LUMA-GIS Thesis no. 13

Mobile Map Client API

Design and Implementation for Android

Mårten Karlberg

2011
Department of Earth and Ecosystem Sciences
Division of Physical Geography and Ecosystem Analysis
Centre for Geographical Information Systems
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

A Master thesis presented to

Department of Physical Geography and Ecosystem Analysis

Centre for Geographical Information Systems

of

LUND
UNIVERSITY

by

Mårten Karlberg

in partial fulfilment of the requirements

for the degree of Master in Geographical Information Science

Supervisors:

Ola Hall, Lund University

Peter Axelsson, Digpro Solutions AB

ABSTRACT

The fast development of computational power of the mobile phone makes it a suitable platform for running map applications. Both public and field working professionals can benefit from easy access to a mobile map client application with features such as route planning, location based services and simple GIS operations. This master's thesis describes the mobile operating system (OS) Android from a geographic information aspect and relates it to other major mobile OS.

Available map client application programming interfaces (API) are investigated. It is concluded that Android is a good platform choice for implementing mobile map applications. But there is need of a generic open source API for Android. Such an API is implemented, resulting in a good performing map client. Though it needs additional development to perform all features aimed for in the suggested API design.

Keywords:

Physical Geography, Ecosystem Analysis, Mobile Map Client, Android, WMS, KML

Table of Contents

1 Introduction

Map clients and other geographic data applications are ideal for mobile use but present both benefits and disadvantages in comparison to desktop solutions. Benefits are the extended functionality that the mobility makes possible. Location Based Services (LBS) provide information about the local surroundings or adapt the application's behaviour to the current location of the user, realised through integration with GPS and other sensors. Field data collection and route planning are other areas where your mobile phone could be the ideal hosting device. Disadvantages are small screen size and limitations in network connection, processing and memory resources and battery power.

Digpro Solutions is a company specialised in geographic information technology (GIT) and network information systems. The motivation for this project is to complement Digpro's desktop map clients with a mobile map client for the mobile operating system Android. This will widen their product line with the possibility of mobile access to geographic data, whenever and wherever the users need it.

One of the reasons for choosing Android as the platform for the mobile map client is its open nature. Android is open source, meaning the source code is open for community based development and free use. Digpro take an encouraging position to an open source business model and see the benefits in contributing to the realisation of a generic mobile map client API of high quality. This thesis is the start of such an open source project, which could result in future profit for both public and business participants.

This report mainly addresses geographic information system (GIS) specialists and computer software developers with an interest in technical solutions for distribution and presentation of spatial data. With an ambition to explain most technical terms, hopefully less technical readers in the GIS field will also understand the main parts of this report.

Next chapter provide some background knowledge that is necessary to understand the problem statement of this work. Then follow a detailed problem description. A description of key concepts used in the solving of the problem precedes the chapter describing the methods used for solving the problem. The results are split into two parts. The first part relates Android to other mobile platforms and describes currently available map API:s. The second part describes the design and implementation of a generic map API for Android. The report finishes with a summary of the conclusions drawn from this work.

2 Background

The operating system (OS) is the bridge to control performance between hardware and applications on a computer. This involves e.g. resource allocation, security tasks, activity management and providing a user interface. Just like a desktop computer has its fundamental functionality controlled by Windows, MacOS or Linux, mobile phones depend on a similar flora of OS:s or platforms. Leading smartphone platforms of today include Nokia's Symbian, RIM Blackberry, Apple's iPhone and Windows Mobile. In 2007 Google, in collaboration with several other technology companies, announced the founding of the Open Handset Alliance, a consortium with the goal to develop open standards for mobile devices. At the same time they launched their first product, the mobile OS Android. Applications for mobiles can be implemented in two different ways:

- *Native application* that can only execute on a particular OS or
- *Web application* that runs in a web browser context and therefore is OS independent as long as a compliant web browser is available.

Both approaches have its drawbacks and possibilities. A native application have better access to hardware and OS services and could provide more advanced features and be made to run more efficiently than a web browser application. The drawback regards interoperability. An application must be more or less rewritten in different programming languages in order to run on different platforms. The web browser application on the other hand is only written once and can run on any OS. This OS independence is achieved by programming in a standard scripting language interpreted by the web browser. Another benefit from this is that the web application does not have to be installed to the system like a native application does, but run directly as the code is downloaded from a web site.

An API (Application Programming Interface) provides software developers with functions to implement applications within a system or technology. The API hides lower level functions that are common to all applications within the system and lets the developer concentrate on higher level functions of the specific application being implemented. A mobile map client API could include functions for e.g. collecting map data from servers, projections or coordinate system transformations, choosing what map layers to display or measuring distances on the map display.

3 Problem description

3.1 Problem statement

The goal for this project was to develop a map client API for the Android mobile operating system. An API, much like the Google maps API but aiming at a more generic product, with a wider choice of map data sources, coordinate systems and offline capabilities, was designed and partly implemented. This implementation is related to other possible platform choices, such as web application, competing mobile operating systems and similar efforts done for Android, regarding performance, user-friendliness and accessibility. The main functionality of the map client API is to provide a good performing map viewer, showing spatial data from various sources. But present in the design is an awareness of favouring data structures that also could enable for performing client side GIS operations on the spatial data and uploading data back to server. The API design aims at effective data and resource use, a logic hierarchic class structure for convenient maintenance and expandability and easy-to-use for the map client API user, i.e. the application programmer. The programmer should be provided basic functions for quickly setting up a map client without detailed knowledge of underlying structures, but should also be able to customise the map application with more advanced API functions. The end-user should be presented a good looking map view with smooth graphics rendering and intuitive and responsive controls for interaction.

3.2 Implications of the research

This work complements the research of mobile mapping by looking at the capabilities of the mobile operating system Android regarding geo-spatial data use and visualisation. Similar work has previously been carried out on other platforms, e.g. Java ME (Harun et al, 2009, Tsou et al, 2005) and Symbian (Biuk-Aghai, 2005), but, as far as I can find, no such attempt has yet been done for the more modern platform Android. Another important implication is to identify problem areas in the mobile map design model for further development. This could include aspects regarding memory and computing resource optimisation, user interaction and application responsiveness. For example how to avoid waiting time when downloading and displaying map data or aspects of implementing GIS functionality and OGC standards in a mobile context. The work also attempts to reveal issues and differences of uses when implementing a mobile map client as either a native application or a platform independent web browser application. Knowledge will be gained about the balance between interoperability and performance costs when choosing approach of implementation.

4 Key concepts

This chapter introduce key concepts that are important for the understanding of this work. The first part describes special issues present in a mobile computing environment that must be taken into consideration in the software development process. Some of these issues, such as memory resources and processor power, are also relevant to traditional desktop software development but can often be neglected because of more powerful computing resources. The second part describes relevant data sources and formats. The third part is an introduction to the Android platform and summarizes software development fundamentals on Android.

4.1 The mobile computing environment – special issues

4.1.1 Limited resources and performance

The size of mobile devices sets obvious limits to its performance. As the electronic components get smaller, faster and more power effective, these limitations are reduced for every year. But still saving resources in platform and application development is crucial for long battery life. Optimising for performance is tricky and often do more harm than good (Bloch, 2008). The Android Developer's guide has a good article about the subject, where more detailed tips complement the two basic rules: “Don't do work that you don't need to do.” and “Don't allocate memory if you can avoid it.” (Android Developers, 2010). Relevant performance test parameters for mobile computing are transmitted bytes and bandwidth consumption, battery consumption, use of CPU resources and local memory and communication latency (Davis Jr. et al., 2009). User-friendliness and readability are other key questions when testing mobile map applications (Dillemuth, 2005).

4.1.2 User interface

The small screen size of mobile phones makes the graphical design different from PC applications. From a mapping point of view it is important to be able to show the spatial data in a readable way at different zoom levels regardless of different screen sizes. Generalisation and grouping of concealing objects are examples of how to achieve this. These issues are not further discussed in this work.

Central to the latest generation smartphones is the touch screen control. This presents one of the issues for web applications. There is no common standard similar to how mouse behaviour is handled in the web browser environment. In the browser window, a touch move event triggers scrolling of the whole website window. But how should that scroll be directed to the map window only, for scrolling the map? This problem may have to be solved differently for different browsers. Starlight[1] is an open source project started by Nokia with the aim to propose a touch event standard to W3C.

A research group in Munich, Germany has began evaluating mobile touch-screen mobiles using Android applications, aiming at developer assistance on which specific user interface (UI) elements should be used for different interactions (Balagtas-Fernandez et al, 2009), rather than general design pattern guidelines as provided by Nilsson (2009) and many websites, including the official website of Android developers[2]. UI aspects directly associated with mobile map design are described by van Tonder and Wesson (2008), focusing on user adaptive interfaces.

4.2 Geographic data web services and formats

4.2.1 Tiled raster map data

Tiled raster images are a popular format for distributing map images using web services. The whole map image is cut up in seamless quadratic tiles of equal size, typically 256 pixels in square. One set of tiles is created for each zoom level. Each higher zoom level double the pixel

1 http://opensource.nokia.com/starlight

coordinate resolution so that four times more tiles are needed to cover the same region of the previous zoom level. This gives the opportunity to render a more detailed map image as illustrated in figure 4.1.

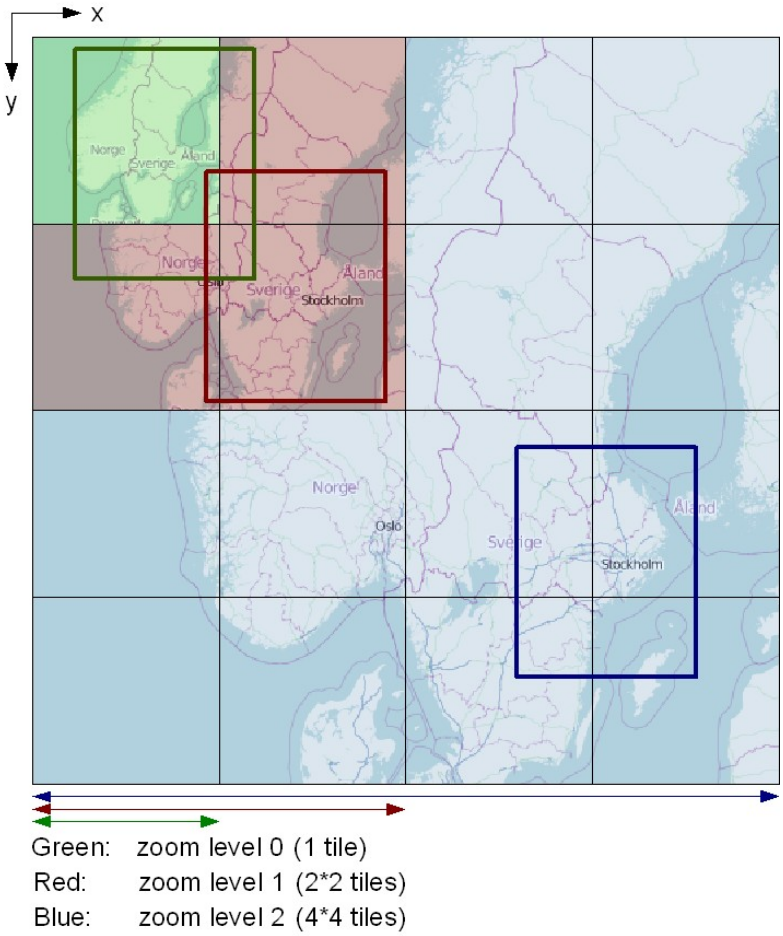

Figure 4.1: The principle of zoom levels on a tiled raster map service. At each increased zoom level the number of uniformly sized tiles, and therefore also the pixel resolution, over the same geographic area are quadrupled and extra detailed information is included in the images by the map renderer. Rectangles represent the map window at each zoom level when zooming in on Stockholm (blue) from a view of Scandinavia (green).

4.2.2 Web Map Service (WMS)

Web Map Service (WMS) is an OGC (Open Geospatial Consortium) standard interface for setting up web map services (OGC, 2010-1). It specifies the structure for an XML (Extensible Mark-up Language) based communication between server and client. A WMS must provide at least two operations:

- `GetCapabilities` returns the capabilities of a WMS, such as available layers, projections and bounds, in a XML-file.
- `GetMap` returns a map rendered from one or multiple map layers at the server as an image.

The returning map images can be pre-rendered raster tiles. Then each `GetMap` call do not have to result in map rendering and the work load on the server is reduced. A `GetMap` call from a tiled WMS is expressed as:

```
www.somesite.org/wms/map?
FORMAT=image/png&SERVICE=WMS&VERSION=1.1.1&TILED=true&REQUEST=
GetMap&STYLES=&SRS=EPSG:3011&WIDTH=tileSize&HEIGHT=tileSize&BB
OX=minX,minY,maxX,maxY
```

where `tileSize` specify the height and width of the returned image tile and `BBOX` (bounding box) holds the geographic bounds of the tile as minimum and maximum values in *x* and *y* dimension. The bounding box must be correctly formulated to align with the bounding of the pre-rendered tiles at specific intervals computed from the map source's outer bounding and defined resolutions of each available zoom level. Information for the client to calculate valid tiles from should be provided in the `GetCapabilities` advertisement. If the parameter `TILED` is set to false, a new image will be rendered and returned if the bounding box values do not align with the tile bounding. (OSGeo, 2010-1)

According to Davis Jr. et al. (2009), implementing WMS for mobiles comes with challenges due to high communication overhead and limited scalability potential. The high overhead is associated with the `GetCapabilities`-request as the returned document is in XML-format and could potentially be larger in file size than the map data intended to be required. They, as do other researchers (Brinkhoff, 2008), suggest a middle tier, providing caching and reduced request complexity, in the WMS network for efficiently serving mobile devices. However, many implementations will typically work with “known” pre-rendered tiled WMS-servers, making the getCapabilities-request superfluous at execution.

4.2.3 Tiled Map Service (TMS)

Tiled Map Service (TMS) is another OGC standard for tiled web map services. (OSGeo, 2010-2) It is used by Google and OpenStreetMap among others. An URI to access a tile can be formulated in different ways depending on the provider. For example:

```
tiles.somemapprovider.org/2/1/0.png
```

would describe access to the tile [x = 1, y = 0, zoom level = 2] as a png (portable network graphics) image from a TMS-server.

TMS is more straight forward than WMS but WMS provides means for forcing rendering of map images with other shapes and features, enabling for a higher level of customisation.

4.2.4 Web Feature Service (WFS)

Web Feature Service is also an OGC standard providing an interface for web services providing map data. Whereas requests to WMS and TMS only result in raster images, WFS can distribute the data behind the map image. The data is often distributed using the XML-format Geographic Markup Language (GML) but any format could be supported. WFS may also be used for sending new or modified data from the client back to the server by implementing the optional Transaction interface (OGC, 2010-2).

4.2.5 OpenStreetMap

OpenStreetMap (OSM)[1] is a free and open map data provider. It is an open source project that aim at creating and providing free geographic data. Anyone can contribute to the map database by adding GPS tracks, street names, topology or other geocoded data. Different places in the world are differently covered as the data gathering depends on local enthusiasm and contribution. The map data is constantly re-rendered and made available from different servers and map services, e.g. the Mapnik TMS.

1 http://www.openstreetmap.org/

4.2.6 Keyhole Markup Language (KML)

Keyhole Markup Language (KML) is an XML-based language schema for expressing geographic annotation and visualization in map applications and 3D “geobrowsers” (OGC, 2010-3). It was first developed by Keyhole Inc. which was acquired by Google in 2004. KML version 2.2 is an open international standard of OGC. Beside Google Earth and Maps, KML can be viewed and edited in e.g. NASA WorldWind, ESRI ArcGIS Explorer, Adobe PhotoShop and AutoCAD.

KML basically specifies features (e.g. images, geometries, text), their location in three dimensions, and optionally a preferred location from where to look at them.

KML share common geometry representations and features with another open standard XML schema, Geography Markup Language (GML). KML can be used to visualize GML content and GML can be reduced to KML, though in the later case losing about 90 % of the data (e.g. metadata and coordinate reference systems).

A KML file does not specify coordinate reference system (CRS). It is assumed that longitude and latitude coordinates are defined in WGS84 and altitude in meters above sea level measured from WGS84 EGM96 Geoid Vertical Datum. If a KML service needs to publish coordinates in another CRS, the consuming map application must be made aware of the CRS in some way. One solution of this problem is to set up a dynamic KML web service, which always returns coordinates in the CRS specified by the user.

4.3 The Android platform

4.3.1 Introduction

Android is an open source mobile platform closely related to Google. The project was initiated in July 2005 as Google acquired the small mobile phone software company Android Inc. The Open Handset Alliance was announced on 5th November 2007 as a collaboration between several leading technology and mobile phone companies, including Google, HTC and Motorola, with the common goal to develop Android as ”the first truly open and comprehensive platform for mobile devices.” (Open Handset Alliance, 2007)

The first Android powered mobile phone, HTC Dream, was released in October 2008. The source code was made available during the same month. It is licensed under the Apache licence version 2.0, which makes it completely free and open for developers to make changes and extensions. The Apache license also permits adding proprietary code. In this way vendors can customise their Android products without having to open up their own code. (Apache Software Foundation, 2010) On the Google IO 2010 conference Google announces that 100,000 Android devices are activated every day (Gundotra, 2010).

4.3.2 Android system architecture

This overview of the Android system and Android application development is based on the information found on the Android Developer website[1]. The Android system consists of components in an architecture as shown in figure 4.2. Its foundation is the Linux kernel, which provides memory and process management, security, drivers and hardware abstraction and other core system services. On top of the kernel are C/C++ libraries that implement specific parts of the system. These include:

- Bionic – BSD-based system C library tuned for Android's embedded Linux kernel,
- Media framework – PacketVideo's OpenCORE-based libraries for many popular audio, video and image formats,
- Surface Manager – manages display and graphics,
- LibWebCore – WebKit based web browser engine,

1 http://developer.android.com

- SGL – 2D graphics engine,
- 3D libraries – OpenGL ES 1.0 based 3D graphics engine that support both hardware acceleration and software rasterisation,
- FreeType – bitmap and vector font rendering,
- SQLite – relational database engine,
- SSL – OpenSSL based Secure Socket Layer and Transport Layer Security for encrypted communication.

The libraries are accessed by developers through the application framework. The framework provide Java API:s to all core functionality and enables development of rich and robust applications. The bundled core applications of Android, such as phone, messages, calendar, alarm and web browser, are built in the same way, using the same API:s as any additional or third party developed application. One important feature for reuse of components is that any application can publish its capabilities, or *Content* as it is referred to in the Android system, so that any other application can make use of them, or replace them.

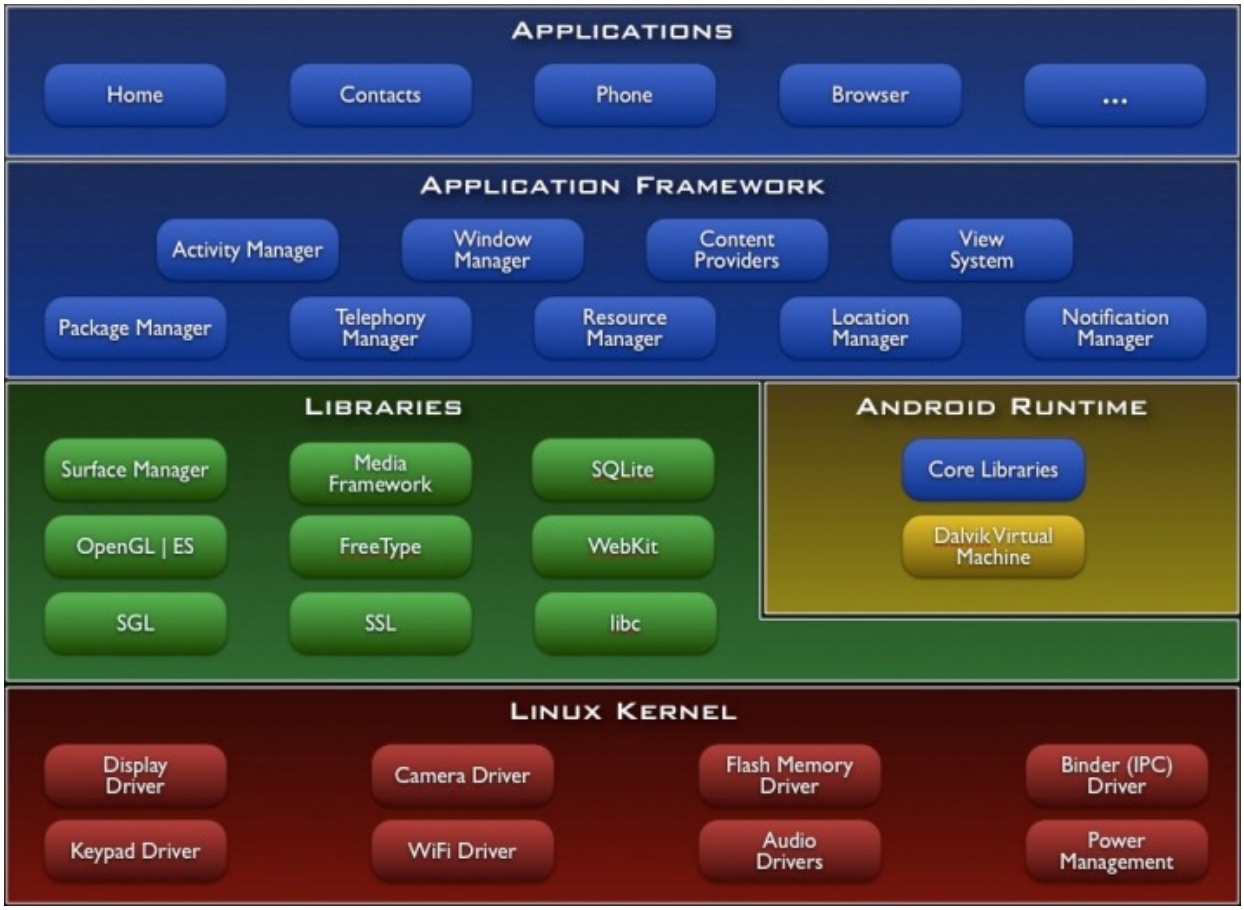

Figure 4.2: Major components of the Android operating system (http://developer.android.com/guide/basics/what-is-android.html)

4.3.3 Android software development

Android applications are written in the Java programming language. Note that Android's Java is a customised dialect and not directly compatible with other Java runtime, e.g. Java ME (Java Platform Micro Edition). The Java code is compiled into the Dalvik Executable (.dex) format which runs on the Dalvik Virtual Machine, similar to Sun Microsystem's Java Virtual Machine. Each application runs on its own process and instance of Dalvik. A software development kit (SDK) is available from the Android Developers website[5]. The SDK includes framework APIs,

tools, sample code, a hardware emulator and documentation. The open source software development platform Eclipse, extended with the Android Development Tools (ADT) plug-in, is officially recommended for coding and debugging.

4.3.4 Application structure

An Android application is bundled and distributed as an archive file called an Android package, with an apk-suffix. It consists of:

- the compiled Java code in the dex format,
- a *Manifest* file in XML format, which could be seen as an initialisation or configuration file,
- additional resources and files, such as images or text string files.

Android applications are built up by any number and combination of four different types of components that the system can initiate and run as needed. The components are implemented as subclasses of the four base classes below:

1. Activity – a visual user interface that often fills the whole screen but also could act as a pop-up or floating on top of another Activity. The visual user interface is built up by objects extending the base class View. Examples of pre-defined Views are: buttons, text fields and scroll bars.
2. Service – a background service without visual user interface. It typically run for an indefinite time and could be used for data communication over the network, making asynchronous calculations or playing music in the background while using other applications.
3. Broadcast receiver – receive and react on broadcast announcements such as low battery level or user changed preferences. Broadcasts typically originate from the system core but can also be initiated by applications. A broadcast receiver can start an activity to react on the broadcast, or use the NotificationManager to alert the user in various ways.
4. Content provider – makes specific parts of an application's data available to other applications. The ContentProvider base class provide a standard set of methods to reach the application's data. These methods are called indirectly by other applications through a ContentResolver object. A ContentResolver can talk to any Contentprovider and handles all necessary inter process communication.

The first three components are activated by something called *intents*. Intents are asynchronous messages that name the requested action of an activity or service, specify what data to act upon or name the action being announced for broadcast receivers, among other things. Intents do not need to explicitly name a target component. In that case the system looks through different component's *intent filters* to decide what component could perform the intent. There are a number of methods that can be used to send intents to activate different components. The choice of method depends on how you want it to act, if it should return anything or from where you activate it.

Context is a base interface to global information about an application environment. Among other things, it provides functions for invoking, controlling and changing the behaviour of components. The activity base class also include functions for starting new activities. Different functions for shutting down activities and services are also provided. The system may also shut down components that are not used or to free memory for more active components. Broadcast receivers and content providers are only active under the short time they have work to do and do not have to be explicitly shut down.

An Android application does not include a `main` function like standard Java applications. This is because it needs the ability to have many different entry points in order to provide reuse of its

contents for other applications. Instead the main entry point to an application is stated in its Manifest file. The Manifest file also declares the application's components, names any linked libraries besides the default Android library and states permissions to external components that the application needs to access. An application, as it appears to the user, is a combination of activities. These activities could be running in different processes if activities of other applications are reused. To hold together the appearance of a user application all its activities are handled in the same *task*. A task is a stack of activity objects. Once a new activity is started within a task it is pushed upon the stack. When the user pushes the back button the task pops back the previous activity from the stack.

5 Methods

5.1 State-of-the-art investigation

The investigation includes a state-of-the-art description of available major smartphone platforms and map API:s. Issues from a technical and user perspective are discussed as well as an analysis of what new features and uses can be expected from future development within the field of mobile mapping. The investigation is done by referencing available scientific papers, technical articles and relevant web sites.

5.2 API design and implementation

The work on the Android map API is divided into a theoretical design and a practical implementation. The design defines the overall functionality, some specialised features and cohesive data structures. The implementation takes the first step to realising the design into an API and executable map client applications. Implementing the API is the part of the project on which most time was spent. The development was carried out in an iterative manner. Starting with complementing a basic, incomplete open source map API to gain understanding of mobile map API design and avoid re-inventing the wheel. As the project got along, more fundamental changes where made and the design matured. Performance was tested on the hardware emulator integrated in the Android SDK and on a real device, Motorola Milestone running Android version 2.1. The functions of the API are partitioned into distinct modules, enabling performance comparison between different implementations and optimising for future extensions. Not all parts of the design are implemented due to lack of time. Focus is on fundamental map functionality including:

- Map data requests via WMS and TMS.
- Projection structure for transforming and mixing data in different coordinate reference system (CRS).
- KML data reading and drawing, exemplifying vector data overlays.
- User-interface with touch-screen navigation in the map view and a simple menu system.
- Data caching, providing offline data access and reduced bandwidth use.

6 Results

6.1 Comparison of mobile platforms

In this chapter available major mobile platforms are being compared. Only so called smartphones are included and not simpler mobile phones. There is no standard definition of what a smartphone is but it should include extended computing power and an operating system able to run applications in a PC-like fashion. The development is fast though. What was a high-end feature only some years ago, could today be standard on the simplest devices. And no signs indicate a stop in the development. On the contrary, new platforms are introduced every year and those available develop to meet costumer expectations on the hard competition on this immature and fast evolving market. The facts about the different platforms are taken from each websites except where else noted.

6.1.1 Symbian

Symbian was the leading smartphone platform for many years. It started in 1998 as a collaboration between the mobile software development company Psion and phone manufacturers Ericsson, Motorola and Nokia. After a decade of success Nokia purchased Symbian and founded the Symbian Foundation for developing the platform as an open source project. A non-standard implementation of C++ is the native programming language of Symbian. Different SDK's for different manufacturers and devices is available but since the unification of the UI there should be less diversity from 2010. Native Symbian is much specialised with a steep learning curve due to the specialised low level programming techniques demanded for the performance optimising real-time core. But fortunately Symbian applications can also be programmed using Python, Java ME, Flash Lite and Ruby.

6.1.2 iPhone OS

Introduced in 2007, Apple's iPhone[1] quickly took shares on the smartphone market. The OS, simply named iPhone OS or iOS, was derived from Mac OS X to run mobile devices. Today it also runs Apple's iPod touch and iPad. The platform is proprietary closed source and initially third party application development was only possible as web applications. However, an SDK for third party native application development was released in 2008 for the objective-C programming language. An iPhone Developer Program fee has to be paid for the application to be installed appropriately. Apple also keep hard control and can keep applications off the only distribution channel, the iPhone app store, if they think it contains improper material or features competing with Apple's applications.

6.1.3 BlackBerry

BlackBerry[2] is Research In Motion's software platform for their line of smartphones. It was introduced in 1999 as a pager, a device for message communication including email and SMS. In 2002 mobile phone and web browser capabilities were integrated. Key focus was on email and it became the most popular smartphone among business users. BlackBerry is proprietary closed source. Third party application development is possible in an extended version of Java ME or as native web browser applications that only work on BlackBerry smartphones connected to a BlackBerry Enterprise Server, a software package integrating mobile devices into an organisation's email system.

6.1.4 Windows Mobile

Windows Mobile[3] is Microsoft's mobile operating system. It is designed to integrate well with desktop Windows in terms of synchronisation, looks and features. Its applications include mobile versions of Microsoft well-known base applications such as Office, Internet Explorer, Outlook,

1 http://developer.apple.com/iphone
2 http://na.blackberry.com/eng/ataglance/get_the_facts/
3 http://www.microsoft.com/Windowsmobile

Media Player and MSN Messenger. It was introduced in 2000, mainly for pocket PC's but also for mobile phones. Windows mobile's greatest success has been among business users. Its market share has dropped steadily the last couple of years, something that Microsoft hope will change with the coming version, Windows Phone 7 Series, which was announced in February 2010. With this version Windows mobile catches up with features that could be seen as standard on smartphone platforms of today, such as native multi touch screen support and better adoption to modern mobile processors. It also broadens focus from strict business use by heavy integration of social networking, multimedia and gaming.

Third party application development is done with .NET, SilverLight and XNA in the Microsoft proprietary Visual Studio environment. A free version, Visual Studio 2010 Express for Windows Phone, has been released for the development of applications to get going before the first device release, planned to the end of 2010. With Windows Phone 7 Series Microsoft break compatibility to previous Windows Mobile versions. They state that it is up to the hardware manufacturers if a device will be able to upgrade from earlier versions. One obstacle may be the number of buttons. Phone 7 require three buttons and older phones typically have more. A full hardware compatibility specification is not available yet.

6.1.5 Palm webOS

Palm webOS[1] supersedes Palm OS, previously implemented in Palm's PDA:s and smartphones. Palm webOS takes an interesting approach in how applications are executed. All applications are developed as web applications. They run in a WebKit-based UI system manager but are installed like a native application and run directly on the device and not in the web browser. Device based native functionality is achieved through a JavaScript API, called Mojo, available in the webOS SDK. As an example, local storage is achieved by HTML5 storage functions. WebOS runs on an embedded Linux kernel. User interactions are mainly, but not necessarily, touch driven (Allen, 2009). The OS was introduced on January 8, 2009 and so far runs on two of Palm's devices. Many of the platform's features are built on open source projects and they take a clear encouraging position towards the open source community, even though proprietary parts are added. The platform as a whole is not open source.

6.1.6 Java ME

Java ME[2] (Java Platform Micro Edition) is a stripped-down version of Sun Microsystem's Java platform intended for mobile devices. It is not an OS but a virtual machine running on top of another OS. Java's core concept is portability. It is supported by many mobile phones as an alternative means of running applications other than native applications. Java ME has been one of the biggest platforms for mobile applications, supported by Symbian and Windows Mobile among others. But the future is unsure as some of the most popular new platforms do not support Java ME, like Android and iPhone.

6.1.7 Linux

There are a vast number of open source projects with the aim to port Linux to mobile phones and other embedded devices. The main differences from full featured desktop distributions are that they use more compact versions of software libraries and utilities and support more specific features and hardware. The range of Linux distributions for smartphones changes rapidly with new projects being announced, like Samsung's Bada in November 2009 (Bada, 2009) and other disappear or merge, like Nokia's Maemo and Intel's Moblin becoming MeeGo in February 2010. (Haddad, 2010) Worth keeping an eye on is the development of the LiMo Foundation Platform, which is supported by many big mobile companies, including Samsung, Panasonic and NEC. What platforms will survive the competition is yet to be seen as the market mature.

1 http://developer.palm.com/
2 http://java.sun.com/javame/

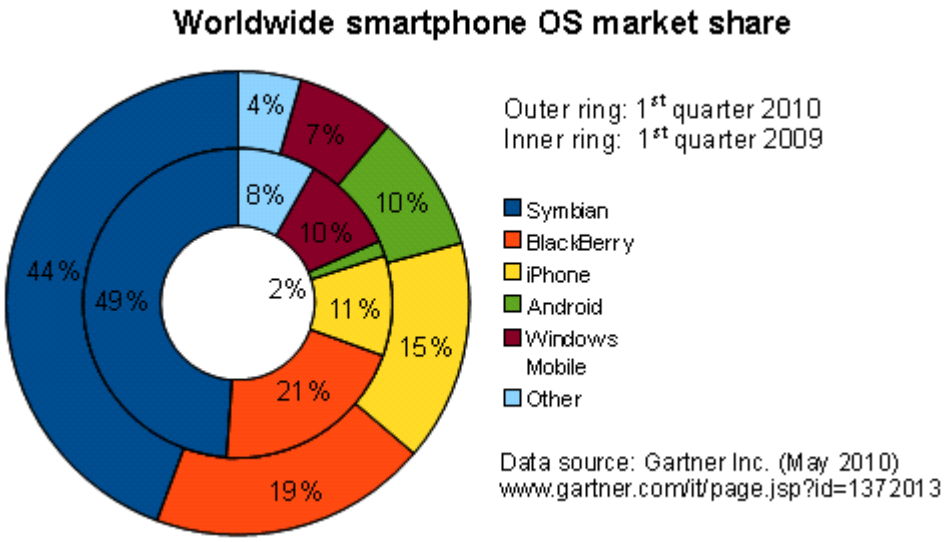

Figure 6.1: Worldwide smartphone OS market share.

6.1.8 Discussion

Smartphone platform market shares vary over different parts of the world, but all together Symbian and Windows Mobile have been dominating. Later years, iPhone has taken a great portion of the sales, especially on personal users. Android has too in short time since its introduction gained much popularity and expectations are high on future success. The analysis company Gartner states that Android is the fastest growing mobile platform and could be expected to have 20% of the world market within three years (Wallin, 2010). Figure 6.1 shows the leading smartphone operating systems based on world market shares.

Technology features of the different platforms does not seem to be the main eliminating factor as features introduced on one platform quickly gets an equivalent feature on the others if proven successful. Important to the success of a platform is the ability to attract third party application developers. Access to a big number of high quality applications is important as the smartphone becomes a more popular device for computing. This will continue and possibly outnumber stationary PC:s and laptops as the main personal computing device especially if the drawbacks of the small format are solved. Future improvements of interfaces for connecting stationary devices, such as bigger screens and more convenient keyboards, could help this (Barton, 2006). Such interfaces are already emerging as many new devices feature e.g. high resolution video-output connections.

Being a new platform, introduced in 2007 and with the first supported hardware released in 2008, not much scientific research has been carried out on Android yet. That may most likely change since the openness of the platform provide extended possibilities for developers and advanced users according to Speckmann's (2008) review paper, in which Android is compared to Symbian and Windows Mobile. Several comparative articles of smartphone platforms are available in scientific literature, but few of them include Android. One mentions Android, but then only as a potential up- comer in relation to now leading platform (Lin, 2009). Godwin-Jones emphasise the possibilities of third party application development for Android and other open source platform projects in contrast to the closed nature of e.g. iPhone and Blackberry (Godwin-Jones, 2008).

6.2 Mobile map API:s for Android

This section is divided into one list of proprietary and one list of open source mobile map API:s and ends with a short discussion about development and usage. The API:s have differences in what they can do and what data they can access. There are more map applications for Android

available but applications with no ambition or clear entry to build new applications upon are left out.

6.2.1 Proprietary mobile map API:s

Since Google is the main developer of Android, of course there are strong links to other Google products. Map applications are easily developed in the Android SDK using the Google maps API add-on[1]. But Google maps have its limitations:

- Map data sources limited to Google's providers.
- Projections and coordinate systems limited to Spherical/Web Mercator.
- 3rd party caching and storing is prohibited which prevents controlled offline usage.
- Requires registration through an API key.
- Google have the right to integrate commercial advertisement in the map view.

CloudMade[2] and Ericsson Labs[3] provide map API:s of slight different approaches. CloudMade provide a set of API:s for accessing OSM data. The OSM data is accessed from CloudMade's own servers for higher speeds and accessibility. They have API:s for most major mobile platforms. The Android API is licensed from Nutiteq. It has more features than Google maps, including various map sources, WMS and KML support, offline capabilities and vector overlays.

Nutiteq's API can be used for free within the terms of the General Public Licence (GPL). That requires that the final application is published under GPL. There are also commercial alternatives at different prices. Ericsson Labs use a quite different technique than the other presented API:s. All data are vector based and the map image is rendered client side. Map data are downloaded from an OpenStreetMap vector server. The API is “experimental in nature, is still in concept development phase and is only made available for testing and feedback purposes.” (Ericsson, 2010) It is free to use but not commercially.

6.2.2 Open source mobile map API:s

A handful open source projects for showing map content on Android can been found. One of them is the Google maps open source clone[4], hereafter referred to as *the clone*. It is an attempt of cloning the Google maps API but using OSM data to avoid registration. It is supposed to be binary compatible with Google maps, i.e. using the same package, class and function names, to be used whenever an application calls for Google maps on totally free and open devices, on which proprietary Google add-ons are replaced with open source clones like this one. The legal aspect of this can be questioned but the project seems to be too small for Google to care about it. The clone follow the Google maps API documentation closely, which makes it easy to understand. Therefore this project was chosen as a foundation of the implementation of this final thesis. The clone implementation is not completed and no activity has been noticed on the homepage since October 2009. The source code is published under an Apache licence.

The open source project gvSIG Mini[5] provide a map viewer application for Android, BlackBerry and Java ME. It uses map data from OpenStreetMap, Yahoo Maps and Microsoft Bing and provide WMS and offline support, address search, GPS integration, a route finder and social media integration. gvSIG Mini is loosely based on the open source desktop GIS project gvSIG, with its base in the Valencian Regional Council for Infrastructure and Transportation, supported by the European Union regional development fund. It is licensed under GNU General Public License (GPL) and free to use and modify source code.

1 http://code.google.com/android/add-ons/google-apis/
2 http://cloudmade.com/products/mobile-sdks
3 https://labs.ericsson.com/apis/mobile-maps/
4 http://gitorious.org/android-maps-api
5 https://confluence.prodevelop.es/display/GVMN/Home

The Android version of gvSIG Mini is not stable and parts of the UI is not obvious how to use. The gvSIG Mini homepage provide quite good software development analysis and class descriptions but not very good code documentation or bug report system. Development relies more on in-house resources at the Spanish software company Prodevelop than on a community based approach. A new more stable version 0.2 was released in April 2010 and more than 10.000 downloads are reported from Android Market.

Two other open source projects are Osmdroid and WMC. Osmdroid[1] provide tools to interact with OpenStreetMap and WMC (Web Map Client)[2] is for viewing WMS data. Both projects are quite simple and no information of future development are presented on their websites.

6.2.3 Discussion

The limitations of Google maps, regarding available features, openness and commercial use, makes obvious needs of additional map API:s for Android. Proprietary alternatives are available with extended features for users willing to pay licence fees. Open source alternatives of different approaches can also be found but suffer from different level of immaturity and poor documentation. OpenStreetMap is the most common map data source for both open source and proprietary API:s.

Relaying on Spherical/Web Mercator CRS is common for all API:s. No API provide a projection interface for convenient mixing data of different CRS. Nutiteq's API is the most complete, with WMS and KML support, vector overlays and offline capabilities. But Nutiteq's licensing requires either commercial fees or totally free GPL publishing of the final application. The most complete open source alternative is gvSIG. But even if the source code is available and open for re-use, it is constructed more like an end-application without good structures for providing an API interface. The other open source projects are merely copies of Google's map API for OpenStreetMap, though good for building simple map applications upon. The immature and small range of open source alternatives motivates the development of another Android map API like the one introduced in this study.

6.3 Mobile web map applications

6.3.1 Background

Another approach for implementing applications is as web applications. A web application executes within a web browser environment, commonly programmed in Hyper Text Markup Language (HTML) and ECMAScript, more often called JavaScript. HTML and JavaScript are standardised by W3C and ISO and supported by all major web browser, which makes web applications potentially platform independent. Add-ons can extend the web browser environment to support other web application formats as well, e.g. Adobe Flash.

The mobile industry has high expectations on web applications. As described earlier Palm base the application runtime fully on a web application environment for webOS and Apple's iPhone was initially intended to support only web applications for third party applications.

What a web browser application gains in interoperability, it may lose in functionality. One example is interactions with other features of the mobile device. Sensor states and control, such as using coordinates from an on-device GPS or how to react on touch-screen gestures, can often be easy handled by native applications through the platform API:s, but accomplishing the same in a web browser application may be tricky. But this is an area of fast progress. Currently much work is invested on developing standard API:s for using on-device features in a web application context. One example of this is next version of W3C's HTML standard, HTML5, which aims at taking HTML from being a language for presenting documentation to building browser independent web applications (W3C, 2010).

1 http://code.google.com/p/osmdroid/
2 http://www.finds.jp/wmc/index.html.en

6.3.2 Map API:s for mobile web applications

OpenLayers[1] is a JavaScript API for building dynamic web map widgets. It was initially developed by MetaCarta but is now a free open source project of the Open Source Geospatial Foundation (OSGeo). OpenLayers map widgets look and work a lot like Google web maps but can use data from more sources. It implements many different data formats and OGC standards such as WMS and WFS. Some effort have been made to port OpenLayers to a touch driven mobile browser context, but with little success. The results so far renders too slow to give a respondent enough user impression. Source code and tested demo applications can be found at the OSGeo wiki[2].

TouchMapLite[3] is an open source JavaScript map viewer optimised for WebKit browsers (Safari, Android, Nokia S60, webOS) running on touch sensitive devices. It renders the map image smoother than OpenLayers when tested on Motorola Milestone, but can not provide any of the extra features of the OpenLayers API, such as overlays and projections. The overall impression is that current web map applications perform well on desktop but give a rather slow rendering appearance on mobile browsers. Whether this depends on immature performance of mobile browsers or map API:s optimised for desktop environment is not answered here. Based on testing the different API demo applications on the Motorola Milestone Android 2.1 browser, it is concluded that no web map API to date is good enough to build a good generic map client upon within the context of this study. Such implementation would be very interesting to compare performance against a native implementation. Future improvements of the technology will most likely change this conclusion, and possibly quite soon. A hint in this direction is the recent release of Android version 2.2. On its presentation on Google's I/O 2010 conference significant speed improvements are demonstrated for both native applications and the web browser thanks to the new JIT-compiler and JavaScript interpreter (Gundotra, 2010).

1 http://www.openlayers.org/
2 http://projects.opengeo.org/mobile/wiki
3 http://sourceforge.net/projects/touchmaplite/

6.4 Android map API – Design and Implementation

6.4.1 Design analysis

The design analysis proposed in this chapter aims at an abstraction model of the map client API. Included are the main functionalities and internal and external dependencies as shown in figure 6.2 where implemented parts can be distinguished from future aims by different colouring.

The Google maps API clone introduced in chapter 6.2.2 was used as a base for the implementation but the memory cache is the only part of it that is not re-worked or extended. The implementation is described in detail in chapter 6.4.2 but the text about the design analysis also includes ideas of how some parts could be implemented. Especially where the Android platform provides convenient standard methods for it, as is the case for the menu system.

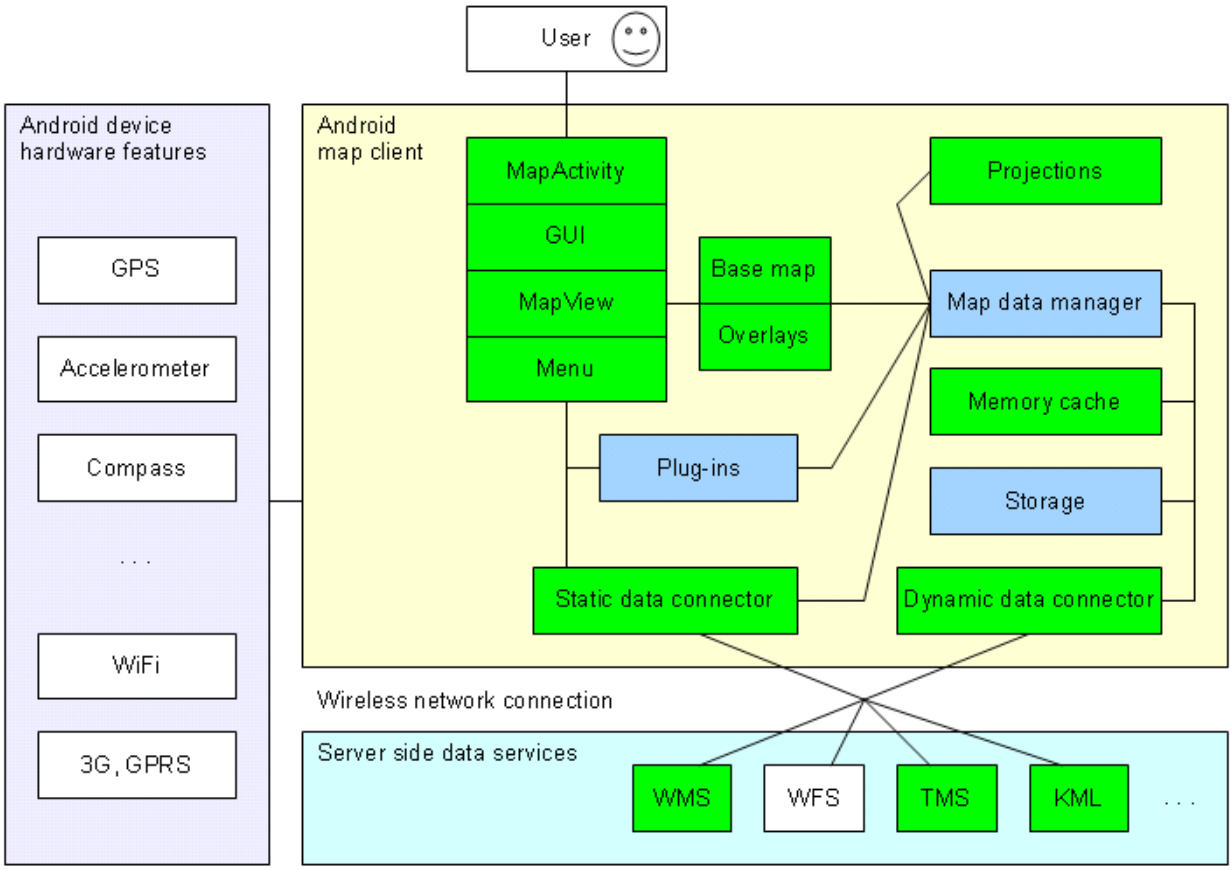

Figure 6.2: Abstraction of the Android map API. Green parts are (partly) implemented. The only part of the implementation that is not new, re-worked or extended from the Google maps API clone is the memory cache used for requested WMS-images.

6.4.1.1 MapActivity

`MapActivity` is the main instance that holds the map client application together. It extends the Android base class `Activity` and manages initialisation of the `MapView` and data management on start-up and collects instance states for reuse on possible re-start when being shut down. It defines UI layout and controls the menu system.

6.4.1.2 MapView

The `MapView` is the map window, visualising map data on screen. It compiles the map data layers into the final map image and updates it following the command of user input and data

updates, such as scrolling or zooming. `MapView` extends the Android base class `View`, that represents graphic objects on screen. `MapView` is built up by three parts:

1. Base map – Underlying map covering the whole map image.
2. Overlays – Layers of interactive map data objects. Tapping an overlay object could for example open a pop-up window showing attribute data or invoke a new Activity for performing work related to that object.
3. Optional view-objects shown on top of the spatial graphics, for example scale bar, zoom controls and indicators.

Screen updates should be asynchronous for not blocking the UI thread. As long as extended time consuming tasks, such as loading data over the network, are handled in its own thread and invokes re-draw whenever new data become available, screen updates should run smoothly following standard Android programming standards.

Figure 6.3 shows a work flow diagram of how a screen update could be performed. First data requests are being made for the area limited by the screen bounding box, referred to as the *viewport*, then an optional animation is performed using the data already available. The animation could be a smooth zoom by computationally light linear pixel transformations. Whenever new data become available, the asynchronous draw method is invoked as needed. As `MapView` is the map client's main window towards the user, it consumes much of the user interaction events. Most events should be different touch gestures (pan, tap, double-tap and so on) but other controls, such as trackball and keyboard, should also be supported.

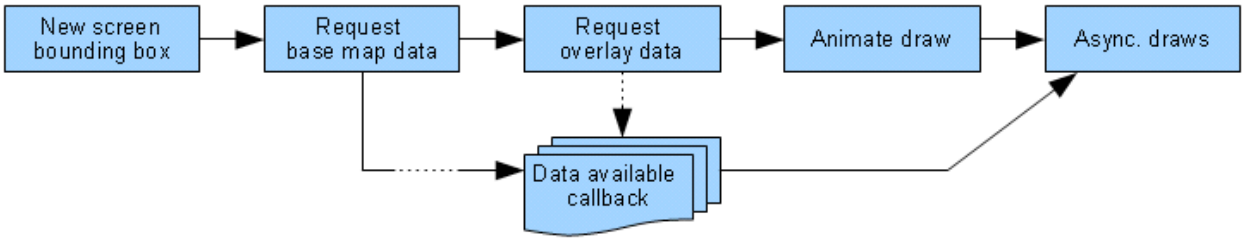

Figure 6.3: Screen update work flow example.

6.4.1.3 Menu

Menus are an integrated part of the Android application structure. Standard Android devices have a button for reaching the options menu of the current running `Activity`. It is easiest implemented through the `Menu` interface in the `Android.view` package. The menu should include sub-menus for controlling visibility of map layers, invoking tools or plug-ins and changing settings. Settings could include: show compass and user location, rotate map with compass to automatically align with north-south direction, centralise map on user location or force offline mode to avoid costs related to data transactions over the network.

6.4.1.4 Plug-ins

The purpose of plug-ins is to provide a standard interface for extending the map clients functionality. A typical implementation of a plug-in would be a tool reached from the menu for manipulating map data, perform searches, force `MapView` to move to pre-defined locations, integrate route navigation, detailed WMS interaction with e.g. selectable layers through the returned XML-file from a `getCapabilities` query, measurements and GIS operations, data export and import or offline data management.

6.4.1.5 Data management

The Android platform includes a SQLite relational database. It would probably provide the best way to store data on the device. The alternative is to read and write data directly to the file system, which does not provide the convenient query tools of the database approach. The database storage would serve as a middle tier between objects in memory and data downloaded from servers in various formats. Once server data is parsed, it is stored in a common data model for generic access. The common data model is not further investigated but should consist of a number of common geometries (at minimum point, line and polygon), a generic object model and attribute tables. SQLite has no support for geometric data types or spatial queries but SpatiaLite[1] is an extension providing that. It has not been tested in this work but it would be interesting to try on Android.

A map data manager (MDM) serves as an interface for getting and storing map data. In its simplest form it serves as a data provider. For example, whenever `MapView` needs to update its graphics, it makes requests to one or several MDM for getting map data. A more advanced implementation could also handle two-ways data transactions, so that a plug-in could be able to perform changes to the data and save the changes to the server through the MDM. The MDM handles all memory and storage issues and communication with servers over the wireless network. MDM is typically implemented as a service, running in its own thread. At least the communication tasks that can be expected to take some time and should not be allowed to block the UI thread.

Figure 6.4 shows the internal design of the MDM used by `MapView` for getting tiled raster data for the base map. But the internal design of a MDM for getting other types of data may look quite different. For example, vector data can not be divided into seamless bounding boxes like raster data. A method for avoiding redundant data transfer and storage must be included here.

Other features that could be useful is adding age criteria or client-server data synchronisation for data in storage if it is sensitive to changes. But such details are beyond the scope of this model. The MDM should support data in any projection. It should be able to integrate data of different CRS through a common projection interface, transforming coordinate data of different CRS to the internal CRS.

Data transaction with server side services can be static or dynamic. Static means that transaction is invoked at a specific time with a specified bounding box or file. One example is downloading a KML overlay at start-up. The KML file is parsed and stored and then the static data connection is closed. A dynamic data connection is active throughout the execution time. It is invoked dynamically by the MDM for getting new data as the user scroll into new areas. The base map raster tile provider is dynamic as it only downloads tiles as they come “in view” and not the whole data set.

1 www.gaia-gis.it/spatialite

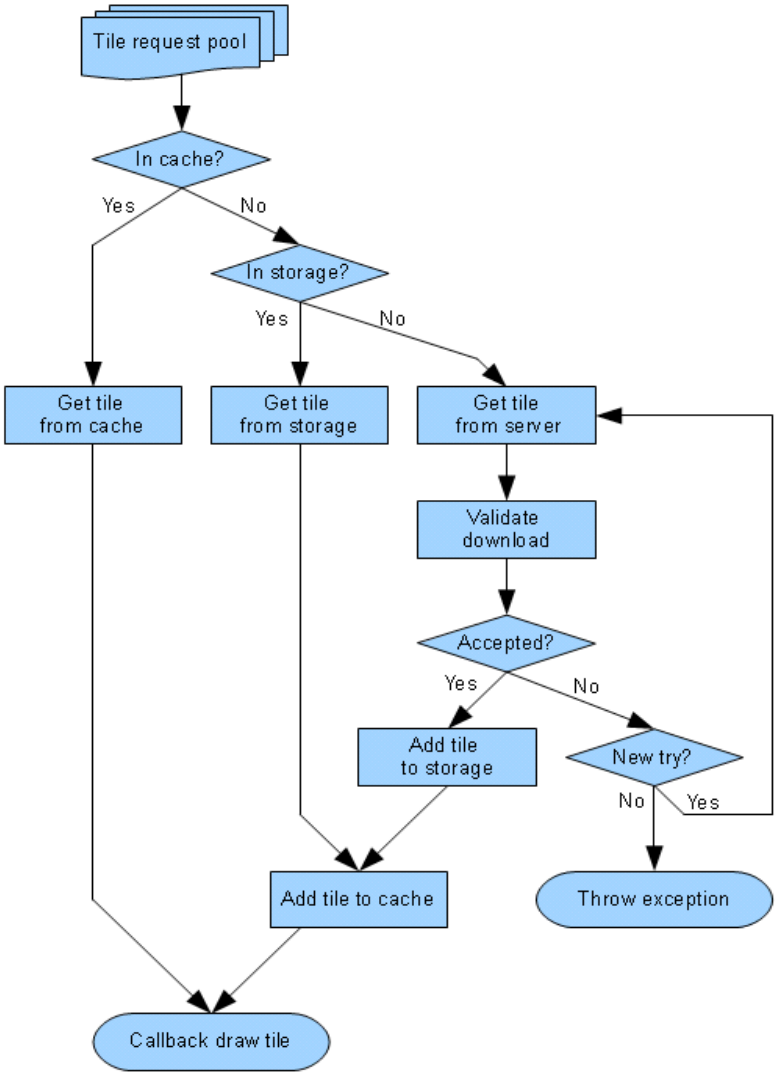

Figure 6.4: Flow chart of the tiled raster map data manager design.

6.4.2 Implementation

6.4.2.1 Introduction – Beyond the Google maps API clone

The Google maps API clone open source project introduced in chapter 6.2.2 was used as a basis for the Android map API development. The clone project is merely in a starting phase but provide the basic classes for creating a map application with simple point overlays. Unfortunately the development has stagnated because of the developer's focusing on other projects. The clone closely follows the architecture of the original google maps API but uses map data from OpenStreetMap instead of Google. Even the package name is equal to Google's in order to be binary-compatible for running map applications on platforms without the Google maps API installed. This approach could be questioned from a legal aspect and breaks the Java convention of distinguishing package names. Therefore the package name for the API developed in this study was changed to *se.digpro.android.maps*. Such a change, and changes such as changing or adding code, is no legal problem since the code is published under the open source Apache license. Permission was also confirmed personally by the original developer, Jim Ancona, via e-mail. The Apache license even permits making any post-developed software proprietary. The present plan at Digpro is to publish this new API as another open source project, open for other developers' use and contribution.

Quite big changes have been made through the development of the API. Starting with filling gaps of "the clone" implementation soon turned to adding new features and breaking the strict following of the Google maps API documentation. This had to be done in order to efficiently implement features beyond the limitations of one single projection (Mercator) as described in the next paragraph.

6.4.2.2 Internal map coordinate system

The most extensive change from the Google maps API clone was replacing the internal map CRS. The clone, like Google maps, uses Spherical/Web Mercator latitude and longitude coordinates as all its data are in that projection. However, as this map API can access data in any projection, a pixel grid presents a computationally lighter internal CRS. This especially holds true for tiled WMS raster data where the tile containing a certain pixel can be given quickly without a computationally expensive transformation from the base map projection to geographic coordinates. The pixel grid coordinate system also results in lighter graphics computing when animating pan and zoom.

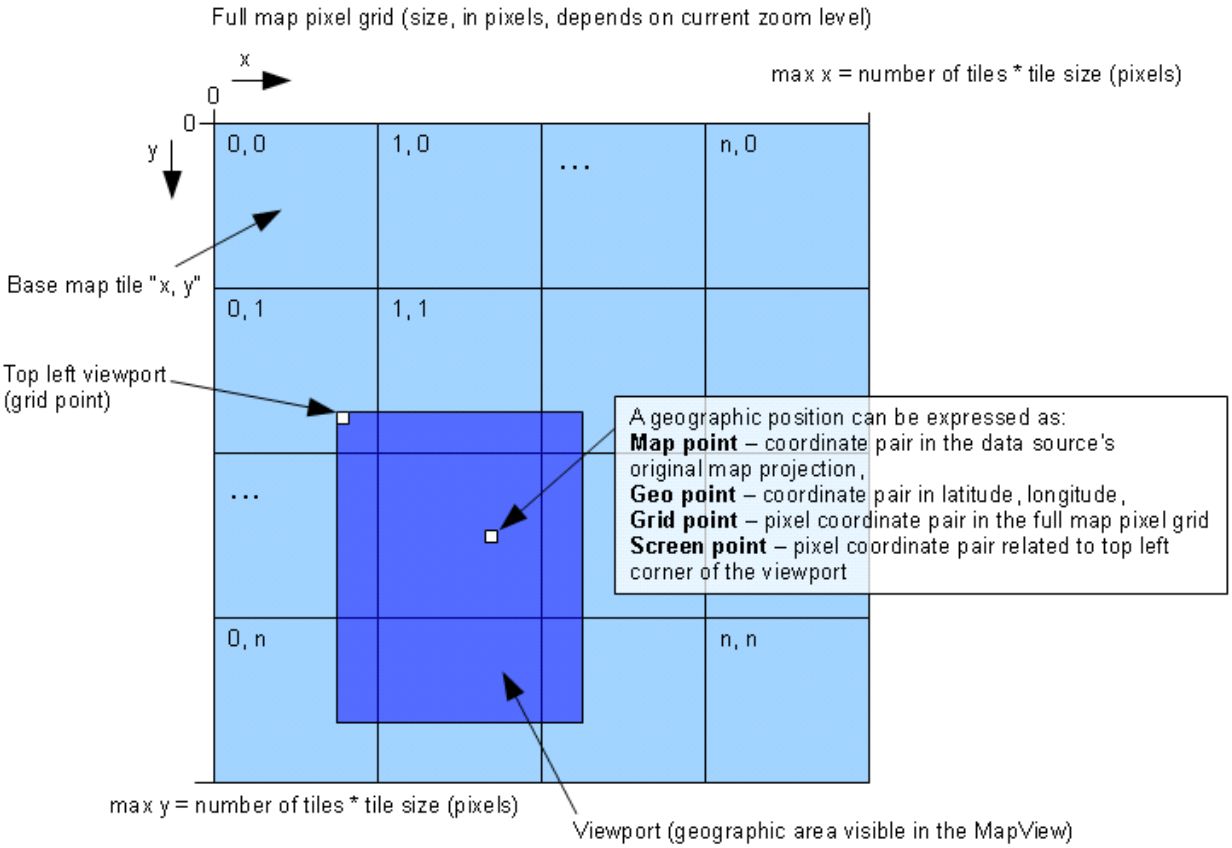

Figure 6.5: Map reference systems and alternative ways of representing geographic positions.

The viewport, the map contents currently visible in the `MapView`, is related to the pixel grid coordinate system as shown in figure 6.5. A point in the pixel grid is defined by a `GridPoint` object. The pixel grid has its origin in the top left corner of the full base map. Note that the size of the base map vary with the pixel resolution of different zoom levels (maximum pixel value = number of tiles * tile size), and so does the fundamental pixel coordinate system. This approach optimises performance when scrolling the map view, as all coordinates are recalculated with a simple offset. Positions of user input events, such as tapping an object by touching its graphic representation on screen, is reported in pixel coordinates related to the top left corner of the viewport, which is referred to as a screen point and implemented as a basic `Point` object.

Corresponding position in the full map pixel grid is computed by offsetting the screen point by the important grid point `topLeftViewport`, which specifies the position of the top left corner of the viewport in the pixel grid coordinate system.

6.4.2.3 Projections

Map sources and overlay data with different CRS are transformed through a projection interface. Each implemented projection is responsible for transforming positions between its own CRS, latitude, longitude coordinates, pixel coordinates and containing tile in the base map. Each data source has its own projection specified so that they can interact in the same map view. Two projections are implemented for testing: Spherical/Web Mercator (EPSG:3785) for accessing OpenStreetMap data and standard projections for the Swedish referens system SWEREF99 (EPSG:3007-3018).

A possibility for future development could be to make use of some open source standard projection library for convenient access to a wide range of projections with transformations support. One such example that might work with Android is JH Labs Java Map Projection Library[1].

6.4.2.4 Screen update

Effective graphic update is important to get a fluent map appearance. It should be respondent and sensitive to constant changes as the user explores the map data. Screen updates can be invoked from different sources. Typical user interactions invoking screen updates are panning or zooming. In these cases the drawing is preceded by functions for calculating the new zoom level and position of the viewport. On each screen update the `dispatchDraw` function of the MapView decides what tiles should be visible within the viewport and requests them from the cache. Tiles not available are downloaded over the network. The storage middle tier suggested in the design model above is not yet implemented. After requests have been made for base map tiles, overlays are drawn on top of the base map by calling the overlay objects' draw functions. Finally indicators and zoom controls are drawn. This function is more or less unchanged from “the clone” implementation. It uses an open source solutions from CommonsWare[2] for asynchronously downloading and caching map tiles. It works decently but lacks methods for validating downloaded tiles. Sometimes download fails and return an empty tile image. This is not recognised and results in a black square in the map view with no chance to repair the map as long as the empty tile is kept in cache.

There is also a problem with how overlays are drawn. The base map tiles, indicators and controls are implemented as separate `View` objects, organised in a `ViewGroup`. But the overlays are drawn directly on the `Canvas`, which is the final graphic representation on the screen. This means the overlays are drawn incorrectly on top of the indicators and controls. This problem has to be solved. Probably by adding a transparent `View` object covering the whole `MapView` to draw overlay objects upon.

6.4.2.5 Map sources

The API use a common interface for accessing base map sources. “The clone” only implements the OpenStreetMap Mapnik TMS as base map data source. A WMS map source class was added to access WMS data, which also required some changes to the map source interface. The screen update algorithm depends on tiled map data for drawing and caching. But that does not mean the server side WMS must be tiled in order to serve as a map source. As long as the WMS bounding box and image size is computed unequivocally, the URI for accessing map images will be expressed equally as WMS tiles and are therefore handled just like tiles internally.

1 http://www.jhlabs.com/java/maps/proj/
2 http://commonsware.com

6.4.2.6 Overlays

Overlays are used for showing complementary data on top of the base map. The overlays are managed in layers which enable for customized mapping and GIS operations. Overlay data could be represented as both rasters and vectors and be distributed in many different formats. Vector data distributed in KML is used in a first test to implement vector overlays. At present only a simple test structure for static KML overlays is implemented. The structure consists of a simplified KML parser and a class hierarchy for representing different KML objects. The parser is called at the MapActivity creation to gather the KML file from a specified URL. It parses the KML data and creates necessary objects to represent the KML objects, identified by the KML-tag *placeholder*.

A KML object is represented by the KMLItem class, which holds geometry data, description and label texts and style data for drawing properties, such as colour and line width. Coordinates are transformed into the local pixel CRS in order to perform fast re-draw of all vector data when panning the map view. New coordinates are simply calculated by an offset of the number of pixels panned. A set of coordinates are also stored as grid points at the maximum zoom level. This is used when changing zoom level. The coordinates at the new zoom level is down sampled from the maximum zoom level grid in order to preserve exactness of the coordinates. The maximum zoom level grid can be seen as a maximum pixel resolution CRS. It is necessary to avoid another transformation from the original KML coordinates when zooming into higher pixel resolutions. The original KML coordinates are also stored in its original CRS. These can be used to re-project the data. One example of when this is done is when changing base map, which could be using different CRS and provide different zoom level resolutions. This doublet representation of the vector data held in memory is far from optimal and must be reconsidered in future work.

This simple implementation does not use the common geometry base objects as suggested in the design analysis. That is a shortcoming that should be corrected for in future development. By now the Java representation of KML data closely follow the hierarchy of the KML standard. Objects and variables often have a corresponding KML tag. The tags <Point>, <LineString>, <LinearRing> and <Polygon> are the fundamental geometry representatives in KML and each of them has a corresponding class in the Java implementation. These classes perform more or less successful when being drawn on screen, partly because they are represented by different Android library classes as shown in table 6.1, something to consider in the coming implementing of the common geometry base objects.

Table 6.1: KML geometry object representation and performance.

Android class (representation)	**Android Canvas draw function**	**API class**	**Performance**	**Might be replaced with**
Path (inner and outer bounds)	drawPath	KMLLinearRing, KMLPolygon	Good	-
Rect (containing a bitmap icon)	drawBitmap	KMLPoint	Good	-
Float[] (line vertices)	drawLine	KMLLineString	Unsure	(unclosed) Path, if drawPath renders faster than drawLine.

Some experimenting has been carried out for showing labels with overlay objects. The difficulty is to show labels rotated along the direction of a line object. The calculations take to much time for getting smooth map rendering at scrolling. This is a good example of where calculations should be performed asynchronous for not slowing down the UI appearance. One possible solution could be to not include the actual vector data in the real-time screen update, but only offsetting the graphic raster image representation of the overlays drawn on the previous screen update. Then request an asynchronous re-draw of all overlays. This approach would guarantee high UI responsiveness no matter what slow drawing algorithms the overlays implement.

6.4.3 Demo application

A demo application was developed using the implemented map API. The scenario for the use of the application was the needs of one of Digpro's typical customers, an energy company distributing district heating to the end users in a warm water pipe line system. Field work, such as inspection, connecting new customers or repairment, require up-to-date information about the system. Using a mobile phone or tablet with a smart map application could help the worker a lot. The requirements for the demo application was to show vector data about the pipe line system on top of a background map.

The application has two base maps to choose between, a custom WMS or OpenStreetMap. The vector overlays are different KML files distributed by a custom KML-service. The vectors are clickable to show information about the features. Depending on what data the user is interested in the different map layer's visibility can be altered in the meny system. Figure 6.6 shows three screen-shots from the demo application. Note that all data used are fake and only for demonstrational purpose.

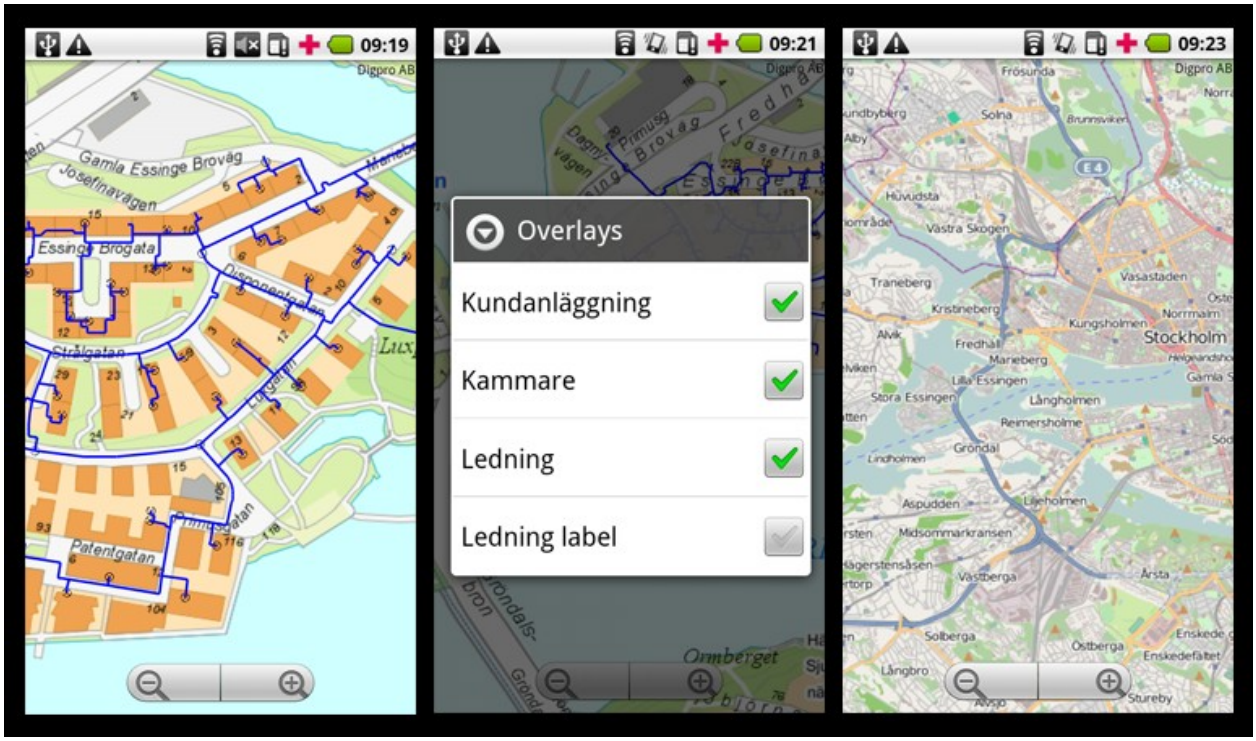

Figure 6.6: Screen-shots from the demo application.. From left to right: 1) WMS base map with KML vector data overlay of a district heating pipe line system. 2) Options sub-menu view for choosing visible layers. 3) OpenStreetMap base map of western parts of Stockholm, Sweden.

7 Discussion

With the demo application running on an Android powered mobile phone the main goal of this study is accomplished. Maybe more detailed questions could have been formulated and answered if the scope of the study had been more narrow and the method more specific than to developing software features additively to as much as there were time for. Selecting a couple of related problems and focusing on them would maybe have been a better way to be able to come up with more new knowledge about mobile GIS.

One of the main issues implementing the API was the chain of linked questions concerning how to efficiently represent and store map data and to decide what map tiles and vector features to draw on screen updates and at what location to draw them. Different implementations have different advantages and backsides and perform differently in different situations. At the time of implementing support for simultaneous display of map data with different coordinate referens systems (CRS) it seemed like a good idea to be using the screen pixel coordinate system as the internal common CRS. Because in the end all coordinates would anyway have to be projected into this in order to be displayed. The mistake made was that the pixel coordinate space of current zoom level was chosen as the internal CRS. This works perfectly fine when panning the map (all coordinates are just shifted the number of panned pixels) and when zooming out (all coordinates are divided by a factor). The problems arise when zooming into finer detail levels. Then the coordinates can not just be multiplied by a factor. That would not result in any better resolution. Instead the coordinates would have to be reprojected from the original map data or, as implemented in this study, down sampled from an additional instance of coordinate set stored as the pixel coordinates at the highest possible zoom level. When writing this discussion it seems more reasonable that the pixel coordinate space of the highest zoom level should have been used as internal CRS as downsampling coordinates by the right factor at zooming and panning would not be very demanding arithmetic operations.

One of the objectives of this study was to compare performance of map applications running either natively or as a web application. That comparison was not done as no good enough API for making the web application was found. Since then some of the developers of the open source web application map API project OpenLayers have put some effort on trimming OpenLayers for mobile use. So now would be the time to do the comparison.

The results of this study clearly indicate that mobile handheld devices of today are capable of running powerful map applications. The overall question of how far GIS can be taken on a handheld device with limited power and resources is still open for further investigation. Future work springing directly from this study include to further refine the API design, complete the implementation and realising more GIS-functionallity. A very interesting topic that could make a nice topic for another final thesis within the field of mobile GIS would be to investigate how Android's SQLite database could perform as a spatial database, with support for geometric shapes and spatial indexing.

8 Conclusions

Android is a good platform choice for implementing generic mobile map client applications. It provides the speed, resources and user interface of a modern mobile operating system. Computing performance differences does not seem to be the main issue when choosing platform. Performance differences are marginal between concurrent platforms and one platform's step ahead in performance and new features will soon be followed by concurrent systems. Intended uses and user groups are important when choosing platform for a specific application. Knowing these could make the choice easier when analysing determining factors such as available application distribution structures, market shares, openness, security and integration with other systems. The expected growing market shares and the openness of Android make it a good platform for third party application development.

Mobile web application implementations look promising but are not yet mature. No suitable open source API was found to build a test implementation of the proposed map client design upon. Available API:s does not perform well enough to put effort on such development yet. But that should change rather soon, as the mobile and internet industry work hard on performance boosts and standards for web applications.

There are holes to fill in the open source community regarding generic map API:s for Android. The design and implementation introduced in this work provide the basis for such an open source project. Though it needs additional developing, many fundamental features are implemented, such as accessing various map data sources (WMS, TMS/OpenStreetMap), vector overlays (KML), a projection interface for transforming between coordinate reference systems and a simple menu system. The design also provide suggestions for taking the work further towards a common internal data model with database storage for effective offline use and data manipulation, and a plug-in interface for customised functionality.

The development of the map API is a first step to extend the knowledge of how well a modern mobile platform like Android can perform as a host for mobile GIS. Possible extensions of this work could be to investigate to what level of completion a GIS could be implemented on a mobile device. It would include further work to identify limiting factors and what problems must be solved regarding spatial databases and GIS operations in a limited computing environment, client-server communication optimization and user interface design.

References

Allen, M. 2009. *Palm webOS Rough Cuts - The Insider's Guide to Developing Applications in JavaScript using the Palm Mojo Framework*. [e-book] Available from: http://palmone.r3h.net/downloads.palm.com/webos_chap1.pdf, published by O'Reilly Media, CA.

Android Developers. 2010. *Designing for Performance* [online]. Available from: http://developer.android.com/guide/practices/design/performance.html [Accessed 6 July 2010].

Apache Software Foundation. 2010. *Frequent Questions about Apache Licensing* [online]. Available from: http://www.apache.org/foundation/licence-FAQ.html [Accessed 6 July 2010].

Bada. 2009. *Samsung launches open mobile platform* [online]. Available from: http://www.bada.com/samsung-launches-open-mobile-platform/ [Accessed 7 July 2010].

Balagtas-Fernandez, F., Forrai, J. and Hussmann, H. 2009. Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices. In *Human-Computer Interaction - INTERACT 2009 - 12th IFIP TC 13 International Conference, Proceedings*, Uppsala, Sweden, August 24-28, ISBN-13 9783642036545, published by Springer Verlag, Heidelberg, Germany, pp. 243-246.

Barton, J. J., Zhai, S. and Cousins, S. 2006. Mobile Phones Will Become the Primary Personal Computing Devices. In *Proceedings 2006. 7th IEEE Workshop on Mobile Computing Systems and Applications*, Orcas Island, WA, USA, 6-7 April.

Biuk-Aghai, R. P. 2005. Performance tuning in the MacauMap mobile map application. In *Proceedings - 3rd International Conference on Information Technology and Applications*, Sydney, Australia, July 4-7, ISBN-13 9780769523163, published by Institute of Electrical and Electronics Engineers Computer Society, pp. 316-321.

Brinkhoff, T. 2008. Supporting Mobile GIS Applications by Geospatial Web Services. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII.* Proceedings of Commission IV. Part B4. ISPRS Congress, Beijing 2008, ISSN 1682-1750, pp. 865-870.

Davis Jr., C. A., Kimo, Y. J. and Duarte-Figueiredo, F. L.P. 2009, OGC Web Map Service Implementation Challenges for Mobile Computers. In *17th International Conference on Geoinformatics*, Fairfax, VA, United states, August 12-14, ISBN-13 9781424445639, published by IEEE Computer Society, Piscataway, NJ, United States.

Dillemuth, J. 2005. Map Design Evaluation for Mobile Display. In *Cartography and Geographic Information Science*, 32(4), October, ISSN 15230406, Published by American Congress on Surveying and Mapping, pp. 285-301.

Bloch, J. 2008. *Effective Java*. 2nd edition, ISBN-13 978-0321356680, published by Pearson Education, Boston, MA, USA.

Ericsson. 2010. Mobile Maps Terms of Use [online]. Available from: https://labs.ericsson.com/apis/mobile-maps/terms-of-use [Accessed 6 July 2010].

Godwin-Jones, R. 2008. Emerging Technologies – Mobile-Computing Trends: Lighter, Faster, Smarter. In *Language Learning & Technology*, 12(3), October, ISSN 1094-3501, pp. 3-9.

Gundotra, V. 2010, *Google I/O Keynote Day 2* [online video]. Available from: http://www.youtube.com/watch?v=IY3U2GXhz44 [Accessed 6 July 2010].

Haddad, I. 2010. *Introduction to the MeeGo Project* [online]. The Linux Foundation. Available from: http://wiki.meego.com/images/MeeGo_Introduction.pdf [Accessed 7 July 2010]

Harun, H. , Jailani, N., Bakar, M. A., Zakaria, M. S. and Abdullah, S. 2009. A Generic Framework for Developing Map-Based Mobile Application. In *Proceedings of the 2009 International Conference on Electrical Engineering and Informatics*, Selangor, Malaysia, August 5-7, ISBN-13 9781424449132, published by IEEE Computer Society, Piscataway, NJ, United States, pp. 434-440.

Lin, F. and Ye, W. 2009. Operating System Battle in the Ecosystem of Smartphone Industry. In *Proceedings - 2009 International Symposium on Information Engineering and Electronic Commerce*, Ternopil, Ukraine, May 16-17, ISBN-13 9780769536866, published by IEEE Computer Society, Piscataway, NJ, United States, pp. 617-621.

Nilsson, E. G. 2009. Design patterns for user interface for mobile applications. In *Advances in Engineering Software*, 40(12), December, ISSN 09659978, published by Elsevier Ltd, Oxford, United Kingdom, pp. 1318-1328.

OGC. 2010 (1). *WMS* [online]. Available from: http://www.opengeospatial.org/standards/wms [Accessed 6 July 2010].

OGC. 2010 (2). *WFS* [online]. Available from: http://www.opengeospatial.org/standards/wfs [Accessed 6 July 2010].

OGC. 2010 (3). *KML* [online]. Available from: http://www.opengeospatial.org/standards/kml [Accessed 6 July 2010].

Open Handset Alliance. 2007. *Industry Leaders Announce Open Platform for Mobile Devices* [online]. Available from: http://www.openhandsetalliance.com/press_110507.html [Accessed 6 July 2010].

OSGeo. 2010 (1). *WMS Tiling Client Recommendation* [online]. Available from: http://wiki.osgeo.org/wiki/WMS_Tiling_Client_Recommendation [Accessed 6 July 2010].

OSGeo. 2010 (2). *Tile Map Service Specification* [online]. Available from: http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification [Accessed 6 July 2010].

Speckmann, B. 2008. The Android mobile platform. MSc thesis, Department of Computer Science, Eastern Michigan University.

Tsou, M., Guo, L. and Howser, A. 2005. A Web-Based Java Framework for Cross-Platform Mobile GIS and Remote Sensing Applications. In *GIScience and Remote Sensing*, 42(4), October/December, ISSN 15481603, published by V.H. Winston and Son Inc., pp. 333-357.

van Tonder, B. and Wesson, J. 2008. Using Adaptive Interfaces to Improve Mobile Map-Based Visualisation. In *Proceedings of the 2008 Annual Research Conference of the South African*

Institute of Computer Scientists and Information Technologists, Wilderness, South Africa, October 6-8, ISBN-13 9781605582863, published by Association for Computing Machinery, New York, United States, pp. 257-266.

W3C. 2010. *HTML5 - A vocabulary and associated APIs for HTML and XHTML*, Working Draft 24 June 2010 [online]. Available from: http://www.w3.org/TR/2010/WD-html5-20100624/ [Accessed 7 July 2010].

Wallin, L. 2010. Cited in Android succé för Sony Ericsson, *Computer Sweden*, 19 May, published by IDG, Stockholm, Sweden.

Series from Lund University
Department of Physical Geography and Ecosystem Science

Master Thesis in Geographical Information Science (LUMA-GIS)

1. *Anthony Lawther:* The application of GIS-based binary logistic regression for slope failure susceptibility mapping in the Western Grampian Mountains, Scotland. (2008).
2. *Rickard Hansen:* Daily mobility in Grenoble Metropolitan Region, France. Applied GIS methods in time geographical research. (2008).
3. *Emil Bayramov:* Environmental monitoring of bio-restoration activities using GIS and Remote Sensing. (2009).
4. *Rafael Villarreal Pacheco:* Applications of Geographic Information Systems as an analytical and visualization tool for mass real estate valuation: a case study of Fontibon District, Bogota, Columbia. (2009).
5. *Siri Oestreich Waage:* a case study of route solving for oversized transport: The use of GIS functionalities in transport of transformers, as part of maintaining a reliable power infrastructure (2010).
6. *Edgar Pimiento:* Shallow landslide susceptibility – Modelling and validation (2010).
7. *Martina Schäfer:* Near real-time mapping of floodwater mosquito breeding sites using aerial photographs (2010)
8. *August Pieter van Waarden-Nagel:* Land use evaluation to assess the outcome of the programme of rehabilitation measures for the river Rhine in the Netherlands (2010)
9. *Samira Muhammad:* Development and implementation of air quality data mart for Ontario, Canada: A case study of air quality in Ontario using OLAP tool. (2010)
10. *Fredros Oketch Okumu*: Using remotely sensed data to explore spatial and temporal relationships between photosynthetic productivity of vegetation and malaria transmission intensities in selected parts of Africa (2011)
11. *Svajunas Plunge:* Advanced decision support methods for solving diffuse water pollution problems (2011)
12. *Jonathan Higgins:* Monitoring urban growth in greater Lagos: A case study using GIS to monitor the urban growth of Lagos 1990 - 2008 and produce future growth prospects for the city (2011).
13. *Mårten Karlberg:* Mobile Map Client API: Design and Implementation for Android (2011).
14. *Jeanette McBride:* Mapping Chicago area urban tree canopy using color infrared imagery (2011)
15. *Andrew Farina:* Exploring the relationship between land surface temperature and vegetation abundance for urban heat island mitigation in Seville, Spain (2011)
16. *David Kanyari*: Nairobi City Journey Planner An online and a Mobile Application (2011)