Mobile Web Browser Extensions

Utilizing local device functionality in mobile web applications

TOMAS JOELSSON

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-07

Mobile Web Browser Extensions

Utilizing local device functionality in mobile web applications

TOMAS JOELSSON

Master's Thesis at KTH
Academic supervisor and examiner: Gerald Q. Maguire Jr.
Company supervisors: Stefan Ålund and Per-Erik Brodin, Ericsson Research

12 April 2008

Abstract

Mobile web browsers of today have many of the same capabilities as their desktop counterparts. However, among the capabilities they lack is a way for web applications to interact with local devices. While today's mobile phones commonly include GPS receivers and digital cameras, these local devices are currently not accessible from within the browser. The only means of utilizing these devices is by using standalone applications, but such applications lack the versatility of web browsers. If a mobile browser could utilize these local devices, then a mobile application could run within the browser, thus avoiding the need for specialized client software.

This thesis suggests an approach for adding such capabilities to mobile browsers. In the proposed method, scripted access to local device functionality is facilitated by a local Java application. This application acts as a proxy server and allows the browser to call methods exposed by the local Java APIs. Both the benefits and some security concerns of this approach are examined. The benefits are further highlighted through two example web applications which utilize local devices.

Sammanfattning

Utökad funktionalitet för mobila webbläsare

I dagens mobila webbläsare återfinns det mesta av funktionaliteten från webbläsare för datorer. Det som dock fortfarande saknas är möjligheten för webbapplikationer att komma åt lokala telefonfunktioner. Dagens mobiltelefoner är ofta utrustade med GPS-mottagare och digitalkameror, men dessa kan för närvarande ej nås från webbläsaren. Det enda sättet att utnyttja dessa inbyggda funktioner är genom separata applikationer, men sådana applikationer är inte lika mångsidiga som webbläsare. Om en mobil webbläsare kunde utnyttja de inbyggda funktionerna, så skulle en mobil applikation kunna köras i webbläsaren istället för att ha separat klientprogramvara.

Det här examensarbetet föreslår ett sätt att ge denna möjlighet till mobila webbläsare. I den föreslagna metoden används en lokal Java-applikation för att ge tillgång till inbyggda funktioner via skript. Denna applikation fungerar som en proxy-server och låter webbläsaren anropa metoder exponerade av lokala Java-API. Både fördelar och några säkerhetsproblem med den här lösningen undersöks. Fördelarna visas ytterligare genom två exempel på webbapplikationer som utnyttjar inbyggda telefonfunktioner.

Table of Contents

List of Figures

List of Abbreviations

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
BOM	Browser Object Model
CDC	Connected Device Configuration
cHTML	Compact HyperText Transfer Protocol
CLDC	Connected Limited Device Configuration
CSS	Cascading Style Sheets
DHTML	Dynamic HyperText Transfer Protocol
DNS	Domain Name System
DOM	Document Object Model
FTP	File Transfer Protocol
GCF	Generic Connection Framework
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
iHTML	Inline HyperText Transfer Protocol
IMEI	International Mobile Equipment Identity
JAD	Java Application Descriptor
JAR	Java Archive
Java EE	Java Platform, Enterprise Edition
Java ME	Java Platform, Micro Edition
Java SE	Java Platform, Standard Edition
JCP	Java Community Process
JNLP	Java Network Launching Protocol
JP-7	Java Platform 7
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JSR	Java Specification Request
JTWI	Java Technology for the Wireless Industry
JVM	Java Virtual Machine

MIDI	Musical Instrument Digital Interface
MIDP	Mobile Information Device Profile
MMAPI	Mobile Media Application Programming Interface
MSA	Mobile Service Architecture
MWS	Mobile Web Server
NMEA	National Marine Electronics Association
NPAPI	Netscape Plugin Application Programming Interface
OMA	Open Mobile Alliance
OTA	Over-The-Air
PDA	Personal Digital Assistant
SIP	Session Initiation Protocol
SMS	Short Message Service
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TURN	Traversal Using Relay NAT
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WGS 84	World Geodetic System 1984
WLAN	Wireless Local Area Network
WMA	Wireless Messaging API
WML	Wireless Markup Language
WSH	Windows Scripting Host
WTAI	Wireless Telephony Applications Interface
WTP	Wireless Transaction Protocol
XHTML	eXtensible HyperText Markup Language
XHTML MP	eXtensible HyperText Markup Language Mobile Profile
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Problem statement

This master's thesis concerns the continuing development of mobile devices, specifically mobile phones. As more and more people use their mobile phones to access the Web, the demand for new services and better user interfaces is growing. Many of the new mobile phones sold today have a web browser pre-installed. As mobile phones become more advanced, with more processing power and larger screens, these browsers are approaching the capabilities and performance of browsers running on PCs. This has enabled developers to build rich user interfaces, resulting in applications that are easy to utilize. There are, however, still limitations as to what these browser based applications can do. This thesis will examine these limitations and show how to overcome many of them.

Unfortunately, much of the new functionality built into mobile phones is not accessible from within the phone's current web environment. To utilize a Global Positioning System (GPS) receiver, a Bluetooth interface, or a built-in digital camera programmers must write programs that work outside the web browser, but these programs need to interface to the web browser - in order to easily integrate other applications which the user is used to using. These other applications are generally written in Java, C, C++, C#, VB.NET, or Python and execute on the phone itself. By writing Java code for mobile phones the application can access many of the local device Application Programming Interfaces (APIs). If this functionality can be integrated with the web browser it will facilitate access from many web applications.

1.2 Proposed solution

To bridge the gap between mobile applications and the mobile web, we have to creatively combine them. We can take advantage of the fact that modern phones can run background processes. Such a background process can act as a local web server - thus providing local device functionality which can be accessed via the built-in

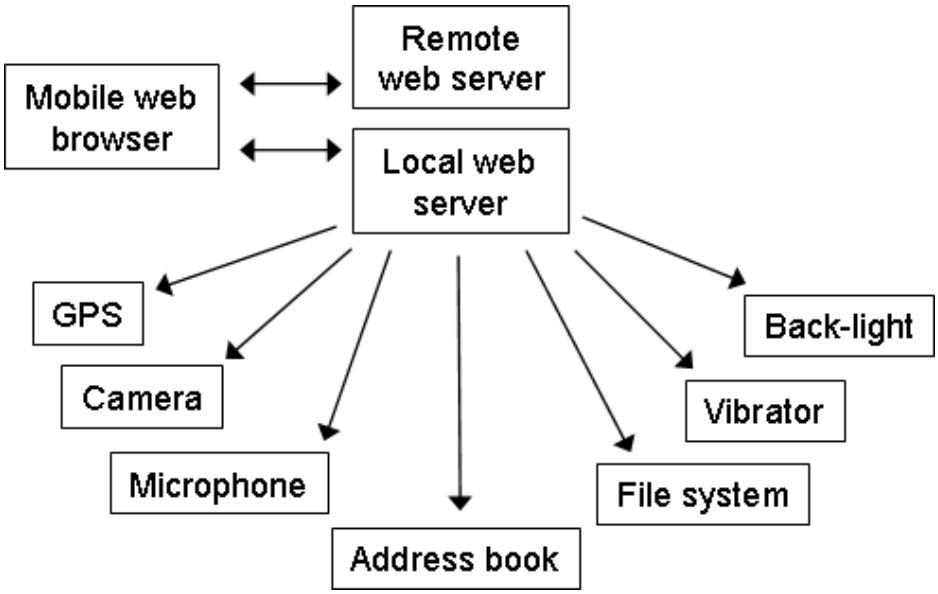


Figure 1.1. System structure.

browser (see Figure 1.1). A page downloaded from the Web could call functions within this local server using HyperText Transfer Protocol (HTTP) requests, for example using Asynchronous JavaScript and XML (Ajax) (see section 2.1.1). This would allow local APIs to be called from applications running in the phone's browser. A few obstacles have to be overcome in order for this to work. Most notably, the "same origin policy" which prohibits scripts from one site from accessing another site, must be addressed (see section 2.2.4). Section 5.2 explains how to get around this problem.

The main goal of this project is to show how a local web server on a mobile phone can facilitate adding new functionality to applications running in the mobile browser. If this approach can be implemented on a mobile phone, there will be two additional goals. First, access to local phone features will be implemented and tested. These features should be made available for use by web applications. The final goal will be to implement a basic security scheme.

1.3 Example usage

The proposed system could be used in many different applications. This section will briefly cover a few example applications.

Digital maps have been around on the Web for some time; however, they are just now becoming available on mobile phones. The introduction of both built-in or external GPS receivers offer a natural extension from simply viewing maps to geo-location based services. While such applications are already available on the

Web while using a laptop or handheld computer, running such an application via a mobile browser offers little advantage *unless the relevant GPS data is available*. If GPS data were available, then it can be used by the application to tailor the digital map to the mobile device's current location. The proposed system could enable this by extracting the GPS data via the local server. Only the local server needs to be able to access the GPS receiver. Using such a local server also isolates the web application from the details of accessing the GPS receiver. For example, the local server could use a proprietary interface to access a particular GPS receiver or could parse NMEA messages coming from the GPS receiver via a serial interface.

Today new mobile phones frequently have an integrated digital camera. The ability to take pictures and uploading them directly to a web site would facilitate the creation of many new services. For example, one could update a photo blog in real time using one's phone. If GPS data was available along with the photos, then each of the photos could automatically be labeled with the coordinates of the camera when the photo was taken.

Mobile games are quite commonly written in Java; however, there is no reason why one could not create games that run in a browser. However, when writing interactive games for mobile devices, developers want to use the complete set of input and output interfaces available on the device. Such interactivity can involve vibration, sounds, and special input (for example, of orientation via gyroscopes which are built into some new phone models). As long as the game's application logic can be written using JavaScript, and the user interface modeled with browser supported markup, the proposed system's local functions would only have to be called upon for specific local events.

Utilizing the proposed system does not necessarily involve building a new application from scratch. If the server has the ability to download documents by itself and relay them to the browser (by acting as a proxy), information could be dynamically added to extend the functionality of many web applications. Mozilla's Firefox web browser has an extension called Greasemonkey that enables users to add or replace parts of the web sites they are viewing [55]. This proposed system could enable similar functions by applying scripts or locally adding data to documents **before** sending them along to the browser.

1.4 Security

There are obvious security risks with this solution. Opening up local APIs to web applications exposes the device to a number of possible attacks, which could affect the user's privacy and personal data. The security implications and threats have to be taken into serious consideration in order for this proposed solution to be viable. Fortunately, there are a number of things that can be done to increase the security. Three steps towards a secure solution are outlined below. For this approach to make sense, it is absolutely crucial that the user can trust the software responsible for exposing the local functionality. The proposed application, which is to run in

the background, must be certified by some authority trusted by the user. It could, for example, be signed using a key issued by an authority whose root certificate is installed on the device. This ensures authenticity of the software, which assumes that the signer of the software is responsible for considering and evaluating the security of the exposed functions.

The first step is to maintain a list of trusted domains. Only pages located within these domains would be allowed to use local functions. The list could either be a static part of the system, updated manually by the user, or downloaded automatically from a central location. The user could be queried each time an application requests to use the exposed APIs. If the user answers yes, the address of the application could optionally be added to a local list, in order to be accepted automatically in the future.

The next step is to handle access rights through sessions. Once the user grants permission to a web application, a random session key would be generated and sent to the application by the background process. Each request would then only be accepted if it was accompanied by the session key, only known to this specific application.

Asking the user for consent to use local functionality could be a problem. If there is only one choice, to allow an application or not, the user might not understand the implications of saying yes or no. On the other hand, if queries were made for each request with a detailed description of the reason, it may become too distracting. Therefore, the third and final step in this approach is to sort the different functions into levels of possible impact. For example, lighting and vibration could be classified as fairly harmless functionality, while access to the user's private data would be on the other end of the scale. The user would then be able to set a desired security level for a certain web application.

Chapter 2

Background

2.1 PC browsers

This chapter examines some of the technologies used in web browsers to extend their functionality. To understand how the mobile web is developing and in what direction it is heading, we look at what has happened with regard to the development of fixed computer based web browsing and fixed web servers. As the capabilities of mobile devices increase, their usage becomes increasingly similar to that of fixed computers. For this reason, many believe that the current trends of the (fixed) Web may offer insights into the future development of the mobile web.

There have been several attempts to make the Web more dynamic. The reasons for this are the need for richer designs, the availability of better user interfaces, and the desire for increased interactivity. Each of these technologies will be examined in the following section, focusing mostly on what this technology can provide in terms of new resources.

2.1.1 DHTML

Dynamic HyperText Transfer Protocol (DHTML) encompasses a range of technologies used together to create interactive web sites. The following subsections will describe some of these technologies and the functions which they offer.

JavaScript

Most of the widely used web browsers include support for scripts. Although people generally call it JavaScript, the standard language for client side scripting is ECMAScript [79]. The actual script implementation in browsers consists of three parts: ECMAScript, the Document Object Model (DOM), and the Browser Object Model (BOM). The ECMAScript implementations are quite similar in different browsers, but the DOM and BOM APIs are often not very compatible.

The ECMAScript part provides basic programming functionality. It includes descriptions of types, objects, keywords, operators, and general syntax.

ECMAScript is not limited to implementations of JavaScript, but is also used as a base for other scripting languages such as Windows Scripting Host (WSH) [79] and ActionScript [3].

DOM is an API for viewing and editing the structure and content of a document. It maps the structure of an HyperText Markup Language (HTML) or eXtensible Markup Language (XML) document onto a tree. The nodes of this tree represent all the elements of the document and individual nodes can be edited or deleted at will. One can also add nodes to create new elements in the document. This allows web developers to create pages that dynamically change within the browser, without needing to reload the page from the server.

BOM provides similar functionality to DOM, but instead of accessing web page content, it lets you change windows and other browser related objects. One can for example set the status bar text and move or pop up new windows using BOM. The BOM implementation differs a lot between browsers since this functionality is closely related to the internal structure of the browser. Some parts are almost always available, such as the window object and the navigator object which provides details about the web browser itself. For more information about DOM and BOM see [50] and [54] respectively.

CSS

Cascading Style Sheets (CSS) is a language that defines style information for documents written in markup languages such as XML and HTML [11]. CSS code provides information about **how** to display data. It can define properties such as color, font, and border layout. Its main purpose is to separate document content from document presentation. CSS also has the ability to make pages appear properly on different media. By supplying appropriate presentation properties, the same document can for example be viewed in both a regular web browser and on a mobile phone. This is accomplished by writing style information for specific media types, such as 'screen', 'print', or 'handheld'. The style sheet code can be supplied in different ways and from several sources. Developers can embed CSS in an HTML document or access it via a link to an external file. Users can provide their own style information to override the provided presentation. Additionally, web browsers usually have a default style for rendering documents.

Ajax

Ajax is a web development technique to enable browsers to communicate with servers without reloading entire pages [80]. This can be accomplished through a number of means, including using frames or the XMLHttpRequest object. The data is often transmitted in XML format, but this is not a requirement. Any format can be used, including plain text. Regardless of format, the data can be dynamically sent to the server at any time. As the name suggests, it is an asynchronous process where data returned by the server can be handled by a predefined callback function.

There are several benefits to this technique. Since no page reload is required, updates are faster and generate less traffic. Web interfaces can be made to look more like desktop applications and feel more natural to the user.

2.1.2 Plug-ins

Common for all plug-ins is that application logic executes on the client computer. These applications can usually benefit by having access to local functionality.

ActiveX controls

An ActiveX control is a software component used in Microsoft environments. It is usually designed for visual presentation purposes, for example, to provide Excel spreadsheets or other Microsoft software technology to any Windows platform application, including web pages [35]. The Internet Explorer web browser supports ActiveX and can download and use ActiveX controls from web sites. Once a control is downloaded and installed it can provide resources and access the local system's APIs. This could be, and frequently is, used by malicious software to take control of the client computer. Another downside to this technology, which has contributed to its unpopularity among developers, is the fact that it only works properly in Microsoft's Internet Explorer. If support is needed in additional browsers, one needs to employ other technologies.

NPAPI

Netscape Plugin Application Programming Interface (NPAPI) is a plug-in architecture supported by many web browsers, although it is **not** included even in recent versions of Microsoft's Internet Explorer. It works by having plug-ins declare which media types they can handle. When the browser encounters such media it loads the appropriate plug-in. Some browsers also support interaction between plug-ins and JavaScript. There have been several extensions developed for this purpose. For example, LiveConnect (introduced with Netscape 4) enables Java applets to access the DOM, and JavaScript to call Java methods. The most recent extension agreed upon by most of the major browser developers (excluding Microsoft), is called npruntime. Npruntime is independent of Java and is more powerful and flexible than the earlier protocols [37].

Flash and Shockwave

Flash and Shockwave are technologies mainly used for embedding rich media in web pages. They were both developed by Macromedia, which was acquired by Adobe in 2005. Adobe supplies a browser plug-in for each of them [4, 5]. Flash provides a lightweight solution that can be used for video playback and building interactive web sites. It supports scripting through an ECMAScript based language called ActionScript. Shockwave on the other hand has more extensive functionality and is

designed for larger applications. It can display advanced media such as 3D games and interactive product demonstrations. Both are widely utilized around the world. According to Adobe, the Shockwave plug-in is currently installed on 58.5% [7] of all Internet-enabled desktops in mature markets (US, Canada, UK, France, Germany, Japan), while the Flash plug-in is available on 99.1% [6].

2.1.3 Java

Java applications can run on almost any platform and are therefore well suited for extending browser functionality. The only requirement is that there is a Java Virtual Machine (JVM) installed. There are two common ways of downloading and executing Java code: Applets and Java Web Start.

Applets

Applets are small applications that run inside a web browser. A JVM is started and acts as a "sandbox" where an applet can run with limited resources [66]. The applet can either run as an embedded object or start up in a new window. Either way, the browser needs to have a Java plug-in installed. The necessary Java plug-in is available for nearly all browsers running on almost any platform.

Java Web Start

A more recent approach to loading Java applications from the Web is Java Web Start. Instead of the browser controlling the program, the browser simply downloads a Java Network Launching Protocol (JNLP) description file with information about the application. If configured properly, the browser passes this file to the JVM. The JVM immediately downloads and starts the indicated Java code. An important feature of Java Web Start is its ability to make sure that the proper version of an application and the correct Java Runtime Environment (JRE) are used [67].

Reflection API

In Java, a class can be loaded and its properties examined at runtime. This allows developers to dynamically load and use classes without knowing their exact specification. An important component in this process, which makes methods and variables visible, is the reflection API [32]. This functionality could for example be used when mapping objects between Java and another language. The system developed in this thesis includes remote procedure calls from JavaScript to Java, which such object mapping facilitates. Unfortunately the reflection API is not part of the current CLDC[1] standard.

[1]Connected Limited Device Configuration (CLDC) is a basic Java configuration for mobile phones (see section 2.3).

2.2 Mobile browsers

Web browsers designed for mobile devices are different from regular web browsers in certain aspects. To present web pages on small screens the rendering has to be adjusted to the device's screen size. Limited processing power and memory means the web browser must be efficient and can not take up too much memory. The browser needs to be easily controlled by the often limited input devices (for example a small keypad for character input). Since mobile devices are battery powered, applications suffer from a limited power supply. Finally, the issue of varying bandwidth and unreliable connectivity may also have to be dealt with.

There are quite a number of browsers available specifically for mobile devices. Some are optimized for devices with very limited resources, such as mobile phones with small screen size and little processing power. Opera's Mini is just such a browser [65]. To make browsing faster, initial rendering is done on a proxy server before being sent to the device in a binary format. This makes it possible to use not only specific mobile web sites, but even sites designed for viewing on traditional fixed computer screens. However, it means that the proxy will have to have access to the content, hence introducing a security hole. Opera also has a more extensive browser called Opera Mobile [46]. As the performance of devices and their memory capacity has increased, many new devices are shipped with Opera Mobile pre-installed. Opera Mobile is supported by many platforms [47]. Opera Mobile uses the same page rendering engine as their PC version, but scales down web pages for viewing on small screens.

Another similar browser, which is pre-installed on many new mobile phones, is NetFront [1]. It supports many different web standards and is also available for more capable (i.e., with greater resources) platforms. It was developed by a Japanese company called Access.

There are several browsers based on the WebKit layout engine. Among them are popular browsers such as the Nokia S60 Browser for the Symbian S60 platform and Apple's Safari for Mac OS X and their iPhone and iPod Touch. The engine was originally based on a fork from Konqueror's KHTML software library which is used for the KDE browser, but WebKit is now a separate open source project [74].

All mobile browsers mentioned above (except Opera Mini) support modern web technologies, such as JavaScript and CSS. They are in fact very similar to web browsers on fixed computers. Limitations in functionality are mainly caused by the mobile device's hardware.

Other browser developers have in a similar fashion created mobile versions of their PC browser. The Mozilla Project's Minimo is a scaled down version of their popular browser [36]. It uses the same Gecko layout engine as Firefox, but lacks some of the more advanced features such as support for the File Transfer Protocol (FTP) and SVG[2]. Minimo is mainly used on slightly larger devices, such as Personal

[2]Scalable Vector Graphics (SVG) is an XML based language for describing two-dimensional vector graphics. For more information see W3C's SVG page [72].

Digital Assistants (PDAs) and high-end mobile phones, and only has ports for the mobile Windows platform and various Linux distributions. Another commonly used browser for PDAs is Microsoft's Internet Explorer Mobile.

There are a number of different protocols used by mobile web browsers for Internet access. The next sections describe their basic functions and limitations.

2.2.1 WAP

The Wireless Application Protocol (WAP) is an open standard for Internet access on mobile devices, put forth by the Open Mobile Alliance (OMA) [78]. It includes a collection of protocols initially similar to the HTTP stack, with specific security and compression features which were believed to be important for the mobile environment. The transport layer equivalent works similarly to the User Datagram Protocol (UDP). On top of this is the Wireless Transaction Protocol (WTP) which performs error checking and re-sends lost packets.

In the first versions of WAP (1.x), documents were not downloaded directly from a web server. Instead, all requests sent from the WAP browser were handled by a WAP gateway. The gateway downloaded the document, transformed it into a WAP specific format and sent it back to the WAP browser [75]. This made it possible for simple devices with little processing power to display web content. Unfortunately, it had a very large number of security problems.

Newer mobile browsers can read and interpret eXtensible HyperText Markup Language Mobile Profile (XHTML MP), a subset of eXtensible HyperText Markup Language (XHTML), which eliminates the need for a gateway for conversion. The WAP 2.0 specification makes the gateway optional as HTTP is used end-to-end, i.e. all the way from the browser to the web server and back. In addition to XHTML MP, WAP version 2.0 also supports a mobile version of CSS, called WAP CSS. In fact, WAP 2.0 is almost identical to the usual HTTP/TCP/IP stack.

WML

The Wireless Markup Language (WML) is a content format language specifically created for presenting web content on mobile devices [76]. It was widely used in the early days of mobile web development, but has now been replaced by XHTML and HTML. WML was based on standard XML, thus it has many of the same features as HTML, including hyperlinks, forms, and image embedding. However, when writing WML, developers need to think of the documents as "decks". The "cards" of the deck are the pages and each page enables one interaction with the user. This is very similar to an earlier programming model called Hypercard which was developed for the Apple Macintosh computer [51].

WTAI

Wireless Telephony Applications Interface (WTAI) is part of the WAP standard. It provides script functions and WML tags for utilizing **basic** phone functionality.

Even though today's mobile web sites use HTML or XHTML rather than WML, the tags are still supported by browsers. The tags are written as protocol identifiers at the start of Uniform Resource Locators (URLs), and tell the phone how to handle the URL. For example, there is a tag for initiating a phone call to a specified number, and another tag for adding a name and phone number to the phone's address book [77].

2.2.2 i-mode

The Japanese company, NTT DoCoMo, developed another standard for mobile web browsing, called i-mode [43]. There were initially some major differences in approach as compared to WAP. I-mode used packet-switched communication right from the start. This enabled clients to constantly stay online, instead of connecting every time they wanted to access the Web. While WAP could not use packet-switched communication until later, when General Packet Radio Service (GPRS) was introduced.

The markup language used with i-mode is not XML based, unlike WML, but is instead a subset of HTML called Compact HTML (cHTML or sometimes iHTML) [25]. It also includes some special i-mode tags. The supported embedded media formats are ones commonly used on the Web. This and the use of cHTML makes it very easy to adapt HTML documents for use with i-mode.

2.2.3 Plug-ins

Flash Lite

Just as for PC browsers, there is a Flash plug-in for mobile browsers. Flash Lite is a scaled-down version of Flash 8 and can be used to view the Flash content available for PCs (with a few limitations). It supports ActionScript 2.0, but lacks some computationally intensive graphic functions such as filters and blend modes [59].

2.2.4 JavaScript

Although not as powerful as on PCs, JavaScript support is being implemented in many mobile browsers. There are standards proposed by Ecma International (ECMAScript Compact Profile) [13] and OMA (ECMAScript Mobile Profile) [45] specifically for mobile browser developers to conform to, but neither has gotten wide support.

JSON-RPC

JSON-RPC is a remote procedure call protocol using the lightweight data interchange format JavaScript Object Notation (JSON) [24]. JSON is based on ECMAScript, but is completely language independent. It represents common data

types and structures in a way that is very familiar to C++ and Java programmers. JSON-RPC can use this language to communicate over different protocols, although TCP/IP is recommended. Remote procedure calls can for example be made from a client using HTTP to call methods on an HTTP server. The JSON data is sent in the body of an HTTP POST request.

Same origin policy

The same origin policy states that scripts from one origin may not set or get properties of a document from a different origin [58]. This rule is applied by browsers to prevent the use of cross-site scripting. To explain exactly what defines an origin we examine the distinction made in the Mozilla browsers. We may change a remote document as long as it is located on the same server as the script. Changing directories is also allowed, but any change to the protocol, port, or domain is considered a change in origin. While there is a way to change the domain within a script, the new domain has to be a suffix of the previous one (i.e., be a parent domain of the previous domain). For example, `http://some.domain.com/` can be changed to `http://domain.com/`, but not to `http://another.domain.com/`. In addition, `http://domain.com/` now becomes the origin and it is not possible to consider documents from `http://some.domain.com/` as having the same origin any longer.

SunSpider

SunSpider is a JavaScript benchmark put together by the WebKit developers [64]. Their goal is to collect and create tests based on code that is used in real web applications. It consists of both calculations from the Web of today and the kind of demands that they expect to find in newer more advanced applications. The current version (0.9) includes benchmarks from areas such as 3D rendering, bit operations, cryptography, and strings. It also has tools for statistical analysis and comparison of results.

2.3 Java ME

The popularity of advanced mobile phones and other small form factor devices has led to fierce competition and many different models of many products being produced. More over, each of these devices has its own features, errors, etc.; this makes it hard for developers, writing applications for these devices, to keep up. There are so many platforms and standards that it is nearly impossible to port software to all of them. Java is thought to provide a solution to this. In theory all Java code can run on any device supporting a JVM, which means developers only need to write a Java program once. In practice though, there are some limitations concerning specific technical features of the target device which have to be considered.

To facilitate adapting code to specific device models, Sun Microsystems has developed a standard API with classifications for different platforms. This standard, called Java Platform, Micro Edition (Java ME), is basically a subset of Java Platform, Standard Edition (Java SE), with certain additions for mobile device functionality. The additions facilitate compatibility between devices with similar features. The omitted classes include many data structures and convenient methods for string manipulation. Sun created a completely new networking API for Java ME. The Generic Connection Framework (GCF) provides I/O capabilities with a smaller memory footprint than the original Java connection APIs. This can make it difficult for developers to port network applications to Java ME, since most of the classes handling connections have to be replaced. Sun provides a reference version of Java ME, then allows device manufacturers to create their own implementation. Such an implementation should consist of three parts:

- A configuration
- A profile
- Optional packages

The most basic libraries needed to run a Java application are bundled into a given configuration. The Connected Limited Device Configuration (CLDC) is a configuration aimed at small devices such as mobile phones. Connected Device Configuration (CDC) includes more libraries and is meant for larger devices, such as smart communicators, PDAs, and set-top boxes (see Figure 2.1). Profiles include additional APIs to utilize features of a range of devices and extend the capabilities of the underlying configuration. The Mobile Information Device Profile (MIDP) is the most common profile being used with CLDC on mobile phones. Together the configuration and the profile provide a specific Java application environment. On top of a configuration and a profile, a device can also have optional packages. These are typically APIs for specific technology available on the device.

Standardization of the Java ME technology is handled by the Java Community Process (JCP). JCP guides the development of Java and approves technical specifications known as Java Specification Requests (JSRs). Anyone is allowed to participate in the process, which is designed to ensure the stability and cross-platform compatibility of the Java (family of) platform(s). For further information, see the JCP web site [23] or Sun's Java ME page [69].

2.3.1 JTWI

Java Technology for the Wireless Industry (JTWI) was introduced in 2003 to extend the current Java ME standard. The MIDP profile was considered to lack strict definitions, and thereby creating fragmentation in the functionality between different devices. MIDP's vague hardware requirements also caused problems with regard to portability. JTWI addresses these issues by enforcing stricter definitions and requirements for Java enabled devices [27].

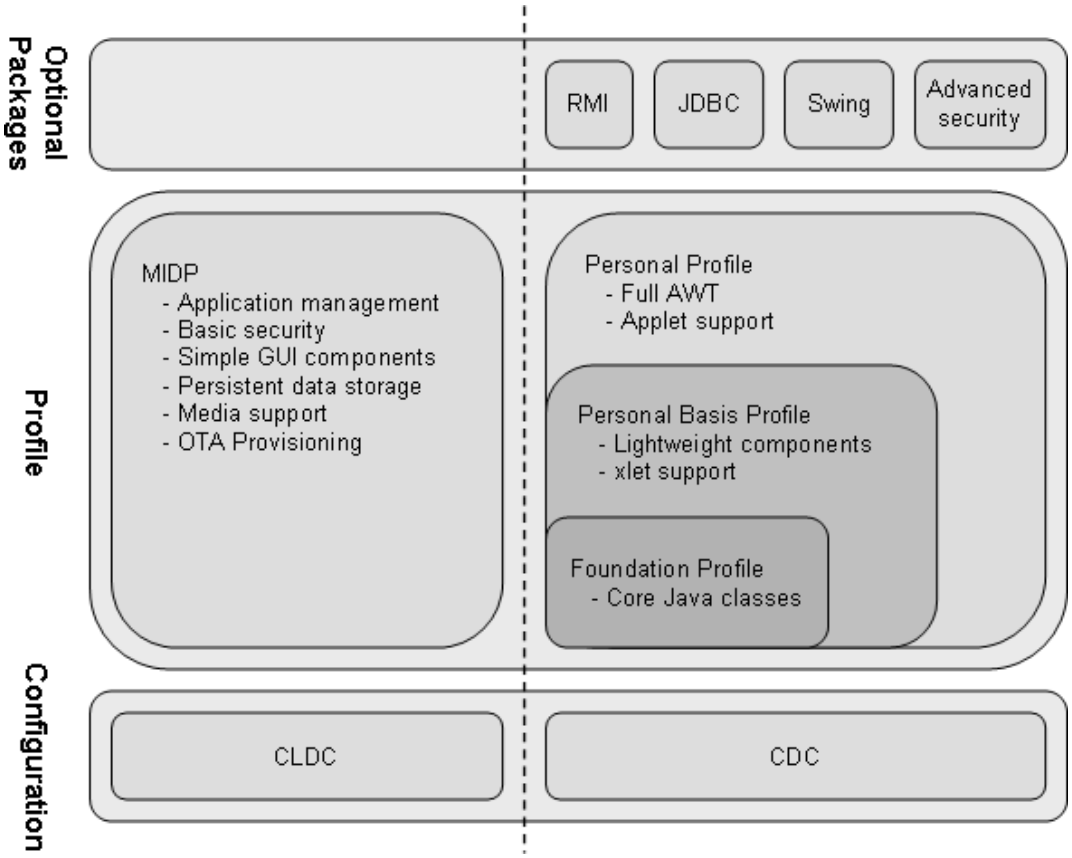


Figure 2.1. CLDC and CDC architecture. CLDC can also be extended with optional packages (see Figure 2.2).

A device supporting JTWI is required to support CLDC 1.0, MIDP 2.0, and the Wireless Messaging 1.1 API (WMA). The Mobile Media 1.1 API (MMAPI) is required if the JVM exposes video playback, audio capture, or video/image capture functions.

JTWI clarifies several, previously vaguely defined, requirements. A conforming Java device must, for example, support a Java Archive (JAR) size of at least 64KB. JTWI also recommends 256KB of available heap memory, compared to MIDP 2.0's required 128KB. Other requirements include Short Message Service (SMS), phone book access, and JPEG support. Musical Instrument Digital Interface (MIDI) and tone-sequence content must be supported and, if the MMAPI is included, a minimum quality for audio and video capture is imposed. JTWI also increases requirements on the security model. More information can be found in the JTWI specification [17].

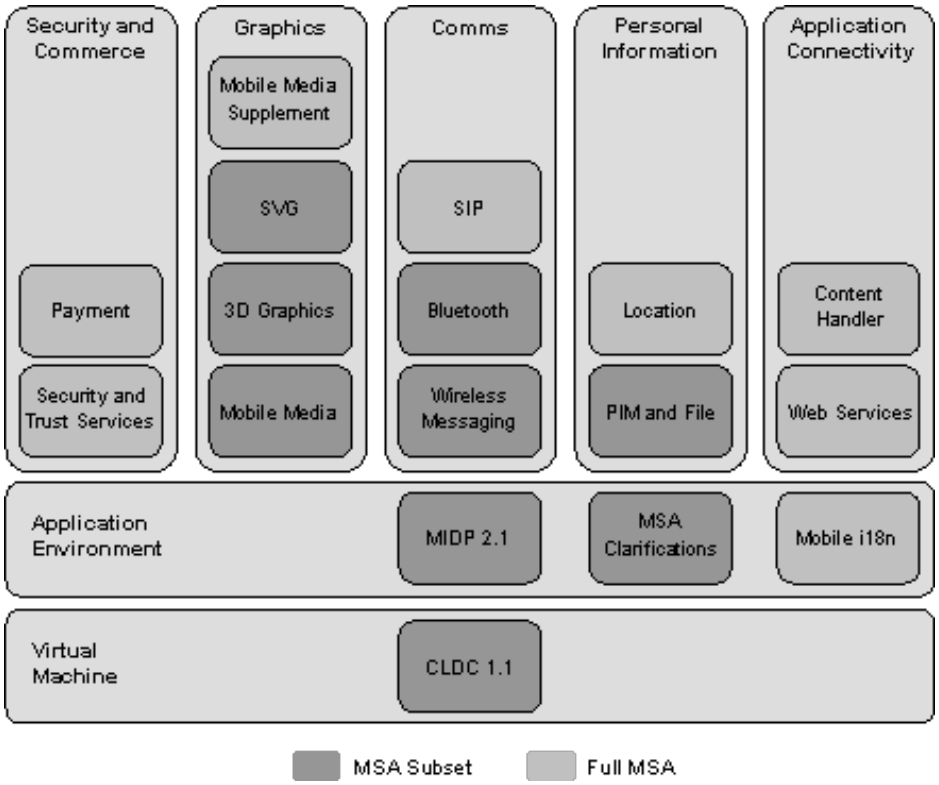


Figure 2.2. Mobile Service Architecture (adapted from figure in [71]).

2.3.2 MSA

To accommodate the introduction of new technology in mobile devices, a new platform extending Java ME has been specified. The Mobile Service Architecture (MSA) [71] builds on the current specifications (i.e. CLDC, MIDP, and JTWI) creating a new standard for the next generation of mobile devices. MSA provides a new set of application functionality, but also clarifies interactions in existing standards. Since mobile devices have varying capabilities, there are two choices for implementation: to implement a predefined subset of MSA or the entire MSA specification. To be MSA compatible, a device must either support all of the predefined subset or all of the full MSA (see Figure 2.2).

Most of the components in both the subset and the full MSA are mandatory. However, a few are only conditionally mandatory. These are only required if the device supports the underlying hardware needed for the functionality. The Blutetooth and location APIs are such components. If, for example, a device is to support the MSA subset and has Bluetooth capabilities, then the Bluetooth Java API must be implemented. However, if the device lacks Bluetooth connectivity, it can support the subset without this API.

2.3.3 MIDlets

Java web applications can be run inside a browser environment on a computer. These programs are called Applets. Unfortunately, currently it is not yet possible to do so for mobile browsers. Java on mobile devices has to be started as a separate application. These separate Java applications are known as MIDlets [28].

Installing

Installing MIDlets can be done from a PC or using 'over-the-air' (OTA) provisioning [48]. Using a PC, the Java application is first downloaded, then using a cable or a wireless connection (such as Bluetooth or WLAN) the file is transmitted to and installed on the device. OTA provisioning was introduced as a recommended practice after the MIDP 1.0 specification. In version 2.0 of MIDP, OTA provisioning was improved and made part of the base specification. OTA is now the standard for finding, downloading, and installing Java applications on a device over a wide area wireless network. In order for a mobile device to support OTA it must be capable of using both the HTTP protocol and HTTP authentication methods. The device is also required to have software that can locate and discover MIDlets.

Starting

MIDlets can be started manually by the user, activated remotely, or started automatically. Automated and remote starting is handled by the MIDP 2.0 push registry [49]. A MIDlet can be registered to start at a certain time or after every boot. There are several ways to activate a MIDlet remotely. A MIDlet can be registered to start upon receiving an SMS message, or a UDP datagram or TCP socket connection (such as an HTTP connection) [16].

Signing

The security of Java ME applications is based on protection domains. Each potentially dangerous action requires a certain permission, based upon which the action is accepted, denied, or permission is requested from the user, depending on which domain the MIDlet is installed in. An unsigned application will request permission from the user each time. Developers can sign their MIDlets using a certificate from a recognized authority, and thereby become identifiable as the authors. For an authority to be recognized, their root certificate must be available on the device. A signed MIDlet can have certain permissions granted without consulting the user or be given so-called "blanket" permissions. Blanket permissions are only granted by the user once, then accepted subsequently without further asking the user.

Background MIDlets

Some Java enabled phone models support running minimized applications, i.e. MIDlets running in the background without user interaction. These MIDlets can either be applications without a user interface or be explicitly defined as background applications in the Java Application Descriptor (JAD) file. Sony Ericsson refers to the latter as standby MIDlets. Such a MIDlet can be used as a wallpaper which is started when the phone enters standby mode. Standby MIDlets can not interact with the user without first being activated. A regular MIDlet without a user interface can not have user interaction either, but if the phone allows it, the MIDlet can activate itself and present a user interface. This can be used to temporarily take control of the screen for alerts and urgent user input. To separate these MIDlets from standby MIDlets, this report will refer to them as background MIDlets.

MIDlet as a server

Java ME lacks built-in support for accepting HTTP connections, unlike Java SE does. It does, however, have support for opening local sockets. A MIDlet running on a mobile phone can open a socket for listening and parse (for example) an HTTP request coming from a browser. If a mobile browser on the device can connect to `localhost` (or equivalently its own IP address), then the MIDlet could act as a local server. It could even be set up as an HTTP proxy, relaying traffic from any application on the Web.

The Connector class works as a factory for all connections in a Java ME implementation. It takes URLs of supported formats and creates appropriate connection objects. For example, if a correctly formed HTTP URL is inputted, a connection is opened and handled by an instance of HTTPConnection.

Chapter 3

Related work

This section examines existing projects relating to this thesis. Some of them have functions that could be used in the proposed system.

3.1 Mobile Web Server

Researchers at Nokia have developed a web server for their S60 platform, called the Mobile Web Server (MWS) [40]. It is a version of the Apache HTTP Server ported to run on Nokia's S60 platform. Nokia has added several extensions to the server and it supports web development using Python. The extensions are closed source components, but both the Apache HTTP Server and Python are open source [8, 56]. The goal of the project is to provide a new way of publishing personal information and media on the Web. Instead of uploading resources to a regular Internet server, data already on the mobile device is made available directly through a local web server. Examples of usage include sharing photos and videos with your friends, straight from your camera phone. It also enables users to access and control their mobile device from any personal computer (or other device) equipped with a web browser. New entries can remotely or locally be added to the phone's address book. SMS messages can be sent using plug-ins made available via this web server to the web browser which is running on their computer.

During development of this system many issues and problems had to be addressed. Mobile devices usually have very limited resources compared to general purpose computers. The processing power and available bandwidth is often lower than for devices connected to main power and with fixed broadband connections. If a mobile device runs on batteries this obviously limits its available power. There are also issues concerning security and connectivity (few service providers allow unrestricted incoming connections to their subscribers). Connecting to a mobile device requires that you know its address. In IP based networks this would be an IP-address. Addresses for devices in a wide area cellular network are usually assigned dynamically, which means you may not get the same address every time you turn on the device. The problem of needing to know the current IP address could

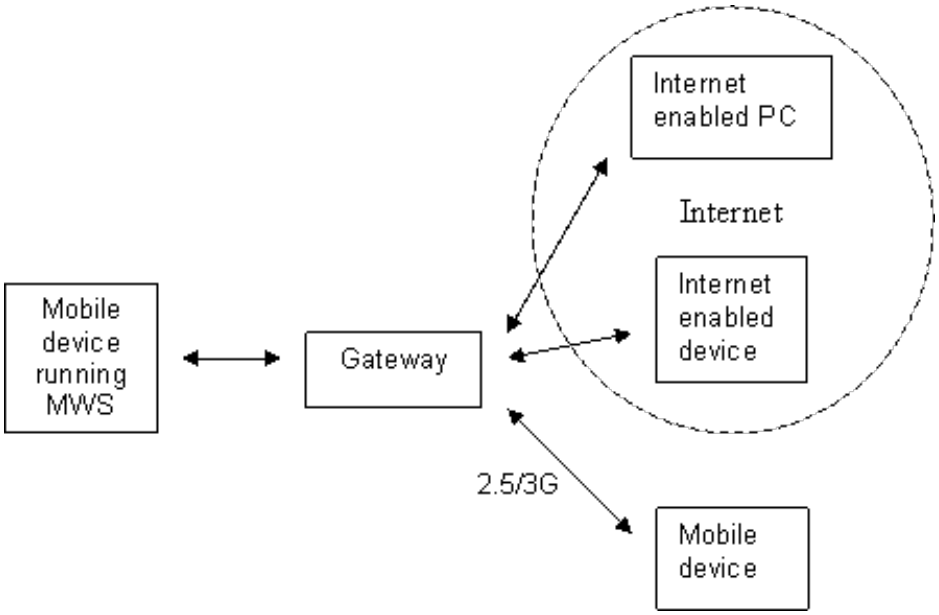


Figure 3.1. The MWS's communication paths (adapted from figure in [41]).

be solved via dynamic Domain Name System (DNS) or via Mobile IP. However, it is also possible that the address is a private local address and therefore only reachable from within a protected network. Nokia has solved this by routing all traffic through a special gateway, the so-called Raccoon Mobile Web Server gateway (see Figure 3.1). When a remote client wishes to access the phone's web server it first has to look up the domain name of the phone by using DNS. The domain names associated with all the mobile web servers point to a Raccoon Mobile Web Server gateway. This means that all requests are first processed by the gateway, which redirects them to the proper mobile device. Since only the gateway knows how to contact the MWSs, all incoming traffic has to pass through it (at least initially). A related method is to use Rendezvous (see Gustav Söderström's thesis [60]) or a SIP TURN server [57].

Mobile devices running on batteries could become unreachable when the batteries run out of power or when there is no network connectivity. This means the mobile web site could sometimes be unavailable. To inform users of this, a default page can be created and stored on the gateway. If someone tries to access the mobile device's MWS while it is unavailable, they are presented with this default page.

3.2 S60 Web Run-Time

In the Feature Pack 2 release of Nokia's S60 3rd Edition system, support for widgets is included [42]. The Web Run-Time enables developers to build small web applications (widgets) using common industry standards, such as HTML, CSS,

JavaScript, and Ajax. The applications are downloaded and installed on the system, then they can gather and display content from the Web.

The JavaScript implementation follows the ECMA-262 specification, but also includes some additional APIs specifically for widget development. The extended scripting features enables retrieval of the following system information:

- Battery level and charging status
- Reception and network information
- System language
- Memory size and current usage
- File system listing and available space

The new APIs can also be used to store persistent values (associated with the current application, not shared), and trigger certain system resources. Functions for the following actions are available:

- Keypad illumination
- Trigger back-light
- Vibration
- Play tone
- Launch native application (not other widget)

Adding these APIs is a big step towards opening up local functionality to web applications. However, some features have been left out in this release. Neither local data access nor positioning functions are included. The reason for this most likely relates to security and privacy concerns.

3.3 Ajax for Java ME

Sun Microsystems has created an open source library for creating Ajax applications in Java ME [9]. The idea is to combine the simplicity and familiarity of the Ajax model with the rich and (supposedly) secure Java ME environment. Sun's MSA specification provides functionality for multimedia and animated graphics that can be used to create interactive user-friendly interfaces for mobile applications (see section 2.3.2). In this context the Ajax model has three parts: asynchronous remote calls using the MIDP GCF, data represented in either JSON or XML, and a user interface based on a DOM enabled markup such as XHTML or SVG.

Building a web application using Java has several advantages over a browser based solution. The Java ME environment has a strong security architecture and

provides access to many APIs. Extra features on mobile devices such as cameras and GPS receivers can be integrated directly into applications. There are also functions for accessing the phone's address book and local storage. However, exactly what functionality can be used depends on the Java ME implementation on the device.

The Ajax for Java ME library provides a way of writing web applications that can integrate local device functionality, with dynamic Graphical User Interfaces (GUIs) written in a markup language. Such applications could theoretically replace web browsers. However, currently the only markup supported by Java ME is SVG Tiny 1.1. JSRs for additional languages are being reviewed, including JSR 287 [18] for extended SVG support and JSR 290 [19] for the common web languages (XHTML, CSS, JavaScript).

3.4 JSON-RPC-Java

The company Metaparadigm has developed a framework for creating Ajax applications using JSON-RPC. It uses JSON instead of XML to represent data. JSON can represent basic data structures and has a simpler syntax than XML. JSON-RPC-Java enables server side Java methods to be called from local JavaScript [33]. A lightweight JavaScript script is used on the client side and the Java code runs in a Servlet container on a Java EE application server[1]. Method calls are dynamically mapped from JavaScript using Java Reflection.

3.5 Location acquisition

3.5.1 LocationAware

According to the Location Aware Working Group, location-aware content and functionality will become more and more common in web applications. Therefore, they are working with browser vendors, device manufacturers, developers, and content providers to standardize the way location data is handled [29]. Location data can be obtained through several methods, such as GPS, Wi-Fi triangulation, and IP geo-location, yet there is no standard way for browsers to acquire this data. There is also a need for standard methods to access the data from within web applications, without compromising the user's privacy. The latest working draft includes sample code, showing how JavaScript access to location data might look:

```
var geolocator = navigator.getGeolocator();
geolocator.request(function(location) {
   alert(location.latitude+', '+location.longitude);
});
```

[1] Java Platform, Enterprise Edition (Java EE) is a Java platform used for server programming. In addition to the Java SE APIs, it includes libraries for deploying distributed components on application servers. For more information see [70] or Sun's Java EE page [68].

Name	Platform	Running on	Client	Server	Ajax support
Mobile Web Server	S60	Nokia, LG, Lenovo, Panasonic, Samsung, etc.	No	Yes	-
S60 Web Run-Time	S60	Nokia, LG, Lenovo, Panasonic, Samsung, etc.	Yes (using browser)	No	Yes
Ajax for Java ME	Java ME	Java enabled devices	Yes	No	Yes
JSON-RPC-Java	Java EE	Computers	No	Yes	-

Table 3.1. Feature summary of related projects.

The code snippet requests location data through a geo-location object, and adds a callback function which displays the returned values for latitude and longitude in a pop-up box.

3.5.2 EZweb

The EZweb mobile browser from KDDI (a Japanese company) has a built-in function for requesting location data from the device it is running on. It utilizes a special protocol that can be used in, for example, links to online maps. Such a link might look like:

```
device:location?url=http://server/location.cgi
```

When a request using this protocol is made, the browser acquires location data and appends it as GET variables in the URL. This would be the form of a URL requested by the browser:

```
http://server/location.cgi?datum=AAA&unit=BBB&lat=XXX&lon=YYY
```

The capital letters represent the location data and included information provided by the positioning system [26].

Name	Proxy classes	Proxy code size	JAR size
RabbIT	61	196KB	241KB
Super Proxy System	11	56KB	149KB
PAW	6	33.3KB	107KB

Table 3.2. Open source Java proxy servers. Proxy classes denotes the number of class files used for the core proxy features and proxy code size is the total size of those files. JAR size is the size of the whole application.

3.6 Google Gears

Google Gears is a plug-in for web browsers, adding extra functionality to allow web applications to run offline. It lets developers create offline browser based applications by adding new JavaScript APIs. The plug-in consists of three parts:

- Web server
- Database
- Worker thread pool

The web server is used to serve offline content such as HTML documents, JavaScript, and CSS. A lightweight database allows applications to save persistent data, while the worker thread pool handles long-running background operations. Google Gears supports several browsers, including Firefox and Internet Explorer. It is also meant to be used on mobile devices, but it currently only supports devices running Windows Mobile (version 5 or higher). Google Gears is still in the early stages of development and is not yet suitable for production applications [14].

3.7 GlassFish

It is possible to create an application server small enough to fit on a mobile phone. This has been shown by Sun Microsystems. In 2005 they launched Project GlassFish. It was an initiative to make their Java EE application server open source. The third release, which is currently under development, has a very small basic kernel size. Pelegri-Llopart, et al. wrote (about GlassFish v3): "Its architecture is modular by default, its kernel is extremely small (under 100Kb which makes it suitable for desktop and even mobile use), and its startup time is under a second" [53].

3.8 Java proxy servers

There are many open source proxy servers written in Java available. These can be extended for other applications and the code can be reused in other software. Three

typical such proxy implementations are RabbIT [44], Super Proxy System [31], and PAW [52]. They are written for Java SE and depend on the standard Java APIs for networking. PAW also uses external frameworks for server and parsing capabilities. The size of each of the three proxy servers can be seen in Table 3.2.

Chapter 4

Evaluation methods

4.1 Usability

Evaluation of usability is important when creating mobile applications just as it is when developing for PCs. Boonlit Adipat and Zhang Dongsong wrote in a report on usability testing that: "Usability testing is an evaluation method used to measure how well users can use a specific software system. It provides a third-party assessment of the ease with which end users view content or execute an application on a mobile device" [2]. From their description, we can see that there are several things that could be evaluated. Some of the questions that should be answered are:

- Can users easily search for specific information?
- Is the menu design and link structure easy to understand?
- Does the data entry method enable fast and easy input?
- How is the user experience affected by the mobile context?

The usage of mobile device applications differs from that of PCs in several aspects. Some of the main differences concern connectivity, screen size, display resolution, processing, battery power, and available user input methods. These physical limitations all have an effect on the overall usability of the software. The usage context is also somewhat different when it comes to testing mobile applications. There are additional situations to deal with. For example, users of mobile devices tend to work standing up or walking around. They also do not always have proper lighting and sometimes there is too much light. These differences have to be taken into account when designing an evaluation method.

There are two distinct categories of testing available for evaluation. One can either perform the tests in a (closed) laboratory environment or send people out of the laboratory to conduct field tests. Laboratory testing enables easy supervision and direct control of participants. Mobility can be simulated by asking the participants to move around while using the device. Details about the environment

can be controlled, such as lighting and noise. It also makes it easy to collect different kinds of data, and is therefore well suited for usability testing. Testing in the field does not provide the same ease of observation and supervision. It makes it harder to control the context and properties of the environment, but potentially yields more realistic information. The dynamic and sometimes unreliable nature of wireless connections is naturally present when testing in a real-life situation. Connected applications tested in the field can be used to predict more accurately the real life experiences which users will have.

When conducting tests in a laboratory there is a choice to be made whether to use emulators or real mobile devices. Emulators make it easy to collect data, but lack the real context of the application. Using real devices creates a more realistic user context. Consequently, while emulators are suited for initial testing of layout and menu structure, real devices should be used for final testing.

4.2 Performance analysis

Mobile devices use wireless connections to communicate with services and other devices in different locations. Connections can be made both on a local peer-to-peer basis (for example, using Bluetooth) or via wide area networks. Many mobile applications are created to work in this context and use network connections to enhance interactivity. These applications can therefore be seen as distributed systems, and evaluated as such. There are two aspects of mobility in distributed mobile applications that should be considered: physical mobility which concerns the actual movement of the device, and code mobility which refers to the positioning of the distributed software parts.

Antinisca Di Marco and Cecilia Mascolo wrote: "Physical mobility is a requirement which developers have not yet considered with the due care despite it having a huge impact on the performance of the system" [30]. Analyzing physical mobility involves identifying mobile user patterns and looking at the impact they have on the system performance. Changing mobile contexts can affect connection speeds, and must be thoroughly considered when evaluating the overall performance.

In the case of a mobile distributed application, different parts of the code run on different hardware. Therefore, the application logic can be distributed across a network. To evaluate performance in such a system, code mobility has to be considered. Since mobile devices often have limited processing capabilities, higher performance can sometimes be achieved by moving heavy computation to servers which have greater resources. However, because transferring data between system parts can be slow and expensive, this is not always the best solution. Another way of improving performance is moving application components closer together, thereby turning previously remote interactions into local interactions. If this kind of software mobility were dynamic, then code location could be optimized during execution. This could increase both flexibility and performance of the system [10].

Integrating performance analysis in the system specification is a critical part of

software design. When designing mobile applications it is therefore important to plan for this kind of evaluation. Several researchers have suggested methodologies for performance analysis in the mobile context [10, 15]. The methodologies usually involve creating evaluation models based on UML diagrams.

Chapter 5

Implementation

This thesis project was carried out at Ericsson Research in Kista. The main goal was to show how a local web server (proxy), running as a background process on a mobile phone, can facilitate adding new functionality to web applications running in the phone's built-in browser. Based on the success of the main goal, there were two additional goals: implementing several functions for use in real web applications and setting up a basic security scheme.

A background application for mobile phones was implemented to prove that the concept works. The application acts as a proxy server for the mobile browser. It also accepts HTTP requests with commands for accessing local phone data and triggering phone specific functions.

5.1 Platform

When choosing a suitable device to develop the software for, there were two important aspects to consider. The phone had to have some built-in functions that could be utilized. For example, enabling use of a digital camera was a definite requirement. A GPS receiver would also be of use, if it could be accessed by the application. Secondly, since Ericsson is supporting this thesis project, the device would have to be of the Sony Ericsson brand. The currently available models support either Symbian or Java ME applications. Since few of the Symbian models had cameras, but several of the Java enabled phones had cameras, the latter was a natural choice. None of the currently available Sony Ericsson phones had a built-in GPS receiver, but some of the later models have support for an external GPS device. Ericsson supplied a Sony Ericsson K800i [61] phone to be used during development. The K800i supports Sony Ericsson's Java Platform 7 (JP-7) [62], which includes among other things, JTWI. However, full MSA compatibility was not implemented until Java Platform 8. The K800i supports the Sony Ericsson HGE-100, a GPS enabler/hands-free device, of which one was also supplied for use during development.

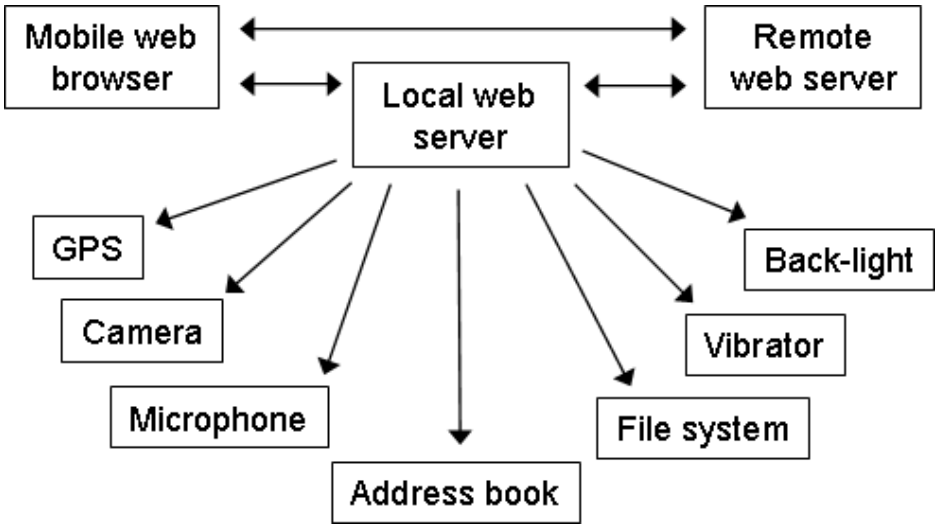


Figure 5.1. Final system structure. Web content can be loaded via the local server.

5.2 Proxy function

Before any specific features could be implemented, the problem of the same origin policy had to be solved. Web browsers will not allow JavaScript from one domain to read data downloaded from another domain. Data can be sent, in the form of a URL, by programmatically adding page content located at the receiving server. However, once the content is loaded, the same origin policy prohibits the script from accessing it. Some of the desired features, such as positioning and reading user data, would be impossible to implement using this method.

To get around the same origin policy, the script must be downloaded from the very host accepting the local feature requests. This means that scripts must originate from the local server, or at least the browser must think they do. This can be accomplished by implementing a MIDlet working as a proxy, relaying content from the Web to the browser (see Figure 5.1). A normal proxy server is not treated as the origin of a page loaded through it. However, if the proxy server accepts requests as if it was the actual destination server, then the browser would treat it as the origin. A MIDlet was therefore implemented to accept both requests for local features and fetching of remote content. The real address of the content is simply added to the URL, as if it was the path of a file on the local server.

An HTTP request is made to the local IP address `127.0.0.1`, where the MIDlet is listening for connections, and the URL of the actual document is appended to the request Uniform Resource Identifier (URI). The complete URL of a request for a document `index.html` on the server `www.example.com` would look like this:

```
http://127.0.0.1/www.example.com/index.html
```

When the MIDlet receives this request, it extracts the correct address (`www.example.com/index.html`), downloads the file, and sends the response back to the client. As far as the browser knows, the document is downloaded from the local server on `127.0.0.1`.

A different solution would be to rewrite an open source proxy server and add the necessary features to it. To simplify development, the MIDlet could work as a regular proxy. Code could be added to intercept certain requests and handle them locally. In this solution the URL would not have to be changed. However, the user would have to change the browser's proxy settings. The problem with this solution, as with any adaptation of existing software for this purpose, is that there are no suitable open source applications written for Java ME. The Java SE based applications described in sections 3.7 and 3.8 were not used, since the work of porting them would exceed any possible gain in development speed.

5.3 Features

To distinguish requests for local device functionality from proxy requests, the URI of the former will begin with the word local (note that this must be a complete match to the first string of the URL and not simply a partial match, to avoid matching against a domain name such as `local.org`). A request URL for the function `some_function` would look like this:

```
http://127.0.0.1/local/some_function
```

To make the syntax more intuitive for programmers, the function names are formed using the same naming scheme as Java methods and can take zero or more arguments:

`someFunction()`, `someFunction(arg)`, or `someFunction(arg1,arg2)`, etc.

The MIDlet sends a valid HTTP response to every request, even if the response does not include any data. The data is sent in the form of JSON objects, which can easily be interpreted in JavaScript. Each response has a field called type, which determines the type of data enclosed in the JSON object. The data, which can be either a string message, a number, an array, or an object, is located in a field called value. If the MIDlet can not execute the request properly, the type field is set to 'error' and the value field to a string message explaining what went wrong.

5.3.1 Retrieving data

As a first trial to see if the concept would work in practice, a simple function was implemented to retrieve the phone's International Mobile Equipment Identity (IMEI) number. The IMEI number can be found in Java on the K800i by calling the

Java method `System.getProperty("com.sonyericsson.imei")`. This method returns the number as a string which was then sent back to the browser as a JSON string. After testing this function using a simple JavaScript, it was concluded that the concept worked. Other data, such as the current time on the phone, could similarly be retrieved. However, some of the potentially useful values such as battery level and network signal strength, are not available from within the Java ME environment. See [63] for a complete list of retrievable data.

5.3.2 Alerting the user

The K800i, like many other phones, can vibrate, play programmable melodies, and light up the screen with a back-light. If a web application had control of these features, it could for example be used to alert the user to certain events. JP-7 has specific methods for using these features. Playing sounds works fine, but using the vibrator and back-light does not work while the MIDlet is in the background. Sony Ericssons's solution to this involves using iMelody files. iMelody is a simple standardized tone sequence format, which supports both vibration and back-light [22]. Using the support for iMelody playback, functions were implemented to trigger vibration and back-light flashing, as well as simple sound and melodies. Also, because the melodies are defined in a simple string format, a function which takes a melody string as input was written. This allows web application developers to add dynamic tone sequence playback using JavaScript.

5.3.3 Taking pictures

The APIs included in the K800i's Java platform provide two ways of controlling the phone's built-in camera (only the backside camera, not the small one on the front). The MMAPI has methods for displaying streaming video from the camera. Snapshots can be saved from this stream. The snapshots are returned as byte arrays, which can be saved to local memory or processed by the MIDlet. Another way of utilizing the camera is provided by the Advanced Multimedia Supplements API. It provides additional control over camera settings and allows burst shooting. The pictures are saved directly as image files to a local storage folder, which must be selected before shooting.

Both approaches were attempted through HTTP requests to the MIDlet. By temporarily taking control of the screen, it was possible to display video from the viewfinder. Unfortunately neither of the snapshot methods worked properly while the phone's web browser was still running. The MMAPI method triggered the camera click sound, but no data was returned. The other method simply failed with the debug output 'STORAGE_ERROR'. The outcome was the same whether the picture was to be saved to phone memory or memory card (both had plenty of free space available at the time of testing).

Name	String	Description
Current Mobile Country Code	com.sonyericsson.net.cmcc	Unique country number
Current Mobile Network Code	com.sonyericsson.net.cmnc	Operator number
Location Area Code	com.sonyericsson.net.lac	Number of current area
Cell Identity	com.sonyericsson.net.cellid	Base station number

Table 5.1. Network properties that can be used for positioning purposes.

5.3.4 Audio capture

All mobile phones are equipped with a microphone for speech input when making calls. Using the MMAPI, it is possible to record the audio stream of the microphone. Functions have been implemented to handle recording, playback, and uploading of audio. This could for example be used to create an audio blog application.

5.3.5 Positioning

The Sony Ericsson HGE-100 is an external GPS receiver combined with a remote control for the phone's music player, a 3.5mm audio jack for stereo output, and an analog microphone to add hands-free capability. It is connected by proprietary cable to the phone and communicates with the phone through a virtual serial port. The output is, as with most GPS devices, in the form of NMEA 0183 sentences [39]. The HGE-100 outputs several sentences, but the most relevant ones are GGA and RMC (Recommended Minimum). Both these sentences can include latitude and longitude coordinates, but GGA can also provide additional information, such as altitude. The coordinates are only available in the World Geodetic System 1984 (WGS 84) [38] format.

The MIDlet uses a separate thread to handle the GPS receiver. While the receiver is connected, the thread reads the NMEA sentences at regular intervals. To reduce the MIDlet's work load when not being used, the data is not processed until actually needed. When a web application requests information about the current position, the most current data is parsed and the coordinates converted from degrees, minutes, and seconds to decimal degree format.

The K800i (and other Sony Ericsson phones with JP-7.3 or higher) support the extraction of certain properties of the wide area wireless network it is connected to. The properties are returned by calls to the method `System.getProperty()` with specific strings as input. The properties of interest to positioning and their respective input strings are shown in Table 5.1. (A complete list of all the properties which can be retrieved can be found at [63].) A combination of these values represent a specific base station of the connected network. By mapping the combined properties to geographic coordinates, a positioning system can be created. Such a system can be used in combination with GPS or separately when a GPS receiver is unavailable.

Several online services exist where users collaboratively identify the location of base stations to build databases for positioning without GPS. The locations can be identified manually, as on CellSpotting [12], or automatically, as on Navizon [34] where devices with GPS receivers combine network properties with GPS coordinates to update the database.

5.3.6 Local memory access

The PDA Optional Packages API [20] can be used to access local file systems as well as the address book and calendar on the device. On the K800i this means one can read and write to the phone's local storage and to inserted memory cards. Documents and other content can be served from the MIDlet (just as from a web server), media can be permanently stored on the phone, and web content downloaded through the MIDlet can be cached. The API also allows contacts to be retrieved or added to the address book, along with notes, URLs, pictures, and other data. The calendar can be viewed and edited as well. These functions can be used to synchronize contacts and events with web applications.

5.3.7 Wireless connectivity

Many mobile phones and other devices have the ability to establish local wireless connections. Using technology such as Bluetooth or IrDA, devices can be located, queried for information, and remotely controlled. Java ME has APIs (included in JP-7) for making Bluetooth connections (see [21] for details). These can be used on the K800i to search for devices and make use of any services they provide.

5.4 Control script

For a web application to make use of the MIDlet's features it can make HTTP requests using JavaScript. To facilitate application development, an example script with simple functions to send such requests, has been created. The script will also automatically detect if the MIDlet is running. If it is not currently running, then the script can start the installation procedure (see section 5.4.2).

5.4.1 Detection

Before allowing the application to make any function calls, the script has to make sure the MIDlet is active. As mentioned earlier, web pages that are to use the extra functionality must be loaded through the MIDlet. The script therefore checks for an active MIDlet, then redirects the client through its proxy function. This detection is performed by adding a hidden image to the page. The image is a single pixel GIF hard coded into the MIDlet and mapped to a special URI. If the image is loaded properly, then the MIDlet must be active, hence the client is redirected and the MIDlet's features become accessible.

5.4.2 Installation

If the test image is not loaded within a certain amount of time (currently set to three seconds), then detection has failed, and the user is given a choice. A confirm pop-up dialog is shown, where the user can choose to either continue using the application without extra features from the MIDlet, or download and install the MIDlet before continuing. If the user answers yes, then the JAD file is loaded in an iFrame on the page, which initiates the download and installation process. Once the installation process is over, then the script will try to detect the MIDlet again at regular intervals (currently set to five seconds). Once it is detected, the MIDlet's features will be available to the application.

5.5 Security

While advanced security schemes should definitely be used in a commercial application of this kind, some simple measures will suffice for the security of this prototype implementation. The first step in securing this setup is making sure the user can trust the MIDlet to handle security. This is done by signing the MIDlet with a certificate to identify who the authors are. A certificate was acquired from Thawte Consulting, which is an authority recognized by modern Sony Ericsson phones. The MIDlet was signed and can now be installed in the trusted third party domain. A trusted third party application can be traced back to its authors and has a less strict policy on acquiring permissions during execution.

All scripts which are able to use the MIDlet features must be loaded through the proxy function. Restricting the features to trusted applications is therefore simply a matter of filtering out untrustworthy proxy requests. When the MIDlet receives a request for content on a remote web server, it temporarily takes control of the screen and displays a confirmation dialog with four choices. The user can choose 'Yes', 'No', 'Always', or 'Never', and thereby allow or deny the request, applied only this one time or as a blanket decision. If the user selects 'Always' or 'Never', then the server address is stored by the MIDlet and any subsequent requests matching that address will automatically be treated the same way.

When a request for a new web application has been permitted, a random session key is created. The web browser is redirected to a new URI where this key is inserted before the remote address. When the browser sends a request with this new URI, the remote content will be downloaded and sent back to the browser. The complete URL of a trusted web application with the session key `abc123456789` would look like this:

```
http://127.0.0.1/abc123456789/www.trusteddomain.com/webapp.html
```

The session key must be used in any requests for local functions. If the key is missing or invalid, the request will be denied. To make sure the request is not coming from an unauthorized application, the referrer HTTP header field must be

included and contain a local server URL. This is ensured by the browser if the application is loaded through the MIDlet. Once the client has been redirected, the control script extracts the session key, and will automatically use it whenever the application requests local functions from the MIDlet. These requests will have URLs of the form:

```
http://127.0.0.1/abc123456789/local/someFunction()
```

5.6 Example applications

To show how the functions described in the earlier sections can be used in practice, two applications have been constructed. The first one is a plug-in that adds an extra feature to the mobile web site Hitta.se. The second one is a map application which makes use of the GPS positioning, audio recording, and alert functions.

5.6.1 The plug-in

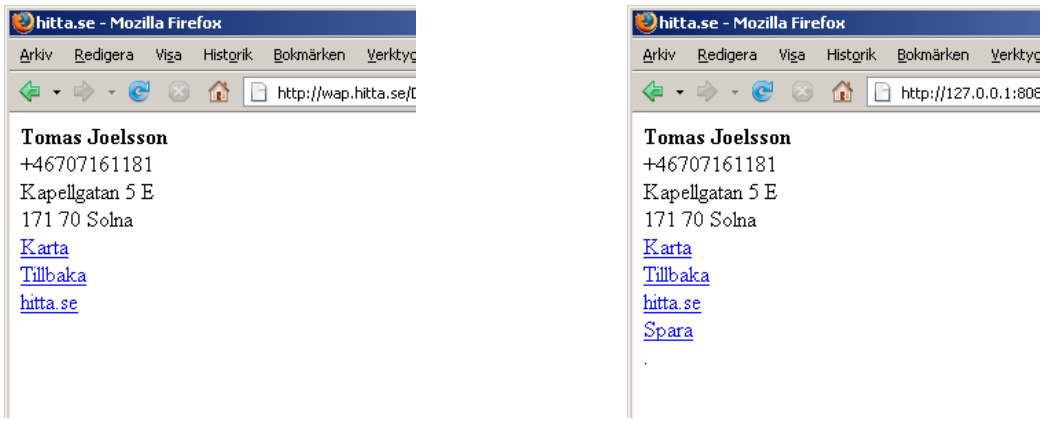
Hitta.se is an online directory of people and organizations in Sweden. Their mobile site (`http://wap.hitta.se/`) consists of a search engine where users can input names, numbers, and/or locations. The result is two lists of entries matching the input data, one for people and one for organizations. The data available in each entry varies, but usually includes one or two phone numbers and an address. If available, it also includes a link to a map centered at the given address, and a link to a photo of the building.

On a mobile phone, the site can be used to find phone numbers to be dialed or saved in the address book. Only the number can be copied though, not the name or any other information. WTAI provides a link tag for saving names and numbers. vCards[1] can be used to add numbers and detailed personal information to the address book. However, neither of these are available on the site.

Using the plug-in, one can easily add name, address, and multiple phone numbers to the phone's address book. This is done by including a script within the page which is returned by the local proxy to the web browser. This script adds a new link labeled "Spara" (to save) (see Figure 5.2), that when clicked, collects all the available information from the page. The script locates the URL of the map, if there is such a link, and can also save the address of a building's photo. All this information is then sent to the background MIDlet, which in turn, writes this information to the phone's address book.

As mentioned earlier, all applications that wish to make use of the MIDlet's functions must be loaded via the proxy. Since Hitta.se is an existing site which cannot be changed without the website operator's permission, no scripts have been

[1] vCard is a standard for electronic business cards. The cards can contain telephone numbers, email addresses, URLs, photographs, and much more. They can be distributed through any network connection and are supported by many handheld devices [73].



(a) Without MIDlet (b) With MIDlet

Figure 5.2. The mobile web site Hitta.se displaying details of a person. The MIDlet adds an additional link to the page. Clicking "Spara" will save the contact details to the phone's address book.

added to their website to automate this process. Therefore to use the plug-in's features, a client must access the site by following a link from another page, or manually input the URL including the proxy address. The full URL for accessing the site this way is:

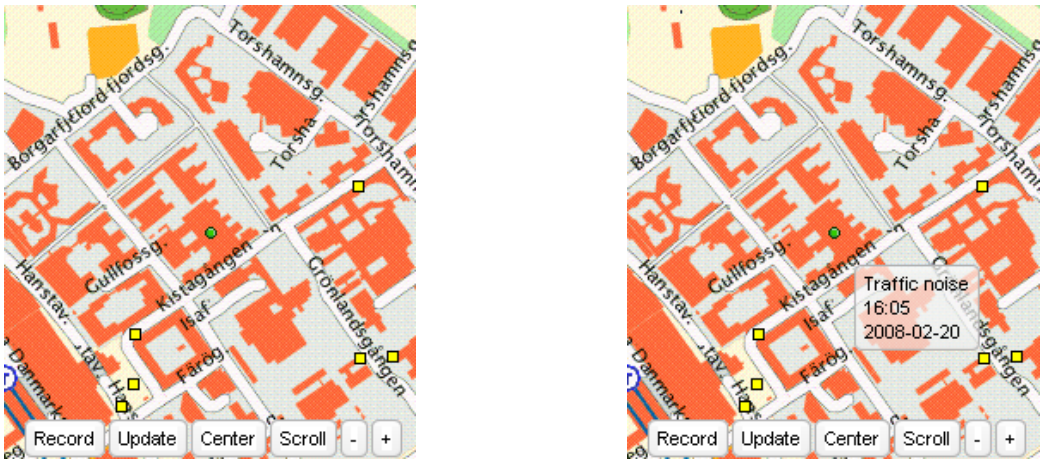
```
http://127.0.0.1/wap.hitta.se/
```

Once the site is loaded this way, the MIDlet can add scripts to appropriate documents by checking for the URI in a table of scripts which can be added on a site specific basis. In our experiments the MIDlet does not edit the documents before sending them to the browser, but simply appends extra script tags at the end. This approach worked well on the K800i running the NetFront browser, but other browsers might object to the incorrect XHTML syntax. Hence to avoid problems, documents should be edited to add the extra lines containing the new script within the head of the XHTML document.

The mapping of scripts to specific sites does not have to be predefined. The MIDlet can add script files with names based on hash values of the URL. The location of these files can be a central server or the referrer address supplied by the first page request. Script tags are added whether the files exist or not and it is up to the browser to load them properly if they are available.

5.6.2 The map application

There are several map applications available for mobile phones, but few of them are browser based. The ones that **are** browser based are often very simple. A major reason for this is the problem of obtaining positioning information from within the



(a) Map showing area with six uploaded audio clips.

(b) A clip has been selected and information about it is being displayed.

Figure 5.3. Two views of the map application.

browser. Today, standalone applications can utilize both internal and external GPS receivers, and are therefore better able to display maps and position them relative to the location of the user. Since this problem can be solved (as explained in section 5.3.5), there is little reason not to create browser based map services.

To show that there is a need for maps in browser based mobile applications, an example map application was built. It not only displays maps centered on the user's location by using GPS positioning, but also allows users to record audio clips that stick to their current location on the map. The audio clips are represented by small yellow boxes. In this example map application, other users viewing the same area may select this icon to hear the audio. If the application is left running, it will automatically update the user's position and check for new audio clips near their current position. If a new clip is found, it will trigger the phone's vibrator and start flashing the back-light to alert the user.

The clips are submitted, by the MIDlet, to a server through an HTTP POST request. Besides the audio file, the request also includes positioning information and a hash value of the phone's IMEI number or SIM subscriber number (if available), to distinguish between users. The server stores the information in a database, which can then be queried for clip information and the URL to the actual file hosted on the server.

The interface is quite simple (see Figure 5.3). After starting the application, the relevant map is automatically loaded and centered at the extracted location. If the MIDlet is not running or no positioning information is available, the map is centered at a default location. The map can also be centered from GET variables in the URI. If such variables are supplied, they will take priority over coordinates extracted from the device.

The user can scroll to view different locations using the phone's navigation keys.

To stop scrolling, one simply presses the middle button ('Select' or 'OK' on some phones). On the bottom of the screen are six buttons. One to enable scrolling, one to start recording, one for manually updating location and map content, one for centering on the user's current location, and two zoom buttons (marked as '+' and '-'). Using the navigation keys, the user can select these buttons or the currently visible audio clip icons. When the cursor is moved to an audio clip icon, a box is displayed with a description and the time and date of recording. To listen to the clip simply select it and the browser will use an appropriate plug-in for playback.

The map is built up from a set of equally sized tiles downloaded from a free mobile map service, such as Eniro (`http://wap.eniro.se/`) or Hitta.se (`http://wap.hitta.se/`). The size of the tiles is configurable and defines the distance the map will move at each scroll event. In addition to the tiles making up the currently visible map, to speed up scrolling the application downloads neighboring tiles bordering the map in each direction. The default scale of the map depends on whether GPS coordinates are available or not. If there are coordinates available, then the map will have a scale detailed enough to show small streets. If GPS data is not available, then the map will show a larger area. To view the map in a specific scale, the user can supply a GET variable in the URI that will overload the default scale variable.

Running the application without the MIDlet will still work. However, no audio can be submitted and the alerts are not available, but the user can still browse the map and listen to previously created audio clips. The application also works with PC browsers, where a larger map view can be displayed.

Chapter 6

Evaluation

The first project goal was to show how a background process could add functionality to mobile web applications. In the last chapter it was described in detail how this was accomplished on a Sony Ericsson K800i phone. The implemented background MIDlet functions as a proxy server for web applications and accepts HTTP requests from these applications to use local phone functions.

Though this worked without problems on the K800i, the concept makes certain demands upon a device it is to run on. The phone must have the ability to open local sockets, and the web browser has to be able to connect to the local IP address. The K800i and other similar devices that were released around the same time, and therefore running similar software, had no problems in doing either of these operations. However, more recent Sony Ericsson models with the Java Platform 8 standard, were unable to make local connections. Since this is a crucial requirement, the MIDlet is not useful on these models. The reason for this limitation is unknown. It could be caused by a bug in the software or the manufacturer could have done it intentionally, to improve performance or to increase security. Unfortunately, it is possible the same limitation will be present in all Sony Ericsson devices released in the near future; hence the method used in this thesis project can not be used with these devices!

During the start of the implementation phase, a choice had to be made whether to start from scratch or build the system on top of existing software. Three proxy servers written in Java were considered. The problem with these applications is that they were not written to run on mobile devices. They were all written for Java SE, which (as explained in section 2.3) is quite different from Java ME, especially when it comes to networking. An HTTP proxy is a network application and requires a fair amount of string manipulation, both of which must be handled differently in CLDC applications. Additionally, the RabbIT and PAW proxy implementations both use many Java SE classes for advanced data structures. PAW also depends on several external packages, which if included in a JAR, would increase the total application size. The Super Proxy System is comparatively simple in its design and does not include many non-Java ME classes. It could be ported to run on a mobile phone

without too much work. However, the proxy function is not actually a very large part of the MIDlet. In the finished implementation, only 27.4KB of code (three class files) is used for the proxy functionality. This can be compared to the 56KB of proxy code in the Super Proxy System. The major part of the implementation consisted of adding methods for the specific services. This code amounts to more than double that of the proxy function (63.6KB). Consequently, rewriting a Java SE proxy application would not speed up the development process significantly. There is also a major difference in total application size. The compiled and packaged MIDlet JAR is only 74.8KB, compared to the Super Proxy System's 149KB JAR file.

Running a Java EE server on a mobile phone would enable a lot of additional functionality. The MIDlet could have been implemented on top of an application server, providing the proxy functionality through a Servlet. It would then be possible to use a framework such as JSON-RPC-Java to facilitate communication between the MIDlet and JavaScript in the browser. That is if one could somehow implement reflection for CLDC. GlassFish v3 might be small enough to fit on a mobile device. However, it is still based on the APIs of Java SE, including data structures and networking, which would have to be changed if ported to CLDC. Even if such a porting was done, the proxy functionality would still have to be added. Thus, the gain of building the system on top of GlassFish would be even smaller than if using a Java SE proxy application as base.

Nokia's MWS has a lot of the same functionality that this thesis aims to develop. However, the purpose of the application is somewhat different. While the Nokia server makes local device features accessible to remote clients, this project aims to give greater access to these local devices to the *local user of the device*. Despite this difference, the approach of using web browsers to increase functionality and versatility in mobile devices is much the same. The core of the MWS is built using open source software. This part could be reused to create a background process similar to the developed MIDlet, but for phones running Symbian. The K800i does not run Symbian but many other phone models do. This approach has not be tried, but it is possible Symbian devices would not have the same problem of accessing the built-in camera while running a web browser. Even if that is not the case, a Symbian application would be a viable alternative to the implementation in this thesis.

While it was shown that having a process running in the background while browsing can add functions to web applications, this will also increase CPU usage. If it affects the browser and makes web applications run slower, this might not be a very practical solution. Interactivity in advanced web applications is often based on client side scripts. Therefore it is important that script execution is not affected too much by the background MIDlet. This would decrease the perceived speed of mobile web applications, which are generally slower than fixed computer web applications. A JavaScript benchmark was therefore used to measure the impact of the MIDlet on local script execution speeds.

The SunSpider JavaScript benchmark tests a lot of common algorithms used in

today's web applications. It was not made for testing mobile browsers, but with a few simple changes (because of a difference in the interpretation of for loops) it could be used with NetFront on the K800i. Unfortunately, two of the tests ('nsieve' and 'tagcloud') had to be removed because they caused the browser to freeze up. These tests are both memory intensive and were therefore probably not well suited to run on a device with limited memory. The overall performance was a lot slower than on a modern PC (see Table 6.1). It was obvious the phone would be slower, but the difference was bigger than expected. This means converting existing web applications for fixed computers to run on mobile phones might not always be feasible. The benchmark could still be used to evaluate the MIDlet's impact, by running the benchmark on the same device and browser, once with and once without the MIDlet.

SunSpider has a function to compare two benchmark results and show where there are significant differences. After running the benchmark and comparing the two results it seems there is only a small difference in performance. Some of the tests were actually completed faster with the MIDlet running, which indicates that the JavaScript execution was not constantly affected by the JVM. Each test was performed five times to get a higher accuracy. If a test is repeated, SunSpider will calculate a 95% confidence interval. By looking at these intervals it would seem that the MIDlet causes variation in the execution speed. This means that some of the test runs were affected more by the JVM than others. Since the MIDlet affected the web browser only at certain times and the impact was different between runs of the same test, it seems the JVM only slows performance when the MIDlet is actively doing something. If this is the case, then the regular activities of the MIDlet (i.e. the regular polling for a connected GPS receiver) is what caused the drop in execution speed. See Table 6.2 for the complete output of the benchmark comparison.

The second project goal was to implement routines for utilizing local functionality and use these routines in actual web applications. The intention was to test which local devices and data could be used from the mobile browser environment on a K800i phone. Most of the implemented features (as described in section 5.3) worked very well. The only major problem was the digital camera. Originally the camera function was meant to be part of the map application, but since it did not work, it was replaced by audio recording. The reason why the phone would not allow pictures to be taken by the MIDlet is still not known. It is clear though, that it was not the application itself, but rather the browser being active which caused the problem. Running the same code without the browser worked fine. Since it is possible to display the viewfinder without exiting the browser, it seems that the browser is blocking some resource needed by the camera specifically for saving high resolution images. At the time of writing, Sony Ericsson has not given any hint on the reason behind this limitation.

Together, the two example applications utilize most of the implemented features. The plug-in writes to both the phone's address book and a local file system. The map application makes use of positioning, audio recording, and alert functions. Both examples show how the background MIDlet can improve the functionality

TEST		COMPARISON	FROM (PC)	TO (K800i)	DETAILS
** TOTAL **:		75500% slower	13319.0ms +/- 1.4%	10055514.6ms +/- 0.1%	significant
3d:		30530% slower	1931.4ms +/- 4.4%	589684.4ms +/- 0.8%	significant
	cube:	28200% slower	506.2ms +/- 8.8%	142750.0ms +/- 0.7%	significant
	morph:	23230% slower	1072.2ms +/- 2.1%	249100.6ms +/- 1.6%	significant
	raytrace:	56040% slower	353.0ms +/- 13.8%	197833.8ms +/- 0.9%	significant
access:		54810% slower	1106.2ms +/- 8.2%	606283.6ms +/- 0.6%	significant
	binary-trees:	34570% slower	181.2ms +/- 38.3%	62646.0ms +/- 1.1%	significant
	fannkuch:	106190% slower	393.8ms +/- 5.4%	418171.8ms +/- 0.8%	significant
	nbody:	23620% slower	531.2ms +/- 16.0%	125465.8ms +/- 0.3%	significant
bitops:		19990% slower	3109.0ms +/- 1.6%	621595.6ms +/- 0.5%	significant
	3bit-bits-in-byte:	46990% slower	246.8ms +/- 3.6%	115983.4ms +/- 0.8%	significant
	bits-in-byte:	73820% slower	234.0ms +/- 0.0%	172740.2ms +/- 0.4%	significant
	bitwise-and:	5930% slower	2365.8ms +/- 1.9%	140329.0ms +/- 1.1%	significant
	nsieve-bits:	73380% slower	262.4ms +/- 12.1%	192543.0ms +/- 2.1%	significant
controlflow:		69740% slower	128.2ms +/- 6.9%	89403.8ms +/- 1.3%	significant
	recursive:	69740% slower	128.2ms +/- 6.9%	89403.8ms +/- 1.3%	significant
crypto:		118710% slower	678.0ms +/- 6.7%	804836.8ms +/- 0.5%	significant
	aes:	52460% slower	243.8ms +/- 28.6%	127895.2ms +/- 2.7%	significant
	md5:	245030% slower	181.2ms +/- 5.8%	444001.6ms +/- 0.0%	significant
	sha1:	92070% slower	253.0ms +/- 18.4%	232940.0ms +/- 0.3%	significant
date:		7140% slower	2462.6ms +/- 1.5%	175769.4ms +/- 0.4%	significant
	format-tofte:	8130% slower	1168.8ms +/- 3.0%	95069.2ms +/- 0.3%	significant
	format-xparb:	6240% slower	1293.8ms +/- 0.6%	80700.2ms +/- 0.6%	significant
math:		31000% slower	1250.2ms +/- 2.4%	387506.4ms +/- 0.5%	significant
	cordic:	36860% slower	575.2ms +/- 2.9%	211990.2ms +/- 0.4%	significant
	partial-sums:	20710% slower	393.6ms +/- 6.4%	81526.8ms +/- 1.6%	significant
	spectral-norm:	33400% slower	281.4ms +/- 0.2%	93989.4ms +/- 0.4%	significant
regexp:		18250% slower	650.0ms +/- 1.7%	118601.6ms +/- 0.2%	significant
	dna:	18250% slower	650.0ms +/- 1.7%	118601.6ms +/- 0.2%	significant
string:		332530% slower	2003.4ms +/- 4.7%	6661833.0ms +/- 0.1%	significant
	base64:	279890% slower	631.2ms +/- 14.6%	1766678.0ms +/- 0.2%	significant
	fasta:	27690% slower	447.0ms +/- 10.0%	123754.0ms +/- 0.5%	significant
	unpack-code:	769790% slower	569.0ms +/- 1.9%	4380114.2ms +/- 0.1%	significant
	validate-input:	109850% slower	356.2ms +/- 12.4%	391286.8ms +/- 0.2%	significant

Table 6.1. SunSpider JavaScript benchmark comparison between an HP Workstation xw4400 running Firefox under Windows 2000 and a K800i running NetFront. Each test was run five times and the values are means with 95% confidence intervals. The word significant in the details column means that the difference between the test results was statistically significant.

TEST		COMPARISON	FROM (with MIDlet)	TO (without MIDlet)	DETAILS
** TOTAL **:		<0,5% difference	10102263.0ms +/- 0.4%	10055514.6ms +/- 0.1%	significant
3d:		2% faster	601494.0ms +/- 0.9%	589684.4ms +/- 0.8%	significant
	cube:	3% faster	146746.0ms +/- 1.3%	142750.0ms +/- 0.7%	significant
	morph:	1% faster	252116.6ms +/- 0.3%	249100.6ms +/- 1.6%	significant
	raytrace:	2% faster	202631.4ms +/- 1.7%	197833.8ms +/- 0.9%	significant
access:		3% faster	627207.4ms +/- 1.9%	606283.6ms +/- 0.6%	significant
	binary-trees:	3% faster	64257.2ms +/- 0.5%	62646.0ms +/- 1.1%	significant
	fannkuch:	4% faster	435797.6ms +/- 2.3%	418171.8ms +/- 0.8%	significant
	nbody:	-	127152.6ms +/- 1.7%	125465.8ms +/- 0.3%	
bitops:		-	629266.6ms +/- 1.4%	621595.6ms +/- 0.5%	
	3bit-bits-in-byte:	-	116489.0ms +/- 1.4%	115983.4ms +/- 0.8%	
	bits-in-byte:	2% faster	175805.2ms +/- 1.6%	172740.2ms +/- 0.4%	significant
	bitwise-and:	-	142032.8ms +/- 2.0%	140329.0ms +/- 1.1%	
	nsieve-bits:	-	194939.6ms +/- 2.0%	192543.0ms +/- 2.1%	
controlflow:		3% faster	91865.2ms +/- 2.4%	89403.8ms +/- 1.3%	significant
	recursive:	3% faster	91865.2ms +/- 2.4%	89403.8ms +/- 1.3%	significant
crypto:		1% faster	814304.4ms +/- 0.8%	804836.8ms +/- 0.5%	significant
	aes:	3% faster	131289.0ms +/- 2.0%	127895.2ms +/- 2.7%	significant
	md5:	1% faster	449537.2ms +/- 0.9%	444001.6ms +/- 0.0%	significant
	sha1:	-	233478.2ms +/- 0.4%	232940.0ms +/- 0.3%	
date:		3% faster	180843.8ms +/- 1.0%	175769.4ms +/- 0.4%	significant
	format-tofte:	4% faster	98821.4ms +/- 1.0%	95069.2ms +/- 0.3%	significant
	format-xparb:	2% faster	82022.4ms +/- 1.5%	80700.2ms +/- 0.6%	significant
math:		-	389249.8ms +/- 1.1%	387506.4ms +/- 0.5%	
	cordic:	-	214245.2ms +/- 1.2%	211990.2ms +/- 0.4%	
	partial-sums:	2% slower	79782.2ms +/- 1.5%	81526.8ms +/- 1.6%	significant
	spectral-norm:	1% faster	95222.4ms +/- 0.6%	93989.4ms +/- 0.4%	significant
regexp:		<0,5% difference	118208.8ms +/- 0.2%	118601.6ms +/- 0.2%	significant
	dna:	<0,5% difference	118208.8ms +/- 0.2%	118601.6ms +/- 0.2%	significant
string:		<0,5% difference	6649823.0ms +/- 0.1%	6661833.0ms +/- 0.1%	significant
	base64:	-	1765249.0ms +/- 0.1%	1766678.0ms +/- 0.2%	
	fasta:	4% slower	118870.6ms +/- 1.4%	123754.0ms +/- 0.5%	significant
	unpack-code:	-	4379268.8ms +/- 0.1%	4380114.2ms +/- 0.1%	
	validate-input:	1% slower	386434.6ms +/- 0.4%	391286.8ms +/- 0.2%	significant

Table 6.2. SunSpider JavaScript benchmark comparison between two runs on a K800i running NetFront. First column values are from testing with the MIDlet running in the background. The word significant in the details column means that the difference between the test results was statistically significant. Therefore a comparison of the results is available for this test.

of online services that are otherwise limited by the web browser. Other currently available browser based solutions for accessing local devices completely lack support for utilizing cameras, audio capture, and non application specific local data (i.e. address book, calendar and local file systems). Google Gears gives web applications additional functionality, but currently only persistent storage of application data and caching content for offline usage. Nokia's Web Run-Time adds a few new features and is clearly a step towards opening up local APIs. However, it still lacks methods for accessing most local devices. Positioning within the browser environment is currently only implemented in one mobile browser (EZweb), which is limited to the Japanese market. Hence, utilizing local device functionality still requires writing separate applications. Building web applications in Java or Symbian does not have the same flexibility as browser based applications. User interfaces and integration with other web applications and services are easier to develop using standard web technologies. Ajax for Java ME provides the means to create MIDlets utilizing web services to create rich web applications, with the ability to incorporate local functionality. However, the existing Java ME APIs for rendering markup are not nearly as comprehensive as today's mobile browsers. Therefore creating user interfaces in Java ME is still not as practical as browser based web development.

The third and final goal of the project was to find a suitable security model. Such a model should ensure the user's privacy and help keep malicious software from accessing local functionality. The security measures implemented here rely on the user to distinguish safe applications and to decide who is allowed to read personal data. Before starting a new web application the user must answer a question of whether this application will be allowed to use the MIDlet's proxy function or not. Only documents downloaded through the proxy are allowed to access local functionality. This is ensured by the browser through the same origin policy. The MIDlet will not accept requests from remote hosts. This will prohibit external access through a network connection. The only way a client other than the browser would be able to utilize the MIDlet's features is if it was a locally executing process. If such a client existed and wanted to request a function from the MIDlet, it would still need a valid session key. Session keys are only given out after receiving permission from the user. A possible security threat could occur if an untrusted application managed to steal a session key from an authorized web application. The key could be stolen if an authorized web application includes external links. The target server would then be able to extract the session key from the referrer field in the HTTP request header. It is therefore important to ensure that trusted applications do not contain such links (it should be mentioned that the browser would not include the referrer field if the local server required the use of secure HTTP connections). If an untrusted application is browser based, even with the session key, it would not be able to access the MIDlet's features. The referrer header field, which can not be changed by a script, would show that the application was not loaded through the MIDlet. The only time the referrer field is not checked is when a web application is requesting authorization, before attaining a session key. Nokia is the one device

manufacturer which has shipped phone software with several new JavaScript APIs for local device functionality (as described in section 3.2). However, Nokia has not implemented access to functions with any major security or privacy concerns. By skipping such functions, they could add new APIs without adding an additional security layer.

Chapter 7

Conclusions and future work

Adding local device functionality to mobile web applications would enrich the online experience of mobile phone users. The currently available alternatives are either browser based with very limited local device access or separate applications with specific individual purposes. The background MIDlet facilitating additional access from the browser environment creates a middle ground, with benefits from both sides. Web applications can utilize standard web technologies and also access local Java APIs. The main problem with allowing local device APIs to be called from scripts executing in the mobile browser, is maintaining an acceptable level of security and privacy. The background MIDlet allows or disallows requests for local functionality based on the user's decisions. This might not be an ideal solution, since it is impossible to know if the user actually reads the questions and if the user knows which web applications to trust.

In an extended implementation, one could imagine giving the user more specific information about which local features a web application needs to use. Currently, the user is only told that an application wants to be loaded through the MIDlet's proxy function to make use of its extended features. The user could be notified of each local feature that is requested. However, this could be a lot of information and the user might choose to ignore it. The security approach in section 1.4 suggests sorting local features into different risk levels. This is yet to be implemented, but could help simplify usage while still keeping the user informed of the possible security risks. By associating each MIDlet feature with a specific level and having web applications request access to a certain level, the user would have a good overview of the implications. For example, one could distinguish low risk features such as light and sound alerts, from high risk features such as audio capture and access to personal data. If access with a low security level was requested, the user would know that only low risk features are utilized by the particular application and so on.

Most mobile browsers have some sort of function for special rendering of web pages, bypassing the original style information. These functions are included for the purpose of displaying information properly on small screens. In reality though,

web designers want full control over page layout, and these functions cause a lot of problems. The NetFront browser on the K800i has such a function, called Smart-Fit, which is enabled by default. If a web application interface depends on absolute positioning of elements, Smart-Fit will cause the browser to ignore this and render the elements differently than intended by the author. This means the user has to change a setting in the browser options before using the application. To get full use of the MIDlet, the user also needs to grant permissions. Access to several restricted APIs is required. The MIDlet obviously needs to accept network connections and download from the Internet, but some of the features also need other permissions. Adding contacts requires access to the Personal Information Manager (PIM) database. To include a contact photo, writing permission for file storage is also needed, because the K800i's PIM only allows locally stored photos to be added. Recording audio needs explicit permission and saving recorded media would require file writing. The HGE-100 external GPS receiver communicates with the MIDlet through a serial connection, which requires an additional permission. All these have to be acquired by the user for any web application to fully utilize the MIDlet's features. This, along with the MIDlet's own security model, means that the user will have to go through a whole process of questions and settings before using local functionality in web applications. It might even add up to the same amount of work required to install a separate application. However, if the MIDlet is installed, settings are made, and permissions permanently set, then the MIDlet will work almost seamlessly in the background while facilitating access to any number of mobile web applications.

Each function that is made available by the MIDlet requires an explicit implementation in Java. Even if the function is an existing Java method there still needs to be a mapping from received HTTP requests. If the mapping could be made dynamically, then the MIDlet would become much more generic. Currently, there is no support for reflection or any kind of dynamic class loading in CLDC, which could otherwise have been used to facilitate calling Java methods through HTTP requests from a web application. Local Java classes could have been utilized from JavaScript executed in the browser. In future versions of Java for mobile phones, this might become possible. A more versatile background MIDlet could then be created, which was less device dependent and left more of the application logic to the actual web application. Even while such capabilities are not supported by the Java platform, this functionality could be added through static method mapping. Code can be generated by parsing available APIs for class names, method names, and signatures. Another improvement, making the MIDlet more generic, would be to standardize the positioning interface. The interface could implement the API proposed by LocationAware (see section 3.5.1), thereby improving the chances of compatibility with future location-aware web applications.

Currently the MIDlet starts automatically after booting the device. This might not be desirable, since it will probably not be used all the time. Using the push registry, it is possible to let MIDlets start from different remote events, such as socket connections and SMS messages. If such an event was triggered by a web

application, the MIDlet could be started automatically when needed. The web application could tell the device to load the MIDlet whenever the user requests a certain service. Sony Ericsson's push registry implementation supports remote starting, but only through SMS messages. A web server could send an SMS message triggering the phone to start the background MIDlet, but to do that, the web server must somehow get hold of the phone number. The number could be sent to the server when starting the application, or if the user is a registered member of the site, it could be associated with the user name and loaded when logging in. If the MIDlet could be started when needed, it could also shut down when it was no longer being used. This can be implemented by listening for an unload event that is triggered when the user leaves the web page or exits the browser. When the unload event occurs, the application sends one last request to the MIDlet telling it to shut down. The MIDlet could also have a timeout which initiates a shut down after a certain period of idle time.

The work done in this project can be continued in a number of directions. There are several questions left unanswered. For example, the concept was only tested on a few different phone models. It did not work on newer Sony Ericsson phones. However, this could possibly be solved by implementing the MIDlet as a real HTTP proxy server. Another area which was not looked into is whether the system could be implemented as a Symbian application. Such an application might not have the same problem of utilizing the built-in camera, as the MIDlet did. Another subject arose from the results of the JavaScript benchmarks. NetFront performed very poorly in these tests, so it would be interesting to see if performance could somehow be improved through the use of a proxy server. It is possible that a Java or Symbian application would execute the calculations faster than a mobile browser. Executing part of the code on a remote server might also have an impact on the overall performance. Thus, code mobility is a subject for further study.

References

[1] Access Co., Ltd. *NetFront Browser v3.5 Mobile Profile*, 2007. http://www.access-company.com/PDF/NFv35_2007.pdf.

[2] Boonlit Adipat and Dongsong Zhang. Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. *International Journal of Human-Computer Interaction*, 18(3):293–308, 2005.

[3] Adobe. *ActionScript Technology Center*. http://www.adobe.com/devnet/actionscript/, Last accessed 2007-10-17.

[4] Adobe. *Adobe Flash Player*. http://www.adobe.com/products/flashplayer/, Last accessed 2007-10-09.

[5] Adobe. *Adobe Shockwave Player*. http://www.adobe.com/products/shockwaveplayer/, Last accessed 2007-10-09.

[6] Adobe. *Flash Player Adoption Statistics*, Dec 2007. http://www.adobe.com/products/player_census/flashplayer/, Last accessed 2007-10-09.

[7] Adobe. *Shockwave Player Adoption Statistics*, Dec 2007. http://www.adobe.com/products/player_census/shockwaveplayer/, Last accessed 2007-10-09.

[8] The Apache Software Foundation. *The Apache HTTP Server Project*. http://httpd.apache.org/, Last accessed 2008-03-28.

[9] Akhil Arora and Vincent Hardy. *Mobile Ajax for Java ME Technology*. Sun Microsystems, 2007. http://www.w3.org/2007/06/mobile-ajax/papers/sun.hardy.mobileAjaxJavaME.pdf.

[10] Simonetta Balsamo and Moreno Marzolla. Towards Performance Evaluation of Mobile Systems in UML. In *Proceedings of ESMc'03 Conference*, pages 61–68. Dipartimento di Informatica Universita Ca Foscari di Venezia, Oct 2003.

[11] Bert Bos. *Cascading Style Sheets*. World Wide Web Consortium. http://www.w3.org/Style/CSS, Last accessed 2007-10-18.

[12] CellSpotting.com. *CellSpotting*. http://www.cellspotting.com/, Last accessed 2008-03-04.

[13] Ecma International. *ECMAScript 3rd Edition Compact Profile*, 2001. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-327.pdf.

[14] Google. *Google Gears API*. http://code.google.com/apis/gears/, Last accessed 2008-04-09.

[15] Vincenzo Grassi and Raffaela Mirandola. PRIMAmob-UML: A Methodology for Performance Analysis of Mobile Software Architectures. In *Proceedings of WOSP'02*, pages 262–274. Dipartimento di Informatica Sistemi e Produzione, Universita di Roma "Torvergata", July 2002.

[16] JSR 118 Expert Group. *JSR 118: Mobile Information Device Profile 2.0*. Java Community Process, 2006. http://jcp.org/en/jsr/detail?id=118, Last accessed 2007-10-09.

[17] JSR 185 Expert Group. *JSR 185: Java Technology for the Wireless Industry*. Java Community Process, 2003. http://jcp.org/en/jsr/detail?id=185, Last accessed 2007-11-09.

[18] JSR 287 Expert Group. *JSR 287: Scalable 2D Vector Graphics API 2.0 for Java ME*. Java Community Process. http://www.jcp.org/en/jsr/detail?id=287, Last accessed 2008-02-28.

[19] JSR 290 Expert Group. *JSR 290: Java Language & XML User Interface Markup Integration*. Java Community Process. http://www.jcp.org/en/jsr/detail?id=290, Last accessed 2008-02-28.

[20] JSR 75 Expert Group. *JSR 75: PDA Optional Packages for the J2ME Platform*. Java Community Process, 2004. http://www.jcp.org/en/jsr/detail?id=75, Last accessed 2008-03-25.

[21] JSR 82 Expert Group. *JSR 82: Java APIs for Bluetooth*. Java Community Process, 2006. http://www.jcp.org/en/jsr/detail?id=82, Last accessed 2008-03-25.

[22] Infrared Data Association. *iMelody V1.2 Approved*, Oct 2000. http://developer.sonyericsson.com/getDocument.do?docId=65053.

[23] Java Community Process. *Community Development of Java Technology Specifications*. http://www.jcp.org/, Last accessed 2007-11-07.

[24] JSON-RPC.ORG. *JSON-RPC Specifications*, 2005. http://json-rpc.org/wiki/specification, Last accessed 2007-10-11.

[25] Tomihisa Kamada. *Compact HTML for Small Information Appliances*. Access Co., Ltd., Feb 1998. http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/.

[26] KDDI Corporation. *EZweb*. http://www.au.kddi.com/ezfactory/tec/spec/eznavi.html, Last accessed 2008-03-10.

[27] Jonathan Knudsen. *Understanding JSR 185*. Sun Microsystems, May 2003. http://developers.sun.com/mobility/midp/articles/jtwi/.

[28] Sing Li and Jonathan Knudsen. *Beginning J2ME Platform, From Novice to Professional*. Apress, third edition, 2005.

[29] Location Aware Working Group. *LocationAware.org*. http://www.locationaware.org/, Last accessed 2008-02-11.

[30] Antinisca Di Marco and Cecilia Mascolo. Performance Analysis and Prediction of Physically Mobile Systems. In *Proceedings of 6th International Workshop on Software and Performance*. ACM Press, Feb 2007.

[31] David Mazières, Helen Nissenbaum, and Zhao Xiaojian. *Super Proxy System*. http://www.scs.cs.nyu.edu/webbug/, Last accessed 2008-03-26.

[32] Glen McCluskey. *Using Java Reflection*. Sun Microsystems, Jan 1998. http://java.sun.com/developer/technicalArticles/ALT/Reflection/.

[33] Metaparadigm Pte Ltd. *JSON-RPC-Java - JavaScript to Java AJAX communications library*. http://oss.metaparadigm.com/jsonrpc/, Last accessed 2007-10-11.

[34] Mexens LLC. *Navizon - Virtual GPS for mobile devices and laptop computers*. http://www.navizon.com/, Last accessed 2008-03-04.

[35] Microsoft. *Introduction to ActiveX Controls*. http://msdn2.microsoft.com/en-us/library/aa751972.aspx, Last accessed 2007-10-09.

[36] Mozilla. *Minimo project*. http://www.mozilla.org/projects/minimo/, Last accessed 2007-10-10.

[37] Mozilla. *Gecko Plugin API Reference:Scripting plugins*, Dec 2007. http://developer.mozilla.org/en/docs/Gecko_Plugin_API_Reference:Scripting_plugins, Last accessed 2008-01-29.

[38] National Imagery and Mapping Agency. *Department of Defense World Geodetic System 1984*, Jan 2000. http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf.

[39] National Marine Electronics Association. *NMEA 0183 Standard*, 2003. http://www.nmea.org/pub/0183/index.html, Last accessed 2008-03-03.

[40] Nokia. *Mobile Web Server*. http://www.forum.nokia.com/main/resources/technologies/mobile_web_server/index.html, Last accessed 2007-10-09.

[41] Nokia. *Mobile Web Server: Mobile Environment Demands*, May 2007. http://sw.nokia.com/id/e3a333a0-9ac8-4a8b-aea0-ec3ceb21c443/Mobile_Web_Server_Mobile_Environment_Demands_v1_0_en.pdf.

[42] Nokia. *Web Run-Time API Reference, Version 1.1*, Nov 2007. http://sw.nokia.com/id/b9ad2b23-07ea-46cd-badf-f0ba3df97da3/Web_Run_Time_API_Reference_v1_1_en.pdf.

[43] NTT DoCoMo, Inc. *i-mode service guideline, Version 1.2.0*, 2002. http://www.nttdocomo.com/binary/technologies/imodetechnology_guideline2002030.pdf.

[44] Robert Olofsson. *RabbIT web proxy*. http://rabbit-proxy.sourceforge.net/, Last accessed 2008-03-26.

[45] Open Mobile Alliance. *ECMAScript Mobile Profile, A Wireless Markup Scripting Language*, 2004. http://www.openmobilealliance.org/release_program/docs/Browsing/V2_1-20040816-C/OMA-WAP-ESMP-V1_0-20040709-C.pdf.

[46] Opera Software. *Opera Mobile*. http://www.opera.com/products/mobile/, Last accessed 2007-10-11.

[47] Opera Software. *Products featuring the Opera Mobile Browser*. http://www.opera.com/products/mobile/products/, Last accessed 2007-10-11.

[48] C. Enrique Ortiz. *Introduction to OTA Application Provisioning*. Sun Microsystems, Nov 2002. http://developers.sun.com/mobility/midp/articles/ota/.

[49] Enrique Ortiz. *The MIDP 2.0 Push Registry*. Sun Microsystems, Jan 2003. http://developers.sun.com/mobility/midp/articles/pushreg/.

[50] OxfordU.net. *Introduction to DOM, by Dr Sam*, 2003. http://www.oxfordu.net/webdesign/dom/straight_text.html, Last accessed 2008-02-15.

[51] Pantechnicon. *HyperCard FAQ039*, 2005. http://pan.uqam.ca/pan/pmwiki.php/HyperCard/FAQ039, Last accessed 2007-10-17.

[52] PAW Project. *PAW Web Filter*. http://paw-project.sourceforge.net/, Last accessed 2008-03-26.

[53] Eduardo Pelegri-Llopart, Yutaka Yoshida, and Alexis Moussine-Pouchkine. *The GlassFish Community Delivering a Java EE Application Server*. Sun Microsystems, 2007. https://glassfish.dev.java.net/faq/v2/GlassFishOverview.pdf.

[54] Jon Perry. *DHTML Part 3: Browser Object Model*. http://www.webdevelopersjournal.com/articles/dhtml3/dhtml3.html, Last accessed 2008-02-15.

[55] Mark Pilgrim. *Dive Into Greasemonkey*, 2005. http://diveintogreasemonkey.org/, Last accessed 2007-10-10.

[56] Python Software Foundation. *Python Programming Language – Official Website*. http://www.python.org/, Last accessed 2008-03-28.

[57] J. Rosenberg, R. Mahy, and C. Huitema. *Traversal Using Relay NAT (TURN) draft*. Internet Engineering Task Force (IETF), 2005. http://tools.ietf.org/html/draft-rosenberg-midcom-turn-08.

[58] Jesse Ruderman. *JavaScript Security: The Same Origin Policy*. Mozilla, 2001. http://www.mozilla.org/projects/security/components/same-origin.html, Last modified 2001-08-24.

[59] Keldon Rush. *Getting started with Adobe Flash Lite*. Smashing Ideas, 2007. http://www.adobe.com/devnet/devices/articles/getting_started_flashlite.pdf.

[60] Gustav Söderström. Virtual networks in the cellular domain. Master's thesis, Department of Microelectronics and Information Technology (IMIT) Royal Institute of Technology (KTH) Kista, SWEDEN, Feb 2003. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/030211-Gustav_Soderstrom.pdf.

[61] Sony Ericsson. *K800*, 2006. http://developer.sonyericsson.com/site/global/products/phonegallery/k800/p_k800.jsp, Last accessed 2008-01-09.

[62] Sony Ericsson. *New Java Platform 7 (JP-7)*, March 2006. http://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsmar06/p_new_javaplatform7.jsp, Last accessed 2008-01-09.

[63] Sony Ericsson. *Developers' Guidelines Java Platform, Micro Edition, CLDC - MIDP 2*, Nov 2007. http://developer.sonyericsson.com/getDocument.do?docId=65067.

[64] Maciej Stachowiak. *Announcing SunSpider 0.9*. WebKit Open Source Project, Dec 2007. http://webkit.org/blog/152/announcing-sunspider-09/, Last accessed 2008-01-25.

[65] David Storey. *Evolving the Internet on your phone: Designing web sites with Opera Mini 4 in mind*. Opera Software, Aug 2007. http://dev.opera.com/articles/view/evolving-the-internet-on-your-phone-des/.

[66] Sun Microsystems. *Code Samples and Apps: Applets*. http://java.sun.com/applets/, Last accessed 2007-10-09.

[67] Sun Microsystems. *Desktop Java: Java Web Start Technology*. http://java.sun.com/products/javawebstart/index.jsp, Last accessed 2007-10-09.

[68] Sun Microsystems. *Java EE at a Glance*. http://java.sun.com/javaee/index.jsp, Last accessed 2007-11-08.

[69] Sun Microsystems. *The Java ME Platform: the Most Ubiquitous Application Platform for Mobile Devices*. http://java.sun.com/javame/index.jsp, Last accessed 2007-11-07.

[70] Sun Microsystems. *Simplified Guide to the Java 2 Platform, Enterprise Edition*, Sept 1999. http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf.

[71] Sun Microsystems. *Mobile Service Architecture: Meeting the Growing Needs of Mobile Handsets*, 2007. http://java.sun.com/javame/reference/docs/msa_datasheet.pdf.

[72] SVG Working Group. *Scalable Vector Graphics (SVG) XML Graphics for the Web*. World Wide Web Consortium. http://www.w3.org/Graphics/SVG/, Last accessed 2008-02-19.

[73] Versit Consortium (now Internet Mail Consortium). *vCard: The Electronic Business Card, Version 2.1*, Jan 1997. http://www.imc.org/pdi/vcardwhite.html, Last accessed 2008-02-01.

[74] WebKit Open Source Project. *The WebKit Open Source Project*. http://webkit.org/, Last accessed 2007-10-10.

[75] Wireless Application Protocol Forum. *Wireless Application Protocol Architecture Specification*, April 1998. http://www.openmobilealliance.org/tech/affiliates/wap/Technical_June2000-20021106%5B1%5D.zip.

[76] Wireless Application Protocol Forum. *Wireless Application Protocol Wireless Markup Language Specification, WAP-191-WML Version 1.3*, 2000. http://www.openmobilealliance.org/release_program/docs/Browsing/V2_1-20050614-C/WAP-191-WML-20000219-a.pdf.

[77] Wireless Application Protocol Forum. *WAP Wireless Telephony Application Interface, WAP-268-WTAI Proposed Version*, July 2001. http://www.wmlclub.com/docs/especwap2.0/WAP-268-WTAI-20010715-p.pdf.

[78] Wireless Application Protocol Forum. *Wireless Application Protocol Architecture Specification, WAP-210-WAPArch-20010712*, 2001. http://www.openmobilealliance.org/tech/affiliates/wap/wap-210-waparch-20010712-a.pdf.

[79] Nicholas C. Zakas. *Professional JavaScript for Web Developers*. Wiley Publishing, Inc., 10475 Crosspoint Boulevard, Indianapolis, IN 46256, USA, 2005.

[80] Nicholas C. Zakas, Jeremy McPeak, and Joe Fawcett. *Professional Ajax*. Wiley Publishing, Inc., 10475 Crosspoint Boulevard, Indianapolis, IN 46256, USA, 2006.

COS/CCS 2008-07

www.kth.se