CHALMERS

Web Operating System for Modern Smartphones

Master of Science Thesis

Henrik Steen
Gustav Tiger

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Web Operating System for Modern Smartphones

Henrik Steen
Gustav Tiger

Examiner: Ulf Assarsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Abstract

The immediate purpose of this thesis work is to further develop the possibility of building a mobile web operating system for smartphones by investigating key areas of modern smartphones operating systems and, if necessary, develop new Application programming interfaces (APIs) for accessing phone resources from a web environment. This report is mainly focused on enabling different kinds of phone calls, messaging services, settings and device information, multitasking, and synchronizing phone book data and other virtual resources.

To this end, the existing functionalities on modern smartphone platforms, as well as the current web standards, were studied. Then, whether or not the functionality could be fully or partly covered by the web standards and how certain functionality could be implemented in a web operating system environment, were examined. As a result, key development guidelines as well as additional or extensions of existing APIs were developed.

Sammanfattning

Det direkta syftet med detta examensarbete är att vidareutveckla möjligheten att bygga ett webboperativsystem för smartphones genom att undersöka nyckelområden i operativsystem för smartphones och om nödvändigt utveckla nya API:er för åtkomst till telefonresurser från en webbmiljö. I den här avhandlingen ligger fokus främst på att möjliggöra olika typer av röstsamtal, skicka och ta emot meddelanden, ändra systeminställningar, tillhandahålla systeminformation samt synkronisera kontaktuppgifter och andra virtuella resurser.

För att åstadkomma detta studerades de befintliga funktionerna i operativsystem för smartphones samt nuvarande webbstandarder. Sedan undersöktes huruvida funktionerna helt eller delvis kan omfattas av webbstandarder samt hur viss funktionalitet kan implementeras i en webboperativsystemsmiljö. Resultatet blev riktlinjer för utveckling samt nya, eller utvidgningar av befintliga, API:er för smartphones.

Acknowledgements

We would like to thank Opera Software, our project initiator Nicklas Larsson, and our supervisor Anders Höckersten for their help and support during this project. We would also like to thank our supervisor and examiner Ulf Assarsson at Chalmers University of Technology for all his help and his admirable modus operandi.

In addition to the people above we would also like to express our thanks, and our gratitude, to the web community, and everyone working for an open web, for their tireless effort to make the Web a better place.

Contents

Chapter 1

Introduction

1.1 Purpose

The immediate purpose of this thesis is to further develop the possibility of building a mobile web operating system for smartphones by investigating key areas of modern smartphones operating systems and, if necessary, develop new APIs for accessing phone resources from a web environment.

1.2 Objective

The objective of this thesis is to investigate the functionality found in modern smartphones and study the possibilities of implementing these concepts using web technologies. If the current web standards are unable to provide sufficient support, extensions to the existing standards or proposals to new ones should be developed.

In addition to the phone's hardware resources there are also several common virtual resources, such as phone books and calendars in smartphones, which potentially could be located online or on the device. Either way, the ability to store resources as online web resources or inside the device's local memory needs to be considered in this project.

1.3 Scope

The term smartphone (see Section 1.5.12) is frequently mentioned in this thesis and denotes a modern mobile device with phone capabilities, usually more connected and with more advanced features compared to an ordinary mobile phone.

Hardware requirements or specifications of the phones supposed to run a web operating system, is not part of this thesis' scope and no specific hardware will be mentioned. The general notion that hardware exists and needs to be in consideration will however be acknowledged.

Regarding the study of web technologies, all studies should be done based upon publicly available open standards. Since the standardization process usually span up to a few years [1] and some of the most related features are mentioned only in new proposed standards that have not yet been accepted, some of the APIs examined in this report are not yet fully reviewed by the web community and is not yet to be considered as standards. [1] [2]

The platforms studied in this thesis are either the most commonly used, the most relevant from the point of view of developing a mobile web application, or contain a solution that brings something different and unique to the table. When investigating operating systems or smartphones, only official supported features should be considered.

This thesis work only covers investigating different platforms and standards, and the development of new APIs – no implementation will take place.

1.4 Method

To get a general idea of concepts regarding smartphone usage, the first step is to study the characteristics, usage and functionality of modern smartphones. This phase aims to find out how the functionality is used in some of the most common smartphone operating systems today.

After researching the concept of smartphones, the next step will be to conduct a study of the current web standards and get a good overview of what is possible to implement using them. When doing this, special care has to be taken to try to understand the web standards' impact when designing applications for use on smartphones.

The third stage will be to study the current web operating systems, how certain problems have been solved and how certain functionality could be implemented. The last step is to specify and design the APIs needed to implement the functionality that could not be covered by the previously researched web standards.

1.5 Background

Recent years of innovation and improvement in web browser technology have shown the increased potential of web based applications. The available web standards and APIs have matured and web applications are now capable of replacing many of the native applications commonly used today [3]. These web based applications could run in a web browser, either online or when disconnected from the Internet [4]. This development, as well as the vast development of mobile devices, opens up for operating systems specially designed for running web based applications on modern smartphones.

A web operating system is an operating system which applications are written using web standards, such as HTML, CSS and JavaScript. Other definitions require the whole user interface to be written in such languages, but this is not part of the definition used in this thesis. The goal will ultimately be that all functionalities covered by modern smartphone applications are also able to be developed with web technologies. For this to be practical, open standards must be used as much as possible and new APIs should only be developed when no open alternative exists.

1.5.1 Web technology

HTML was originally designed as a simple language for sharing documents and linking documents together using hyperlinks. The HTML 4.01 standard was released in 1999 [5], but the two organizations WHATWG and W3C are currently working together in order to develop the next major revision, HTML5. The new standard introduces a number of new elements and attributes, as well as a number of scripting APIs.

1.5.2 The Mobile Web

In 1996, development on a new web language, called HDML, begun. It sprung up as a fork to the commonly referenced standard for HTML and was developed to contain a simplified striped-down version of that standard. HDML was never standardized, but lead to the development of WML which was used as markup language in the first edition of WAP. Later editions of WAP used an XHTML based standard, called XHTML Mobile Profile (XHTML MP), as markup language. Most of the mobile web languages were developed as a subset of other, desktop, languages. This is not the goal of this thesis; the aim here is to add additional features that are not present in web languages today.

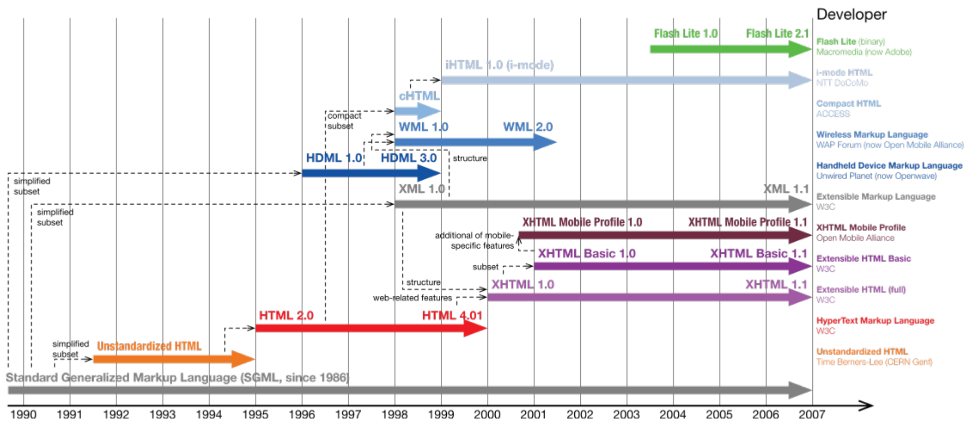

Figure 1.1: An overview of the evolution and branching of mobile web standards [6] (Image: Mobile Web Standards Evolution Vector, David Höffer, Wikimedia Commons)

Figure 1.1 shows an overview of some of the markup languages that have been developed for mobile devices in the past. Most of the languages described were however not targeted towards modern smartphones, but were meant to be used in mid-end devices or devices that are now outdated. That means that, however important the history of the mobile web has been in the past, not much of the above mentioned languages can be used as a building stone in this thesis.

1.5.3 Scripting

The most commonly implemented standard for client-side scripting on the Web is ECMAScript, with famous implementation dialects such as JavaScript, ActionScript and JScript. The ECMAScript standard defines a versatile object-oriented and prototype-based scripting language that originally was meant to be *"a Web scripting language, providing a mechanism to enliven Web pages in browsers and to perform server computation as part of a Web-based client-server architecture" [7].*

ECMAScript is said to be a prototype-based language. This means that each constructor is a function with a property named *prototype*, which is inherited by the objects. In the ECMAScript specification, the relationship and heritage of prototypes are explained:

> *"In a class-based object-oriented language, in general, state is carried by instances, methods are carried by classes, and inheritance is only of structure and behavior. In ECMAScript, the state and methods are carried by objects, and structure, behavior, and state are all inherited [7]."*

During the last decade, the performance of web based scripting has increased exponentially, which naturally has increased the potential for advanced web applications [8]. Figure 1.2 describes the vast development and speed increase of web browser based scripting during the last decade.

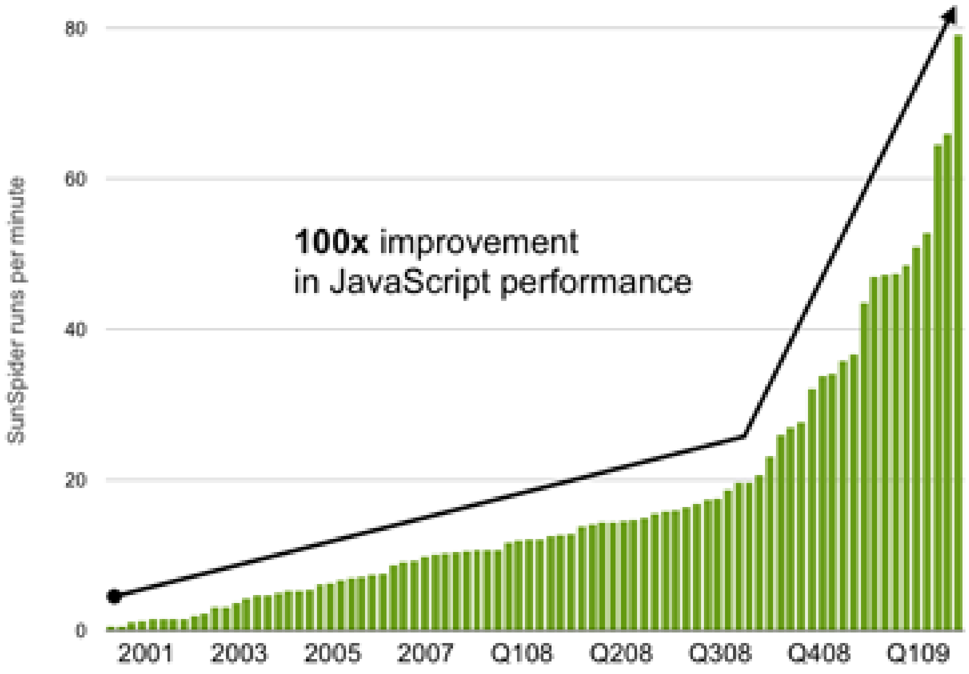

Figure 1.2: According to a study presented by Google Inc., the performance increase in web based scripts has increase with a growing rate since 2001. [8] (Image: Google I/O – Keynote 2009, Google Inc.)

1.5.4 User interaction

Since the release of HTML 4.01, user interaction on the Web has changed dramatically. AJAX allows web pages to initiate new requests, and interact with remote servers, without having to reload the page; making it easier to develop web pages that are more dynamical, faster and easier to use. In addition to this, the new hardware has made the Web more mobile than ever before. A study conducted by International Data Corporation (IDC) in 2009 [9] showed that there were more than 450 million mobile Internet users worldwide in 2009, and that is a number that is expected to more than double by the end of 2013.

The work on HTML5 includes many improvements on user interaction, and makes creating new ways of interacting with the user easier. One new attribute proposed is the `contentEditable` attribute. It allows user manipulation and editing of the specified part of the document, for example text to be inserted and removed as if the user was using a native text editor. This makes development on web based text and document editors easier and more capable than before. [10] [11]

The new HTML standard also defines simple functionality for drag-and-drop, something very common on web pages today. This means that it will be easier to develop web pages with an intuitive user interface. [12]

The Session history and navigation API also contains interesting functionality; it allows manipulations of the web browser's history and determine what should happen when the user tells the browser to go back or forward on a page. [13]

There have been several attempts at creating an open speech recognition API for the Web, and one of the latest is the HTML Speech XG Speech API. It is a proposal for communicating, controlling and interacting with a web page using speech recognition mechanisms. This API is especially interesting for small, mobile devices where other forms of input could be tedious or take too long time to use or get accustomed to. According to the proposition, the human voice could function as input on many different input fields, for example to conduct searches, fill forms, or click links. [14]

1.5.5 Visualization

CSS is a style sheet language for describing how a web document should be presented. It is used by defining rules for how a document, or a specific part of the document, should be styled. With the latest proposed standard, CSS3, new features such as multiple backgrounds, rounded borders, opacity, shadows, animations and transitions are introduced. [15]

The HTML5 proposal also improves visualization for the user, for example by introducing video and audio playback. The audio and video media elements introduced in HTML5 will enable plug-in free multimedia in web browsers. The scripting API provides extensive control to allow, for example, seeking and pausing video and audio playback. For the video element it is also possible to define text tracks where subtitles for different languages or commentaries from the maker of the video could be displayed. [16]

The canvas element can be used to create applications that utilize native acceleration from the device's graphic processor. The element can use different contexts; one for 2D and one for 3D. There is a proposal for an OpenGL based API standard called Web Graphics Library (WebGL), for the 3D context, that could provide smooth 3D graphics to the browser without the need for a plug-in. [17] [18]

Another visual feature drafted by W3C is the Web Notifications API. The API allows for asynchronous alerts to be sent to users outside of a web page, in order to, for example, show that something on another page has been updated or a new mail has been received. A notification contains an icon, a title and a notification body. [19]

1.5.6 Positioning

The Geolocation API contains functionality for finding a person's, or rather the device's, geographical location. This can be done in several ways; the most obvious way is by using the device's local GPS system, but IP addresses and cell tower location can also be used. The Geolocation API can be used to get longitude and latitude position, as well as some other data, from the device. [20]

1.5.7 Local storage

As the web applications become more and more advanced, require more data and faster response times, the need for local storage has grown bigger. Local storage means that the web browser can save larger amount of data locally and thus reduce the network load between the client and the web server. Allowing web applications to store data locally increases the amount of data that an application can consume, as well as lowering the time needed when loading a larger amount of data.

There are several solutions that allow local storage, some more advanced than others.

- The Web Storage API, a W3C working draft, allows local storage by providing key value pairs to be saved. The specification does not specify how much local space that could be used, but instructs implementers to limit that space in order to not overload the client device. The specifications also allow automatic deletion of storage areas, if this is preferred by the user. [21]
- The Indexed Database, or WebSimpleDB, API is close to a real transactional database with indexes, cursors and transactions. [22]
- The File API provides an interface for representing file objects in web applications, for reading and working with files directly in the web browser. [23] [24]

- Offline Web application is a specification for letting users access web applications and documents even when connectivity to the Internet is unavailable. The files specified are cached by the browser in order to keep a local copy to work with offline. [4]

1.5.8 Communication

Since the introduction of AJAX and the XMLHttpRequest (XHR) API there has been a clear change where web pages have become more dynamic and interactive. The AJAX APIs allows browser scripts to contact a web server and get data back without reloading the page, as well as changing and updating a web page asynchronously. This can be used to create dynamic web pages and allows for more specific requests to be made. [25]

XHR is an easy and asynchronous way of sending messages from a client browser to a web server, but there are also other ways of communicating on the Web. The new WebSocket API, for example, which enables a two-way socket communication between a user's web browser and a specified server. It introduces a way for web pages to maintain a bidirectional socket with the server, and not as XHR, which is used for sending non-persistent requests and receiving an answer from a server. [26]

The Web Messaging API is a cross-document messaging API for sending messages between pages on different domains. This communication is, for security and privacy reasons, prevented in most web browsers today, but the Web Messaging API aims to introduce a way for web pages to communicate in a secure manner regardless of their source domain. [27]

Another API worth mentioning in this context is the Video conferencing and peer-to-peer communication API, a living standard from WHATWG that enables video-conferencing using web technologies. It contains methods for recording audio and video stream, as well as connecting to remote peers and accessing each other's streams. [28]

1.5.9 Multithreading

Multicore CPUs are dominating the PC market, but most of the smaller mobile devices are still only using one core. Some mobile phone vendors have begun releasing smartphones with dual-core processors, but they are far from common. Most web pages are built to have one single execution thread for their scripts, but some multithreading solutions are available. [29]

The Web Workers API allows web applications to create background threads that run in parallel. If multiple workers are created, those will be executed asynchronously. Creating a new worker is done by passing a script into a constructor, and the execution begins automatically. The API also provides basic message passing capabilities. [30]

1.5.10 Web applications

In this paper, a web application is defined as an application, with the purpose of performing one task or multiple related ones, written using common web techniques that can run natively in a modern web browser. web operating systems could also allow web applications to be installed on the device, in order to give additional functionality and easy access to the application. This definition does not put any requirements on how advanced an application must be; an application could be anything from as simple as just showing and updating a clock to a whole office suite with spreadsheets, presentations and documents. It is however important that the application must not depend on any plug-ins or extensions to the browser. This disqualifies Flash and Java applications from counting as web based applications.

There is a large amount of different web applications available on the Internet today; it can be everything from e-mail applications, office suites and newspapers to blogs and other social communities.

1.5.11 Web operating systems and other solutions

There are several examples, and several varieties, of web operating systems. Some define it as an operating system that runs inside a web browser, and where a web page creates or emulates a complete desktop space inside the web browser. Another definition defines it as operating systems in the classic sense but where all applications are written using web technology. This is the definition used in this thesis.

Several benefits can be derived from using a web based operating system rather than an ordinary operating system. Some examples of concepts that make a web operating system for smartphones a rather interesting idea could be:

- Easy app development – Developers are already familiar with the development process and very little has to be added to use the phone's native hardware.
- Cross platform applications – Open web standards enables applications to run on multiple platforms
- Customizable – Built-in applications (such as phone book and SMS applications) are easy to change and customize on vendors request, or possibly by the vendors themselves.
- Openness – All applications are written using open web standards and are more open for other developers to see, compared to the rather closed environment of traditional operating systems.
- Security – Web applications are by definition sandboxed in the user agent and do not have direct access to the local file system or hardware.

Google Chrome OS

Google Chrome OS is a web operating system built upon the web browser Chrome, developed partly as an open sourced project supported by Google Inc. The operating system features a fully integrated media player to support the HTML5 video-tag, an integrated flash player, a settings manager, as well as a built-in PDF-viewer. The operating system does also include a built in file browser, from where users can upload files from a plugged in camera or USB device to the Internet through the Google Chrome OS API. [31]

Google Chrome OS also introduces Google Cloud Print, which allows users to send files to a printer over the Internet. The user does not need to be connected to the device, or even physically present, as long as the printer is connected to the Internet, has support for Google Cloud Print, and the user has clearance to use it. This also means that there is no need for the user to install any drivers to use the device.

When introducing the new platform, Google emphasized the security of only running web applications; for example, since all applications can be sandboxed, it is not possible for applications get information that leaks from another application. Sandboxing also means that applications are unable to modify files without the user's consent, and only the operating system itself can access settings such as power configurations or installed applications. [31]

Figure 1.3: A laptop running Chrome OS, planned to be released in the middle of June 2011. (Photograph: Samsung Chromebook Series 5 Chromebook, Amazon [32])

PhoneGap

The PhoneGap open-source framework enables developers to use web technology, such as JavaScript, HTML and CSS, when developing applications for mobile devices. The framework has been implemented for several different platforms, including iOS, Android, Symbian, webOS and Windows Phone 7. [33] The support for each platform differ, and in some cases even the specific version of the underlying operating system makes a difference, but according to the official list of features, some supported features are: [34]

- Accelerometer - Tap into the device's motion sensor.
- Camera - Capture a photo using the device's camera.
- Compass - Obtain the direction that the device is pointing.
- Contacts - Work with the device's contact database.
- Device - Gather device specific information.
- Events - Hook into native events through JavaScript.
- File - Hook into native file system through JavaScript.
- Geolocation - Make your application location aware.
- Media - Record and play back audio files.
- Network - Quickly check the network state.

- Notification - Visual, audible, and tactile device notifications.
- Storage - Hook into the devices native storage options.

Wholesale Applications Community (WAC)

WAC is the name of the organization, originally started by telecommunication operators to unify the different mobile platforms and operating systems, as well as the name of the API they develop to allow development of platform independent applications for mobile devices. [35]

The API is comparable to PhoneGap and does also use existing web standards such as HTML, JavaScript, and CSS for application development. The API also includes parts that, for example, allows accessing contact information, calendar events, device status and sensor information. [36]

Besides the API, WAC is also a broader concept which includes users, developers, telecom operators, and handset manufacturers and OS owners. Operators can run application stores that enable application developers to charge their users through the operators existing billing systems by sharing some of the revenue. [36]

WAC is also supported by a large number of operators, developers, and handset manufactures. Some well-known examples of the more than 70 member are GSMA, China Mobile, Vodaphone, Ericsson and Huawei. [37]

The WAC 2.0 Specification includes the following modules for developer to use:

- The deviceapis module – The base object for accessing WAC Device APIs.
- The accelerometer module – API that allows using the device accelerometer sensor.
- The orientation module – API that allows using the device orientation sensor.
- The camera module – API that enables capturing media through the device camera.
- The devicestatus module – API that provides access to the device status information.
- The filesystem module – API that allows accessing the device file system.
- The messaging module – API that allows message sending and retrieval.
- The geolocation module – API that exposes the device location (as specified in W3C).
- The pim module – API that exposes the different PIM (Personal Informa-

tion Management) functionalities.
- The contact module – API that enables the management of contact information.
- The calendar module – API that enables the management of calendar information.
- The task module – API that enables the management of task information.
- The deviceinteration module – API that enables the interaction with the end user through different device capabilities.

webOS

> *"Architecturally, Palm webOS is an embedded Linux operating system that hosts a custom User Interface (UI) System Manager built on standard browser technology. The System Manager provides a full range of system user interface features including: navigation, application launching and lifecycle management, event management and notifications, system status, local and Web searches, and rendering application HTML/CSS/JavaScript code [38]."* – HP/Palm, Overview of HP webOS

WebOS is a mobile operating system for smartphones, developed by HP/Palm. Most applications, both in the software suite that is included in the operating system and the third party applications available for installation, is written using HTML, CSS, and JavaScript. HP/Palm has created two frameworks, Mojo and Enyo, to help developer create functional web applications with ability to interact with some of the phones hardware and storage system, as well as use the same layout rules as all the other applications.

This type of web environment is enough to power most applications, but for some heavy graphic applications, or applications that need a closer interaction with the device's hardware, webOS provides C/C++ support together with OpenGL and SDL. Those applications will be deployed as a plug-in, using the webOS Plug-In Development Kit (PDK), to the underlying operating system, which ease the interaction with the rest of the operating system, and allows the plug-ins to utilize the same JavaScript libraries as the web applications.

For background applications, also called services, webOS allows usage of node.js, a framework for building scalable network programs using JavaScript. By using this kind of non-graphical web frameworks, the services and the application can run in the same environment and use the same APIs. Developers will only have to learn one environment and interaction between

Figure 1.4: The HP Pre3 running webOS 2.2, the next major version of webOS featuring Enyo (see Section 1.5.11), is said to be released in the summer of 2011. (Photograph: HP Pre3, HP [39])

applications and services gets easier to handle.

Mojo Mojo is a framework and SDK for developing applications for webOS. An application written with Mojo will contain one or multiple views, or pages, called *scenes*. A scene will be rendered on a *stage*, which corresponds to to a tab or a window in a desktop browser. Each scene could consist of some predefined widgets, buttons, forms, or anything the developer has chosen to display. It could for example just render an ordinary web site.

The application listens for events given by the Mojo framework – events that could spring up from the user interacting with the interface, pressing a button or making a gesture. The framework provides a number of Service APIs, through which the application can communicate with the underlying system and hardware. The API does also provide methods for communicating with remote servers or other applications on the same device, accessing the calendar and contact information on the device, and so on. [40] [41]

The webOS Service APIs include:

- Accelerometer – Orientation events and accelerometer data access.
- Accounts – Returns information on established user accounts for use with the HP Synergy feature information manager.
- Alarms – Sets a timer to activate on the device either after a specified interval or at a specified date and time.
- Application Manager – Invokes default handlers for the common resource types or basic device operations.
- Audio – Plays or streams audio by using common audio formats.
- Browser – Loads and views the target specified by a URL.
- Calendar – Various methods for accessing or creating Calendar data.
- Camera – Launches the Camera application to take a picture.
- Connection Manager – Gets connection status and subscribes to notifications of connection status changes.
- Contacts – Various methods for accessing or creating Contacts data.
- Display Manager – Gets events related to the status of the display.
- Document Viewers – Launches the DocViewer application to browse and view common document file types.
- Download Manager – Uploads and downloads files over HTTP.
- Email – Sends an email, and includes options for pre-populating the email contents.
- GPS – Gets the current location coordinates and registers for continuous updates.
- Keys – Gets keypress events from headset and volume buttons.
- Maps – Displays a map based on the various input options.
- Messaging – Sends an IM/SMS/MMS, and includes options for pre-populating the message contents.
- People Picker – Displays a list of contacts for the user to make a selection.
- Phone – Makes a phone call with or without a pre-populated dial string.
- Photos – Views an image in various common image formats.
- Power Management – Enters Sleep mode after a period of inactivity.
- System Properties – Gets the named system properties, including device ID.
- System Service – Accesses various system settings, including systemTime.
- System Sounds – Plays audio feedback in response to user interaction. The sounds play when the message is played, with low latency.
- Video – Plays or streams video by using the common video formats.
- View File – Downloads and/or views a file in various formats or resource types.

Enyo Enyo is the latest generation of web frameworks for developing web applications for webOS. The framework uses the same technologies for the mobile platform as is used on the desktop user agents today. This means that the same developing and debugging tools used for developing desktop applications could be used to develop for other devices, such as smartphones running a web operating system.

The framework is still in development, but devices with the new version of webOS are scheduled to be released in summer of 2011.

1.5.12 Smartphones

The definition of a smartphone, and the distinction between smartphones and other devices, such as other high-end mobile phones, mobile navigation systems, or a small tablet or pocket computer, is difficult. A smartphone could be defined as a mobile device, with a set of basic features that distinguish them from other devices; ability to make phone calls, play music, take photographs with a built in camera, and having a capable web browser together with the connection capabilities required to fully utilize the web, are common requirements on modern smartphones. A smartphone should also include a multitasking operating system and a single or multi-touch screen for user interaction. This definition excludes some of the high-end Symbian and BlackBerry devices that lack touch screen user interfaces, but interaction with such devices is very different from using a touch interface.

Compared to desktop or tablet computers, smartphones are considerably smaller in size and the hardware capabilities are more limited. Each platform handles the problems arising with low available memory and a limited battery supply differently. The latency introduced on wireless connection, and the fact that most cellular plans does not include a flat data rate, does also need to be taken into consideration. User input is another difficult task to handle; a touch interface together with either a small sized hardware keyboard or none at all. All these attributes need to be considered when doing development for smartphones.

Applications

One feature that in some way separates low and mid-end mobile phones from smartphones is how the connectivity can be used to install new third party applications on the device. First party applications are developed by the platform provider, while third party applications are developed by a third party developer. Second party applications are not as common, but could be described as an application developed by a third party with ties to the device or platform manufacturer and who have received access to tools or APIs not normally available to third party developers.

Applications are usually installed on the device, downloaded either through an application market or directly from the developer. Most platforms do, besides normal applications, also support background applications, called services, that do not require a user interface but are able to run in the background. These services are often used, for example, to fetch e-mails or synchronize contact information to the local phone book.

Platforms

The three largest smartphone operating systems as of today are Android, Symbian and iOS, and hundreds of millions of smartphones are sold every year [42]. Nokia has decided to stop the development on the Symbian platform and start developing smartphones for the Windows Phone 7 platform instead [43]. The Windows Phone 7 platform is predicted to become one of the larger platforms for smartphones in a few years [42].

The webOS platform was introduced in Section 1.5.11 and is highly interesting for this thesis, since it solves many of the problems with developing frameworks and APIs for web applications on smartphones. Another interesting platform, built to be more versatile and therefore has to solve some problems a bit differently compared to others, is the Maemo or MeeGo platform.

Virtual resources

Besides the ability to communicate with some of the smartphone's hardware, the applications also need access to a number of virtual resources. These are resources that could be stored anywhere, on the device or somewhere online. For keeping implementation and documentation simple, some kind of APIs are needed to access and manage such resources. The most common virtual resources available in modern smartphones are:

- Phone Book – For accessing contact information
- Calendar – For scheduling events and reminders
- Messages – Short Message Service (SMS), Instant messaging (IM), e-mail, and other messaging services
- Files – Located at flash memory cards, hard drive, cloud storage, and such

- Menu-items – Workspaces, icon placement, widgets, and other menu related configurations
- Settings – System settings, such as current ringtone, current date and time, and other preferences
- Installed applications – The current install applications on the device
- Notes – Notes saved by the user
- Clock/Alarm – Wake up alarms, with ability to wake the phone even when turned off
- Notifications – Alerts to the user about a recently occurred event
- Clipboard – Recently cut or copied information, for pasting or saving.
- Accounts – Account information shared between different applications, such as Twitter, Facebook, or IM services

Having well defined APIs for each virtual resource, and an enforced permission system that keeps applications from using the resources without first getting permission from the user, is a common security benefit, seen in most smartphone operating systems, and a common way for giving restricted applications access to shared data.

Chapter 2

The Phone Book

Contact information is one of the most common virtual resources and much previous work has been done prior to this thesis. Any given solution must include functionality for adding new contact information, remove information and changing previously inserted information. This is the most basic set of features that are required, but some of the following sections will also discuss alternative or additional features that are desired when developing phone book applications for mobile devices using web technologies.

2.1 Previous work

There are a number of different solutions available for accessing phone book data in a mobile device, and even some for implementing phone book applications using web technologies; W3C has been working on a draft, as described in Section 2.1.3, and the webOS platforms includes an extensive API for managing contact information.

2.1.1 Android

The Android API suite includes the Contact API for managing and integrating contact information from multiple sources. Besides the common methods for adding and managing contacts, it also provides methods for aggregating similar contacts together and presenting the result as one contact. [44]

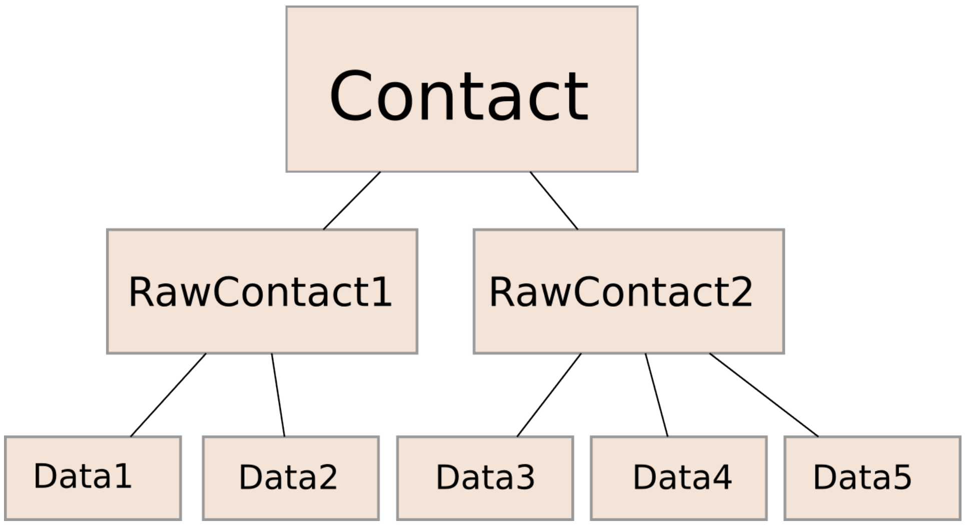

Figure 2.1: A Contact-object consists of information aggregated from multiple sources. [44]

Figure 2.1 describes how a Contact-object in the Android API consists of one or more RawContacts. A RawContact is described as a set of data associated with a single contact source. The idea is to allow synchronization with multiple services, for example Google, Exchange, or Facebook accounts, and let the user merge the RawContacts from each service into one single Contact-object for each person in their phone book. Instead of having several contact objects corresponding to the same person, the information is collected and presented to the user as one entity for each person. Merging or aggregating duplicate contacts is also seen on other platforms, for example Maemo and webOS, and is a common way of simplifying the usage of sorting and finding contact information in a phone book.

Besides letting the user manually handle merging of contacts, there is also an automated routine that looks for information (name, address or e-mail, for example) that is highly similar with other contacts. The routine will, if found, automatically merge the contacts together. If the user later finds that the contacts should not be represented as the same contact, the aggregation is reversible. This is achievable by only linking the RawContacts together; if the change should need to be reversed, all that is needed is to remove the link. [44]

2.1.2 WebOS

The webOS Contacts API is built around the Contact-object, stored in a local database as JSON [45] data objects. The preferred way of creating a new contact is to create the contact object and pass it as an argument to the Contacts application. This will open the application with the edit view of the newly created object.

Schema

The webOS standard library for handling contact information is mainly focused on the Contact-object. The built in phone book application extends the functionality by enabling users to link multiple Contact-objects together in a Person-object container. The container contains all of a person's email addresses and phone numbers, even if the information might be fetched from different services and stored in different Contact-objects. A Contact-object must be tied to one Person, which means that the phone book application only has to make a list of all Person-objects to get all contact information available. This linking functionality is very similar to the one seen in Android; an Android RawContact could be compared to a regular Contact-object in webOS, and an Android Contact-object would then correspond to a webOS Person-object.

Some information in the Person-object container is however not returned as a collection of information from multiple sources, but instead as a single value from the Primary Contact. The Primary Contact is set by the user, defaulting to the first Contact-object that was tied to the Person-object. This means that the type of information that is not supposed to differ between different services is not presented to the user as a list, but as a single value. Other information, such as street address or phone number, is presented as a list. [46]

API calls

The webOS API does not allow applications to search and access contact records that was not created by the application itself. This is for security reasons and to help to protect a user's integrity.

If an application needs contact information that was created by another application, it has to go through the People Picker API. Calling the API will launch the built in phone book application and let the user manually choose a person that will be returned to the application. This only allows the application to select one person to be returned. [47]

2.1.3 W3C Contacts API

W3C is working on a solution to standardize contact handling using a modern web browser. The W3C Contacts API enables applications to read contact information, while the W3C Contacts Writer API handles data input. [48] [49]

The Contacts API defines two interfaces; one for a single contact and one to be used for a group of contacts. The Contacts interface contains the method `find` that can be used both as a search function for finding a list of contacts, or as a get method that returns a Contact-object with a given id. The other interface, Contact, is a full featured Contact-object container. It contains all information needed to represent a Contact-object in a phone book. [48]

The APIs take the integrity of data in consideration by specifying a number of privacy rules for user agents to follow; defining what mechanisms that are needed before an application may access the user's address book. A user must for example give permission before an application may access information and all permissions that do not expire after each session need to be revocable by the user. [48]

Mozilla has begun working on an implementation of the Contacts API under the Mozilla Labs initiative. The implementation also contains a few alterations and extensions on the API, for example for handling contact aggregation. [50]

2.1.4 PhoneGap

PhoneGap provides a small API that allow applications implemented using web techniques to access contacts information. The API contains methods for adding new contacts to the phone book, by simply providing all information associated with the new contact, and to search the phone book for already existing contacts. The API follows the W3C draft of fetching and writing information to the phone book. PhoneGap does however adopt the attributes it provides to fit the attributes available on the underlying platform. On some platforms, for example, the first two addresses added to a contact will be stored as a home address and a work address, no matter what tag the user has added to the two addresses. This could be seen as somewhat of a simplification, but is necessary in order for the API to be compatible with all supported platforms. [51]

As with the W3C solutions, there is no way of keeping a revision history, or merging two Contact-objects into one. [51]

2.2 Analysis

The purpose of this thesis is to, as far as possible, build upon open standards and add only necessary extensions that are not covered by the standards and are not moving away developers from using the open solutions. In this case, the open W3C Contacts and Contacts Writer API is an open draft that covers all of the daily usage of a phone book application implemented in the browser. The specification defines an API for gathering contacts information from multiple sources, both remote and local, into unified phone book, as well as interfaces for accessing the stored contacts.

The Writer API contains methods for creating, updating and removing contacts from the phone book. One feature missing from this specification, which has shown to be of value on other platforms (see Section 2.1.1 and 2.1.2), is functionality to link or merge contacts. If the aim is to only provide this functionality, of managing contacts, in the user agent itself, and only provide users of the API a way of fetching or adding information, there is no need for merging to be added in the standard. If the aim however is to let third party web applications handle the contact management, and the functionality of contact merging is desirable, an extension of the APIs would be beneficial to the developer and make synchronization of new contact information easier.

2.2.1 Permissions

The W3C draft specification states, in the privacy considerations section, that the user agent implementing the specification must acquire permission from the user before allowing access to the API but states that such permissions could be prearranged for example when installing an application. It is however arguable if all applications need access to information about all contacts or if the user should be able to only allowing access to a specified part of the phone book. This is something that the specification touches upon, but does not go into detail on. It could, for example, be an application only handling synchronization of contacts between an external service and the local user agent. An example of a more detailed solution, which might not be included in the specification but could be useful for implementers of the specification to have in mind, is presented below. In this solution, there are three kinds of permissions, each with its own purpose.

The read-only access gives read access to all contact information stored in the phone book. This can be used by applications that need access to fetch information, but do not need to input new information into it. Messaging applications can, for example, use the API to create an auto completion field in the messaging text box. This part is fully covered by the W3C Contacts API.

Synchronization access gives synchronization services access to a specific part of the contact list. This access includes permission to read and write, but is limited to contact information that has previously been added by the service or a domain itself. Each contact, or raw contact, could (internally) be marked with a source and access to this contact would only be granted to the same source. Since this part does not handle linking, it is fully covered by the W3C Contacts and Contacts Writer API.

The last kind of permission includes full read and write access to all contact information in the phone book. This will be used in order to create a phone book application, or to create a complete backup of the whole phone book, for example. This is the permission that enables the new extended API, which works in the same way as the W3C Contacts and Contacts Writer API but with some extra functionality. This means that a simple representation of a phone book, that is only using the basic methods of the W3C standard, should be able to work satisfactory. A more advanced phone book however, that requires linking and awareness of sub-contacts could use some of the methods in the extended API.

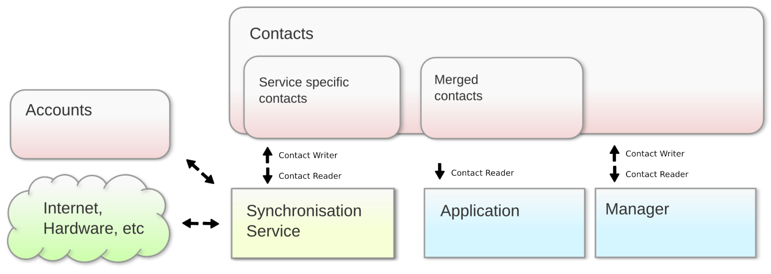

Figure 2.2: Overview of related API:s

Figure 2.2 shows how other APIs relate to the Contacts API and how different parts are communicating. A synchronization service could access an account and its remote contact information, adding or updating previously added raw contacts to the local phone book through the Contacts API. Applications using the API could then access real contacts with information from multiple sources.

2.2.2 Linking

Linking is a way of virtually merging two contacts into one, without destroying their internal representation or changing any information. The internal representation could then be used by synchronization services in order to synchronize only the data that it itself has provided.

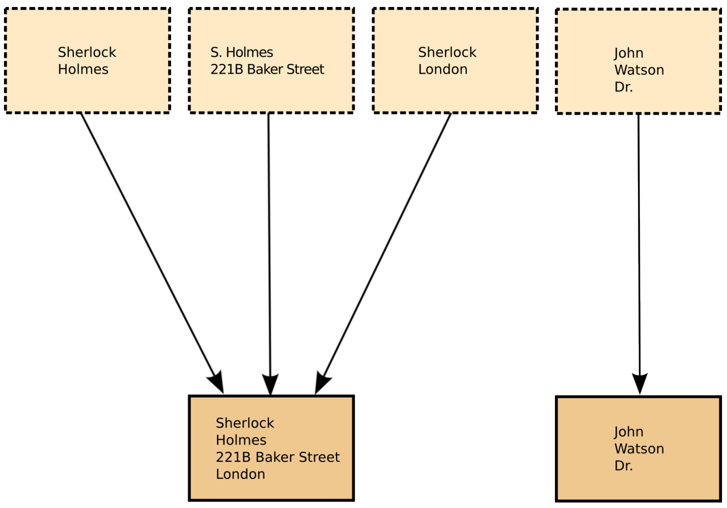

Figure 2.3: The process of linking information from three different sources into one Contact object.

Figure 2.3 describes how contact information from multiple sources can be linked together into Contact-objects. The resulting Contact-objects are only retrievable for applications asking either for the read-only access permission, or the full managing permission. The dotted lines indicate RawContacts (as described in Section 2.1.1) that are either provided by a remote service or managed only in the local phone book.

Note that the linked contacts do not contain all available information; the name S. Holmes in the example above is discarded for being information already filled by a previous information source (in this case Sherlock Holmes).

This information is said to have higher priority than the new information and is therefore not included in the final Contact-object. This does not mean that the information is lost or overwritten.

Android, Windows Phone 7, Maemo, and webOS all uses linking, which shows there is a direct need for linking on mobile platforms. At least when using, and synchronizing, multiple accounts.

2.3 Result

This proposed API extends the W3C Contacts API by introducing four methods: `getContacts`, `setAsPrimary`, `link` and `unlink`. No new object types are needed, since both the Contact-object and the RawContact can be represented with the same object as is used in the W3C proposal.

getContacts The `getContacts` operation will retrieve all RawContact objects that a Contact-object consists of.

```
getContacts(onSuccess, onError, fields)
```

Where:

- `onSuccess([contact])` is the callback function, with the wanted list of RawContact objects as parameter
- `onError(e)` is the callback function that is called when an error, `e`, has occurred
- `fields` (Optional) is a list of the fields that should be returned to the onSuccess callback function

link Link a given RawContact to one or more other RawContacts.

```
link(onSuccess, onError, [contact])
```

Where:

- `onSuccess(contact)` is the callback function that is called when the link operation has successfully executed and `contact` is the Contact-object that now contains information from all the newly linked RawContacts
- `onError(e)` is the callback function that is called when an error, `e`, has occurred

- `[contact]` is a list of contacts that should be merged with the Contact-object

Example:

```
function success(c){
  alert("Successfully linked all RawContacts named "Sherlock" into one contact.");
}

function found(rawcontacts){
  if(rawcontacts.length > 0){
    rawcontacts[0].link(rawcontacts, success, error);
  }
}

navigator.service.rawcontacts.find(['id'], found, error, {filter: "Sherlock"});
```

unlink The unlink operation separates a RawContact from all other RawContact objects that it is currently linked to.

```
  unlink(onSuccess, onError)
```

Where:

- `onSuccess(contact, rawContact)` is the callback function that is called when the unlink operation has successfully executed and `contact` is the Contact-object that is no longer linked to `rawContact`
- `onError(e)` is the callback function that is called when an error, `e`, has occurred

setAsPrimary Set a RawContact to the primary RawContact on its current Contact-object. This means that its information will be prioritized above information from other RawContacts.

```
  setAsPrimary(onSuccess, onError)
```

Where:

- `onSuccess(contact)` is the callback function that is called when the operation has successfully executed and contact is the new representation.
- `onError(e)` is the callback function that is called when an error, `e`, has occurred

Chapter 3

Synchronization

Synchronization is a way of easing the handling of phone book contacts, calendar events, e-mails, and other such personal data. It also introduces a possibility for the users to back up data, by synchronizing information to a remote server. Some algorithms are only capable of fetching information from one or more sources and save it on the local device. Synchronization algorithms do, in addition to fetching information, also have the capabilities of pushing local changes back to the original source and have information on all clients being automatically updated.

3.1 Previous work

There are different solutions to how synchronization should be handled in a smartphone environment; some systems give the developers no or little additional functionality, while some provide a whole architecture with solutions to problems such as secure account handling, synchronization services and integration into the standard system.

3.1.1 webOS

In webOS, synchronization is handled by services called Synergy Connectors. An implementation of a Synergy Connector has the capability of synchronizing contacts, calendar events, or messages. It uses an Account for accessing the information on a remote server. The procedure of having a system account service will allow other services to get access to an account, and the information it is protecting, without having to compromise passwords or any other identification credentials. This could be achieved by, for example, providing an access ticket that only gives access to the information that should be accessed by the requesting service. The service may then use the other APIs to manage the respective data types; using the Contacts API for handling contact information, the Calendar API for event scheduling, and the Message API for synchronizing message data. A brief overview of how this works is showed in Figure 3.1. [52]

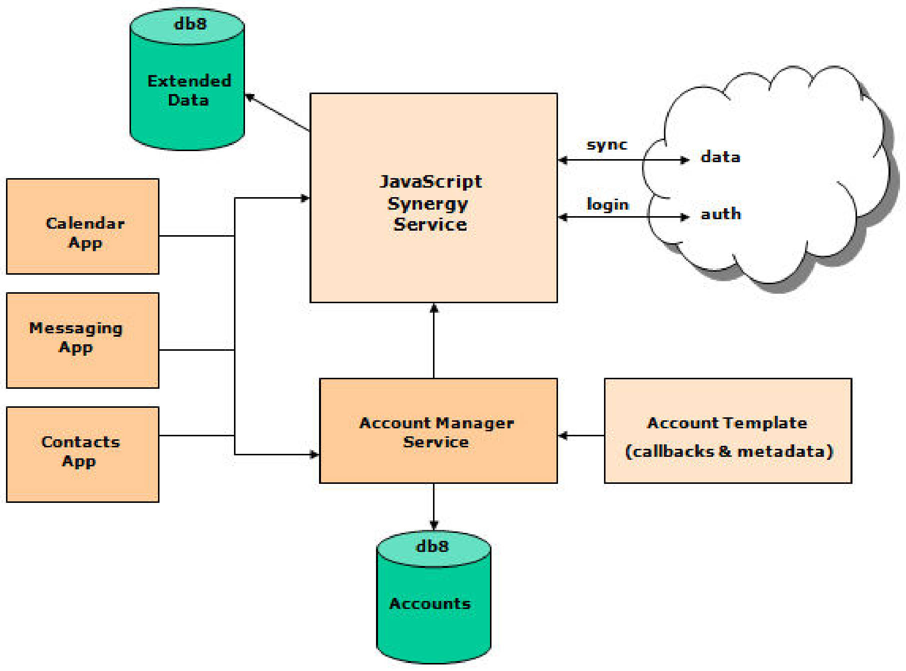

Figure 3.1: An overview of the how an implementation of a Synergy Connector would be able to synchronize calendar events, contacts and messages.

The service will continue to be able to synchronize changes occurring on the server for as long as it is running. This synchronization from a remote server to the client is usually done in intervals. The intervals are decided by the underlying API that, based on the current processor load and network usage, executes a function registered by the service in order to handle the synchronization procedure.

As of webOS 2.0, HP/Palm is also opening up their Synergy service to allow third-party developers to create connectors for Contacts, Calendars, and Messages. This means that the user will be able to synchronize related information with any external site for which there exists a Synergy connector. [52]

A connector is packaged and distributed as any other application, through the webOS App Catalog. The Connector is built as a JavaScript service that creates an account with the Account Manager and stores the data objects. In order to be able to package it as a regular application it also has to include an ordinary application; even if the application, in this case, could consist only of an empty file.

The service inside the connector will register the event handlers, also called assistants, which will be responsible for acting when the user commits changes, updates, removes, or adds new information to a contact. It will also have to implement a trigger callback, which can be called whenever a synchronization event has been triggered, either a manual call by the user or a regular synchronization update. [52]

3.1.2 Android

Android is quite similar to webOS in that both accounts and synchronization services can be created. These services can then synchronize contacts or other information.

A service, which is called a SyncAdapter in Android, performs the synchronization in a background thread while an AccountAuthenticator service handles the account part. [53] [54]

3.1.3 iOS

Synchronization in iOS is built into the operating system and can be used with M4E, MobileMe, Gmail, Yahoo, AOL, and general POP/IMAP, LDAP, CalDAV. There are no special methods designed for third party developer to create new synchronization services, except using the standard APIs for handling and accessing data.

There is only one address book application available, and other applications have to go through it in order to access contact information. Each contact record contains the source from where the information originates. This information is mainly used for displaying the origin for each contact to the user. [55]

3.2 Analysis

The webOS platform provides a stable and secure way of handling user sensitive information, such as username and password, for accounts used for synchronizing. Having global secure accounts does also enable multiple services or applications to share the account. The implementation steps necessary to keep this information hidden are however quite advanced, and additional support on the remote server might be necessary for it to be completely secure. There is also no way for the user to know whether the account information is secure or not, since it all depends on the account implementation for each service. Creating a secure way of handling authentication information, that is both easy to implement and easy to use, is no easy task, and not something suitable for this thesis. Further research is needed for a solution to be developed. [52]

Having a service register a callback method that is called whenever the device is ready to synchronize data is an interesting feature, but has to be extended with more user control. It would be suitable for a synchronization API to provide additional callback methods, for example to have services register methods that would be called whenever a calendar event has been added to the local calendar, so that it would be directly pushed out to the remote server.

3.3 Result

The proposed synchronization API consists of a few registration functions for registration methods that should be called either when the system is ready for a full scale synchronization, or when an object visible for the service has been changed. The synchronization function would only have to be a trigger function, but the others would have to contain information about the changed data.

An important design question that has to be answered is where a service's functions should be registered. This could be done either each time the service is launched, having each service relaunched on system startup, or registered internally when a service is installed. In webOS, the function registration occurs at installation, and since all services is required to be installed on the device, this is viable even in this solution. This approach would also not require each service to be initialized each time the device is turned on, and instead only initialize and call the functions when they are needed.

Explained below are what functions could be registered with the proposed synchronization API. The exact registration of the functions, what metadata is required, and how the registration is handled, is not included in this thesis, but a further discussion can be found in Section 10.3.

Synchronize trigger The synchronize trigger is a function registered to be called when the user agent decides that network activity and system resource usage is low enough and sufficient time has passed since the last synchronization round. The user agent should provide an interface to change the approximate time between synchronization, as well as an interface for manually trigger a call to the synchronization method.

```
onSynchronize()
```

Example:

```
function found(contacts){
  // send the contact information to a remote server
  for(i in contacts){
    if(!contacts.hasOwnProperties(i)) continue;
    sendContactToServer(contacts[i]);
  }
}
function onSynchronize(){
  // get a list of all contacts
  Contacts.find(found, onError)
}
```

Calendar updates This function will be called when a calendar event has been updated. Since this thesis does not cover a more specific calendar API (see Section 10.2) no more details will be specified here.

```
onEventUpdate(newEvent, oldEvent)
```

Where:

- `newEvent` is the new event, containing the updated information
- `oldEvent` is the representation of the old event, before it was updated

Phone Book Update This function will be called when information in the Phone Book have been updated. Note that for services with access only to a limited amount of contacts, only changes on contacts visible for the service will trigger a call to this function.

```
onContactUpdate(newContact, oldContact)
```

Where:

- `newContact` is the new Contact object, containing the updated information
- `oldContact` is the representation of the old Contact, before it was updated

Chapter 4

Messaging

SMS is still the dominant form of instant messaging on mobile phones today. A study done by the Nielsen Company has shown that U.S. teens send an average of over 3 000 texts per month; many teens also say that texting was one of the main reason for them to purchase a mobile phone in the first place. [56]

There are several other ways of sending messages, besides SMS, on modern smartphones – including MMS, XMPP and other types of instant messaging protocols, most of which originate from desktop computers. These services are similar to each other, and contain the same basic functionality of sending short instant text messages. Some of the services have additional functionality, such as attachments or support for multiple recipients.

This report will not try to address e-mail under this section since it is not designed for instant communication.

4.1 Previous work

One common method for allowing third party applications to send instant messages is by letting applications launch the official messaging application; possibly with predefined values, such as message body and recipient. It is also common to support the SMS URI scheme, which allows applications to use links, such as `sms:+15105550101?body=hello%20there`, to launch the standard messaging application. [57].

4.1.1 webOS

The webOS API only provides capabilities to pre-populate fields in the official messaging application. Unlike iOS and Windows Phone 7, webOS uses the same API to send all supported types of messages, including SMS, MMS and IM messages. [58]

Messaging is handled with the Synergy service, which collects messages from different sources and presents them through a single interface. Developers can develop new Synergy Connectors to communicate with new messaging services, which then allow programs to access the messages with the same API. [52]

4.1.2 WAC

The WAC platform provides a number of ways of handling messages. It supports sending SMS, MMS and e-mail as well as subscribing to incoming messages from these services. The services can however not be extended, as in webOS. WAC also provides extensive methods for searching, filter and listing messages based on different parameters.

The API is using the same functions to send all types of messages but some services only support a subset of the attributes. Subscriptions of incoming messages however use separate functions for each message type. [59]

4.1.3 Android

There are two ways of sending an SMS message using the Android API; either by creating an Intent and open the standard SMS application with some pre-defined values, or by calling the SmsManger API. The API has support for sending text based messages, either as a single message or, if the message is too long, in multiple parts, as well as pure data based messages to a specified application port. [60] [61]

In addition to SMS there also exists a full SIP stack in Android. This makes it easier for developers to create applications that handle VoIP calls, message services or other things that makes use of SIP. This is however not comparable to the simple APIs used for sending SMS. [62].

4.1.4 Maemo

The Maemo platform, being based on a standard Linux distribution, provides message handling through the Telepathy framework. Telepathy is a framework for managing voice, message, and video communication. It also supports file transfers, managing contacts, and online status (presence). Just as webOS Synergy this allows developers to add support for additional messaging services without requiring application developers to explicitly add support for each service. [63]

Telepathy is built on top of D-Bus (Desktop Bus), which is an inter-process communication framework, and all components run as separate processes [64] [63].

On top there is the Mission Control, which provides the Account Manager and the Channel Dispatcher. The Account Manager handles all accounts the user has set up and can initiate connections. The Channel Dispatcher is responsible for dispatching applications upon either remote requests from the different protocols or local requests from other programs, for example to start a chat with someone. [65]

Each Telepathy Connection Manager handles Connections for one or more protocols. A Connection is the connection to the protocol and it contains contacts, avatars and other things. It can also be used to create new channels for text messaging, calling, file transfers, and so on. Telepathy supports multiple clients, and lets them use the same Connections and Channels. [63]

4.1.5 The W3C Messaging API

The Messaging API from W3C defines methods for creating and sending messages of different types, including SMS, MMS and e-mail. The API is meant to complement the previously defined URI schemes. The API does not handle receiving of messages.

4.2 Analysis

Since support for the same feature set that is found in other mobile operating systems is required, the Messaging API from W3C and the use of URI schemes are too limited to be useful. Most of these required features are listed in Figure 4.1. With webOS and Telepathy, developers can extend the messaging platform to support new protocols and services and still have them integrated into the standard APIs. This is an elegant solution and the separation will enable developers to develop services that handle network communications, while other third party developers implement messaging applications that utilizes these services.

- Message
  - Text
  - Service Type
  - Attachments
  - Meta information
- Actions
  - Send
- Notification
  - Incoming messages
  - Outgoing messages
  - Sent or Error
- Launch applications on incoming/outgoing messages

Figure 4.1: Common features for messaging.

Protocols that want to relay their own messages into the system need to implement this in a service. This service needs to be able to create native messages from the protocol's messages, as well as converting native messages to the representation used by the protocol. Services should be careful not to use more of the phones resources (for example battery, CPU, and network usage) then needed. XEP-0286: XMPP on Mobile Devices [66] discusses the XMPP protocol from a battery usage perspective, noting for example 3G radio levels and compression.

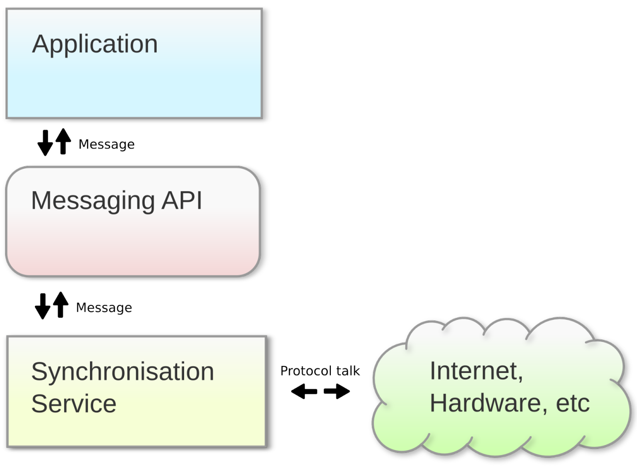

Figure 4.2: Message flow.

For applications, it is preferred that sending and receiving messages should be protocol agnostic. This is to be able to support additional protocols without rewriting the applications. Translation from the application to the services and from the services to the application need to go through a single point. An example of this flow can be seen in Figure 4.2.

All standardized URI schemes, such as the RFC standardized SMS [57], mailto [67] and XMPP [68], as well as the less standardized URI schemes (such as MSN/IM and GTalk) have to be supported to make sure existing web pages and web applications work as expected.

Besides the URI schemes, the Messaging API from W3C also needs to be supported. However, this API is only for sending messages and is missing, among many things, support for reading messages and subscriptions. It is still under development and two vastly different approaches exists – one that is more similar to the WAC approach and one that is based on URI schemes. [69] [70].

4.3 Result

The resulting API is divided into two layers, as can be seen in Figure 4.3. The lower layer (service level) is meant to be used by services to add support for a messaging protocol. The upper layer (application level) is to be used by regular applications that create a user interface for sending messages through the underlying services.

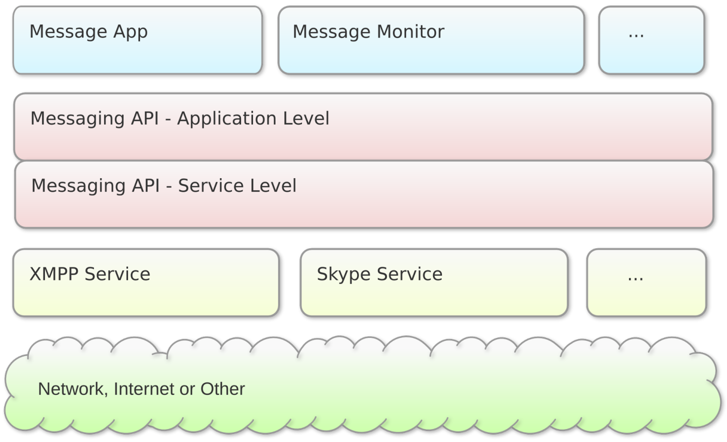
Message App
Message Monitor
...
Messaging API - Application Level
Messaging API - Service Level
XMPP Service
Skype Service
...
Network, Internet or Other

Figure 4.3: Message implementation levels overview.

4.3.1 Data types

Message Different types of messages are abstracted, into this single Message structure. It is used both on the application level and on the service level.

```
Message(To, From, Type, Content)
    To              [String]
    From            String
    Type            String
    Content         String
    Attachments     [File]
Functions
```

MessageState is an indication of which state the message is in, either incoming or outgoing.

```
MessageState = {INCOMING, OUTGOING}
```

4.3.2 Service level

Files

Services that support attachments should read the Attachments property in the Message class and send them together with the message. If the service can not send files, but some are attached, the service should throw an error.

SMS and MMS

The local physical phone hardware will always generate a Message in response to an incoming MMS or SMS. This is a built in service, which is always running inside the operating system, and it abstracts away and hides the complexity of the hardware, and translates it to the application level API just as any other service would do. Direct access to the phone's hardware should not be available to application developers.

SMS messages only support a text length of 160 letters with the 7-bit alphabet, 140 with the 8-bit alphabet and 70 letters using UTF-16. Messages can however be concatenated into one longer message instead. This should be done automatically by this service. Although it is possible to extend SMS and attach files this should not be done by this service. A message with an attachment should not be sent but instead throw an error. [71]

Create a message This function pushes a newly received message to the API, and forwards the message to the application in charge of handling it.

```
createMessage(message)
```

Where:

- `message` is the Message object that was received

Handle Message This function is called when an application or the API wants to know if the service could handle messages of the given type, number format or protocol.

```
bool handleMessageType(type)
bool handleMessageRecipient(number)
bool handleMessageProtocol(protocol)
```

Where:

- `type` is the type of service that is to check whether it is handled by the service.
- `number` is the recipient string (for example a number or a username) to check whether it is handled by the service.
- `protocol` is the protocol string (for example `xmpp` or `gtalk`) to check whether it is handled by the service.

Send a Message This function is called when an application wants to send a message using this service.

```
onMessage(message, onSuccess, onError)
```

Where:

- `message` is the Message object that is to be sent.
- `onSuccess` is the callback that is to be called when the message is sent.
- `onError` is the callback that is to be called if the message could not be sent. This should include a reason for failure; for example if the service cannot handle attachments, or is lacking the permissions required for sending the message.

4.3.3 Application level

Register is a function for registering a function for receiving message and message notifications. The function registered here is called once for each change of state in the message.

```
registerMessageListener(
                function listener(Message message, MessageState state))}
```

Where:

- `message` is the message for which the state just changed.
- `state` is the state of the message. One of the items in MessageState.

Unregister is a function for unregistering a previously registered function.

```
unregisterMessageListener(function)
```

Where:

- `function` is function that were previously registered with the registerMessageListener function.

Send Message Function for sending a message, or rather pushing the message to the underlying service. Before sending a message, the user agent has to prearrange permissions for the application to send messages of the given type.

```
sendMessage(recipient, type, content, onSuccess(message), onError(e))}
```

Where:

- `recipient` is the recipient of the message.
- `type` is the type of message that is to be sent.
- `content` is the message content.
- `onSuccess` is the callback that is to be called when the message is sent.
- `onError` is the callback that is to be called if the message could not be sent. This should include a reason for the failure.

4.3.4 Examples

To further explain the usage, a few examples will be presented. First the service level will be covered with examples on how a service could handle sending and receiving messages. There will also be two examples of application level usage.

Service level

This example will demonstrate the standard routine for a service that receives a message from a remote server, and how this message could be forwarded to the API.

Example 1 Receiving a message

```
[...]

function onPacket(package) {
        if(isCall(package)) {
                [...]
        } else if(isMessage(package)) {
                [...]
                var message = new Message(to, from, type, content);
                message.To      = Myself;
                message.From    = getRecipient(package);
                message.Content = getContent(package);
                message.Type    = MyType;

                createMessage(message);
}
[...]
}

function onMessage(message, onSuccess, onError) {
        sendMessage(message.To[i], message.content);
        [...]
}

function handleRecipient(
        [...]
}

function sendMessage(to, content) {
        [...]
}
```

Application level

The following two examples handle sending and receiving messages using the API.

Example 2 Sending a message

```
function onSuccess() {
        // Message sent correctly
}

function onError(error) {
        // Message not sent
}

sendMessage("+46703283946", "Hello. How are you? Have you"
                + " been alright, through all those lonely lonely lonely lonely"
                + " lonely nights?", onSuccess, onError);
```

Example 3 Receiving and monitoring messages

```
function messageListener(message, state) {
        if (state === MessageState.INCOMING) {
                log("Message from " + message.From + ": " + message.Content);
        } else if (state == MessageState.OUTGOING) {
                log("Message sent to " + message.To + ": " + message.Content);
        }

}

registerMessageListener(messageListener);
```

Chapter 5

Calls

Making and receiving phone calls is one of the central features in mobile phones, and something that all smartphones have to be able to do. Calling includes incoming and outgoing calls with audio, and video, streams, together with other data and events. Besides traditional telephone calls, calling also includes other types of calls, such as Skype, SIP and other instant messaging services.

5.1 Previous work

The `tel` URI describes telephone numbers as a string of decimal digits, which uniquely indicates the network termination point. There are also other URI schemes for voice communication protocols, including XMPP and SIP. [72] [68] [73]

The WebOS, iOS and Windows Phone 7 platforms give developers almost the same functionality with regard to calls. On all three platforms there is no direct way for developers to create applications that place calls. Instead there is functionality for opening the default call application with information already typed in. The user must then confirm, often by pressing the call button, for the actual call to be initiated. On these platforms there is also no way to react to incoming or outgoing calls, or retrieve any information about the cellular network, or even get any information about earlier calls.

WebOS and iOS handle calls by creating a tel URI and launching it with a special method. In webOS this is done using the Application Manager, either by using the open method with the tel URI or using the launch method with the dialer application's id. [74]

In iOS, developer can use the tel URI as a parameter to the openURL function in the UIApplication class, which will launch the phone application with the given phone number already typed in. [75][76]

In Windows Phone 7, third party applications can, through the PhoneCallTask class, set the display name and the phone number shown in the standard phone application, but not handle a call directly. [77]

5.1.1 Android

Calls in Android are initiated by creating an Intent, shown here in Figure 5.1.

```java
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:123456789"));
startActivity(callIntent);
```

Figure 5.1: Making a call in Android

This will, if the application has permission, bring up the Phone application and either call the number directly or show the number, requiring the user to manually press the call button. [61]

Call information, as well as network information, can be gathered using the TelephonyManager class. This class also supports subscription to notifications of incoming telephone calls. [78]

The SIP stack mentioned in Section 4.1.3 is, as specified there, also usable for implementing applications that should be able to handle SIP-enabled calls, even though support for VoIP calls may vary across different Android phones. [62].

5.1.2 Existing Standards

The Video Conferencing and Peer to Peer Communication, a part of the HTML Living Standard, includes multimedia streams for video and audio, as well as methods for handling, creating, and displaying these. It also specifies different network and P2P functions, with the aim of sending and receiving streamed media between peers. [79].

5.2 Analysis

The characteristics of a Call API would have much in common with the Messaging API (see Chapter 4); both need notifications of incoming and outgoing traffic, be able to send and receive calls or messages and have information and service specific information attached to it.

Instead of text messages, the base of calls is the sound or video streams, and the Video conferencing and peer to peer communications standard from WHATWG can be used to add support for these streams. [79]

As with messaging, there are a lot of different services and protocols which support calls and developers need support for adding new types of communication protocols.

Since it is expected that various URI schemes should work, support for this needs to be included in the API. Services such as Skype often have their own URI scheme, which should also be supported if that service is supported on the device, meaning that developers have to be able to add additional URI schemes for supporting their application. [80][81]

Regarding the functionality and information required for placing, handling, or reacting to a call, the functionality showed in Figure 5.2 is considered to be the basic functionality an API must provide in order for third party applications to fill the need of a native phone application.

- Calls
  - Voice or Video
  - State (Ringing, On Hold, Ended)
  - Caller
  - Callee
  - Duration
- Actions
  - Call
  - Hang Up or End call
  - Hold call
  - Resume a previously held call
  - Reject call
- Notification
  - Incoming call
  - Outgoing call
  - Call State change
- Service Related
  - Service Type
  - Cell Tower
  - Cell ID
  - Carrier

Figure 5.2: Common attributes and features for calling services.

5.2.1 Incoming calls

Incoming calls come in through different services and need to be handled by the system centrally. An incoming call needs to generate a notification, just as for messages, but there must also be ways to take an action on the call – answer it or hang up, for example.

There is also additional information that needs to be present together with the call. This includes who is calling, what type of call it is, and other important service specific information.

5.2.2 Outgoing calls

Outgoing calls share most of the requirements for incoming calls. Notifications and actions are required to work here as well.

Applications should have simple methods for calling different numbers without having to worry about which protocol or service that could handle the call, without losing control to an unknown implementation.

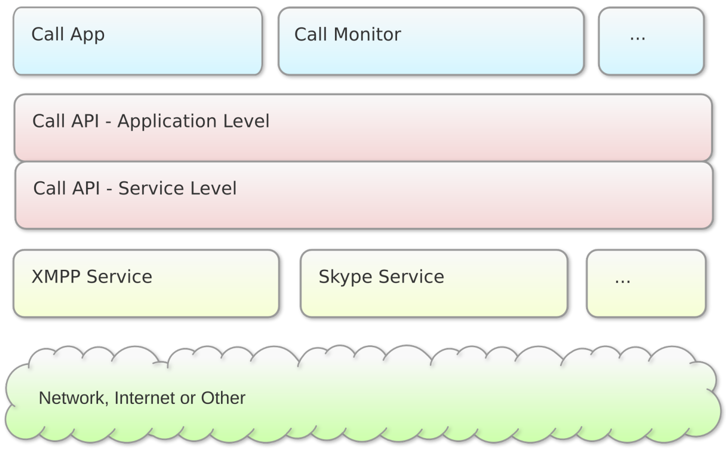

Figure 5.3: Call implementation levels overview

Figure 5.3 shows how the different implementation levels interact. The services handle incoming and outgoing calls directly to the network, which means that they are responsible for communication and call setup between the device and a distant server. In a way, the service can be seen as a translation between the API specification of a call and the individual network representation. The service handles all the setup that is required to initiate and handle a call.

The Call Application in Figure 5.3 represents the application interacting with the user. If the user needs to make a call, the Call Application does an API call that directs information to the underlying service, the service then handles the connection and returns information to the application when the call is ready to be initiated.

5.3 Result

The resulting call API will work as a layer between services and applications, where services handle network and protocol details, while the applications handle user interaction and whatever task it is designed to do.

5.3.1 Data types

```
Call
        Call(inputStream, outputStream, hangup, answer, hold, unhold)
        Hangup(function onSuccess(), function onError())

        getInputStream()
        getOutputStream()
        setInputStream(stream)
        Type
        getNumber()

Incoming Call: Call
        Answer(function onSuccess(), function onError())

CallState = {RINGING, CALLING, ANSWER, HOLD, UNHOLD, HANGUP}
```

Where incoming call are denoted as RINGING, while outgoing calls are denoted as CALLING.

5.3.2 Service level

Services are used to translate low level call handling to the application level. The functions listed in this Section would only be available to services.

The local physical phone hardware will always generate a normal Call object on incoming calls. All actions and streams will also work as expected. This can be thought of as a static service translating from the hardware to the API and vice versa.

All new types of communication will have to generate their own Call object based on incoming traffic to their own service.

Incoming call The `createCall` method is not a constructor, but a method that pushes an incoming Call to the API and forwards it to the application in charge of handling it.

```
createCall(call)
```

Where

- `call` is the Call object that the service has created based on the received data.

Handle There are three methods that a service has to implement in order to tell the API what capabilities the service provides; whether it can handle a specified number, protocol, or call type.

```
bool handleCallType(type)
bool handleCallNumber(number)
bool handleCallProtocol(protocol)
```

Where:

- `type` is the type of service to be checked if the service could handle.
- `number` is the recipient string (for example a number or a username) to be checked if the service is able to handle.
- `protocol` is the protocol string (for example XMPP or SIP) to be checked if the service is able to handle.

Outgoing call The `onCall` method should be implemented by the service in order to be called when an application wants to make a call. In this method the service has to add an outputStream to the call parameter, so that the stream could be utilized by the application.

```
onCall(call, onSuccess, onError)
```

Where:

- `call` is the call to be placed.
- `onSuccess` is the callback that is to be called when the call is placed.
- `onError` is the callback that is to be called if the call could not be placed. This should include a reason for failure. For example if the service cannot handle video streams, or is lacking the permissions required to place the call.

Common functions The `onHangup`, `onHold`, and `onResume` functions are called when an application has requested a change in the state of the call.

```
onHangup(call, onSuccess, onError)
onHold(call, onSuccess, onError)
onResume(call, onSuccess, onError)
```

Where:

- `call` is the call to change the state for.
- `onSuccess` is the callback that is to be called when the call state is changed.
- `onError` is the callback that is to be called if the call status could not be changed. This should include a reason for failure. For example if the service cannot handle the requested status, or if the application is lacking the permissions required to place the call.

The `updateCall` functions can be called by the service if it wants to change the state of the call. This is usually done when the other end has hung up.

```
updateCallState(call, CallState)
```

Where:

- `call` is the call to update state for.
- `CallState` is the new state of the call.

5.3.3 Application level

Notifications The `registerListener` method will let developers register a function for receiving calls and notifications. The method inserted will be called once for each state change of the call.

```
registerListener(function listener(Call call, CallState state))
```

Place a call The `placeCall` method is used in order to place a call to a given number.

```
placeCall("number", inputStream,
        function onSuccess(Call), function onError(e))

placeCall("number", type, inputStream,
        function onSuccess(Call), function onError(e))
```

5.3.4 Examples

This section includes examples of how the API can be used; both how to place a call, receive a call, or how to implement a simple service for introducing a new protocol.

Example 4 Placing a Call

```
navigator.getUserMedia('audio,video user', gotStream, noStream);

function gotStream(inputStream) {
        call = placeCall("+46730455070", inputStream, answer, error);
}

function noStream() {
        // Failed to get stream
}
function answer(call) {
        outputStream = call.getOutputStream();
        // Phone call is active...
        call.hangup()
}

function error(call) {
        // An error occured
}
```

Example 5 Receiving a Call

```
var activeCalls = {};

function onRinging(call) {
        var outputStream = call.getOutPutStream();
        // Set speaker to play outputStream

        navigator.getUserMedia('audio,video user',
                function(inputStream) {
                        call.setInputStream(inputStream);
                        call.answer();

                        // Call is active...

                        call.hangup();
                },
                function() {
                        // Could not get stream
                }
        );
}

function listener(call, state) {
        activeCalls[call] = state;
        if(state == CallState.RINGING) {
                onRinging(call);
        } else if(state == CallState.HANGUP) {
                delete activeCalls[call];
        }
}

call.registerListener(listener);
```

Example 6 Simple Service

```
[...]

function onMessage(message) {
        if(isCallMessage(message)) {

                inputStream  = getInputStream(message);
                outputStream = getOutputStream(message);

                call = new Call(inputStream, outputStream,
                        onAnswer, onHangup, onHold, onUnhold);

                createCall(call);

        } else {
                [...]
        }
[...]
}

function onAnswer(call, onSuccess, onError) {
        [...]
}

[...]
```

Example 7 Simple Service for Receiving Calls

```
[...]

activeCall = null;

function onCall(call, onSuccess, onError) {
        placeProtocolCall(call.getNumber(),
                function(stream) {
                        call.setOutputStream(stream);
                        activeCall = call;
                        onSucess();
                }, function(error) {
                        onError(error);
                }
        );

        [...]
}

function onHangup(call, onSuccess, onError) {
        hangupProtocolCall(call);
        onSuccess();
}

[...]

function onMessage(message) {
        if(isCallMessage(message)) {
                isHangup(message) {
                        updateCall(activeCall, CallState.HANGUP);
                }
        }

        [...]
}

[...]
```

Chapter 6

Settings

Changing settings is a common use case on modern smartphones. It is however not as common to let third party developer implement the user interface for it. The DAP Working Group has no plan of developing any APIs for application configuration or application specific settings and informs that this kind of configuration can be accomplished by using localstorage and with the Widget interface APIs.

The question of a interface for global device settings, such as static IP on the wireless connection or time zone and presentation format, remains.

6.1 Previous work

Neither iOS nor Windows Phone 7 includes any public API solutions that enables third party developers to change global settings on the device. iOS has interfaces for managing volume control, called MPVolumeView [82], but there is no general solution that spans over multiple settings environments, as in Android or webOS.

6.1.1 Android

Android includes an extensive API for changing system settings. It is built as a set of key value pairs, where the key describes what functionality the value holds [83]. The API is divided into two parts, one containing settings to which third-party software can gain only read access [84], and one part containing non-vital settings that third party developers may access and change [85].

To provide a better overview, the API includes a list of predefined keys that cover most settings and preferences that the built in applications and services are using. The read-only API, called Settings.Secure, contains settings for a global http proxy, if the device is able to install applications from other sources than Android Market, whether the USB mass storage is enabled, whether the Wi-Fi should be on or off, a list of Service set identifiers (SSIDs) for which the user has agreed to connect to, and so on. The settings in this API can only be modified through the system user interface. [84]

The other part, called Settings.System, contains the preferences that applications, that have permission from the user, can change or even add new settings to. Here, the predefined list of settings can contain configuration options for the air plane mode, Bluetooth timeout, the date and time format, screen backlight brightness, timeout before the screen turns of when inactive, click sound effects, vibration on and off, all kinds of volume, Wi-Fi settings, and so on. [85]

The Settings.System API can also be extended with additional keys and values by having applications adding keys that are not already defined. These settings will work as any predefined settings, and will be accessible for all applications that have permission to use the settings API. [85]

6.1.2 webOS

In webOS settings are handled as services through the com.palm.systemservice API [86]. The settings are divided into groups, where each group is a service that an application can pass messages to.

The available services, and what kind of preferences they are holding, are:

- Location/Time information – holds settings for what format date and time will be presented and the region and time zone.
- Sound/Tones – could be used to change ringtones, volumes or other sound related settings
- Generic Information – currently holds settings for airplane mode, where network and phone parts of the device are turned off
- Carrier Information – holds the name and home page of the cellphone carrier

- Lock/Timeout Preferences – keeps the timeout settings for when the screen should lock when inactive, if alerts could be shown even if locked, and so on.
- Browser Preferences – contains a list of available web search services available, as well as a setting for the default one

As with the Android API, applications can define their own keys on demand, which enables implementations of similar applications to share settings between each other without having to specify which kind of application that can use the information.

6.2 Analysis

The solutions presented in Section 6.1 had much in common. Both have a very basic and easy to use key-value pair structure for keeping and distributing system settings to applications. Both are also easily extendible and have similar ways of handling application permissions.

The way of splitting preferences into different groups depending on the setting characteristics could enable a more extensive way of handling applications' access to settings. For example, it would be a useful security feature to have permissions to not only use the Settings API, but have the user agree on access to each individual group and not let applications access groups outside of its responsibility.

6.3 Result

By studying the different solutions and what preferences should be available, the following groups are proposed:

- Lock timeouts and limitations – time of inactivity until the device should lock and whether to allow notifications when locked.
- Notifications – settings for what events should generate a notification.
- Sound, volumes, ringtones and vibration settings – for incoming phone calls, received messages, alarms, and other notifications.
- Carrier information – information about the cell phone carrier.
- Location, date and time settings – for example time and date format and synchronization settings.

- Network settings – for static IP, gateways, DNS, search intervals, and other network related settings.
- Screen settings – for example screen resolution and backlight strength.
- Call settings – for example whether to send caller id on outgoing calls, if data roaming should be enabled, or what PIN code should be used on the device.
- Bluetooth and related transmission technologies – settings related to Bluetooth, IR, RFID, FM radio, or other such technologies that might be available on the device.
- Browser preferences – various web browser settings for handling local storage, cookies, and so on.
- GPS settings – settings for GPS, or the location server for network positioning.

Exactly what attributes each category should contain is not specified here but all examples above should be covered.

Regarding security and permission handling, a user agent should not allow changing settings without permission from the user. How this is handled is up to the user agent, but a separate permission for each category would be preferred. Further discussion regarding permission handling is found in Section 10.3.

Chapter 7

System information

This section will consider information about the device, operating system and vital hardware parts and how such information should be accessed. Examples of such information could be for example current memory usage, the name and version of the operating system and what capabilities the device has.

Much of this information can already be obtained by using standard methods, either by standardized scripting APIs, as with current time and date, or with other Internet standards, for example using the client's User Agent to transmit operating system name and version. The W3C Device APIs and Policy (DAP) Working Group have been working on an API to gather Systems information and events, as seen in Section 7.1.6. This chapter investigates whether the W3C standard is sufficient enough for a web based operating system on smartphones or if another API will have to add additional functionality.

Note that this section does not handle local file storage or anything else that could be abstracted out.

7.1 Previous work

Most operating systems for smartphones regard memory information as something that is supposed to be handled exclusively by the operating system itself, and does not like to release that information to any other application. That is why Android and iOS for example, discourage developer from detecting lack of memory, or calculating the amount of available memory on the device. This is however not something that could be assumed for any operating system, and that is why this kind of abstraction cannot be done in this paper.

7.1.1 iOS

The iOS API can be used to retrieve information about the local device by using the UIDevice class. According to the API it can be seen as a singleton representation of the device at hand, where applications can get hold of vital information about the device itself. [87]

The UIDevice class has divided the properties into five different groups.

- Available Features – currently only hold a property to tell whether multitasking is supported on the device or not. The reason for not including additional features is because of the fact that the developer already knows what brand of device he is working on, and can make assumption based on that fact. Such assumptions will not be possible for a more general API that has to support all possible kind of devices.
- Device and Operating System – contains properties for identifying the device, such as a unique id, the model name, and what kind of operating system it is running. Based on this information the application is able to accommodate its services for the current device.
- Device Orientation – enables applications to find out the device's physical orientation, in order to tilt the display to accommodate the user interface with the user's view. This category also contains notifications of device orientation, so that applications can be automatically notified when the screen needs to be tilted.
- Device Battery State – handles information about the current state and level of the device's battery. The state tells the application whether the device is currently charging, unplugged or fully charged, and the level shows the current charge on the battery in percentage.
- Proximity Sensor – indicates whether or not the proximity sensor senses a close object.

This is what Apple has added through their API, but much of the needed information is naturally retrieved by the choice of using objective-C as base. Monitoring the system and the network connectivity is something that is already built into objective-C and that does not require an additional API from Apple. [88]

7.1.2 Android

The Android API utilizes the Java programming language and, in the same way as iOS, it can get much system information directly from the language itself. [89]

As stated in the beginning of this section, the Android API discourages applications from handling memory information directly, but instead relies on the operating system itself. The application can however retrieve the information, using the ActivityManager, as well as tell the system that the memory level is too low and that the system should consider itself as being in a low memory situation. Note also that, for debugging purposes, a more detailed view could be retrieved directly from the kernel. [90]

7.1.3 Windows Phone 7

Since Windows Phone 7 has a memory cap of 90 MB on any application that runs on a device with less than 256 MB of total memory, it is necessary to let applications keep track of memory usage. The DeviceExtendedProperties class keeps properties for both the amount that the application is presently using, the maximum amount of memory the application has used during its lifetime, and the total amount of memory on the device. This class can also be used to get device specific properties such as a unique id, the manufacturer and name. [33]

7.1.4 WAC

WAC specifies a Device API by having access methods in the Device Status module fetch information based on pre-specified attributes, listed in the Device Status Vocabulary. The vocabulary groups attributes together in the following groups:

- Battery – contains attributes for current battery level, and whether the battery is being charged or not.
- CellularHardware – tells whether cellular hardware is available or not.
- CellularNetwork – contains current signal strength, roaming capability, and type of operator of the cellular network in use.
- Device – contains information about the device, such as model number, version and vendor.

- Display – equivalent to the non-standardized Screen Object and contains screen information for the device.
- MemoryUnit – tells the total memory size, as well as the amount of free built in and/or removable memory on the device.
- OperatingSystem – contains information about the operating system: language, version, vendor, et c.
- WebRuntime – represent the current web runtime, with information about the available WAC version.
- WiFiHardware – tells whether the device can be used to connect to Wi-Fi networks.
- WiFiNetwork – contains signal strength, network status and SSID of the current Wi-Fi connection.

7.1.5 PhoneGAP

The PhoneGap framework actually includes an API for getting system information, even if the API is very limited. The framework documentation mention properties for getting the device's name, unique id, platform, operating system version, and version of the PhoneGap API available.

The idea behind the PhoneGap information API is that the developer will only support known devices that are known to have the features required to run a specific application. This makes developing for a pre-specified platform, for example iOS, quite easy, since all iOS devices have very similar hardware features. Problems will however arise when developing a multi-platform application, where availability of some hardware component might not be certain and the device has to manually check for it. That is not possible with the PhoneGap solution. [91]

7.1.6 W3C and the System Information API

The System Information API divides all properties and information into a number of different areas, all with restricted access level that each application has to receive permission from the user in order to access, based on the characteristics of the properties. The standard is still only a draft, but in the latest public version, dated February 2nd 2010, the different areas are described in Figure 7.1, which also gives a clear overview on what information each group contains. [92]

The early work of the System Information API did also include battery status properties, but these have been moved to the Battery Status Event Specification. The specifications do not only include methods for fetching battery information, but also introduce a way of setting an event listener that will receive continues updates of battery level and charging status. [93]

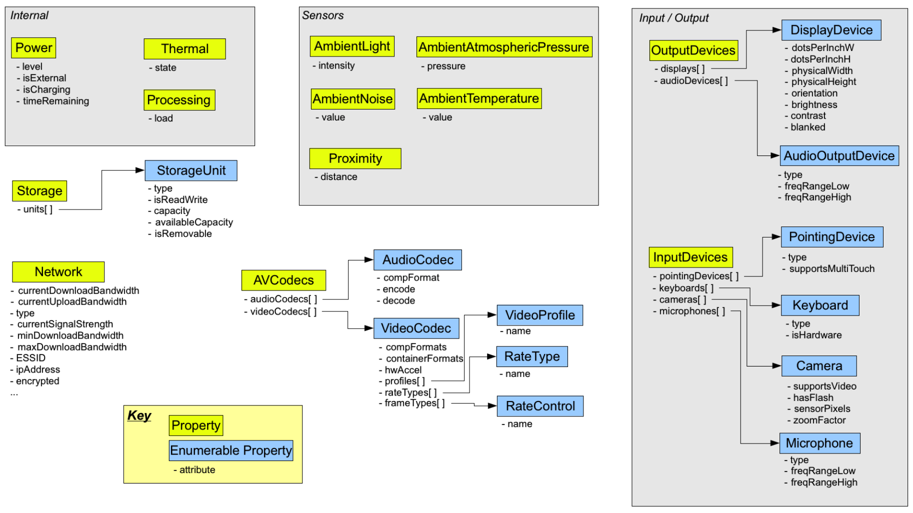

Figure 7.1: An overview of the System Information API [92]

7.2 Analysis

The handling of system information on current smartphone devices are all limited or simplified by assumptions made by the targeted platform. In the case of Windows Phone 7, there is no way for applications to ask the device if a GPS chip is available or not. The developer can however assume that a GPS is always present on a Windows Phone 7 device, since that is one of the requirements made by Microsoft when selling a license. Some simplifications can be drawn when developing applications for Android and iOS as well. These kinds of solutions are however only viable for platform dependent development. When developing the same applications for multiple platforms, it quickly becomes infeasible to handle pre-defined rules about each platform. [33]

The only solutions studied in this paper that do not rely on presumptions made according to the operating system are WAC and the W3C proposed standard. When comparing the two, they both have quite similar properties and work in the same way. Note that the W3C proposal does not state how to access the device, but only provides information about the device and its parts.

7.3 Result

The WAC API and the W3C proposed standard complement each other well. An operating system providing support for both of these would cover all necessary properties for application development in modern smartphones. It would provide a fully usable interface for developing web applications that covers everything from system monitoring to accessing information about input and output devices.

Chapter 8

Multitasking and Application handling

According to Maximiliano Firtman's definition [33] a device needs, among other things, a multitasking operating system in order to be categorized as a smartphone. Other definitions might have a different opinion, but it is clear that multitasking is one of the most important features in modern smartphones and all operating systems studied in this paper have, if not full then at least some, multitasking capabilities.

8.1 Previous work

There are different kinds of multitasking, and almost all vendors have come up with different solutions to balance the battery life, functionality and user friendliness in their operating system. Applications can be minimized, and the user can switch between them, or a process might be running completely in the background where they might always be able to get some processor power. Some operating systems have an automated system for shutting down applications when processing power and memory is starting to get low. [94]

8.1.1 iOS

As of iOS version 4.0, having an application running in the background is no longer limited to first party developers. The developers do however need to work through specially designed APIs to achieve this functionality, and even then the background applications have restrictions that limit their access to the phone's resources.

The applications that need support for accessing resources while running in the background have to register to some of the available services to handle the multi-threading for them. [75] If all an application needs is to finish its current task, like downloading a file, the application can register the task with the Task Completion service and the task could finish even if the application is put to the background. When the task has completed, the application will receive a notification to display to the user.

If all an application needs is to finish its current task, e.g. downloading a file, the application can register the task with the Task Completion service and the task could finish even if the application is put to the background. When the task has completed, the application will receive a notification to display to the user.

Another service, called Background audio, allows applications to continue playing audio when put to the background. This service is commonly used to implement an Internet radio or a music player that needs to be able to play audio even when the user has switched application.

For applications that need to handle incoming network messages, to implement a messaging service, like Skype or GTalk, they may use the service called Messaging service. Here, developers can register a message handler that will receive new incoming messages of a certain type and notify the user accordingly. It is also possible to enable a timed event to occur even in a background application by calling the service Push Notification. The service will alert the application when the timer has run out and execute the timed routine.

The last background service available for applications is the Background Location Service, which enables applications to track the user's movement by enabling the GPS even for programs running in the background. [75]

By having such restricted background access, the operating system does not need to implement a full featured thread scheduler and can save both computation power and battery life, but still give the user the feeling of a multitasking operating system.

As of iOS 4 the operating system also includes an application switcher where users can bring up a list of hibernated applications and let the user browse between them. There is however no native API for letting developers create their own application switcher, manage installed applications or list the services running on the device. [95]

8.1.2 Android

Android has a different approach when it comes to multitasking and switching between applications. The idea is that the user does not have to know whether an application has turned off or is still left in the memory; the operating system will handle that automatically. That is also why the Android operating system does not include an application switcher that lets the user switch between open applications.

Multitasking in Android basically works by having the operating system not turning off applications when the user switches to a new application. Instead, it leaves the hibernated application in the memory, until the memory is needed by another application. By letting the applications stay in memory, the applications will be much faster to start up again. Android gives developers the ability to register its applications as services, giving the application access to actually execute code while running in the background. Since Android 1.5, services that run in the background are limited to using 5-10 percent of the total CPU. This increases both the availability of the operating system but also the battery life of the device. [94]

To free up memory, as more and more applications are left in memory, Android implements an Out of Memory (OOM) killer. The routine works by having two memory thresholds. When the first threshold is met, the background processes are notified and asked to save their state in the persistent storage. When the applications have saved their state, they return back to the OOM routine which, when the second threshold has been reached, starts to turn off the non-critical applications whose state has been reported as saved. Since all applications save their state, they will be able to return to the same state as before when launched the next time. This means that the only thing differentiating between an application that has been turned of and an application that is still running is the time it takes to launch the application. [96]

API

The Android API for interacting with running applications and services, the ActivityManager API, separates applications and services by defining separate namespaces and separate methods for each of them. The API provides methods for fetching a list of running applications or services or, if the application has got permission from the user, shut down a specified application by terminating its processes. It is also possible to get a separate list for applications that are in an error state, or a list of the most recently started applications that include applications that are no longer running.

The API can be used not only for listing running applications, but also to fetch more specific information about each running process; for example the process' PID, its memory usage and how long it has been running for. An application that is in an error state contains a little bit more information, such as the error message and a stack trace.

8.1.3 Windows Phone 7

Multitasking and having applications working in the background is, in Windows Phone 7, limited to selected third party and first party applications only. Regular applications are limited to receiving and sending notifications which means that there is, for example, no way for those applications to continue playing music when running in the background. The application can use the received notifications in order to; for example, change the icon tile to display the information dynamically. Microsoft had said that they have plans of extending the multitasking capabilities for third party applications, as well as introducing ways of switching between open applications, but has yet to define how. That is also why their API cannot be studied in more detail in this thesis.

When a new application is launched, the old application will get a notification asking it to save its current state in the device's memory and hibernate. The procedure is similar to the solutions in iOS and Android where applications can be turned off by the operating system to free up used memory blocks. Saving the state of each application will enable applications to resume in the same state as when the operating system turned them off.

Windows Phone 7 does not include any native APIs for application management; this is only supposed to be handled by the operating system or first party applications.

8.1.4 webOS

The webOS approach to multitasking is also a bit different. Here the user has full control over which applications that are running and may turn them off at any time. The user interface consists of *cards*, which is basically a rectangular screenshot of the application that the user may flick through. To turn off an application, the user simply grabs the application's card and throws it off the screen. [97]

Open web applications that are running in the background are limited to only accessing operations that are considered to use a moderate amount of memory and CPU to keep the battery from draining too fast. Constant data requests, or accelerometer access, are examples of prohibited usage. Applications using the PDK are also a bit more restricted when running in the background and may no longer allocate more memory or use the graphic APIs.

Applications that are turned off by the user are typically completely shut down by the operating system, but the user also has the possibility of letting the application stay alive even after it has been terminated. This is done by putting the application icon in the webOS Dashboard, making it a Dashboard Application that, if implemented to support it, may run as a service even after it has been turned off. [97]

If the user has too many application cards open, and the operating system runs out of memory, the user will be prompted with a warning and forced to turn off applications in order to be able to continue using the device. This leaves the user in full control over which applications that should continue to run, and which should be shut down. However, there is no indication of how much memory each application is allocating and the user might close low memory applications unnecessarily, before finding an application that has allocated a larger amount of memory. [94]

8.1.5 Maemo

Multitasking in the Maemo operating system is more similar to personal computer's operating systems and is one of the few, or possibly the only, operating system for smartphones to implement real memory swapping as well as a desktop like interface for application switching. It also includes full multi-task capabilities for minimized applications, enabling all applications to continue to run even when the user has minimized them or opened some other application. The applications receive notifications, stageActive and stageDeactive, when minimized or activated so they can optimize the CPU and memory usage accordingly.

The application switcher is displayed whenever the user presses the menu button. From here the user can choose a minimized application to open or press the menu button again to display the application launcher, with a list of installed applications.

8.2 Analysis

As seen in the previous sections, there are a wide variety of solutions to multitasking on different smartphone platforms. The problem whether to minimize or close applications could be solved in different ways and is something that should be up to the platform to decide; it could be seen as a question that depends too much on memory capacity of the device and the implementation goals of the vendor for it to be answered in this thesis. Guidelines and general tips on what kinds of simplifications that could be made when exclusively working with web based applications however, could fit the scope, as well as a small proposal on which services should be provided in order for the applications to work in the background and optimize memory usage.

Something common with all solutions, previously mentioned in this chapter, is that an API for getting information about installed and running applications is practical to allow applications to handle listing, switching, launching, closing, installing, and removal of applications. The question in this case however, is whether the operating system should allow such tasks to be carried out by web applications at all, or if this is something that is best handled in the operating system itself. Web pages are normally sandboxed inside the web browser and not aware of any other open applications or tabs. Introducing such a feature would break this fundamental security feature that the Web is built on today [98]

Installing applications, such as extensions and add-ons, is something common in modern desktop browsers and even some mobile browsers, today and the installation schema should be very similar. By comparing the installation procedures of installing applications on smartphones with installing extensions in browsers, it can quickly be concluded that the cases are very similar. This means that installing applications could very well adopt a similar behavior to how extensions are currently handled today.

Removing extensions is another question, since removing an application or extension means exposing installed application data to other applications, which could be a serious problem of integrity. Browsers have dealt with this problem by not allowing extensions to list or remove other extensions and most smartphones only allow the built in applications to handle such tasks. [33]

Since web based scripts are running in a sandboxed environment, there is currently no API for handling application switching or list any running applications, with the current web standards, at least not by the definition of an application used in this thesis. It is possible for applications to get information about what is happening, for example if an application has lost focus and been minimized or if it is about to be shut down. But having applications fetching information about other running applications is a question of integrity and in order to find a definite answer, further studies are needed and the question is unfortunately not something that could be answered in this thesis. This means that a safe way of proceeding with this could be possible.

8.3 Result

Multitasking is required in modern smartphones, and everybody is doing it to some degree. To maintain full compatibility with current web applications the operating system needs, at least regarding this topic, to function as a standard web browser.

Since compatibility with existing web applications is wanted, all scripts in the application need to continue to run even while the application is not active. This does not mean that full priority needs to be given to the background applications. A smaller subset of the full performance should be sufficient, just as is done in webOS. [97]

It is not necessary to render anything for the applications that are not visible. If a thumbnail of the application is needed, one can be taken when the application is going out of view. [97]

Web applications that supports being able to be automatically terminated by the operating system, and then resumed in the same state, can be created today by using onLoad and onUnload/onBeforeUnload events. Applications can receive updates for when they are being started or shut down. Since support for pre-existing web applications is wanted, it is hard to implement the solutions that Android, Windows Phone 7, and iOS have implemented. The solution in webOS however, is reasonable.

Swapping out memory to the hard drive, as has been done in desktop operating systems for many years, is a solution that has proven to work well in Maemo. Moving the memory swapping up the chain, into the web browser, would probably mean that some heuristics could be applied to make it more effective. [33]

Chapter 9

Discussion

The smartphone market is growing, and together with continuous improvements of desktop web applications the need for further development of an open mobile web is growing. Using open web standards means that the resulting application could be platform independent, and the same application can be deployed on several markets directly at launch.

During this thesis work, some of the most common fields of smartphone usage have been studied and even if all areas in this thesis is not to be considered as fully solved, this thesis has aimed to get a step closer to a mobile web. How different problems should be solved, and what solution is preferable, is often a matter of opinion and the solutions presented in this thesis are only to be considered as proposals.

The proposals described have been designed based upon common usage on multiple platforms, both mobile and not, in order for them to be fully compatible with how developers are expected to work with similar APIs and what kind of functionality that should be included.

There has been much previous work done in this field, and much have happened even during the thesis period; the Enyo framework was only a concept and an internal framework in the beginning of the year but is now available in a preview version in the webOS Early Access program for developers.

Frameworks such as Mojo and Enyo are however not to be considered fully open standards, and have therefore only been used as guidelines or concepts in this thesis. The Device APIs and Policy (DAP) Working Group has however published a number of drafts with promises for the future.

A few things that were meant to be a part of this thesis have been excluded for various reasons. One such area is sound and video streaming, and the APIs are needed to fully implement a media player as a web application. This is an area that has already received a large amount of the web community's focus and, even with the problems with the media codecs [99], the media playing elements discussed in Section 1.5.5, have shown potential for the future. [100]

Chapter 10

Further Work

In this thesis, some of the tasks commonly used in smartphones have been discovered to be able to utilize many of the common web techniques, even if they in some cases have needed an extended API. There is however much left to study before any attempts on launching an operating system only capable of running web applications. Some things where not covered at all in this study, and some where only partly studied.

In this chapter, some areas where future studies are needed will briefly be presented; what has been done in the past, and what needs to be done in the future?

10.1 Media capture

The W3C DAP Working Group has published a specification for providing file access to media capture devices. The specifications do however not cover streaming those media, and instead only provides the media as a file. This could be sufficient for an ordinary camera application, but without providing access to the stream itself it would never have the same functionalities as a native application.

There has been work in standardizing access to a device's audiovisual parts. The <*device*> element, for example, was introduced for allowing streaming access to video cameras and microphones on mobile devices, but is no longer included in the HTML 5 draft. Instead, the *getUserMedia* method was introduced, with a clear focus on video and audio streaming. [28] [101]

10.2 Calendar

It is fairly common today to use a mobile device as an alarm clock or event scheduler. On phones and other devices, people expect to be notified of occurring events even when the phone is turned off. This is currently not possible in any open standards, and even though it might not be a necessity for most users, it still is a feature that is commonly used on modern smartphones, and thus needs to be investigated further.

Accessing calendar events is fundamentally related to contact information (see Chapter 2) and other similar virtual resources (see Section 1.5.12); it is related when it comes to required methods, such as adding, altering, removing and retrieving items, as well as the synchronization procedures and security and permissions requirements.

The W3C DAP Working Group has begun working on a calendar API. A notable difference between calendar events and phone book information is that there is no need for merging events together, since calendar items in different calendars are commonly displayed separated.

10.3 Distributing applications

Some interesting development areas that have not been covered in this thesis are problems related to packaging, updating, distributing, and installing web applications on mobile devices.

There are some market places that specialize in distributing web based applications. Google has a Chrome Web Store for its desktop browser, Apple has a category for it in their App Store for iOS devices, and Opera has Opera Widgets for web based widgets.

The problem of permissions has been partly covered in this thesis; how packaging and a well formed installation process can handle a web application's permissions to access certain APIs. Instead of having one permission request for each action that is performed, as is common when visiting web pages, an application installation process with knowledge of what APIs the application will access can utilize this information and only prompt the user with one request. This procedure is currently in use in several platforms, such as Android and Chrome Web Store. [102]

Section 3.3 contains an additional example how metadata could be used during the installation process; how applications or services could register trigger methods or event handlers when being installed.

10.4 Managing and browsing the local file system

Due to fundamental sandboxing of web based scripts, there is no way to access the local file system. The File and FileReader APIs enables applications to access selected files on the local file system, but the selection is handled by the user agent. Through the FileWriter API, web applications also have the ability to save files to the local file system, even though this API is currently not widely implemented. [23] [24]

This solution, together with the local storage solutions presented in Section 1.5.7, could be proven to be sufficient but would require more investigations when regarding mobile devices.

List of Figures

Bibliography

[1] Rigo Wenning. W3c web standardization. http://www.w3.org/Talks/2009/01-rw-brux-ssoku/ (2011-05-09), 2009. 2

[2] Ian Jacobs. World wide web consortium process document. http://www.w3.org/Consortium/Process-20010719/process.html (2011-05-09), 2001. 2

[3] Dean Hachamovitch. Native html5: First ie10 platform preview. http://blogs.msdn.com/b/ie/archive/2011/04/12/native-html5-first-ie10-platform-preview-available-for-download.aspx (2011-04-13), Apr 2011. 3

[4] W3C. Offline web applications. http://www.w3.org/TR/offline-webapps/ (2011-05-09), 2008. 3, 9

[5] W3C. HTML 4.01 Specification – W3C Recommendation 24 December 1999. ECMAScript Language Specification, 1999. 3

[6] David Höffer. Mobile web standards evolution vector. http://en.wikipedia.org/wiki/File:Mobile_Web_Standards_Evolution_Vector.svg (2011-05-03), July 2007. 4, 81

[7] Ecma International. ECMAScript Language Specification. ECMAScript Language Specification, 2009. 5

[8] Vic Gundotra. Google's html 5 work: What's next? Google I/O – Keynote 2009, May 2009. 5, 6, 81

[9] IDC. Number of mobile devices accessing the internet expected to surpass one billion by 2013. *IDC Press Release*, Dec 2009. 6

[10] W3C. HTML Living Standard – User interaction. http://www.w3.org/

TR/html5/editing.html#editing (2011-05-06), 2011. 7

[11] Mark Pilgrim. The road to html 5: contenteditable. *The WHATWG Blog*, Mar 2009. 7

[12] W3C. Drag and drop. HTML5 – Editor's Draft 10 May 2011, 2011. 7

[13] W3C. Session history and navigation. HTML5 – Editor's Draft 10 May 2011, 2011. 7

[14] W3C. Html speech xg speech api proposal. HTML5 – Editor's Draft 10 May 2011, 2011. 7

[15] W3C. Introduction to css3. http://www.w3.org/TR/css3-roadmap/ (2011-05-02), 2011. 7

[16] W3C. HTML Living Standard – the video element. http://www.w3.org/TR/html5/video.html (2011-05-06), 2011. 7

[17] W3C. Html canvas 2d context. http://dev.w3.org/html5/2dcontext/ (2011-05-02), 2011. 8

[18] Khronos Group. Webgl - opengl es 2.0 for the web. http://www.khronos.org/webgl/ (2011-05-02), 2011. 8

[19] W3C. Web notifications. http://dev.w3.org/2006/webapi/WebNotifications/publish/Notifications.html (2011-05-02), 2011. 8

[20] Andrei Popescu. Geolocation API Specification. Geolocation API Specification - W3C Candidate Recommendation 07 September 2010, Sep 2010. 8

[21] W3C. HTML Living Standard – web storage. http://www.w3.org/TR/webstorage/ (2011-05-06), 2011. 8

[22] Nikunj Mehta, Jonas Sicking, Eliot Graff, Andrei Popescu, and Jeremy Orlow. Indexed database api. http://www.w3.org/TR/IndexedDB/ (2011-05-18), 2011. 8

[23] Arun Ranganathan and Jonas Sicking. File api. http://www.w3.org/TR/FileAPI/ (2011-05-18), 2010. 8, 80

[24] Eric Uhrhane. File api. http://www.w3.org/TR/file-writer-api/ (2011-05-18), 2011. 8, 80

[25] Anne van Kesteren. Xmlhttprequest – w3c candidate recommendation 3 august 2010. `http://www.w3.org/TR/XMLHttpRequest/` (2011-05-18), 2010. 9

[26] Anne van Kesteren. The websocket api – editor's draft 12 may 2011. `http://dev.w3.org/html5/websockets/` (2011-05-18), 2011. 9

[27] Ian Hickson. Html5 web messaging – editor's draft 12 may 2011. `http://dev.w3.org/html5/postmsg/` (2011-05-18), 2011. 9

[28] Ian Hickson. Video conferencing and peer-to-peer communication. `http://www.whatwg.org/specs/web-apps/current-work/complete/video-conferencing-and-peer-to-peer-communication.html` (2011-05-19), 2011. 9, 78

[29] Stuart Robinson. Multi-core processors to penetrate 45 percent of smartphones by 2015. *StrategyAnalytics*, Jan 2011. 10

[30] Ian Hickson. Web Workers. Web Workers - W3C Working Draft 10 March 2011, Mar 2011. 10

[31] Google Inc. Google chrome os. `http://www.google.com/chromebook` (2011-05-12), May 2011. 11

[32] Amazon. Amazon – samsung series 5 3g chromebook. `http://www.amazon.com/gp/product/B004Z6NWAU` (2011-05-04), 2011. 12, 81

[33] Maximiliano Firtman. *Programming the Mobile Web*. O'Reilly Media, Inc., first edition, 2010. 12, 64, 66, 68, 74, 75

[34] The PhoneGap project. PhoneGap – supported features. `http://www.phonegap.com/features` (2011-05-13), 2011. 12

[35] WAC Application Services Ltd. WAC – About WAC. `http://www.wacapps.net/web/portal/about` (2011-05-23), 2011. 13

[36] WAC Application Services Ltd. WAC – FAQ. `http://www.wacapps.net/web/portal/faq` (2011-05-23), 2011. 13

[37] WAC Application Services Ltd. WAC – Membership. `http://www.wacapps.net/web/portal/membership` (2011-05-23), 2011. 13

[38] HP. Application framework and OS. `https://wiki.mozilla.org/Labs/Contacts/ContentAPI` (2011-03-20), 2011. 14

[39] HP. Hp pre3. `http://www.palm.com/Pre3` (2011-05-04), 2011. 15, 81

[40] HP. Developing Mojo applications. `https://developer.palm.com/content/api/dev-guide/mojo.html` (2011-05-12), 2011. 15

[41] HP. webOS Service APIs. `https://developer.palm.com/content/api/reference/services.html` (2011-05-12), 2011. 15

[42] IDC. Worldwide smartphone market to grow by nearly 50 percent in 2011. *IDC Press Release*, Mars 2011. 18

[43] Nathan Olivarez-Giles. Nokia to cut 7,000 jobs, stop developing symbian operating system. *Los Aneles Times*, April 2011. 18

[44] Google Inc. Using the Contacts API. `http://developer.android.com/resources/articles/contacts.html` (2011-05-09), 2010. 20, 21, 81

[45] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. 22

[46] HP. Contacts. `https://developer.palm.com/content/api/reference/data-types/contacts.html` (2011-04-15), 2011. 22

[47] HP. People Picker. `https://developer.palm.com/content/api/reference/services/people-picker.html` (2011-04-15), 2011. 23

[48] Richard Tibbett. Contacts API. Contacts API - W3C Working Draft 09 December 2010, 2010. 23

[49] Richard Tibbett. Contacts API. Contacts Writer API - W3C Editor's Draft 04 October 2010, 2010. 23

[50] Michael Hanson. Labs/Contacts/ContentAPI. `https://wiki.mozilla.org/Labs/Contacts/ContentAPI` (2011-03-20), 2010. 23

[51] The PhoneGap project. PhoneGap Documentation – Contacts. `http://docs.phonegap.com/phonegap_contacts_contacts.md.html` (2011-05-04), 2011. 24

[52] HP. Developing Synergy Connectors. `https://developer.palm.com/content/api/dev-guide/synergy/overview.html` (2011-04-15), 2011. 30, 31, 32, 36

[53] Google Inc. Android SyncAdapter. `http://developer.android.`

com/reference/android/content/AbstractThreadedSyncAdapter.html (2011-05-17), 2010. 31

[54] Google Inc. Android AccountAuthenticator. http://developer.android.com/reference/android/accounts/AbstractAccountAuthenticator.html (2011-05-17), 2010. 31

[55] Apple Inc. iOS Developer Library – Address Book Programming Guide for iOS. http://developer.apple.com/library/ios/documentation/ContactData/Conceptual/AddressBookProgrammingGuideforiPhone/ (2011-05-13), 2010. 32

[56] The Nielsen Company. U.S. Teen Mobile Report: Calling Yesterday, Texting Today, Using Apps Tomorrow. http://blog.nielsen.com/nielsenwire/online_mobile/u-s-teen-mobile-report-calling-yesterday-texting-today-using-apps-tomorrow/ (2011-05-24), October 2010. 35

[57] E. Wilde and A. Vaha-Sipila. URI Scheme for Global System for Mobile Communications (GSM) Short Message Service (SMS). RFC 5724 (Proposed Standard), January 2010. 35, 39

[58] HP. webOS Service API – Messaging. https://developer.palm.com/content/api/reference/services/messaging.html (2011-05-09), 2011. 36

[59] WAC Application Services Ltd. WAC 2.0 – The messaging module. http://specs.wacapps.net/2.0/feb2011/deviceapis/messaging.html (2011-05-10), Jan 2011. Proposed Release Version (PRV). 36

[60] Google Inc. Android SmsManager. http://developer.android.com/reference/android/telephony/SmsManager.html (2011-05-09), 2010. 36

[61] Google Inc. Android Intent. http://developer.android.com/reference/android/content/Intent.html (2011-05-04), 2010. 36, 47

[62] Google Inc. Session Initiation Protocol. http://developer.android.com/guide/topics/network/sip.html (2011-05-23), 2010. 36, 47

[63] Murray Cumming Danielle Madeley. Telepathy Developer's Manual. http://telepathy.freedesktop.org/doc/book/ (2011-05-04), 2009. 37

[64] Ross Burton. IBM developerWorks – connect desktop apps using D-BUS. http://www.ibm.com/developerworks/linux/library/l-dbus/

index.html (2011-05-04), 2004. 37

[65] Sumana Harihareswara. Telepathy, Empathy and Mission Control 5 in GNOME 2.28. *The GNOME Journal*, Nov 2009. 37

[66] Dave Cridland. XEP-0286: XMPP on Mobile Devices. http://xmpp.org/extensions/xep-0286.html (2011-05-12), 2010. 38

[67] P. Hoffman, L. Masinter, and J. Zawinski. The mailto URL scheme. RFC 2368 (Proposed Standard), July 1998. Obsoleted by RFC 6068. 39

[68] P. Saint-Andre. Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP). RFC 5122 (Proposed Standard), February 2008. 39, 46

[69] Suresh Chitturi, Daniel Coloma, Max Froumentin, Maria Angeles Oteo, Niklas Widell, and Anssi Kostiainen. The Messaging API. The Messaging API - W3C Working Draft 20 January 2011, 2011. 39

[70] Dominique Hazaël-Massieux, Suresh Chitturi, Max Froumentin Maria Angeles Oteo, and Niklas Widell. The Messaging API. The Messaging API -W3C Editor's Draft 05 May 2011, 2011. 39

[71] European Telecommunications Standards Institute (ETSI). ETSI TS 123 038 - Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Alphabets and language-specific information. http://pda.etsi.org/exchangefolder/ts_123038v100000p.pdf (2011-05-24), March 2011. 41

[72] H. Schulzrinne. The tel URI for Telephone Numbers. RFC 3966 (Proposed Standard), December 2004. Updated by RFC 5341. 46

[73] G. Camarillo. The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP). RFC 3969 (Best Current Practice), December 2004. Updated by RFC 5727. 46

[74] HP. webOS Service API – Phone. https://developer.palm.com/content/api/reference/services/phone.html (2011-05-02), 2011. 47

[75] Apple Inc. iOS Developer Library – UIApplication. http://developer.apple.com/library/ios/DOCUMENTATION/UIKit/

Reference/UIApplication_Class/Reference/Reference.html (2011-05-03), 2010. 47, 69

[76] Apple Inc. iOS Developer Library – Apple URL Scheme Reference – Phone Links. http://developer.apple.com/library/ios/featuredarticles/iPhoneURLScheme_Reference/Articles/PhoneLinks.html (2011-05-03), 2010. 47

[77] Microsoft. Windows Phone Development – PhoneCallTask. http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.phonecalltask(v=VS.92).aspx (2011-05-02), 2011. 47

[78] Google Inc. Android telephonymanager. http://developer.android.com/reference/android/telephony/TelephonyManager.html (2011-05-03), 2010. 47

[79] HTML Living Standard – Video conferencing and peer-to-peer communication. http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#video-conferencing-and-peer-to-peer-communication (2011-05-02), 2011. 47, 48

[80] Skype Limited. The Skype Public API. http://developer.skype.com/accessories (2011-05-03), 2011. 48

[81] W3C. The Application Launcher API. http://dev.w3.org/2009/dap/app-launcher/ (2011-05-02), 2011. 48

[82] Apple Inc. Mpvolumeview class reference. http://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPVolumeView_Class/Reference/Reference.html (2011-04-15), 2010. 58

[83] Google Inc. Android settings. http://developer.android.com/reference/android/provider/Settings.html (2011-04-15), 2010. 58

[84] Google Inc. Android settings.secure. http://developer.android.com/reference/android/provider/Settings.Secure.html (2011-04-15), 2010. 59

[85] Google Inc. Android settings.system. http://developer.android.com/reference/android/provider/Settings.System.html (2011-04-15), 2010. 59

[86] HP. webOS Service API – System Services. `https://developer.palm.com/content/api/reference/services/system-services.html` (2011-04-15), 2011. 59

[87] Apple Inc. iOS Developer Library – UIDevice Class Reference. `http://developer.apple.com/library/ios/documentation/uikit/reference/UIDevice_Class/Reference/UIDevice.html` (2011-05-03), 2010. 63

[88] iPhoneHacks. Apple tells developers to remove 'free memory' feature from iphone apps but.. *iPhoneHacks*, Aug 2009. 63

[89] Google Inc. Android android.os. `http://developer.android.com/reference/android/os/package-summary.html` (2011-04-03), 2010. 64

[90] Google Inc. Android activitymanager. `http://developer.android.com/reference/android/app/ActivityManager.MemoryInfo.html` (2011-04-03), 2010. 64

[91] The PhoneGap project. PhoneGap Documentation – Device. `http://docs.phonegap.com/phonegap_device_device.md.html#device.version` (2011-05-04), 2011. 65

[92] Dzung Tran and Max Froumentin. The System Information API. The System Information API - W3C Working Draft 02 February 2010, 2010. 65, 66, 81

[93] Anssi Kostiainen. Battery Status Event Specification. Battery Status Event Specification - W3C Working Draft 26 April 2011, 2011. 66

[94] Matt Buchanan. How multitasking works on a phone. *Gizmodo*, Apr 2010. 68, 70, 72

[95] Jason Kincaid. Bump now lets you swap app recommendations with a tap. *TechCrunch*, Jan 2011. 70

[96] Jonathan Corbet Jonathan Corbet, Jake Edge. Kernel development. *LWN*, Feb 2009. 70

[97] Mitch Allen. *Palm® webOS*^{TM}. O'Reilly Media, Inc., first edition, 2009. 72, 74

[98] Michal Zalewski. Same-origin policy. *Browser Security Handbook*, Mar 2011. 73

[99] Fabian Topfstedt Roland Ehle. Why we won't enjoy html5's video tag. *Gruppe*, May 2009. 77

[100] Tom Leadbetter. The video element. *HTML5 Doctor*, Jun 2009. 77

[101] Rich Tibbett. Native webcam support and orientation events – technology preview. `http://my.opera.com/core/blog/2011/03/23/webcam-orientation-preview` (2011-05-20), Mar 2011. 78

[102] Google Inc. Repeated requests for permission are annoying. `http://code.google.com/chrome/apps/docs/index.html` (2011-05-20), 2011. 79