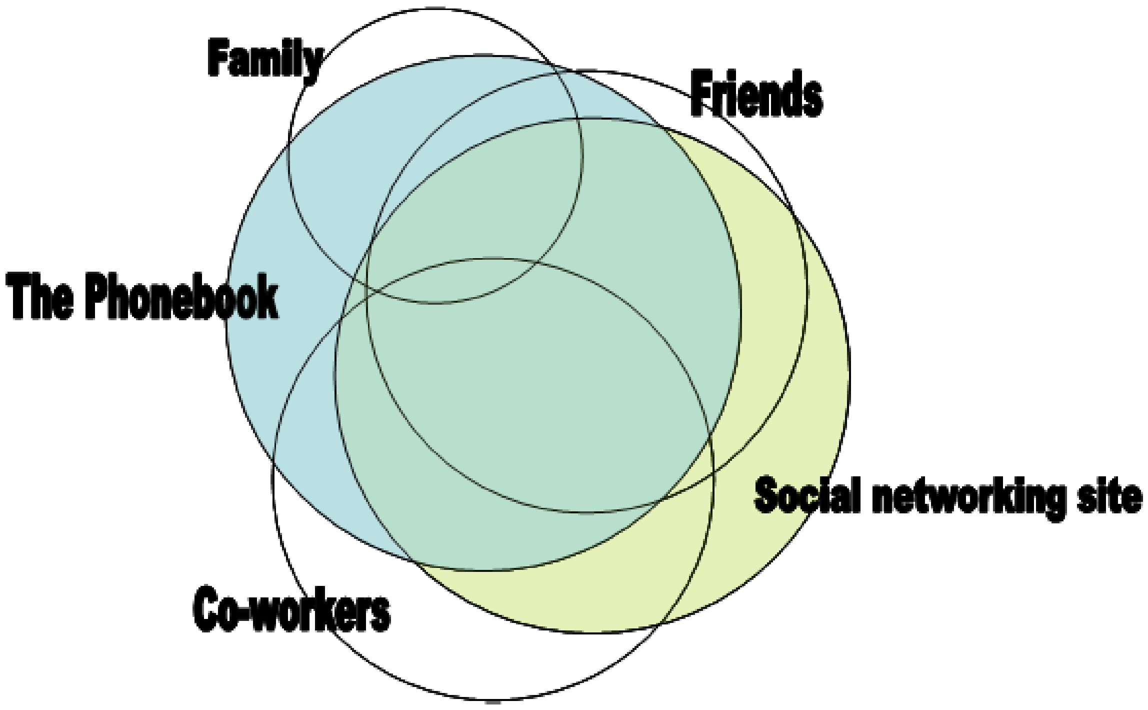


Figure 1: Social network overlap

2.1.1 Naming

The process of automatically matching contacts from the phonebook with contacts from online social networking services is difficult for a number of reasons. Naming of a contact is done differently in the two domains, because of the difference in the primary way in which the contact information is used. For example a telephone number is used in the mobile domain and the e-mail address is commonly used for naming online. Even though it may be possible to store email addresses in their phonebook, people usually do not do so. This is because a mobile phone was viewed by many people as not being a suitable device for generating e-mail (particularly on devices not equipped with a full keyboard). Of course this point of view was not universal, for example – Blackberry users are very used to using e-mail addresses as the primary way of identifying users. Additionally, even if an e-mail address is available in the mobile device, this e-mail address may **not** be the same one used to identify the contact within the social networking service. In a similar fashion, while it is possible to specify phone numbers in a social networking profile, few people do this. Part of the reason for not using phone numbers could be that people do not want a single identification – as they wish (or need to) have different roles and want to separate these roles. These problems make synchronization harder because there is no common means of identification in both the mobile and the Internet domain[2].

Desirable is a way to identify a person in a social networking service using the phone number for this person in our phonebook. However, to maintain personal integrity some people may not want their phone numbers to be publically available. A way to solve this could be to have a hash of the phone number and only use this hash value as the identifier. Unfortunately, a phone number is a maximum of 15 digits it would be computationally feasible to find the phone number using the hash value, rendering this solution unsuitable.

2.1.2 OpenID

OpenID enables Internet users log on to many different web sites using a single digital identity, thus reducing the number of user names and passwords each person needs. It is a decentralized, free, and open standard that lets users control what information they want to provide. OpenID is becoming more popular and is supported by many large organizations including: AOL, Google, IBM, Microsoft, Orange, VeriSign, Sun, Novell, and Yahoo [12]. However, despite these organizations saying they support OpenID - few of these organizations have actually implemented OpenID authentication when signing in to their services.

2.1.3 XHTML Friends Network

XHTML Friends Network (XFN), is not actually a social network, but is a microformat used to represent human relationships using links in an easy way. Thus web authors can indicate their relationships to other people by adding one or more keywords as the "rel" attribute to <a href> tag in HTML encoded homepages[3]. XFN is mostly used by bloggers to indicate their relationships to people in their blogroll. This information can easily be parsed by computers compared to parsing the whole blog and extract user identities; therefore a computer can easily compute a social network using this information.

By publishing the following link on their homepage a person could indicate that they are friends with Jonas Exempel and that they have met in real life [13]. Of course none of this information needs to be true, thus it is possible for computer to collect this information and generate a social network which is false!

<a href="http://jonas.exempel.org" rel="friend met">Jonas Exempel</a>

[2] Many might say that there should be no single identification for a user for reasons of privacy and personal integrity. For example, in some countries it is not permitted to have such a single identifier; while in other countries such single identifier exist – but using them would subject the person or organization storing this identifier and other information to be subject to laws concerning personal data (eg., in Sweden PUL).

[3] The anchor element is used to create links to other online resources in a HTML encoded web page.

2.1.4 Friend of a Friend

The Friend of a Friend (FOAF) project aims at creating a web of machine-readable pages describing people, the relationships with others and the things that they create and do [14]. It allows groups of people to create social networks without the need for a centralized database. In fact, from this information multiple social networks can be computed – some of them of interest to the user and some which can be used for malicious purposes.

Anyone can create their own FOAF file, in which they describe themselves and their relationships and post this file on their web site. We assume that each FOAF file has a unique identifier (such as the person's e-mail address or a URI of the homepage or weblog of the person), which is used as a unique ID. Applications can use the information in the FOAF file and the unique ID to compute a network of relationships between people.

The FOAF project is an application of the semantic web [15]. The semantic web provides a common framework that allows data to be easily shared and reused across applications. FOAF is an extension to the Resource Description Framework (RDF) which is one of the fundamental building blocks for the semantic web [14].

2.1.5 SyncML

SyncML is an open, platform independent, information synchronizing and device management standard. Development of SyncML was started by the SyncML initiative, but was later consolidated into the Open Mobile Alliance (OMA). It is currently supported by many mobile device manufacturers and SyncML clients are pre-installed in most mobile phones and devices [16].

Several mobile network operators provide SyncML servers for their customers to synchronize their mobile phones with [17] [18]. There are also several open source SyncML servers available [19] [20]; these could be run on any network attached machine which is reachable from the mobile device. SyncML is commonly used to synchronize contacts, but can also be used to synchronize other information, such as calendars.

2.2 Developing applications for mobile devices

Since mobile phones have become increasingly powerful and because the platforms have become increasingly open to developers (outside the vendor itself) in recent years the interest in developing applications for them has grown. However, because there are so many different manufacturers and many different software environments (with small variations even between models of devices in the same family) it is both tedious and expensive to port every application to all these different devices.

There are many different programming platforms available for developing applications for mobile devices: Android[21], Qualcomm's BREW[22], mobile browser based, Adobe's Flash Lite[23], Sun Microsystems' Java ME [24], Lazarus, Python, Microsoft's .Net Compact [25]; as well as several different operating systems: Microsoft's Windows Mobile [26], Garnet OS [27], Linux, and Symbian [28]. Today browser based and Java ME are the most widely supported platforms used with mobile devices [29]. Browser based applications development will be covered in section 2.2.4. Next we will examine the Java ME platform.

2.2.1 Java ME

Sun Microsystems developed Java ME in order to port Java mobile applications to multiple platforms. The Java ME platform is a subset of the Java platform and APIs. This subset was intended for devices that have limited resources, such as mobile phones and other portable devices. For a device to be able to run a Java ME application it needs to have a Java Virtual Machine (JVM) installed [24] and to have a number of libraries present.

Most APIs that Java ME supports have been specifically adapted to be very small (in memory and computation requirements) in order to run as smoothly as possible on limited resource devices. Unfortunately, this also means that many functions that are supported in Java SE have been removed, thus breaking the basic Java goal of "Write once, run everywhere".

Java ME technology is based on three elements:

A configuration	provides the basic set of libraries and virtual machine capabilities for a broad range of devices
A profile	a set of APIs that support a narrower range of devices
An optional package	a set of technology-specific APIs

Over time the Java ME platform has been divided into two base configurations, one for *very* limited mobile devices and one for more capable mobile devices, such as smart-phones. The configuration for the first class of devices is called the Connected Limited Device Configuration (CLDC) and the configuration for more capable configuration is called the Connected Device Profile (CDC) [24]. The CLDC configuration has been used in this thesis project. The CDC configuration is commonly used when creating application for more advanced portable devices including smart communicators, high-end personal digital assistants, and set-top boxes.

CLDC was specifically designed to provide a Java platform to run on devices with limited resources. The Virtual Machine that comes with the CLDC is smaller than a regular JVM and is called a Kilobyte Virtual Machine (KVM) because the size is usually measured in kilobytes. On top of the different configurations of the Java ME platform a number of profiles define a set of higher-level APIs. A widely adopted combination is CLDC with the Mobile Information Device Profile (MIDP) to provide a Java application environment for mobile phones and similar devices. A device manufacturer can choose to implement optional APIs, in order to support hardware that is specific to the device, such as a GPS receiver [24].

2.2.2 Java ME applications

An application created for a device that implements CLDC and MIDP is called a MIDlet. A MIDlet consists of a Java ARchive (JAR) file and optionally a Java Application Descriptor (JAD) file. The JAD describes the MIDlets that are distributed in the JAR file. The JAR file contains the executable code and all of the associated static data. It is possible to run a MIDlet using just the JAR file; however, there are cases when the JAD file needs to be present. For example, if the vendor signs the MIDlet, then the MIDlet can access some functionality without asking the user; or if the MIDlet is to be installed over the Internet, usually called Over-The-Air (OTA) installation, then additional information needed to do this needs to be specified in the JAD [30].

A major benefit of using Java ME is that applications easily can access resources on the device without requiring much, if any, porting of the application – while at the same time allowing the same MIDlet to be used on a wide variety of devices. The downside of developing applications using Java ME is that users of this application need to download and install the program as well as any subsequent updates [31]. Additionally, it is harder to create good-looking user interfaces in Java ME, as compared to the browser based approach.

2.2.3 Mobile Service Architecture

The Mobile Service Architecture (MSA) Specification, JSR 248, is a Java architecture definition that describes the essential Java client components of an end-to-end wireless environment [32]. The MSA specification defines a set of Java ME technologies and shows how these technologies can be integrated in a mobile device to create a mobile Java platform [32].

The MSA platform builds on the earlier Java ME specifications, by including MIDP, CLDC, and Java Technology for the Wireless Industry . As the wireless device market continues to evolve and incorporate new technologies and services in mass-market handsets, there has been a requirement to create a platform that standardizes these new technologies [33]. However, the number of new specifications seems to make this a moving target.

There were three design goals for the MSA specifications. The main goal was to minimize the fragmentation of mobile Java environments, by defining a predictable and highly interoperable application and service environment for developers. The second was to enable this environment to be used in a wide variety of different markets and customer segments. This was achieved by introducing two platform definitions: MSA and MSA Subset (see Figure 2). The third goal was to ensure the highest level of consistency in the definition of both MSA and the upcoming MSA Advanced environment [32]. (Thus introducing yet another platform. So much for the first goal!)

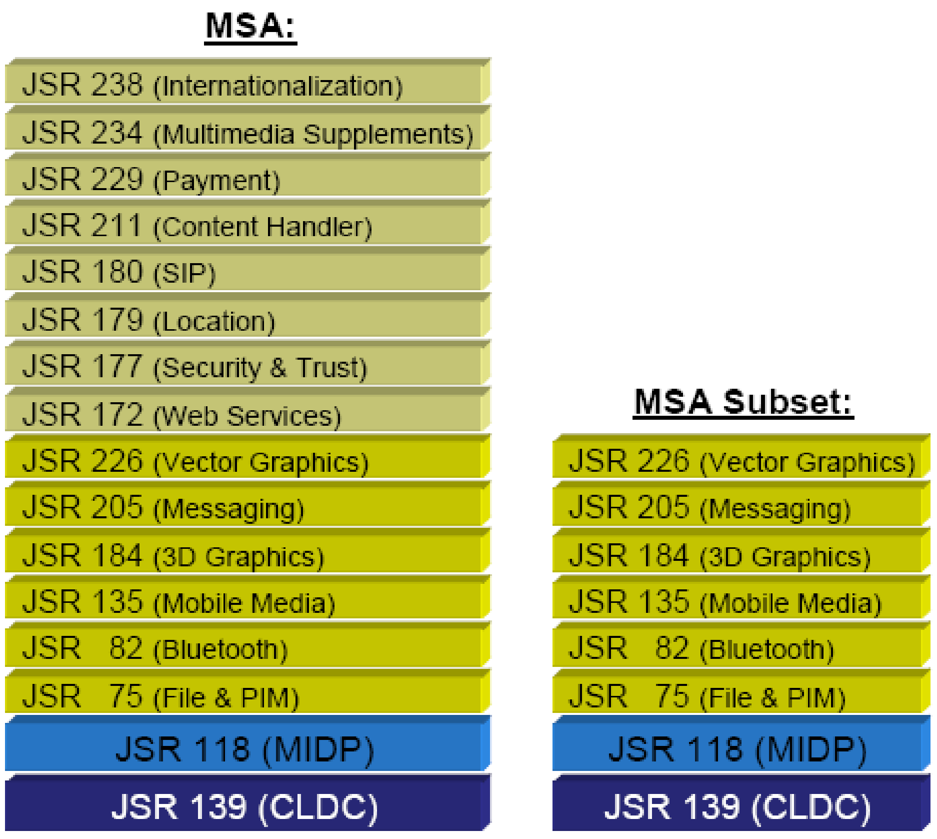


Figure 2: The JSRs required by the MSA and MSA subset definitions [32].

Note that the JSR 211: Content Handler (CHAPI) APIs (included only in the MSA platform) provides functions to register a MIDlet as a content handler [34]. This means that a registered MIDlet can be launched by other applications either by using the content handler id, URL, or MIME type. This reduces the necessary user interaction, as the user does not need to explicitly run this MIDlet.

2.2.4 Mobile browser based

Web browsers for mobile devices have over the last several years gone from being stripped down versions of regular browsers to supporting most web technologies; such as XHTML, CSS, JavaScript, and SVG. This evolution has occurred because the devices have greater resources (thus eliminating many of the perceived limitations that lead to the various subsets of Java, BREW etc.). The most popular mobile browsers include: Opera mobile [35], Nokia Web browser [36], and Access's Netfront [37].

There are several major advantages of developing browser based mobile applications compared to other widely used techniques. One is that the applications do not need to be installed on the phone, so when an update or a new version of the application is released, the user does not need to reinstall (or update) the application. Another major advantage is that cross platform porting of the application is not necessary. A third advantage is that the techniques used

(XHTML, CSS, JavaScript, AJAX) are widely used, provide very forgiving error handling, and have a short learning curve (especially for existing web developers). For further information see [31].

2.2.5 JavaScript

JavaScript is an interpreted programming language with object-oriented capabilities. The core JavaScript language syntactically resembles Java, but the resemblance ends there. JavaScript was developed by Brendan Eich of the Mozilla organization and its official name following standardization by the European Computer Manufacturers Association (ECMA) is ECMAScript. However, in practice everyone still calls it JavaScript. JavaScript is usually used for client side scripting and to run code inside a web browser in order to make the web page more dynamic [38]. With JavaScript it is also possible to communicate with the server without reloading the page. This causes a web page to act much like an application. This method of programming is called Ajax (Asynchronous JavaScript and XML) and was originally intended to enhance a web page's interactivity, speed, functionality, and usability. Additionally, because most mobile phones have a web browser and most of them act in the same way, this often provides a good alternative to Java ME when developing applications for mobile devices.

JavaScript code running in the browser uses the XMLHttpRequest API to exchange data with the server without reloading the entire page. The information received from the server is processed by the JavaScript code and only updates specific parts of the web page. The updating of the page is done using the Document Object Model (DOM) [38]. In this way entire web pages do not have to be reloaded each time there is a need to fetch data from the server. By only asking for relevant data and updating only small parts of the page the responsiveness of the page is enhanced (and the communications traffic reduced). These small amounts of data are often exchanged in the XML data format, but other data formats can also be used, for example: JSON, HTML, and plain text.

There is, unfortunately, no access to device functionality from the browser, which greatly limits what these applications can accomplish. One way to solve this problem is using a proxy server that can be contacted by making HTTP requests from the browser. In that way the same origin policy is bypassed and scripts running in the browser can make Ajax calls to utilize resources in the proxy that can access device functionality (see Tomas Joelsson’s thesis [39]).

One big security issue when developing web applications is cross site scripting (XSS). Web applications which allow such dynamic content may face code injection by malicious web pages. A XSS may enable an attacker to gain elevated access privileges to sensitive page content, session cookies, and other objects. There are currently three types of cross site scripting, for further details see [40].

2.2.6 Rhino

Rhino is an open source JavaScript engine. It was implemented entirely in Java SE and is managed by the Mozilla Foundation [41]. Rhino converts JavaScript scripts into Java classes, thus making it possible for users to script Java (i.e., turning JavaScript code into Java). Rhino works in both compiled as well as interpreted mode. Rhino is an implementation of the core language only and does not contain objects or methods for manipulating HTML documents (although a library could be implemented to make this functionality accessible for a given browser). The motivation for embedding JavaScript within Java applications is typically to allow users to customize the application using a JavaScript script. The Firefox web browser, for example, uses JavaScript scripts to control the behavior of its user interface [38].

Rhino includes a feature that allows a JavaScript script to query and set fields and invoke the methods of Java objects. Even if a script is not passed any references to Java objects it can create its own references to Java objects. Host objects are special JavaScript objects that provide special access to the host environment. For example, in a browser environment, the Window and Document objects are host objects [42].

Each JavaScript script is executed in a Context. The Rhino Context object is used to store thread-specific information about the execution environment. There should be one and only one Context associated with each thread that will be executing JavaScript. A script must be executed in a scope. A scope is a set of JavaScript objects and it was created by a context. Note that a scope is independent from the context and can be shared between contexts [43]; thus a scope allows cross thread communication.

2.3 Accessing online social networks

According to Alexa Internet [4], the largest social networking services at the moment are MySpace [44], Facebook [45], Hi5 [46], and Orkut [47]. These communities all provide APIs for remote access to user and friend information as well as other information - without using their websites. MySpace, Hi5, and Orkut provide this by supporting the OpenSocial API [48].

The OpenSocial API is a set of common APIs for building social applications on many websites. OpenSocial was developed by Google and is currently supported by most of the major social networking sites. It gives a programmer the ability to develop applications for social networks or applications that are hosted by another site, but accesses information from these networks. There are two ways to access the OpenSocial API: client-side access using a JavaScript API and server-side access using RESTful data APIs (see section 2.3.1). This means that you can either develop your social application using the JavaScript APIs and install it on your profile page of your social network site; or, if you want to create an application and put it on another website, then you can use the RESTful APIs for server to server communication and retrieve data from any of the OpenSocial sites. The RESTful API serves as a common protocol understood by all OpenSocial version 0.8 compliant clients and servers [48]. However, I have currently not been able to find any OpenSocial servers that implement the RESTful API.

In addition to OpenSocial, Hi5 has also implemented a SOAP API; as well as support for some RESTful commands of their own [49]. MySpace also gives developers access to information through a proprietary RESTful API [50].

Facebook provides a developer platform that is a standards-based web service with methods for accessing and contributing Facebook data. The platform supports a REST-like interface, as well as an SQL-style interface called Facebook Query Language (FQL) [51]. In order for a user to use an application to access the REST resources both the user and the application needs to be authenticated. The application is authenticated by supplying an application key during authentication of the user. This application key is uniquely assigned to the vendor, and identifies, among other things, the list of acceptable source IPs for the call.

2.3.1 Representational State Transfer (REST)

REST is an architectural style for building large-scale networked applications that strictly refers to a collection of network architecture principles which outline how resources are defined and addressed [52]. The concept of REST was first introduced in the doctoral dissertation of Roy Fielding [53]. REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations [53].

REST supporters claim that the web's scalability relies on a few key design principles [54]:

- Application state and functionality are divided into resources
- Every resource is uniquely addressable using a universal syntax for use in hypermedia links
- All resources share a uniform interface for the transfer of state between client and resource, consisting of:
    - A constrained set of well-defined operations
    - A constrained set of content types, optionally supporting code on demand
- A protocol which is:
    - Client-server
    - Stateless
    - Cacheable
    - Layered

REST describes a networked system in terms of:

- data elements (resource, resource identifier, representation)
- connectors (client, server, cache, resolver, tunnel)
- components (origin server, gateway, proxy, user agent)

A central concept of REST is the existence of resources and that each resource can be referred to using global identifiers such as Uniform Resource Identifiers (URIs). A resource is a source of specific information that components of the network can access and manipulate. In order to do that the components, clients and servers, communicate using standardized interfaces (for example HTTP) and exchange representations of these resources. Systems which follow Fielding's REST principles are often referred to as RESTful.

It is possible to design web services following the REST principles. The resulting RESTful web service has many benefits, such as:

- Supports caching

- Stateless
- Requires less client side software since many resources are accessible through the browser
- Uses widely used protocols

Most RESTful web services today use URIs to identify resources. They support the HTTP requests GET and POST, and often also PUT, HEAD, and DELETE, to modify or retrieve resources. The representations are usually transferred in XML or JSON over the HTTP protocol.

2.4 Widgets

Widgets are a class of client-side, single purpose, web application for displaying and updating local or remote data. Widgets typically run as stand alone applications outside of a web browser, but it is possible to embed them into web pages. Examples range from simple clocks, stock tickers, news casters, games and weather forecasters, to complex applications that pull data from multiple sources to be "mashed-up" and presented to a user in some interesting and useful way

The first widgets appeared on personal computers around the year 2003 and more recently on web enabled mobile devices [55]. For a device to be able to run widgets it needs to have a widget engine, also known as widget user agent, installed. Note that these widgets require the use of a widget engine and are desired to be an application; unlike graphical user interface widgets which were designed to be part of a toolkit to use by application developers.

A widget provides users with access to online services that they commonly use. This means that, in a lot of instances, users do not need to open up a web browser to get the information that they want. This is an aspect of widgets that makes them particularly attractive on mobile devices, where the monetary cost of downloading web pages is currently an issue for many users [55]. However, this may quickly cease to be an issue due to the ongoing shift to flat-rate (wireless) broadband service subscriptions. Nonetheless, the relatively easier way of accessing online resource that widgets enable is the major benefit that has made them popular on desktop computers. This will probably lead to popularity also on mobile devices, where browsing the web is often considered bothersome.

To make the development of widgets as easy as possible for developers most widgets are developed using technologies traditionally used to develop web pages, such as XHTML, CSS, and JavaScript. Widget engines therefore mimic, in many ways, the behavior of web browsers. In fact, an increasing number of widgets are actually built directly on top of web browsers so they are able to render web pages, while others incorporate web browser components such as JavaScript interpreters [55].

Widgets for all major widget engines are created using web technologies. However, other aspects of widgets are currently undergoing standardization by the W3C. Aspects that need standardization in order to make widgets device independent include: packaging, configuration, an API for widgets, security, and widget engine aspects (such as how automatic updates, persistent storage, and multiple instances of the same widget should be handled) [56].

Widgets and Java applets share many commonalities. For instance, both widgets and applets rely on a pre-installed runtime engine for execution: Java applets rely on the presence of the Java Runtime Environment, while widgets rely on the presence of their target widget engine. Widget and Java applets also share many similar functional aspects, such as being able to do asynchronous HTTP requests to download resources from the web [55].

It is argued that the most notable difference between widgets and Java applets is that widgets are easier for authors to create than Java applets. This argument is made because widgets are created using HTML, CSS, and JavaScript, which have very forgiving error handling and a shorter learning curve compared to Java [55]. Another difference is that Java Applets are intended to run inside web pages, while widgets generally serve the purpose of stand-alone applications that run outside of a web browser [55] (but in fact need to use most of the functions built into a web browser – so the code for rendering pages, etc. either has to be duplicated or made sharable to the widget engine).

2.4.1 Widget Resources

All resource files required by the widget are usually encapsulated into a single file for the purpose of distribution and deployment. The de facto standard for packaging widgets is the ZIP file format, but vendors usually use their own specific file extension (for example .widget, .gadget or .wgz).

The current working draft of widgets standardization from the W3C specifies that a widget contains a start file and may contain one or more configuration documents, icons, and additional resources. A widget may also be digitally signed. A start file is the resource that when instantiated represents the widget. A start file could be defined in the configuration document or, if that is missing, a default start file called index.html or index.htm must be present. Most major widget engines require that a widget resource include a configuration document, in which the author declares metadata and/or configuration parameters that a widget user agent may use to configure a widget upon instantiation. Other files present in the widget resource usually are of the following types: JavaScript, CSS, HTML, PNG, JPEG, and GIF [55].

3 Related work

This section examines existing projects that are relevant in one or another way to this thesis project. Since the abilities of mobile devices and the data rates of mobile networks have been fairly limited, until recently, most of these projects are still under development or have only recently been released.

3.1 Synchronizing Facebook with mobile devices

Most mobile devices manufacturers provide applications to synchronize the device's phonebook and calendar with Microsoft's Outlook. This functionality has been exploited by some people to create several Facebook applications that synchronize mobile devices with Facebook via Outlook [57] [58]. However, there are some problems with this approach. One of these is that Facebook does not allow applications or third party organizations to collect user information for any purpose, this makes it hard to match a contact in Outlook with a contact on Facebook; thus most applications force the user to match contacts manually. An approach that is used by Vringo is to have the user provide his or her phone number in order to be able to match a name with a number [59]. However, Facebook does not provide access to contact information to any third party application, this significantly restricts what information the application is able to synchronize and consequently most applications only synchronize profile pictures and possibly birthdays [60].

3.2 iLabs mobile toolbox

Telenor Research and Innovation's iLabs mobile toolbox aims to facilitate development of mobile clients for participation in popular social communities which communicate via open communication standards. They provide open source Java ME APIs on MIDP terminals for the XMPP [61], HTML [62], and ATOMS [63] standards. These APIs also include a Facebook API which can be used to develop MIDlets that access Facebook, a simple feed API that handles several different news and blogger feeds, as well as a simple XMPP instant messaging client API [64].

The Facebook API is a translation of Facebook's own Java API to Java ME. This API simplifies access to the REST resources that Facebook provides. In order to access the REST resources an application key is required. Since this application key identifies acceptable source IP addresses – this can be used to control who has access to this interface. This requirement for an application key greatly reduces the benefit of using the Facebook API in a mobile application; therefore little testing of the iLabs APIs have been done in this thesis project.

3.3 ZYB

ZYB is a Danish mobile social network recently acquired by Vodafone[65]. They provide a Java ME application that acts as a phonebook, called the ZYB Phonebook. ZYB also provide a backup service for mobile phones. The backup service uses the SyncML protocol to transfer data between the ZYB servers and the mobile device [66]. Once the data is received by ZYB's servers the user can log in and add information about the different contacts, such as Facebook ID. This information is then synchronized with the ZYB Phonebook application on the user's mobile device. This enables the user to communicate with this contact on different online social networks, using the ZYB Phonebook.

Besides the backup service ZYB also provides an application called the ZYB Phonebook. ZYB claims that this application will "transform" our experience of how a mobile phone address book operates, by turning it into a live and interactive experience of your contacts. The application was recently released and is currently in beta testing.

The ZYB Phonebook (according to ZYB) provides several services, for example:

- See the physical location of friends
- See the availability of friends, including what time zone they are in
- Share your calendar with friends
- Receive online activity streams about your friends from services such as Facebook and Twitter
- See photos that your friends posted on photo sharing sites like Flickr

Besides these services the ZYB Phonebook will also be connected to the existing ZYB network [66]. The ZYB network is an online social network that is closely integrated with ZYB's backup service for mobile phones.

3.5 Geotagging

Knowing your physical location is getting easier with low cost GPS receivers, access to the cell ID via your mobile phone, Wi-Fi triangulation services such as Loki [67], mapping of IP addresses to geographical locations, as well as maps and satellite pictures easily available on the web. This has lead to an increasing demand for standardized ways of adding geographical metadata to different kinds of media such as blogs, photos, and websites. This is usually called Geotagging or Geocoding.

GeoRss is a standard for encoding geographical location as part of an RSS feed. This can be used to add geographic location information to web feeds such as blog posts [68].

Geo is a microformat used for marking up geographical coordinates in HTML, XHTML, Atom, RSS, and arbitrary XML. The Geo specification is part of the vCard standard in HTML which is one of several open microformat standards, therefore if a publisher is publishing the name as well as geographic location, then he or she must use vCard. Geo was developed because there was need for a way to simply and easily publish visible, extractable, geographic location information on the web. This demand was driven by bloggers and numerous other sites that publish information. Microformats are becoming more accepted, for example the next Firefox web browser will include native support for several of them, including Geo [69].

Twittervision is a real time visualization of posts to Twitter. It is mashup web application created using Twitter's public feed and Google maps. It places the most recent public Twitter posts that include a location and a picture on a map. You can also change your location by including “L:” followed by your location in your post. This location can be a city, an address, a postal code, a known place, or specified by longitude and latitude [70].

3.6 Loki

Loki is a positioning service provided by Skyhook Wireless Inc. [67]. The service uses Skyhooks Wi-Fi Positioning System to automatically detect a user's physical location (if they are nearby any of the Wi-Fi access points that are registed in the Loki database). They also provide a service for users to upload their current position to Loki’s web site and exchange location information with friends by presenting the location on a profile page. The service also gives the user the ability share their location in other ways. Loki provides links to either a map that shows where the user is or a picture that indicates what city the user is in. This link can be included on a homepage, profile page, or in an e-mail. They also provide a public RSS feed for each user that says what city the user is in and a Facebook application that also indicates what city the user is in [67].

Loki has developed applications for both Microsoft’s Windows mobile and the Symbian series 60. With this application users can change their public position from their mobile phones [67]. A user does not need to use the positioning service provided by Loki. For example, it is possible to manually change the position either by specifying a geographical name or coordinates.

3.7 GyPSii

GyPSii allows users to share their real life experiences in the virtual world using mobile devices and the web [71]. It is a social networking, search, and location based suite of integrated mobile and web applications. It enables users to share, view, and upload pictures, video, text, and points of interest with a Geo-location and find people and places, points of interest, map, and navigate to them all.

The GyPSii software consist of an application that allows you to access maps and see where you, your friends, and interesting places are on the map; as well as send text messages and e-mail to your friends [71]. However, the application is currently only available for Symbian S60, Blackberry, and Microsoft’s Windows mobile devices that have built-in GPS receivers [71].

3.8 Mobile Web Server

The Mobile Web Server (MWS) was developed by Nokia for the Symbian S60 platform. It consists of a port of the Apache web server to Symbian along with a solution that provides a mobile device with a global URL and HTTP access to it [72]. This solution is implemented to circumvent the firewalls that many mobile operator employ to prevent access from the Internet to mobile devices in their network. The solution consists of a gateway that runs on a computer on the Internet and an application that runs on the mobile device. Together with an appropriate DNS configuration they provide a mobile device with a URL in the operator networks of today [72].

MWS enables users to set up a web server on their mobile device and access resources on this device via the Web. Its purpose was to provide new ways to publish and access information on the web. For example, a user is able to access the device's contact list, add calendar entries, and send SMS messages, etc. from a personal computer or another mobile device. Example applications include accessing your friend's and co-worker's calendars as well as mash up applications that use multiple data sources when creating mobile Web sites.

3.9 Web Runtime

Web Runtime (WRT) adds a widget runtime environment to the web browser on Nokia S60 3rd Edition, Feature Pack 2 devices. The Web Runtime platform enables widgets to run independently from the web browser, thereby allowing a number of widgets to run simultaneously in the system. However, due to the physical limitations of handheld device screen sizes, only one widget can be in the foreground at a time. It is also possible to start widgets directly from the S60 idle screen or from the applications menu, which speeds up access to the widget. However, this additional human interaction would seem to not be in keeping with the goals of minimizing the required human input task.

The WRT platform support widgets developed using the following web technologies:

- HTML 4.01, XHTML 1.0 (basic and mobile profile)
- CSS Level 2 revision 1 (CSS 2.1)
- JavaScript 1.5 (ECMA-262 3rd Edition)
- DOM Level 2

There are a few differences between WRT widgets and widgets for other widget engines. Nokia claims that porting widgets to the WRT environment takes very little effort [73]. However, it would seem that this will also lead to yet more fragmentation, due to the additional porting efforts required by the non-standard widget engine.

The Web Runtime uses the open-source WebKit browser engine [74] also used in the S60 Web browser to render HTML. The WebKit browser engine is also used by Apple's Dashboard widget engine [75] and the Yahoo! Widget engine [76] to render HTML [55]. There are many similarities between how widgets for Nokia's WRT and Apple's Dashboard engines are configured and packaged, and it seems like Nokia borrowed many design features from Apple. This makes the job of porting widgets between the two engines very easy. Unfortunately, there are some differences in what APIs are available, what WebKit version is used to run them, and what UI resources are available [77].

Besides the standard features offered by the supported web technologies, the following additional JavaScript features are available to widgets running in the WRT environment [78]:

- Utilizing the underlying user input features of the mobile device to interact with the user
- Self-updating widgets
- UI navigation using either a cursor or tabs
- Rotating the screen: portrait and landscape orientation
- Launching S60 mobile applications

- Localization framework with automatic detection of the device language settings
- Querying the battery power
- Querying the network signal strength and other network information
- Controlling the device’s display back light and keypad illumination
- Controlling the device's vibration functionality

According to a press release from Nokia, a widget will be able to access the S60's numerous applications and services, such as calendar, contacts, GPS, messaging, audio, video [79]. However, there is currently no support for widget access to GPS.

3.10 WidSets

WidSets is a mobile widget runtime technology provided by Nokia. The technology is based on Java MIDP 2.0. Therefore the WidSets runtime can be run on any MIDP 2.0 mobile compliant device [80].

Developing widgets for WidSets requires learning a Java-like, strongly typed, proprietary scripting/programming language called WidSets Scripting Language (WSL). The reason for not writing the code directly in Java is that MIDP 2.0 does not support dynamic classloading. Therefore it is impossible to load new Java classes to a running MIDP application and all functionality needs to reside inside a single JAR file. The WidSets approach has, however, a certain benefit: compiled WSL code will be compressed to smaller size than the same amount of Java bytecode (even when obfuscated). WidSets widgets therefore consume less storage space in the phone memory. Smaller size translates to less data traffic when the code is being transferred to a mobile device [81].

A single WidSets widget consists of an XML file, a varying number of PNG images and a WSL source code file. It is also possible to add styles using stylesheets that resembles CSS. The stylesheet definition can be a reference to a separate file in the widget package, or it can be given as an embedded resource in the XML file [82] [83].

3.11 WidX

WidX is a MIDlet developed by joemoby that allows the user to run widgets on a mobile device [84]. According to joemoby, WidX is currently the only standards-compliant widget engine that runs on Java ME. WidX supports its own subset of the following standards:

- JavaScript
- XMLHttpRequest
- XHTML with W3C Level 1 DOM
- CSS

A typical MIDlet is about 180KB in size. Content downloaded by the MIDlet is cached for offline use. Anyone can create and upload a widget to be used on the WidX platform for noncommercial use. However, according to joemoby there is “opportunity to charge brands / corporates to deploy widgets which gives them full usage statistics and user metrics as well as preferential placement” [85]. This seems to mean that anyone who wishes to develop a widget for commercial use must pay joemoby to make the widget available for the WidX engine. WidX is still in the Alpha testing phase and there are currently only six widgets available for download. However, development of the application appears to have stopped.

3.12 Jaxer

Jaxer is an open source web server developed by Aptana. According to Aptana, Jaxer is the world's first true Ajax server [87]. Jaxer is built on top of the Mozilla engine which provides standards-based, well-tested parsing and APIs for HTML, JavaScript, and CSS; as well as support for XMLHttpRequests, JSON, DOM scripting, etc. Jaxer comes with the Apache web server and as a server it offers access to databases, files, and networking, as well as logging, process management, security, integration APIs, and extensibility [87].

Jaxer offers a unified development model, in which the developer can continue to use exactly the same well-known technologies from the client (JavaScript, DOM, and CSS) on the server, without requiring any server-side technologies other than Jaxer. JavaScript functions on the client can call JavaScript functions on the server to fetch information or access back-end Java objects and network resources. The DOM can be prepared on the server using the same Ajax libraries that will be used on the client when the page is delivered. This makes it possible for an entire rich web application to be written in a single HTML document and since Jaxer is based wholly on established web standards, there is no reliance on proprietary markup or protocols [88].

At a very high level Jaxer works as follows [89]:

1. Jaxer works closely with a web server to process and serve web content and respond to callback requests.
2. Jaxer reads HTML pages from the web server (Apache, Jetty, etc.) before they are sent to the browser, processes them, and returns the processed HTML to the web server, which forwards them to the browser — there are no proprietary XML formats or browser plug-ins needed, Ajax is used all the way.
3. Jaxer integrates the Mozilla engine — the industrial-strength engine that powers Firefox 3 — to provide rich, standards-based, well-tested parsing and APIs for HTML, CSS and JavaScript.
4. To allow seamless calling of server-side JavaScript functions from the browser, Jaxer automatically injects a bit of JavaScript wrapper code into served pages. The actual server-side code remains securely on the server.
5. When you call a server-side JavaScript function from the browser, Jaxer wraps the function's name and arguments in a JSON string, sends them via an XMLHttpRequest back to Jaxer (via the web server), which unwraps the information, calls your server-side function, wraps the results in a JSON string, and returns them to the browser, which unwraps them and passes them back as if the server-side function was running right in your browser. Essentially this is a remote procedure call much like SOAP uses.

The lifecycle of a typical web page built with Jaxer is [89]:

1. The HTML document starts life on the server, either as a static HTML file read from disk or as a dynamic page (generated by PHP, Ruby, Java, etc.).
2. Jaxer receives the document acting as an output (post-process) filter for the web server. It parses and executes it, just as a browser would. Jaxer creates and populates the DOM, executes the JavaScript code designated to run on the server, and so on until the entire document is consumed.
3. The result is a DOM modified by Jaxer and by the code created by the developer: in particular, proxies automatically replace server-side client-callable functions. Some important side effects include storing designated JavaScript functions as callbacks and persisting session-type data.
4. The new DOM is serialized as an HTML document and streamed out to the client as usual.
5. The client receives the HTML document and the processing continues, recreating the DOM from the HTML. Now it will execute the client-side JavaScript placed in the page.
6. When one of the client-side proxy functions is called, its parameters are automatically serialized into a JSON format string, and an XMLHttpRequest is sent to the server to invoke the original function with these parameters.
7. When the server receives this request, the parameters are deserialized, the function invoked with these parameters, and the results are serialized into a JSON string.
8. The data is returned to the client, where it is deserialized and returned as the result of the proxy (or a corresponding client-side exception is thrown).

The JavaScript code designated to run on the server is identified by a “runat” attribute added to the script tag in the HTML document [87]. If the value of the runat attribute is "server" or "both", then the code is executed on the server. If the value is “both”, then the code is also executed in the browser.

It is also possible to register callback functions on the server. In this case code running in the browser can call a function that will be executed on the server the same way it would call a function that would be executed in the browser (similar to a Remote Procedure Call). A function is designated as callable from the browser if it is in a script block with the value “server-proxy" of the runat attribute, or if it has a "proxy" property with a value of true, or if it is included in the Jaxer.proxies array in the page. Jaxer makes a function callable by adding some wrapper code during the processing of the page in the server. Additionally, a feature Aptana calls DOM scraping is available in Jaxer. DOM scraping enables code running in the server to create window objects and load documents from remote URLs into the window object, having that content execute, then being able to go into that window object and pull DOM elements out. This enables code running in the server to create a document and using DOM scraping insert information into that document from several different sites.

4 Implementation

4.1 Background

The goal of this project has been to study social networks on the web, how they apply in a mobile context, and investigate how such networks could be integrated with a mobile phonebook. The popularity of developing mobile applications using web technologies has grown over the last years since mobile browsers have become more capable. Because of this popularity the web technologies have been used in combination with Java ME, which was chosen to enable access to local device functionality, in the development of a prototype solution.

A prototype application has been developed which when integrated with the phonebook simplifies communication with contacts from one or more online social networks. The functionality specific to the different social networks has been implemented using background and foreground widgets. These background widgets are executed on a widget engine that has been developed as part of this project. The foreground widgets are primarily executed in the browser. An initial desire on the design of the application was that no additional infrastructure, besides the mobile device, should have to be set up for the application to work. This goal meant that all logic of the application should reside in the local device. By avoiding fixed servers the application is made less vulnerable by eliminating on of the single point of failure entities in the system. Extremely fast growth in popularity of an application can lead to the server being overloaded, thus rendering the application useless to all users.

4.1.1 The Sony Ericsson K800i

For development and testing of the implementation a Sony Ericsson K800i was supplied by Ericsson Research. The K800i supports Java ME and the Java platform is the Sony Ericsson Java Platform 7. This supports CLDC 1.1, MIDP 2.0, and Java Technology for the Wireless Industry. With this platform it is possible to have several MIDlets running at once and to have MIDlets running in the background. The browser on the device is Access Netfront 3.3 [90].

The browser's support for the major web technologies is fairly good. However, the support for AJAX is poor since XMLHttpRequest is not supported by the browser, which complicates development. Different methods for remote scripting in Netfront 3.3 were covered in an earlier thesis project [31]. The method used in that project was to use hidden iframes. When a new asynchronous request is made the location property of one of the hidden iframes is set to where the requested resource is. You can then set a callback handler as the eventhandler for the load event that is triggered when the iframe has finished loading. This approach has been used in this project; both when requesting information from the server and when invoking functions in the background widgets. There are several bugs and other problems with this web browser that causes the development of a web application for this browser to be difficult and tedious. These problems include lack of support for the navigation key and several standard HTML attributes that are not supported, for example it is impossible to turn off the scrollbars by using the scrolling attribute. It was possible to find a work around for some of these problems, while others simply have to be tolerated (or avoided).

Another problem with the browser is the cache, the browser often thinks that it has something in the cache when it does not. The solution to this problem is to use the Pragma, Cache-Control, Expires, and Date headers in the HTTP reply from the server to tell the browser not to cache the content of the reply.

4.1.2 Geo-location information

The Sony Ericsson HGE-100 GPS enabler (an external GPS receiver) was used in this project to get information about the handset's current location. The HGE-100 connects via a cable to the K800i and provides a large amount of information (some of which is useful and some less useful, for the purpose for this thesis). The information provided includes: longitude, latitude, altitude, longitude hemisphere, latitude hemisphere, speed, etc. The geographic coordinates are given in the World Geodetic System 1984, which is the reference system being used by the Global Positioning System, represented in degrees, minutes, and seconds.

Data provided by the GPS receiver is encoded in National Marine Electronics Association [91] messages and must be parsed accordingly. Sony Ericsson provides a Java ME class that handle communication and parsing of the NMEA sentences as well as a callback interface. This class is used in this project to make use of the HGE-100 [92].

4.2 Architecture

The application is started from the phonebook by selecting a contact's URL, which has automatically been added by the application. The URL specifies a resource in the MIDlet web server and might look as follows:

http://127.0.0.1:8080/local/126

The last number in the URL identifies the requested user. This is the same (local) user ID that is used by the personal information manager (PIM) to uniquely identify a user in the phonebook. By accessing the application this way it can be considered that the application is integrated with the phonebook, which was one of the initial goals with the application.

When selecting a URL the device's browser is automatically started. Because of this the application consists of two parts. The first part is a MIDlet that acts as a web server. It enables access to the local device's functionality, executes background widgets, and also executes parts of the foreground widgets. The second part is the graphical user interface which is rendered in the device's browser. Communication between the two parts is made by asynchronous HTTP requests (See Figure 3).

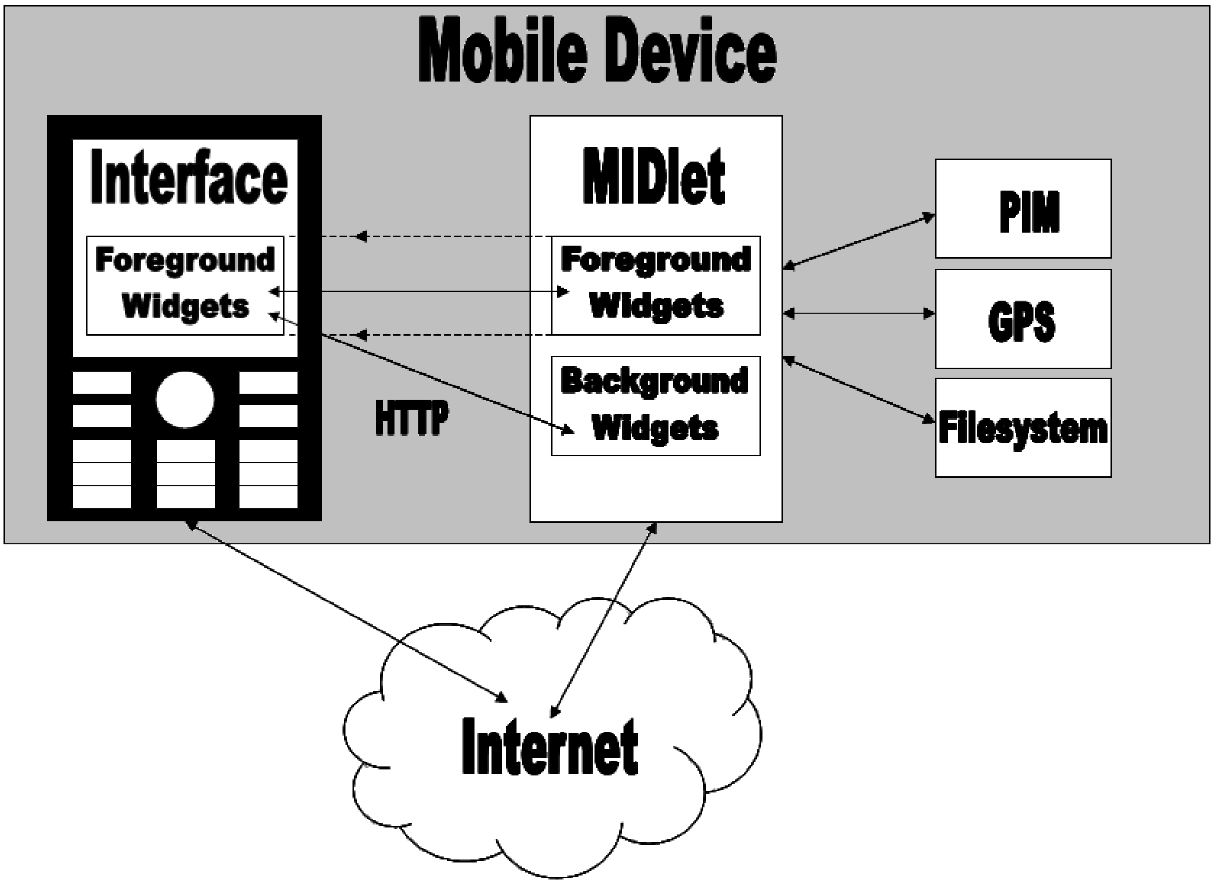


Figure 3: Architecture of the prototype implementation

4.3 MIDlet

The MIDlet consists of a web server that manipulates a set of resources located in the device's file system. As a security precaution the server only accepts incoming connections originating from the local host. The MIDlet server executes background widgets, parts of the foreground widgets, and enables communication between these two. Besides this the server also enables access to local device functionalities, such as PIM and GPS, from the widgets.

When the MIDlet is started it reads a configuration file, called "config.json". There is only one configuration file per application, this file is located on the device's file system. The information in the configuration file states which background widgets should be started (see section 4.5.2). Other information in the configuration file includes: last documented position of the user, what foreground widgets exist, and application specific information about the different widgets such as username and password, that might be needed to log in to different web services (These usernames and passwords are, unfortunately, saved in clear text. Encryption of these are desired but implementing that is considered being outside the scope of this thesis.). The background widgets are written in JavaScript and executed by the JavaScript engine which is integrated with the MIDlet server. The JavaScript engine is a port of the Rhino JavaScript engine (see section 2.2.6) to Java ME and is called Nelson. The porting has been done by other researchers at Ericsson Research.

At startup the MIDlet server starts listening for incoming connections on TCP port 8080. As a security precaution the MIDlet only accepts requests from the localhost. Upon accepting an incoming request a new thread is started that processes the request. This new thread checks if the path part of the URI starts with the word local. If this is the case, then the thread continues parsing the path and matches it to acceptable requests. Request URIs of the following forms are accepted and processed:

1. http://127.0.0.1:8080/local/util/**function name/arguments**
2. http://127.0.0.1:8080/local/**user ID/**json/**widget name**
3. http://127.0.0.1:8080/local/image/**image name**
4. http://127.0.0.1:8080/local/js/**JavaScript filename**
5. http://127.0.0.1:8080/local/**user ID/Valid foreground widget name**
6. http://127.0.0.1:8080/local/**user ID**

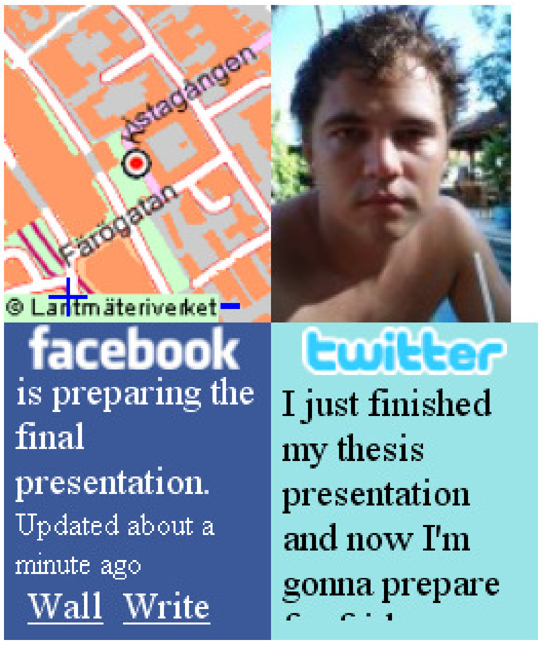
The first form enables background widgets to dynamically register URIs to functions or resources they want accessible from the user interface. The second form enables foreground widgets executed in the browser to access information about the contacts cached in the MIDlet. The third enables the browser to request locally stored images which when loaded are used by the user interface. The fourth accesses JavaScript files that are part of the foreground widget (in order to execute them in the browser). The fifth URI form retrieves and starts foreground widgets. The last URI form is the format of the URIs stored in the phonebook; such a URI is used to initially access information about a specific contact.

4.4 Graphical User Interface

The graphical user interface was developed using common web technologies, such as XHTML, CSS, and JavaScript. This implementation enables the user to change the existing user interface and create new interfaces. The interface also allows the look of each contact to be customized with help of CSS. Thus each contact can have a unique look and interacting with this contact can be highly personalized.

The device's browser is used to render the user interface. This requires less functionality from the MIDlet, as compared to having to produce an equally good rendering of HTML in the MIDlet itself. Furthermore, this also makes the installed application smaller and easier to develop. Additionally, by including the URI in the phonebook entry, the browser automatically starts when a URL is selected in the phone book, thus starting our application. This way of starting the application is very appealing as it means that the graphical user interface of our application is only executed if the user actively uses it and the user does not need to explicitly start the application.

The application is activated by selecting the URL of a contact in the phonebook. This starts the browser and information about the contact is shown on the screen. What information is shown depends on which foreground widgets that have been chosen for the requested contact. The start page for a contact could look like Figure 4.



is preparing the final presentation.
Updated about a minute ago
Wall Write

I just finished my thesis presentation and now I'm gonna prepare

Figure 4: The graphical user interface might look like this after selecting a user's URI in the phonebook.

4.5 Widgets

There are many different social networking services and the popularity of them is changing constantly. By implementing functionality for each of them in widgets, the application is more adaptable to the desires of the user. If a user does not use Facebook for example, then the Facebook widget does not need to be downloaded. If this application had been implemented completely in Java ME, then support for every social networking service would have been needed in the MIDlet, because of the lack of dynamically loading classes in Java ME.

4.5.1 Host objects

Access to Java classes using JavaScript from within the JavaScript engine is provided by a number of host objects (see section 2.2.6). The reason for having host objects is to enable JavaScript scripts to access Java classes. Using host objects the developer can specify which Java classes that the JavaScript code may access. The specific host objects available in the prototype enable, among other things, access to local device functionality such as getting GPS information, using HTTP connections, and access to persistent storage and phonebook information.

Access to the GPS device is enabled by an implementation of the Geolocator class proposed in the working draft from the Location Aware working group [93]. Three host objects are provided to enable this, the Geolocation, the Geolocator, and the Navigator. The Geolocation is an object that symbolizes a geographic location. The geographical location given by the GPS receiver are World Geodetic System 1984 coordinates represented in degrees, minutes, and seconds. To facilitate using this information in calculations they are converted to decimal degree format, instead of degree, minute, and seconds. A Geolocator object is acquired from the

Navigator object. The Geolocator object handles an asynchronous request for the current location. The user can also register a callback function with the request to the Geolocator object – so that the callback code is executed when the geographic location is determined. A request to the Geolocator for the current location eventually returns a Geolocation object, which represents the location in decimal degree format representation. The navigator object handles requests for new Geolocators and decoding the location data coming from the GPS device. Note that the Navigator object is agnostic to the source of the location data, thus this location data could come from a variety of sources.

The HTTP connection is provided by an implementation of the XMLHttpRequest API [94]. The implemented host object enables both synchronous and asynchronous communication over HTTP with remote web servers. It can also handle callback functions that are executed once a response is received. As described earlier in section 2.2.5, the XMLHttpRequest object is one of the central parts of the Ajax programming technique.

Persistent storage is provided by a host object called Storage. It gets its name and some features from the Storage interface from the HTML 5 [95] specifications but there are several differences. The constructor takes a filename as argument. If a file with this filename is found, then the content of the file is read and made the initial content of the object. Otherwise the Storage object will be empty. The Storage object has three functions: put, get, and save. The get function takes a key as an argument and returns a JSON formatted object with information regarding that key, or an empty object if no information was found. It is also possible to put information into the Storage object with the put method. The save function writes all information in the Storage object on the file system, it takes a filename as argument.

An almost complete implementation of the HTML Document Object Model (DOM) level 2 [96] is also available through host objects, developed at Ericsson Research. The DOM implementation enables widgets to access nodes in XML or HTML documents in a convenient and standardized way. DOM is a fundamental part of the Ajax programming technique and it enables scripts to dynamically access and update the content, structure, and style of documents. Instead of changing XML or HTML documents using string operations, DOM provides an object oriented way of modifying a document. The HTML DOM implementation also ensures that the resulting document is correctly written and that all closing tags are present.

Access to the PIM is also enabled by a host object. The pimhandler host object provides methods for accessing and changing contact information in the phonebook. A method that changes the URL of all the contacts was added to simplify configuring new contacts and reducing the amount of user input that is required. Note that if a contact already has a URL specified in the phonebook this URL will be replaced.

In order to simplify executing JavaScript functions on regular intervals or after a specific period of time a window host object has been implemented. This window object offers the *setTimeout*, *setInterval*, *clearTimeout*, and *clearInterval* methods found in the window object specifications from the W3C [97].

Another host object is the Cache object which provides a key to value mapping session storage mechanism. This cache is not widget specific and therefore can be used to share information between widgets. The Util host object enables a widget to dynamically register the URL of a function. The result is that this function is now callable from the browser. It is also possible to pass arguments to such a function, by including them in the URL (see URI form 1 in section 4.5). In addition to these host objects, there are several more, for example a BasicAuth object which encodes a string to the form required by HTTP basic authentication. A complete list of all the host objects can be found in Appendix A. Figure 5 try to illustrate how widgets executing on the MIDlet server can access local device functionality using host objects.

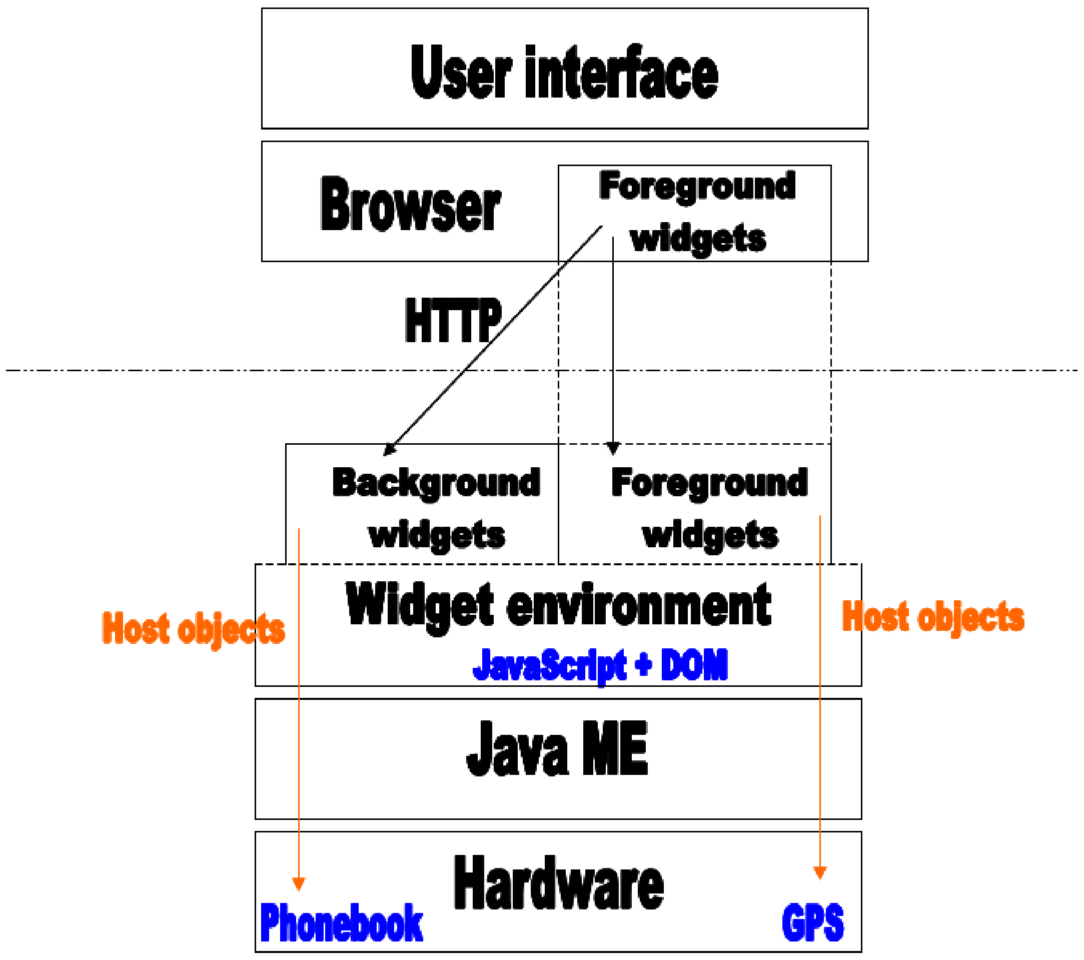


Figure 5: Hierarchical architecture diagram.

4.5.2 Background widgets

Background widgets are written in JavaScript and executed in the MIDlet when the application is started. A background widget does not have a user interface and only executes in the server. Hence, the background widget consists of only one resource, the JavaScript file. The names of the background widgets that should be executed are stored in the configuration file. The name of the JavaScript file is always the name of the widget with the word "widget" appended and the file extension ".js". For example, if there is a background widget called "GPS" the name would be "GPSwidget.js".

The background widget will only be executed once, on each start up. However, by using the Window host object a widget can register an expression to be evaluated either at specified intervals or after a specified number of milliseconds. By using this host object a background widget is able to register an update function, in this implementation this is used to maintain and keep an up to date view of the user's contacts.

As shown in the masters thesis "*Mobile Web Browser Extensions*" by Tomas Joelsson [39] and also supported by tests conducted in this project (see Table 2 on page 42) the JavaScript engine in the Netfront browser is very slow. Because of this it is desirable to place as much functionality as possible in the MIDlet, therefore in this implementation most of the functionality and processing has been implemented in background widgets.

4.5.3 Foreground widgets

A widget is called a foreground widget if it has a user interface. In order to have a user interface it **must** be executed in the foreground. The user interface is rendered in the browser and most of the logic of the foreground widgets also executes in the browser. However, some of the logic is executed in the server before the widget is transferred to the browser.

A foreground widget has at least one resource. The filename of that resource must be the name of the widget appended with the word "template" and the file extension ".html". So if there is a foreground widget called "map" the filename must be "maptemplate.html". Other acceptable resources are images, JavaScript files, and CSS files. These resources must be located on the device's file system.

Figure 6 illustrates the process of initiating a foreground widget and transferring it to the browser. First of all, when a foreground widget is requested, the server reads the document and populates the DOM. Then the server checks if the document has any scripts designated to run on the server. Such scripts are identified by a *runat* attribute. These scripts are executed and removed in the same way as scripts are in the Jaxer server (see section 3.12). Scripts that have a *runat* attribute with the value *proxy* usually add contact specific information available on the server to the document so it can be shown in the user interface. However, it is possible for foreground widgets to register a function with the Util host object and make it callable from the browser. After all the scripts that should be run on the server are finished the document is serialized from DOM to HTML again, and then sent to the browser for rendering.

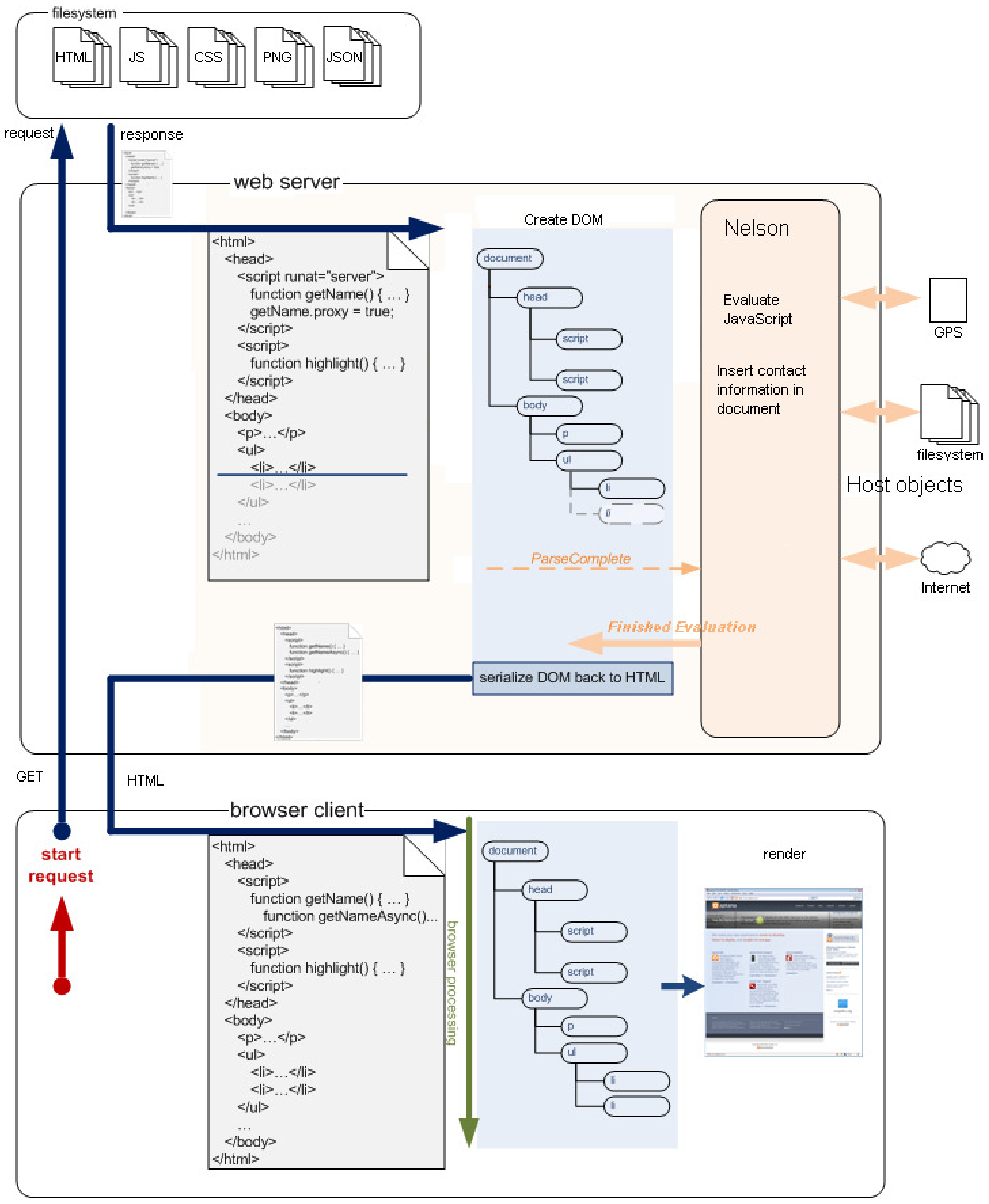


Figure 6: Foreground widget initiation process, adapted from [89]

4.6 Implemented example widgets

Four background widgets (a Facebook widget, a Twitter widget, a GPS widget, and a phonebook widget) and six foreground widgets (one each for Facebook, Twitter, map, image, configure, and index) have been implemented. The background widgets perform most of the logic, while the foreground widgets primarily act as user interfaces. Each of these widgets will be described in some detail below.

4.6.1 Implemented background widgets

4.6.1.1 The Facebook background widget

The Facebook background widget consists of one resource ("facebookwidget.js"). To be able to access the information from Facebook the user has to be authenticated. If the application wants to use the RESTful API it needs to be authenticated, this requires an API key which must be obtained from Facebook. Furthermore, Facebook restricts which IP addresses are allowed to originate requests, based on the application key. Because of this restriction, it was decided that the RESTful API should not be used. Instead the Facebook widget authenticates itself the same way that someone using a browser would. Thus it uses a HTTP POST request to the Facebook login server which, following a successful authentication, returns an HTTP cookie that should be used with each subsequent request.

After authentication the application requests the Facebook profile page of each contact specified in the configuration file. The reason for storing this information in the configuration file is that there is no free space in the phonebook and storing information outside the mobile device was not desired. The profile pages are XHTML coded, which simplifies creating DOM of them. By using the DOM implementation the widget can easily collect all the relevant information from the profile pages. The information that is saved is the username, latest status update, what other people have written on the wall[4], the URL to the contact's profile picture, and a link for posting to the wall.

After retrieving the above information, the Facebook widget registers a function for posting to walls. By using the Util host object the widget dynamically registers a URL on the server. The function which is registered takes two arguments: the user ID of the contact to post to and the message to post. This information is then sent to Facebook and the request is authenticated based upon the cookie.

Additionally, to keep the information about the contacts current the Facebook background widget registers an update function with the window object. This update function logs into the Facebook server every 120 seconds and extracts all of the relevant information about all the contacts as specified in the configuration file. When the widget is started it downloads a list of all the user's friends. This list is later used by the Facebook foreground widget if a contact is requested that has not yet been configured.

[4] The wall is the guestbook found on each Facebook profile page.

4.6.1.2 The Twitter background widget

The Twitter background widget also consists of one resource ("twitterwidget.js"). This widget uses the RESTful interface that Twitter provides for application developers. By using HTTP GET requests information about Twitter user's can be fetched in either JSON format or in XML. There are roughly 30 different methods provided by Twitter's RESTful interface. The method used by the Twitter background widget is "friends_timeline". The JSON format has been chosen due to the ease of using such an object in JavaScript. Twitter requires a user to be authenticated when calling the friends_timeline method. This authentication is done using HTTP Basic Authentication (via the BasicAuth object described in section 4.5.1). The information gathered from Twitter includes: last post, profile picture URL, location, and time of last update.

Just like the Facebook widget, this widget also registers an update function with the window object. This function runs every 120 seconds, in order to maintain an up to date view of the contacts specified in the configuration file.

4.6.1.3 The GPS background widget

The GPS background widget makes use of the Navigator, Geolocator, and Geolocation host objects to find out the device's current location. By registering an update function with the window object, the current location can be updated at regular intervals. When a location has been found, the GPS background object saves the location in the configuration file and submits this location to Twitter. Twitter's RESTful interface provide a method called "update_location", that allows an authenticated user to update his or hers profile with their location. This method is used by the Twitter background widget to enable users to easily share their position with friends, while and at the same time controlling who is allowed to learn this location. If a new location is not found, for example because there is no GPS signal available, then neither the update of configuration file nor the Twitter location are made. The GPS widget consists of one resource ("gpswidget.js").

4.6.1.4 The phonebook background widget

The phonebook background widget checks if any new contacts has been added to the phonebook. A function is registered using the window host object to be executed every five minutes. This widget uses the pimhandler host object to access the phonebook and changes the URL of each contact in the phonebook, so that the URL specifies a resource on the MIDlet server.

4.6.2 Implemented foreground widgets

4.6.2.1 The index foreground widget

When a contact is first selected in the phonebook the index foreground widgets is started. This widget consists of an XHTML document ("index.html"). This document has four iframes that each points to another foreground widget. Which foreground widgets these iframes indicate is read from a file containing information about the contact. The file is located on the file system and not on the Internet in order to reduce delay.

When a contact is requested for the first time there is no file available containing information for that contact. In this case the user is presented with a list of available foreground widgets and is requested to choose which widgets he or she wants to use for this specific contact. This information is then sent back to a function running on the server and saved in a file for the contact.

4.6.2.2 The Facebook foreground widget

The Facebook foreground widget consists of the required resource ("facebooktemplate.html") as well as a Facebook logo, "HTTP.js", "view.js", and "controls.js". Together these resources provide several features that are executed in the browser.

The "facebooktemple.html" document has one script that is evaluated in the server. This script adds the latest status update to the page, and then the page is serialized and sent to the browser. Using the getJSON function of the HTTP object in "HTTP.js" all information cached by the Facebook background widget is requested and transferred to the browser. This enables the user to see what is written on this contact's wall.

Another feature is the ability to write on the contacts wall from the foreground widget. As writing on someone's wall requires authentication, this authentication is done based upon the use of cookies. In this case, the cookie is stored in the background widget. Writing on a wall is implemented by making an HTTP GET request to the URL assigned by the background widget to the wall post function. This function takes two arguments: user ID of the contact and the message to be written on the wall. These arguments are included in the URL and separated by slashes ("/"). A request for a wall post that writes the message "Hej" on the user 126's wall would look like this:

http://127.0.0.1:8080/local/util/wall_post/126/Hej

The "view.js" and "controls.js" resources enable the user to scroll in the foreground widget. This is quite tricky due to some limitations of the Netfront browser. One limitation is the lack of support for events being triggered when the navigation key is moved. Only when the navigation key is pressed is an event triggered. A solution to this problem was presented in a masters thesis by Per-Erik Svensson and this solution is also used in this project [31].

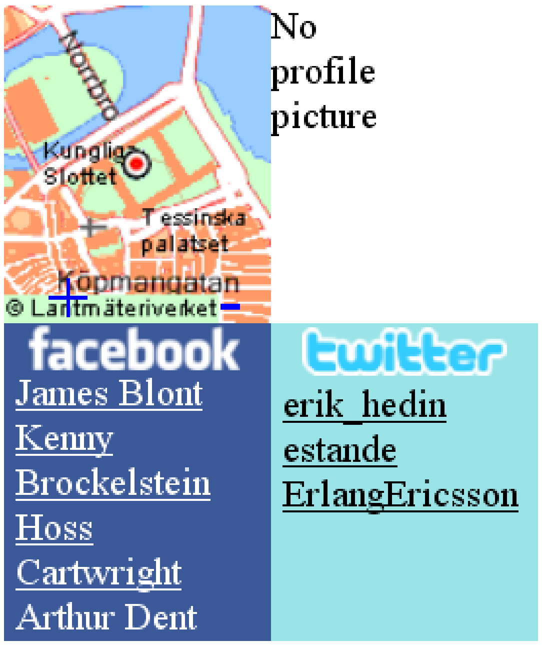
The first time a contact is requested from the Facebook foreground widget it does not know that contacts Facebook ID. In this case the user is presented with a list of all available contacts and is requested to match the contact in the phonebook to the contact on Facebook (see Figure 7). A function is registered with the Util object to listen for a response from the user. The response contains the Facebook ID of the selected user, this information is saved in the contacts configuration file.

4.6.2.3 The Twitter foreground widget

The resources that make up the Twitter foreground widget are the required "twittertemplate.html", a Twitter logo, "HTTP.js", "view.js", and "controls.js". Just like the Facebook foreground widget the Twitter foreground widget evaluates some JavaScript code in the browser that adds information to the page before it is sent to the browser for rendering. The information added is the latest message submitted by the requested contact.

A Twitter message can be up to 160 characters, which is too long to show in the designated HTML "div" element, therefore it would cause the scrollbars to appear. To prevent the scrollbars from appearing and making the user interface looking untidy, the style property "overflow" is set to "hidden". "View.js" and "controls.js" enables scrolling through the message content without using the browser's built in scrolling, thus keeping the scrollbars hidden. Scrolling is enabled by pressing the Twitter logo and disabled by pressing the navigation key left or right. Note that the "view.js" and "controls.js" files for the Facebook and Twitter foreground widgets are the same.

Just like with the Facebook foreground widget the user needs to manually enter the contact's username on Twitter. This matching is simplified by the foreground widget that presents a list of friends from Twitter. The user is asked to mach the contact to a friend's username from Twitter. This might look like Figure 7.



James Blont
Kenny
Brockelstein
Hoss
Cartwright
Arthur Dent

erik_hedin
estande
ErlangEricsson

Figure 7: Manual matching of Twitter and Facebook.

4.6.2.4 The map foreground widget

The map foreground widget shows the location of a contact on a map. This location is retrieved from the contact's Twitter profile. The map widget uses Eniro's online map service (http://wap.eniro.se/) which uses the decimal degree format to encode geographical locations. This is the same format as provided by the GPS background widget. By appending coordinates and screen size to a URL, the Eniro map service responds with a picture of a map centered at the requested location, with a dot at the specified location and with the requested screen size. Also included in the URL is an attribute called "zoomlevel". By changing the value of this attribute, it is possible to zoom in or out of the map. To enable zooming of the map in the widget two zoom buttons (marked as '+' and '-') are found at the bottom of the user interface. If either of these buttons is pressed, the value of the zoomlevel attribute is changed and the map zooms either in or out. Note that the URL is added to the page by a script *in the server* before the page is sent to the browser.

4.6.2.5 The image foreground widget

The image foreground widget displays the contact's profile picture as retrieved from Facebook or from Twitter, the later being used if there is none available from Facebook. The URL of the picture is added to the user interface by a script in the server. This picture might be of a different size than what is preferred by the user interface; this would trigger the scrollbars. Therefore, in order to avoid this, rescaling of the picture is done by an onload eventhandler function.

4.6.2.6 The config foreground widget

The first time the user tries to access a contact the config foreground widget is run. This widget asks the user for username and password to Facebook and Twitter. This information is then sent to the MIDlet server and saved in the configuration file.

5 Evaluation

The goal of the project was to expand the functionality of the phonebook in order to facilitate the user's communication with contacts which are a result of their participation in online social networks. It was decided at the start of the project that the application should be started from the phonebook by selecting the URL of a contact. Two solutions were considered for how this could be achieved: the first was to have the URL specify a resource on a web server running on the device. The second alternative was to use the content handler API [34] and register a content handler that would respond to the URL and start a MIDlet. The content handler approach was rejected for two main reasons. First there are few devices that support the content handler API and second the K800i, which was used for development and testing, does not support this approach. Therefore the first alternative was utilized.

A second requirement that was decided at the start of the project was that all of the application logic should be placed on the mobile device. It was specifically decided to avoid using a remote server for any part of the application. Such a solution could reduce the amount of data downloaded by the mobile device, resulting in less computation being required on the device. However, the need for servers by the application might also restrain their usability. The possibly sudden growth in popularity of an application might lead to exponential growth in the number of users. Such fast growth, sometimes called viral distribution, can be demanding on the infrastructure required for the application to function properly. This might result in the need for costly updates of infrastructure, such as more powerful servers and faster Internet access, or in the worst case, making the application unpopular with the users.

Additionally, there were two alternatives for how to implement the user interface. The first option was to use the internal web browser which can be started on demand. This was an option because the URL is used to start the application and this automatically starts the web browser. This enables the option of rendering the user interface in the browser. The other option was to create a user interface directly in Java ME. The first option was chosen because it enables the use of web technologies to create the user interface. This was preferred because of the recent growth of popularity of these web technologies. The second option would have required more code to be implemented in the MIDlet. This code would have made the application larger, thus making it more likely to get updates.

Since the user interface would be rendered in the browser it was possible to have most or all of the logic of the application running in the browser. Unfortunately, the browser's JavaScript engine is very slow (see section 5.3 and [39]). A slow JavaScript engine would increase the time it would take for the application to generate responses, thus decreasing quality of the user's experience. Because of this, it was decided that the majority of the code would be executed in the MIDlet server. Another reason for running social network specific code in the MIDlet is caching information is possible in a more controlled way than in the browser. Such cache management was highly desired since round trip delays are very high in mobile networks. Loading content on demand would make the application respond very slowly, which would also reduce the quality of the user's experience.

Early during development an option of using a port of the Rhino JavaScript engine, called Nelson, was considered. This JavaScript engine is written in Java ME, which enables it to be directly integrated with the server. Rhino is written in Java SE and can therefore not be run on a mobile device that only supports Java ME. By using this JavaScript engine much of the functionality of the application could be loaded at runtime. This makes the server both more flexible and allows it to be updated without being reinstalled. Another benefit is that this solution enables developers to use the same techniques for both the client side and server side functionality – thus reducing development time and complexity.

These approaches result in a server with functionalities very similar to Jaxer (see section 3.12), as well as the common functionalities of a widget engine. However, the Jaxer server is not written in Java ME, which makes it unsuitable for mobile devices. Furthermore, Jaxer is intended to be used as a remote web server. While these are fundamental difference, the idea of simplifying server side development by exploiting JavaScript and providing a DOM is very similar in both solutions.

Another initial goal of the project was to investigate if it is possible to automatically match the contacts in the device's phonebook with the contacts found on online social networks. This is a very interesting challenge and was initially given much attention. However, early in the project it was realized that accomplishing this would require a considerable amount of time. Because of this, trying to find a solution to the automatic matching goal was given a lower priority, resulting in that development of the novel widget platform was given much more attention. The current application facilitates the manual matching of contacts that the user needs to do (see section 4.6.2.2 and 4.6.2.3).

5.1 Developing for the browser

A great limitation of web applications is that they are executed within another application, the browser. This introduces several significant issues for both developers and end-users when deploying and using applications within the browser. The web browser was originally designed to deliver and display HTML-based documents. A fundamental conflict between document-focused and application-focused functionality creates a number of problems when deploying applications via the browser [98]. Web applications have their own user interface, which often conflicts with the user interface of the browser. These conflicts can lead to user confusion, as well as more severe problems, such as application failure. One of the most common problems concerns the browser's "back" button. If the "back" button is pressed when using a web application, it could force the browser to load a previously visited page, thus causing the web application to loose state and data. A non-standard solution to this problem was introduced in Internet Explorer 4.0 and is now implemented in other browsers. The solution is to generate an event that fires just *before* a page is unloaded. This event is called *beforeunload*. Unfortunately, this event is not available in the Netfront 3.3 browser. Therefore the user **must not** hit the "back" key when the browser is being used as an application user interface.

Many mobile browsers, including the Netfront 3.3 browser, automatically render web pages in a different way than intended by the creator. They do this so the layout of the web page can be optimized for the small screens of mobile devices. This function in Netfront is called *smart-fit* and causes all CSS absolute positioning to be removed, resulting in all elements being placed below each other in the order they appear in the markup. While the smart-fit function can be turned off in Netfront, this is only possible by having the user turn it off manually.

Despite these drawbacks, using the browser as a platform offers many positive benefits. For example, it is easier to create a user interface using web technologies than using Java ME. Another benefit is that it is easier to update and configure the application, while an update of java code would inevitably lead to reinstallation of the Java application. In comparison, an update of web technology code could be done automatically, without the user ever noticing. Configuring the application and choosing which features should be available is also easier with the use of web technologies. Additionally, the widget specific logic does not need to be placed in Java code, making the application more dynamic.

5.2 Accessing online social networks

Facebook and Twitter provide different ways of exchanging information. Twitter offers a very convenient RESTful interface for developers to use. While Facebook also provide a RESTful interface, however, as mentioned earlier, this interface is meant for applications running on computers with static IP addresses (hence access control is based upon matching against specific IP addresses). Currently, the only form of communication that Facebook offer Swedish users are via their home page.

Through the RESTful interface Twitter offer roughly 30 different methods that can be used by an application. This enables application developers to request on the information relevant for their specific application. In many cases this also limits the number of requests an application needs to do to make to a single request. By using HTTP basic authentication Twitter further reduces the number of requests needed, since no special authentication procedure is required. It is also possible to specify what format the response should be encoded in. Both the XML and JSON formats are available, which simplifies the process of extracting relevant information.

Twitter has a limitation on how many requests each user is allowed to make to the RESTful interface. This limit is usually 70 requests per minute, which might constraint the application somewhat. It is possible to register an application with Twitter, this makes it possible for a user to make an unlimited number of requests, assuming that the registered application name is included in the request. However, the widgets developed in this project make at most one request every minute. Thus the limit is not a practical limit for this thesis, unless the users are making request to Twitter's RESTful interface using some other application as well.

Since the RESTful interface was not available, information from Facebook is gathered from profile pages, encoded in XHTML. Authentication is provided by a HTTP cookie. This cookie is obtained by making a HTTP or HTTPS POST request containing the username and password to the Facebook login server. Each profile page can then be downloaded and with the help of the DOM implementation the relevant information from the pages can be extracted fairly easily.

When comparing the two ways of accessing online social networks, it is obvious that Twitter's RESTful approach provides a much more convenient interface for application developers. There are several aspects that simplify the development process and, at the same time, also make the application more appealing for mobile users. The first aspect is that the RESTful interface greatly reduces the number of HTTP requests. As mentioned earlier, usually only one request is needed to get all the information required from Twitter. While when requesting information from Facebook a new request is needed *for each contact*, this in addition to the request needed for authentication. By reducing the number of HTTP requests the amount of processing needed to gather all the information is greatly reduced.

Another aspect is that the amount of data transmitted is reduced. Since the data received from Facebook is intended to be displayed in a browser it also includes information that would be used by the browser. However, this information is not relevant for the application and can therefore be thrown away (unfortunately in the case of Facebook – if this filtering is done at the mobile device, then you have lots of processing of the received data only to throw most of it away!). Reducing the amount of data that is received is possibly the greatest benefit of RESTful interfaces. Less data reduces the processing needed by the receiver and might also lead to lower costs for the receiver, since still today many users pay for the data they download.

To further investigate this amount of savings (in numbers of bytes which do not need to be transmitted), data from the Facebook background widget was collected. Table 1 shows the data collected. This table contains the size of 18 XHTML encoded profile pages received from Facebook, it also shows the number of bytes extracted from the profile pages that is actually displayed by the Facebook background widget. From this data a 95% confidence interval of the portion of data saved is calculated. The result shows that 15% ± 2% of the downloaded bytes are saved by the Facebook background widget. This result is interesting since it shows that most of the data downloaded is usually used to display the information in the browser, and is therefore less useful to this application. Please note that this application will update the information about the configured contacts every 2 minutes. If five contacts are configured for Facebook and a profile pages is on average 5 KB large this will lead to approximately 18 MB of data being downloaded, just from Facebook, each day. If the user does not have a flat rate subscription this will probably lead to great costs. A similar test was conducted to see how much of the downloaded information from Twitter that is actually displayed to the user. The test downloaded the 20 most recent messages sent to Twitter. The amount of data downloaded was 12338 bytes and the amount of data extracted by the background widget was 4274 bytes. This shows that about 35% of the downloaded information is saved by the widget. It also shows that information about 20 messages and the users who posted the messages is on average smaller than three Facebook profile pages.

The most obvious advantage of the RESTful interface encountered in this project is the stability of the API. Facebook has changed the layout of their web pages several times during the project. The need to modify the code used to extract the relevant data following each of these changes has slowed down development considerably, since much time was required to investigating what changes were made and how the new data can be parsed efficiently from the new layout. Being able to choose which format the response should be encoded in (in the case of RESTful) further reduces both development and runtime. Additionally, any changes in the layout of the page are now irrelevant to the application – hence the application will continue to work for all of the users who are using it – unlike the case for Facebook.

5.3 Sunspider JavaScript benchmark

One of the main reasons for moving as much logic as possible to the MIDlet was to decrease the execution time of the widgets. This idea is a result of a previous thesis project carried out by Tomas Joelsson [39]. To accurately measure the difference in execution speed between the JavaScript engine in the Netfront browser and the JavaScript engine integrated in the MIDlet the Sunspider JavaScript benchmark [99] was used.

The Sunspider benchmark tests the core JavaScript language only, not the DOM or other browser APIs. It is designed to compare different versions of the same browser or different browsers to each other. The test tries to focus on real world problems and the kinds of actual problems developers solve with JavaScript today, and the problems they may want to tackle in the future as execution of code written in JavaScript gets faster. The developers of the tests also have tried to make them balanced between different areas of the language and different types of code. It is not all maths, all string processing, or all timing of simple loops. In addition to having tests in many categories, the individual tests are balanced to take similar amounts of time on currently shipping versions of popular browsers [99].

In the original version of the benchmark the individual tests are loaded from a remote server and run in the browser. By doing some changes to the server and how the tests are loaded it was also possible to do a benchmark of the JavaScript engine in the MIDlet.

However, these tests were not designed to run on mobile devices and they expect the underlying platform to have significantly more resources. This introduced some problems and one of the tests ("string-tagcloud") could not be evaluated in either the mobile device's browser or in the MIDlet. Another test ("access-nsieve") caused the browser to freeze, but had no problem executing in the MIDlet, therefore it has also been removed from the comparison between the two.

In the changed version the code running in the browser asks the server to run a test. When the test is completed the execution time is returned to the browser, which saves the result and asks the server to run a new test. This introduced a problem, one of the tests ("string-unpack-code") took quite a long time to execute. Because of this the TCP connection timed out and no result could be reported by the server for that test. Another problem was that two of the tests ("string-validate-input" and "regexp-dna") required too much memory and caused the MIDlet to throw an "out of memory exception", thus no result could be reported by the server for these tests.

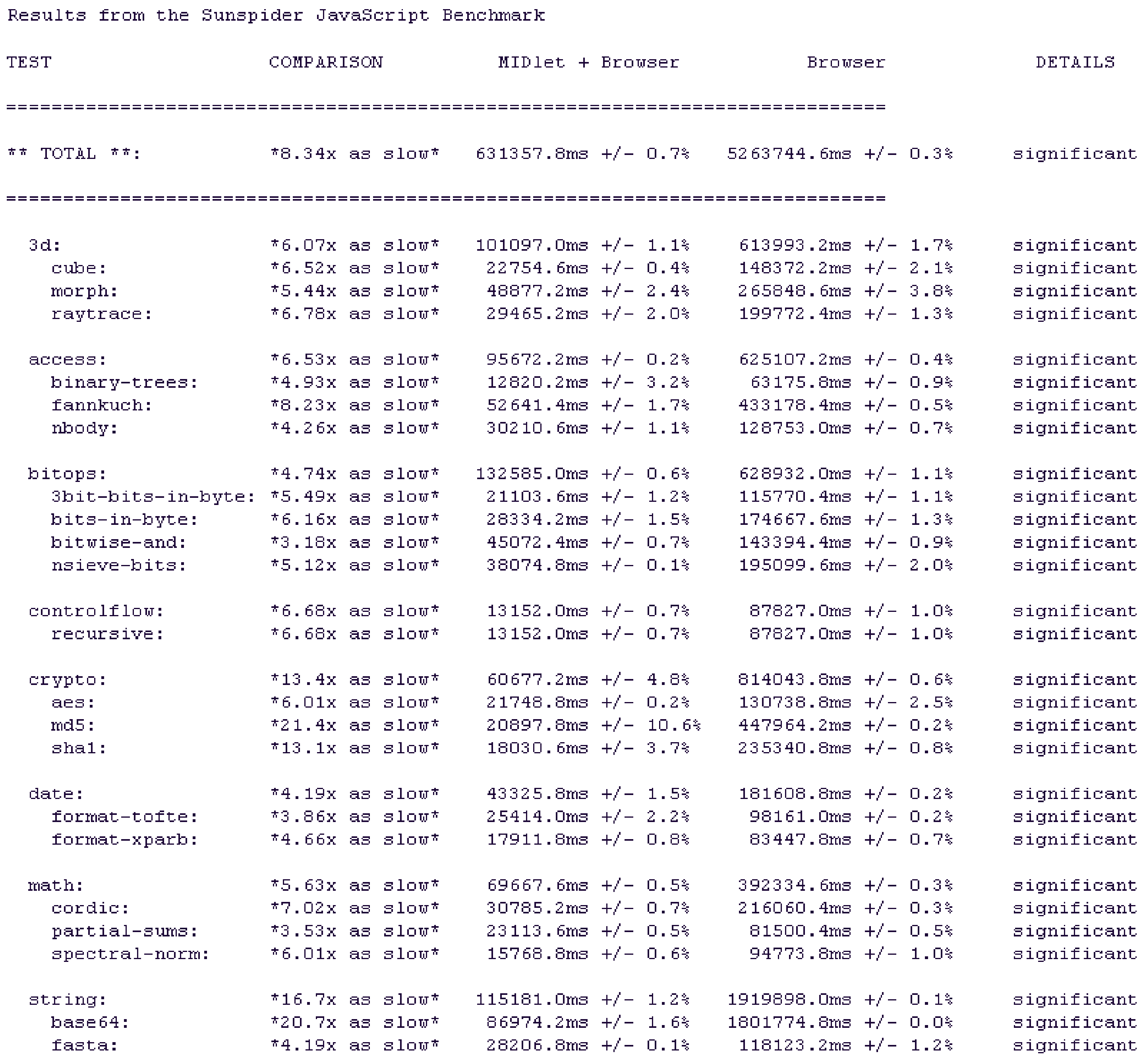
After removing these five tests there are still 21 tests left from the original sunspider benchmark. The 21 tests were run five times each to produce a 95% confidence interval of the execution times. The results from these tests show that the Netfront browser is approximately 8,34 times slower than the MIDlet server at evaluating JavaScript (see Table 1). This nearly order of magnitude difference in performance supports the decision to place most of the logic in background widgets. The results also support having the MIDlet server perform all of the heavy processing, especially processing of data *received* from the different social networks.

Further tests were done to see how much the browser influences the execution speed of JavaScript code running in the server. To perform these tests a new MIDlet was created that has a user interface in order to facilitate starting the tests; as well as displaying of results. The results of these tests are shown in table 2. These results indicate that the browser increases the execution time of the MIDlet by 9%. This increase in execution time is due to the fact that the browser requires both processor time and memory. The main reason, however, is probably due to the available amount of memory being reduced, which increases the frequency of garbage collection. This can be verified by recording the number of times the KVM garbage collector executed. The result shows that performing as much of the processing as possible in the background widgets was a good choice.

Size of Facebook profile page versus the amount of data actually displayed to the user

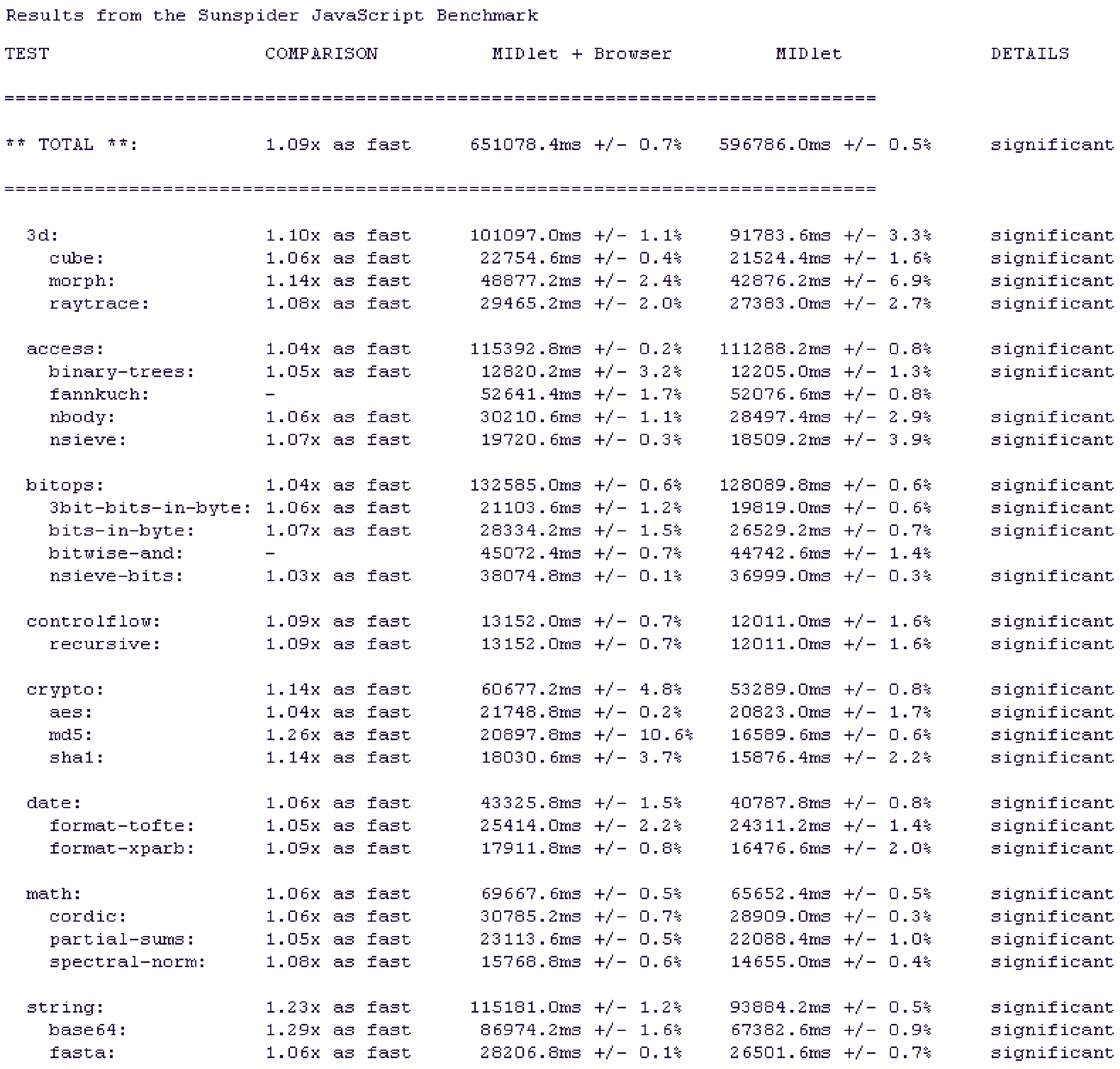
Profile page number	*Page size(bytes)*	*Displayed (bytes)*
1	4726	483
2	4497	684
3	4440	460
4	5173	862
5	5379	513
6	5234	1110
7	6357	1279
8	5002	610
9	5292	1048
10	5391	962
11	5797	1045
12	5276	1124
13	4800	854
14	5919	729
15	5534	894
16	5165	793
17	5030	514
18	5481	630
Total	94493	14594

Table 1: Data showing the size of Facebook profile pages and the amount of data actually displayed by the Facebook foreground widget.



```
Results from the Sunspider JavaScript Benchmark

TEST                   COMPARISON         MIDlet + Browser          Browser              DETAILS

=============================================================================

** TOTAL **:           *8.34x as slow*   631357.8ms +/- 0.7%   5263744.6ms +/- 0.3%     significant

=============================================================================

  3d:                  *6.07x as slow*   101097.0ms +/- 1.1%    613993.2ms +/- 1.7%     significant
    cube:              *6.52x as slow*    22754.6ms +/- 0.4%    148372.2ms +/- 2.1%     significant
    morph:             *5.44x as slow*    48877.2ms +/- 2.4%    265848.6ms +/- 3.8%     significant
    raytrace:          *6.78x as slow*    29465.2ms +/- 2.0%    199772.4ms +/- 1.3%     significant

  access:              *6.53x as slow*    95672.2ms +/- 0.2%    625107.2ms +/- 0.4%     significant
    binary-trees:      *4.93x as slow*    12820.2ms +/- 3.2%     63175.8ms +/- 0.9%     significant
    fannkuch:          *8.23x as slow*    52641.4ms +/- 1.7%    433178.4ms +/- 0.5%     significant
    nbody:             *4.26x as slow*    30210.6ms +/- 1.1%    128753.0ms +/- 0.7%     significant

  bitops:              *4.74x as slow*   132585.0ms +/- 0.6%    628932.0ms +/- 1.1%     significant
    3bit-bits-in-byte: *5.49x as slow*    21103.6ms +/- 1.2%    115770.4ms +/- 1.1%     significant
    bits-in-byte:      *6.16x as slow*    28334.2ms +/- 1.5%    174667.6ms +/- 1.3%     significant
    bitwise-and:       *3.18x as slow*    45072.4ms +/- 0.7%    143394.4ms +/- 0.9%     significant
    nsieve-bits:       *5.12x as slow*    38074.8ms +/- 0.1%    195099.6ms +/- 2.0%     significant

  controlflow:         *6.68x as slow*    13152.0ms +/- 0.7%     87827.0ms +/- 1.0%     significant
    recursive:         *6.68x as slow*    13152.0ms +/- 0.7%     87827.0ms +/- 1.0%     significant

  crypto:              *13.4x as slow*    60677.2ms +/- 4.8%    814043.8ms +/- 0.6%     significant
    aes:               *6.01x as slow*    21748.8ms +/- 0.2%    130738.8ms +/- 2.5%     significant
    md5:               *21.4x as slow*    20897.8ms +/- 10.6%   447964.2ms +/- 0.2%     significant
    sha1:              *13.1x as slow*    18030.6ms +/- 3.7%    235340.8ms +/- 0.8%     significant

  date:                *4.19x as slow*    43325.8ms +/- 1.5%    181608.8ms +/- 0.2%     significant
    format-tofte:      *3.86x as slow*    25414.0ms +/- 2.2%     98161.0ms +/- 0.2%     significant
    format-xparb:      *4.66x as slow*    17911.8ms +/- 0.8%     83447.8ms +/- 0.7%     significant

  math:                *5.63x as slow*    69667.6ms +/- 0.5%    392334.6ms +/- 0.3%     significant
    cordic:            *7.02x as slow*    30785.2ms +/- 0.7%    216060.4ms +/- 0.3%     significant
    partial-sums:      *3.53x as slow*    23113.6ms +/- 0.5%     81500.4ms +/- 0.5%     significant
    spectral-norm:     *6.01x as slow*    15768.8ms +/- 0.6%     94773.8ms +/- 1.0%     significant

  string:              *16.7x as slow*   115181.0ms +/- 1.2%   1919898.0ms +/- 0.1%     significant
    base64:            *20.7x as slow*    86974.2ms +/- 1.6%   1801774.8ms +/- 0.0%     significant
    fasta:             *4.19x as slow*    28206.8ms +/- 0.1%    118123.2ms +/- 1.2%     significant
```

Table 2: Comparison between results from when the tests are executed in the browser and results from when the tests are executed in the MIDlet server when the browser is running.



```
Results from the Sunspider JavaScript Benchmark

TEST                   COMPARISON          MIDlet + Browser          MIDlet             DETAILS

=============================================================================

** TOTAL **:           1.09x as fast     651078.4ms +/- 0.7%   596786.0ms +/- 0.5%     significant

=============================================================================

  3d:                  1.10x as fast     101097.0ms +/- 1.1%    91783.6ms +/- 3.3%     significant
    cube:              1.06x as fast      22754.6ms +/- 0.4%    21524.4ms +/- 1.6%     significant
    morph:             1.14x as fast      48877.2ms +/- 2.4%    42876.2ms +/- 6.9%     significant
    raytrace:          1.08x as fast      29465.2ms +/- 2.0%    27383.0ms +/- 2.7%     significant

  access:              1.04x as fast     115392.8ms +/- 0.2%   111288.2ms +/- 0.8%     significant
    binary-trees:      1.05x as fast      12820.2ms +/- 3.2%    12205.0ms +/- 1.3%     significant
    fannkuch:          -                  52641.4ms +/- 1.7%    52076.6ms +/- 0.8%
    nbody:             1.06x as fast      30210.6ms +/- 1.1%    28497.4ms +/- 2.9%     significant
    nsieve:            1.07x as fast      19720.6ms +/- 0.3%    18509.2ms +/- 3.9%     significant

  bitops:              1.04x as fast     132585.0ms +/- 0.6%   128089.8ms +/- 0.6%     significant
    3bit-bits-in-byte: 1.06x as fast      21103.6ms +/- 1.2%    19819.0ms +/- 0.6%     significant
    bits-in-byte:      1.07x as fast      28334.2ms +/- 1.5%    26529.2ms +/- 0.7%     significant
    bitwise-and:       -                  45072.4ms +/- 0.7%    44742.6ms +/- 1.4%
    nsieve-bits:       1.03x as fast      38074.8ms +/- 0.1%    36999.0ms +/- 0.3%     significant

  controlflow:         1.09x as fast      13152.0ms +/- 0.7%    12011.0ms +/- 1.6%     significant
    recursive:         1.09x as fast      13152.0ms +/- 0.7%    12011.0ms +/- 1.6%     significant

  crypto:              1.14x as fast      60677.2ms +/- 4.8%    53289.0ms +/- 0.8%     significant
    aes:               1.04x as fast      21748.8ms +/- 0.2%    20823.0ms +/- 1.7%     significant
    md5:               1.26x as fast      20897.8ms +/- 10.6%   16589.6ms +/- 0.6%     significant
    sha1:              1.14x as fast      18030.6ms +/- 3.7%    15876.4ms +/- 2.2%     significant

  date:                1.06x as fast      43325.8ms +/- 1.5%    40787.8ms +/- 0.8%     significant
    format-tofte:      1.05x as fast      25414.0ms +/- 2.2%    24311.2ms +/- 1.4%     significant
    format-xparb:      1.09x as fast      17911.8ms +/- 0.8%    16476.6ms +/- 2.0%     significant

  math:                1.06x as fast      69667.6ms +/- 0.5%    65652.4ms +/- 0.5%     significant
    cordic:            1.06x as fast      30785.2ms +/- 0.7%    28909.0ms +/- 0.3%     significant
    partial-sums:      1.05x as fast      23113.6ms +/- 0.5%    22088.4ms +/- 1.0%     significant
    spectral-norm:     1.08x as fast      15768.8ms +/- 0.6%    14655.0ms +/- 0.4%     significant

  string:              1.23x as fast     115181.0ms +/- 1.2%    93884.2ms +/- 0.5%     significant
    base64:            1.29x as fast      86974.2ms +/- 1.6%    67382.6ms +/- 0.9%     significant
    fasta:             1.06x as fast      28206.8ms +/- 0.1%    26501.6ms +/- 0.7%     significant
```

Table 3: Comparison between results from when the tests are executed in the MIDlet server with and without the browser running at the same time.

6 Conclusions and future work

6.1 Conclusions

The application developed in this project facilitates the users communication on social networking sites and enables the user to configure the application according to his or hers desires. Furthermore, it allows third part developer to create new widgets for the novel widget framework developed. New functionality can easily be created using Ajax technologies for both MIDlet and browser logic.

Widgets greatly simplify the way users access online resources, specially from mobile devices. This is a reason why widgets are so well suited for mobiles and why experts think they are the method mobile users will access the web in the future. A result of this thesis is that widgets developed specifically for mobile devices simplify access to social networking sites (and probably many other online services).

The fact that widgets can be developed using common web technologies simplifies the development process and further adds to their popularity. However, there are currently few widget engines available for mobile devices, which slows down the growth of application development. The lack of support for access to device functionality, such as a GPS receiver and the phonebook, is another problem. Most of the available mobile widget engines do not support access to any local resources, while the remaining few provide access to only a very limited amount of functionality. Widgets are also highly portable, as long as the different widget engines support standardized APIs, which also increases the simplicity of developing code for these different engines. As the benchmark results show, the JavaScript engine in the browser is very slow compared to the engine in the MIDlet. This shows that creating a dedicated JavaScript engine for executing web applications such as widgets is desirable for performance reasons. The use of mobile Internet will probably grow as more mobile widget engines become available. But for them to have a greater impact, greater access to local device functionality needs to be supported by the widget engines and standardization of the APIs that enable this is needed.

The design decisions made in this project have lead to a much more adaptable application, compared to other mobile social networking applications, such as the ZYB phonebook. While the ZYB phonebook is a similar application (since they try to mimic the phonebook and it connects to several different social networking services) and the application developed in this project requires installation of a larger application, but once the application is installed few, if any, subsequent updates will be needed. Access to contact information seems to be more intuitive when using the prototype application developed in this project, as all the user needs to do is to browse to the contact using the built in phonebook. Thus there is no need for the user to remember to start an application.

A positive trend is that many of the major online social networking services are developing RESTful interfaces. One of the conclusions drawn from this project is that RESTful interfaces greatly simplify the development process of web applications. Furthermore, the RESTful interfaces are changed or updated comparably seldom, while web sites usually change the layout more often. This leads to that applications developed for RESTful interfaces are more stable and require fewer updates. By providing RESTful interfaces these organizations make it easier for third party developers to create applications that connected to them. This will probably increase the popularity of these services and increase the number of users they serve.

6.2 Future work

More functionality can be added to the existing widgets as future work. Some of this desirable functionality is: starting an update, enabling the poke feature that Facebook provides, and replying to peoples' Twitter's could quite easily be implemented, but lie outside the scope of this thesis project.

Another result of this project is that solving the problem of automatic synchronization of online contact lists and the device's phonebook is extremely hard without a common means of identification. All users might not even want this to be possible and many people use different aliases on different networks to stay anonymous. Therefore, for a working solution to be found the problem needs to be given much more attention. The application developed in this thesis requires that the user manually enters contacts into their phonebook that he or she wants to communicate with using the application. While the application facilitates configuration of the contacts and matching to existing contacts on Facebook the initial goal of doing this automatically was down prioritized in favor of the novel widget framework developed in this project. A future project could give more attention to the challenges surrounding automatic synchronization. There are several projects working on facilitating creation of social networks, these could possibly be used to enable automatic synchronization.

While automatic synchronization might not be possible, it is possible to make the process of matching contacts on different lists more automatic, possibly by comparing names and ranking contacts with similar names higher. However, there are significant differences in communication patterns between online communities and mobile phones. The mobile phone is still primarily used to communicate with *close* contacts and business contacts; while communities are used to also communicate with more distant contacts (often in areas which the user wishes to purposely keep separate from their business contacts – even if they might involve the same people). Because of this the actual demand for automatic synchronization and a matching function might not be so high. Another corollary to this is a reason why matching of contacts might be difficult, a contact might just be present in one list (or one service).

Another feature that might be useful is a function that monitors how often contacts are viewed. This information could be used to update unpopular contacts less often, which would help reduce data communication costs and also prolong the battery life, since less processing is needed. Functionality to start updates on demand or triggered by specific events (such as moving to a location which is not one of the user's typical locations) could also be a feature that might be desired by users.

More widgets can be developed, especially one that contact OpenSocial sites, whenever a RESTful interface becomes available. As more widgets become available a means of distributing them will be needed, this could be developed in future work. Automatic updates of the installed widgets have not been implemented and a means for doing this also needs to be developed.

It would simplify development of new widgets if more standards were developed regarding both how the widget engine works, how widgets are packaged, and APIs for accessing local device functionality. The W3C WebApps working group is developing recommendations for these, but has not yet released many recommendations and no standards. There are currently only recommendations for widget packaging and digital signatures. These could and should be implemented in the widget engine developed in this project as future work.

Future work could also be to develop an application that uses the Content Handler API to start the MIDlet. A MIDlet that is started this way does not need to be running all the time, which would prolong battery life.

A result of this thesis project and of previous projects is that running a HTTP server on a mobile device works very well. This result has lead to the idea of further extending the functionality of the MIDlet server. Functionality such as instant messaging could be added to the mobile device by implementing the XMPP protocol in the MIDlet server.

References

[1] Informa Telecoms & Media. Mobile Social Networking growth accelerates: Revenues could reach US$52 billion by 2012. Press release for a report by Christine Perey of PEREY Research & Consulting. Informa Plc. February 2008. http://www.informa.com.au/itmgcontent/icoms/s/pressreleases/20017504078.html;jsessionid=676E2FB707FF5889A34F61435E85F4E1
see also: http://www.perey.com/mobilesocialnetworking.html

[2] Vodafone. *Social networking - Mobile Internet - Vodafone*. Vodafone Group, Last accessed on 2008-08-14. http://online.vodafone.co.uk/dispatch/Portal/appmanager/vodafone/wrp?_nfpb=true&_pageLabel=template10&pageID=MI_0012

[3] Jemima Kiss. *Orange to open up mobile social net service*. guardian.co.uk, Guardian News and Media Limited, Wednesday July 09 2008 17:21 BST, Last accessed on 2008-08-14. http://www.guardian.co.uk/media/2008/jul/09/web20.digitalmedia1.

[4] Alexa Internet. Global Top 500. A dynamically generate web page containing rankings data from Alexa Internet, Inc. Last accessed on 2008-05-05. http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none.

[5] Nokia. *Nokia Europe - Extras – Widgets*. Nokia. Last modified on July 3, 2008. Last accessed on 2008-08-19. http://europe.nokia.com/A41013687

[6] China Mobile Limited. *China Mobile Limited, SOFTBANK and Vodafone in agreement to establish a Joint Innovation Lab to develop mobile internet services*. China Mobile Limited. 2008-04-24. http://www.chinamobileltd.com/doc/pr/2008/20080421.htm

[7] Brad Fitzpatrick. *Thoughts on the Social Graph*. Brad Fitzpatrick. 2007-08-17. http://bradfitz.com/social-graph-problem/.

[8] The DataPortability Project. *DataPortability.org - Share and remix data using open standards*. Last accessed on 2008-04-28. http://www.dataportability.org/.

[9] Open Mobile Alliance. *Working Groups and Committees*. Open Mobile Alliance. Last accessed on 2008-04-08. http://www.openmobilealliance.org/Technical/WorkingGroupsCommitees.aspx.

[10] David Pollington. *Mobile Widgets Simplifying the mobile Internet*. Vodafone Group Research and Development. May 2008. Available at http://www.betavine.net/bvcms/documents/widgets/MoMo%20May07%20bullets.pdf

[11] Craig Cumberland. *Mobile Widgets are more valuable than Desktop Widgets!*. Nokia. June 2008. Available at http://www.widgetwebexpo.com/wp-content/uploads/2008/07/widgets-bring-more-value-craig-cumberland.pdf

[12] The OpenID Foundation. *OpenID » What is OpenID?*. The OpenID Foundation. Last accessed on 2008-05-05. http://openid.net/what/.

[13] Global Multimedia Protocols Group. *XFN - XHTML Friends Network*. GMPG. Last accessed on 2008-05-05. http://gmpg.org/xfn/.

[14] FOAF. *The Friend of a Friend (FOAF) project*. Foaf. Last accessed on 2008-08-14. http://www.foaf-project.org/

[15] Ivan Herman (Semantic Web Activity Lead). *W3C Semantic Web Activity*. W3C. 2008-07-20. http://www.w3.org/2001/sw/

[16] Wikipedia. *SyncML – Wikipedia the free encyclopedia*. Wikipedia. Last modified 2008-08-27. Last accessed on 2008-08-28. http://en.wikipedia.org/wiki/Syncml.

[17] Tele2. *Web Phonebook - Tele2*. Tele2. Last accessed in 2008-08-14. http://www.tele2.se/mobiltjanster-web-phonebook.html

[18] Storegate AB. *Telia Säker lagring*. Storegate AB. Last modified 2008-08-22. Last accessed on 2008-08-27. http://www.storegate.se/partners/telia/

[19] Funambol, Inc. *Funambol community forge*. Funambol, Inc. Last accessed on 2008-08-14. http://www.funambol.com/opensource

[20] OpenSync. *OpenSync - A synchronization framework*. OpenSync. Last accessed on 2008-08-14. http://www.opensync.org/

[21] Open Handset Alliance. *Android*. Open Handset Alliance. Last modified 2007-11-12. Last accessed on 2008-08-27. http://www.openhandsetalliance.com/android_overview.html

[22] QUALCOMM Incorporated. *Qualcomm BREW | Home*. QUALCOMM Incorporated. Last accessed on 2008-08-14. http://brew.qualcomm.com/brew/en/

[23] Adobe Systems Incorporated. *Adobe – Flash Lite*. Adobe Systems Incorporated. Last modified 2008-08-07. Last accessed on 2008-08-14. http://www.adobe.com/products/flashlite/

[24] Sun Microsystems, Inc. *Java ME Technology*. Sun Microsystems, Inc. Last accessed on 2008-04-15. http://java.sun.com/javame/technology/index.jsp.

[25] Microsoft Corporation. *.NET Compact Framework*. Microsoft Corporation. Last accessed on 2008-08-14. http://msdn.microsoft.com/en-us/netframework/aa497273.aspx

[26] Microsoft Corporation. *Phones, Software, Help, How-Tos | Smartphone and PDA | Windows Mobile*. Microsoft Corporation. Last accessed on 2008-08-14. http://www.microsoft.com/windowsmobile/en-us/default.mspx

[27] ACCESS CO., LTD. *ACCESS*. ACCESS CO., LTD. Last accessed on 2008-08-14. http://www.access-company.com/products/platforms/garnet/index.html

[28] Symbian. *Symbian OS: the open mobile operating system*. Symbian. Last modified 2008-07-24. Last accessed on 2008-08-14. http://www.symbian.com/

[29] Wikipedia. *Mobile development - Wikipedia, the free encyclopedia*. Wikipedia. Last modified 2008-08-20. Last accessed on 2008-08-28. http://en.wikipedia.org/wiki/Mobile_development.

[30] Richard Marejka. *Learning Path: MIDlet Life Cycle*. Sun Microsystems, Inc. February 2005. http://developers.sun.com/mobility/learn/midp/lifecycle/

[31] Per-Erik Svensson. *Mobile TV as a Web Application*. Masters thesis, Department of Communication Systems, School of Information and Communications Technology, Royal Institute of Technology. August 2007. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/070816-Per-Erik-Svensson-webmobtv_report_final-with-cover.pdf.

[32] JSR 248 Expert Group. *Mobile Service Architecture Specification*. Java Community Process. 2006-09-27. Available at http://jcp.org/aboutJava/communityprocess/final/jsr248/index.html.

[33] Sun Microsystems, Inc. *Java ME Technology - Mobile Service Architecture Overview*. Sun Microsystems, Inc. Last accessed on 2008-04-16. http://java.sun.com/javame/technology/msa/

[34] JSR 211 Expert Group. *Java™ 2 Platform, Micro Edition Content Handler API, Specification, (CHAPI)*. Java Community Process. 2005-06-03. Available at http://jcp.org/en/jsr/detail?id=211.

[35] Opera Software ASA. *Opera Mini - Free mobile Web browser for your phone*. Opera Software ASA. Last accessed on 2008-08-14. http://www.operamini.com/

[36] Nokia. *Nokia Mini Map Browser*. Nokia. Last modified 2008-05-21. Last accessed on 2008-08-14. http://www.nokia.com/browser

[37] Access CO., LTD. *NetFront™ Browser*. Last accessed on 2008-08-14. http://www.access-company.com/products/mobile_solutions/netfrontmobile/browser/index.html

[38] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media. August 2006. ISBN: 0-596-10199-6

[39] Tomas Joelsson. *Mobile Web Browser Extensions*. Masters thesis, Department of Communication Systems, School of Information and Communications Technology, Royal

Institute of Technology. April 2008. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080412-Tomas_Joelsson-with-cover.pdf

[40] Wikipedia. *Cross-site scripting - Wikipedia, the free encyclopedia*. Wikipedia. Last accessed on 2008-04-22. http://en.wikipedia.org/wiki/Cross-site_scripting.

[41] Mozilla Foundation. *Rhino - JavaScript for Java*. Mozilla Foundation. Last modified 2007-08-30. Last accessed on 2008-08-14. http://www.mozilla.org/rhino/

[42] Mozilla Foundation. *JavaScript Runtime*. Last accessed on 2008-08-14. http://www.mozilla.org/rhino/runtime.html

[43] Mozilla Foundation. *Scopes and Contexts*. Mozilla Foundation. Last modified 2006-11-17. Last accessed on 2008-08-14. http://www.mozilla.org/rhino/scopes.html

[44] MySpace.com. *MySpace*. MySpace.com. Last accessed on 2008-08-14. http://www.myspace.com/

[45] Facebook. *Välkommen till Facebook! | Facebook*. Facebook. Last accessed on 2008-08-14. http://sv.facebook.com/

[46] hi5 Networks. *hi5 | Your Friends. Your World*. hi5 Networks. Last accessed on 2008-08-14. http://hi5.com/

[47] Orkut. *Orkut – om*. Orkut . Last accessed on 2008-08-14. http://www.orkut.com/About.aspx

[48] OpenSocial. *Home(OpenSocial)*. OpenSocial. Last accessed on 2008-08-27. http://www.opensocial.org/.

[49] Hi5.com. *Hi5 API (beta)*. Hi5.com. Last modified 2008-08-23. Last accessed on 2008-08-27. http://api.hi5.com/.

[50] MySpace.com. *MySpace Developer Platform*. MySpace.com. Last accessed on 2008-04-08. http://developer.myspace.com/Community/

[51] Facebook. *Facebook Developers*. Facebook. Last accessed on 2008-04-09. http://developers.facebook.com/.

[52] Wikipedia. *Representational State Transfer - Wikipedia, the free encyclopedia*. Wikipedia. Last accessed on 2008-04-09. http://en.wikipedia.org/wiki/Representational_State_Transfer.

[53] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation, Department of Information and Computer Science, University of California, Irvine. 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[54] RestWiki. *Short Summary of Rest*. RestWiki. Last modified 2006-03-01. Last accessed on 2008-04-09. http://rest.blueoxen.net/cgi-bin/wiki.pl?ShortSummaryOfRest.

[55] Marcos Caceres (editor). *Widgets 1.0: The Widget Landscape (Q1 2008)*. W3C. 2008-04-14. http://www.w3.org/TR/2008/WD-widgets-land-20080414/

[56] Marcos Caceres (editor). *Widgets 1.0: Requirements*. W3C. 2008-06-25. http://www.w3.org/TR/2008/WD-widgets-reqs-20080625/

[57] Vringo. *Vringo (Beta) | Facebook*. Vringo. Last accessed on 2008-04-16. http://www.vringo.com/facebook/.

[58] OutSync. *OutSync | Facebook*. OutSync. Last accessed on 2008-08-27. http://www.facebook.com/apps/application.php?id=6433496082.

[59] Fonebook. *Fonebook | Facebook*. Fonebook. Last accessed on 2008-04-16. http://www.facebook.com/apps/application.php?id=2442338999.

[60] Facebook. *Facebook Principles*. Facebook. Last accessed 2008-05-05. http://www.facebook.com/policy.php.

[61] XMPP Standards Foundation. *XMPP Standards Foundation*. XSF. Last accessed on 2008-08-14. http://www.xmpp.org/

[62] Dave Raggett (editor), Arnaud Le Hors (editor), and Ian Jacobs (editor). *HTML 4.01 Specification*. W3C. 1999-12-24. http://www.w3.org/TR/1999/REC-html401-19991224

[63] Mark Nottingham (editor) and Robert Sayre (editor). The *Atom Syndication Format*. The Internet Society. December 2005. http://www.ietf.org/rfc/rfc4287

[64] iLabs. *iLabs mobile toolbox*. CollabNet, Inc. Last accessed on 2008-04-14. https://ilabsmobiletoolbox.dev.java.net/.

[65] Vodafone. *Welcome to Vodafone – Vodafone*. Vodafone. Last accessed on 2008-08-14. http://www.vodafone.com/hub_page.html

[66] ZYB. *ZYB | Home*. Last accessed on 2008-04-16. ZYB. https://zyb.com/phonebook.

[67] Skyhook Wireless Inc. *Loki - You Can Get There From Here*. Skyhook Wireless Inc. Last accessed on 2008-08-14. http://loki.com/

[68] GeoRSS. *GeoRSS | GeoRSS :: Geographically Encoded Objects for RSS feeds*. GeoRSS. Last accessed on 2008-08-14. http://georss.org/.

[69] Microformats.org. *geo – Microformats*. Microformats.org. Last modified 2008-06-16. Last accessed on 2008-08-27. http://microformats.org/wiki/geo.

[70] David Troy. *Twittermap faq*. David Troy. Last accessed on 2008-04-21. http://Twittervision.com/maps/faq.html.

[71] Gypsii. *Gypsii Webtop*. GeoSolutions, B.V. Last accessed on 2008-05-05. http://www.gypsii.com/.

[72] Johan Wikman and Ferenc Dosa. *MobileWeb Server*. Nokia. Last accessed on 2008-08-27. http://opensource.nokia.com/projects/mobile-web-server/

[73] Nokia. *Forum Nokia – Widgets*. Last accessed on 2008-08-14. http://www.forum.nokia.com/main/resources/technologies/browsing/widgets.html

[74] WebKit Open Source Project. *The WebKit Open Source Project*. WebKit Open Source Project. Last accessed on 2008-08-27. http://webkit.org/

[75] Apple Inc. *Apple - Downloads – Dashboard*. Apple Inc. Last accessed on 2008-08-27. http://www.apple.com/downloads/dashboard/

[76] Yahoo! Inc. *Yahoo! Widgets: useful, fun, beautiful little apps for Mac and Windows*. Yahoo! Inc. Last accessed on 2008-08-27. http://widgets.yahoo.com/

[77] Nokia. *Porting Apple Dashboard Widgets to S60*. Nokia. 2007-10-09. Available at http://sw.nokia.com/id/db501a88-12a1-475b-b880-8d5f6c70f359/Porting_Apple_Dashboard_Widgets_to_S60_v1_0_en.zip

[78] Nokia. *Web Developer's Library 1.0*. Nokia. Last accessed on 2008-08-14. http://www.forum.nokia.com/document/Web_Developers_Library

[79] Nokia. *S60 extending the lead in Internet innovation and experiences*. Nokia. 2008-04-23. http://www.nokia.com/A4136001?newsid=1212541

[80] Nokia. *Category:WidSets - Forum Nokia Wiki*. Nokia. Last modified 2008-08-25. Last accessed on 2008-08-27. http://wiki.forum.nokia.com/index.php/Category:WidSets

[81] Nokia. *Introduction to developing WidSets widgets - Forum Nokia Wiki*. Nokia. Last modified 2008-06-17. Last accessed on 2008-08-14. http://wiki.forum.nokia.com/index.php/Introduction_to_developing_WidSets _widgets

[82] Nokia. *Widget files - Forum Nokia Wiki*. Nokia. Last modified 2008-06-21. Last accessed on 2008-08-14. http://wiki.forum.nokia.com/index.php/Widget_files

[83] Nokia. *Stylesheet - Forum Nokia Wiki*. Nokia. Last modified 2008-08-20. Last accessed on 2008-08-27. http://wiki.forum.nokia.com/index.php/Stylesheet

[84] joemoby. *Get WidX*. Joemoby. Last accessed on 2008-08-14. http://www.joemoby.com

[85] Alex Linde. *Joemoby –WidX Overview*. Joemoby. 2007-09-10. http://www.joemoby.com/images/JoeMobyOverview0p1.pdf

[87] Aptana, Inc. *Aptana Jaxer | Aptana*. Aptana, Inc. Last accessed on 2008-08-14. http://www.aptana.com/jaxer

[88] Aptana, Inc. *Jaxer Documentation | Aptana*. Aptana, Inc. 2008-02-08. Last accessed on 2008-08-14. http://www.aptana.com/jaxer/book

[89] Aptana, Inc. *Architecture | Aptana*. Aptana, Inc. 2008-02-20. Last accessed on 2008-08-14. http://www.aptana.com/node/275

[90] Sony Ericsson Mobile Communications AB. *Sony Ericsson Developer World -K800i*. Sony Ericsson Mobile Communications AB. Last accessed 2008-08-18. https://developer.sonyericsson.com/device/loadDevice.do?id=074bb4a4-997d-45af-b309-88de20148887

[91] National Marine Electronics Association. *Publications and Standards from the National Marine Electronics Association (NMEA) / NMEA 0183*. Last accessed 2008-08-18. http://www.nmea.org/pub/0183/index.html.

[92] Sony Ericsson Mobile Communications AB. *Creating MIDlets for HGE-100 GPS enabler accessory*. Sony Ericsson Mobile Communications AB. December 2007. https://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_midlets_hge100_gpsaccessory.jsp

[93] LocationAware.org. *LocationAware.org*. LocationAware.org. Last accessed on 2008-08-18. http://www.locationaware.org/

[94] Anne van Kesteren (editor). *The XMLHttpRequest Object*. W3C. 2008-04-15. http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/

[95] Ian Hickson (editor). *HTML 5*. Web Hypertext Application Technology Working Group. 2008-08-27. http://www.whatwg.org/specs/web-apps/current-work/#storage0

[96] Johnny Stenback (editor), Philippe Le Hégaret (editor), and Arnaud Le Hors (editor). *Document Object Model (DOM) Level 2 HTML Specification*. W3C. 2003-01-09. http://www.w3.org/TR/DOM-Level-2-HTML/

[97] Ian Davis(editor) and Maciej Stachowiak(editor). *Window Object 1.0*. W3C Working Draft. W3C. 2006-04-07. Last accessed on 2008-08-26. http://www.w3.org/TR/2006/WD-Window-20060407/

[98] Mike Chambers, Daniel Dura, Dragos Georgita, and Kevin Hoyt. *Adobe AIR for JavaScript Developers Pocket Guide*. O'Reilly Media. April 2008. ISBN: 978-0-596-51837-0

[99] webkit.org. *SunSpider JavaScript Benchmark*. webkit.org. Last accessed on 2008-08-19. http://webkit.org/perf/sunspider-0.9/sunspider.html

Appendix A – Documentation of the Host objects

This appendix contains information about all the host objects created for the widget platform.

The BasicAuth host object

The BasicAuth host object provides a method that encodes a user name and password in the format (base 64) that HTTP Basic Authorization requires. The BasicAuth host object is not constructible.

Methods
encode(name, password)

returns a string containing the name and password encoded in the format HTTP Basic Authorization requires.

The Cache host object

The Cache host object provides a key - value based storage. The Cache host object is not constructible. Internally it uses Hashtables to store the information. The same Hashtables are used to store information from all widgets, hence the Cache object can be used to store information from one widget that can be read by another.

Methods
saveCache(widget, key, value)

The first argument is used to identify the widget that put the information in the Hashtable. The second argument is the key and the third is the value to be stored. Returns void.

getCache (widget, key)

Returns the value associated with the keys widget and key.

The console host Object

The console host object enables developers to print information to an available screen from the widget. If the widget engine is running on a mobile device and the console object has a reference to a form it prints the information to that form. If that form is currently in the foreground the information is printed on the screen. Similarly if the widget engine is executed on a mobile device emulator the information is printed on the screen of the emulator. If no form is available the information is printed to standard out.

Methods
log(msg)

prints the message to an available form or to standard out.

The DOM host objects

The DOM host objects is a set of host objects that together provide an almost complete implementation of the DOM level 2 HTML specifications (for further information see http://www.w3.org/TR/DOM-Level-2-HTML/).

The Geolocation host object

The Geolocation host object provides a representation of a geographic location. It is an implementation of the Geolocation interface specification from the LocationAware working draft (see http://www.locationaware.org/wiki/index.php?title=Working_Draft for further information).

The Geolocator host object

The Geolocator host object provides methods for requesting a geographic location, represented as a Geolocation object. It is an implementation of the Geolocator interface specification from the LocationAware working draft (see http://www.locationaware.org/wiki/-index.php?title=Working_Draft for further information).

The Navigator host object

The Navigator host object provides a method for requesting a Geolocator object. The Navigator object virtualizes the source of the location data, in this case the HGE-100 GPS receiver. The Navigator host object is not constructible.

Methods

getGeolocator()

returns a new Geolocator.

The PIM host object

The PIM host object enables widgets to read and write to the local device's phonebook. The PIM host object is not constructible.

Methods

changeURLs(URL)

Changes the URLs of all the contacts in the local devices phonebook. Returns void.

getField(uid, field)

Returns the value found in the requested field in the contact with the specified uid.

setField(uid, field, value)

Sets the requested field to the specified value of the contact that is identified with the uid. Returns void.

The Storage host object

The Storage host object provides a key – value based persistent storage. The Storage host object is constructible.

Constructors

new Storage(file name)

Reads the content of the file with the file name given as argument. This content is used as initial information in the Storage object. If no file is found with the file name the Storage object is initially empty.

Properties

length

The number items in the Storage object

Methods

get(key)

Returns the value associated with the key.

put(key1, key2, value)

Puts the value in the Storage object and associates it with first key1 and then key2.

putObject(key, value)

Puts the value in the Storage object and associates it with the key.

save(file name)

Serializes and writes the content of the Storage object to a file with the file name specified as argument. The file is saved on the local devices file system.

The Util host object

The Util Host object enables widgets to dynamically register a function with a URL on the server. When the URL is requested the function is called. It is also possible for a caller to pass arguments, as parts of the URL. The Util host object is constructible, but two function cannot be registered with the same URL. Subsequent registrations will override previous registrations.

Functions registered with the Util object cannot return any values to the caller. The function is callable from the browser with the following URL:

http://127.0.0.1:8080/local/util/**name/argument1/argument2**

Constructors

new Util()

Returns a new Util object.

Methods

addFunction(name, function)

Registers the function passed as argument with the server under the name passed as argument. Returns void.

The window host object

The window host object provide the *setTimeout*, *setInterval*, *clearTimeout*, and *clearInterval* methods found in the window object specifications from the W3C (see http://www.w3.org/TR/2006/WD-Window-20060407/ for further information). Besides these methods a proprietary method for doing explicit garbage collections has been added.

Methods

gc()

The XMLHttpRequest host object

The XMLHttpRequest host object provides a fully functional implementation of the XMLHttpRequest Object specification from the W3C (see http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/ for further information).

COS/CCS 2008-18