A development approach for geo-enabled mobile applications based on HTML5

A Rwanda case study and prototype

Author: ir. P. (Penny) Broman
E-mail: penny_broman@hotmail.com

Supervisors:
Professor: Prof. dr. M.J. (Menno-Jan) Kraak
Supervisor(s): dr. ir. R.A. (Rolf) de By

Version: 21-7-2012 (version 1.0)

Abstract

This study aims to build a comprehensive development approach for platform-independent geo-enabled mobile app development, based on relevant scientific studies and available development frameworks and approaches. The development approach should offer a suitable technical platform, and usable spatial mobile functions, and include context-awareness. The approach is described by the Spatial Mobile Application Development (SMAD) framework. The Sakaza Muhinzi case study is performed to verify the framework.

From the goals set for the SMAD framework, a framework quadrant is deducted. This quadrant contains four elements:

I. Technical architecture: The architecture presents the set up of the technical environment in the tools and techniques, and its relationships to develop the apps in a modular manner, and benefit from reusable objects and code.
II. Categorization of functions: Functions are categorized by complexity, context-awareness, technique of visualization, and security, and are related to the tools and techniques to successfully integrate these functions in the app.
III. Project approach: Presents a standard application development method to manage the project.
IV. Design and development: Offer the process and techniques/templates to design and build the apps. Techniques of web development, and mobile app development are used. Analysis and testing are also part of the design and development quadrant.

The SMAD framework is built on three building blocks: mobile apps, HTML5, and spatial functionality. Four reference frameworks contributed to the framework, leading to the following contributions:

- the Model-View-Controller principle (Trygve 1979)
- mobile cartography
- the usage of services
- context-awareness in a mobile environment

Three levels of applications are defined that are used to categorize the functions. The three levels of applications are based on the processes that are supported, and on how data is used within the application: (1) the basic applications that provide information in a predefined format, (2) the personalized app that combines sources of data, and aims to support individuals in performing operations and tasks, and (3) the context-aware applications that can be used by multiple users to collaborate, and in which functions are available to combine data, and perform analyses.

The Agile project approach is proposed, because of its incremental and evolutionary processes, which makes the development processes flexible and adaptive to changes. Further, Agile offers modular and lean processes, and is time-based, which builds on iterative and concurrent work cycles.

The design and development strategy is based on Picchi's approach (Picchi 2011), in which the content-out approach has a central role. The content-out approach is an approach, in which the user demands and expectations, and focus on the user interface, are the starting point. This strategy is translated into a stepwise approach of development, which makes the development of an app a straightforward process:

1. Choose a Bootstrap template to start with
2. Create the basic public content
3. Realize the page navigation
4. Choose an appropriate lay out, and styling
5. Include widgets, and other third party services
6. Add simple data functions
7. Add web maps
8. Add more complex data functions

The case study on Sakaza Muhinzi coffee farmers shows how a development project of geo-enable apps is executed, following the SMAD framework. Tools used within the SMAD approach are: TwitterBootstrap, for designing the user-interface with HTML, OpenLayers to serve webmaps, KnockOutJS for developing application logic in JavaScript, and CouchDB to store and serve the data. The SMAD framework proved to be successful looking at the goal to systematically develop geo-enabled mobile applications. The SMAD framework offers an approach , which is based on other scientific research, and on best practices shared by different specialists from the field. OpenLayers was included to offer geo-functions, and to geo-enable the apps. The combination of HTML, JavaScript, and CSS, resulted in achieving platform-independency. These three types of code can all be understood by web browsers.

Biographical sketch

This thesis report is written by ir. Penny Broman, as part of her second MSc thesis: "Geographical Information Management and Applications". Penny graduated in 1998 in "Technical Engineering and Management" at the University of Twente, Enschede, The Netherlands.
After graduation, Penny started working in the corporate consulting services as a Management Consultant ERP for PriceWaterhouseCoopers. After two years of ERP consultancy, Penny switched to consultancy in business intelligence, and data warehousing. Her specialties are, amongst others, user requirements specifications, information analysis, business analysis, and metadata management. In 2008 she started working for the Province of Noord-Brabant, as coordinator data warehouse. A data warehouse in which administrative, and spatial data were integrated, was set up. The interest for integration of spatial and administrative data to improve management information, and the wish to better understand the ideas of geosciences, inspired Penny to start her second study.

Lieve Mika,

Nu ben je alweer 5, en ga je na de grote vakantie naar groep 3!
Wat ben ik trots dat je al kan lezen en schrijven
Op de laptop, tablet, Wii, en mobiel speel je graag
Ik speelde nog Pong en spellen op de Atari/CBS
Miss Pacman was mijn favoriet
Jij bent nu verkikkerd op Angry Birds, en Cut the Rope
Gewoon bij mama op de mobiel
Ik ben bang dat het niet meer lang duurt
Dat ik niet meer volg waarmee jij bezig bent
Wat zal ik trots zijn!

Je Mams

(Samsung Galaxy SII back)

Acknowledgements

Last year, I almost decided to stop my GIMA-studies, having only my thesis work as last hurdle for my graduation. The first time I wrote a thesis was in 1997/1998, finishing my studies in Industrial Engineering and Management. With my fellow student, and friend Fiona Ting A Kee, I performed a thesis project on information needs of the most significant development actors in rural areas, and how ICT could be used to fulfill these needs. It was one of our greatest adventures, travelling to Ghana and Tanzania, and to do field research in the capitals Accra, and Dar es Salaam, as well as in the rural areas around Tamale, and Mbeya. On the other hand, I remembered the amount of work to do a thesis project, especially writing the thesis report, and that part was also the reason to almost quit my second masters.

The first year of GIMA was a busy year, as a full-time student, I managed to finish all modules besides my job at the Province. The second year I started with a period of no study, which in fact took a bit longer than the two months thought of. Last year, just before I had to quit my job at the Province, I decided to go for that final step, trying to finish that masters. Thanks to a few fellow students, my friends, and direct family, I got motivated again. Thanks all, for that!

A special thanks to my partner Jeroen de Cocq van Delwijnen, and son Mika, who had to accept my working late at nights, and missing out some trips. Also the interest showed by some of my best friends, Annemiek Takens, Fiona Ting A Kee, Wendy van Veelen, Sinling Choy, and Mirjam Aardoom, kept me motivated to continue: thank you very much!

I would like to thank dr.ir. R.J.A. (Ron) van Lammeren, and ir. H.J. (John) Stuiver for brainstorming with me on a subject for my thesis project to be performed for the Province of Noord-Brabant, which I have not started. This brainstorming made me realize it is important to pick a subject that really took my interest. I wanted to start on something new, something more visible.
This search for a new topic brought me to my supervisor of this thesis work: dr. ir. R.A. (Rolf) de By. I would like to thank Rolf very much for his inspiring way of thinking, and reasoning. He inspired me to work out the thesis project, the way it is now presented here. His feedback was critical, which made me realize again, I was supposed to write a scientific thesis report, and not just some inconsistent essay. Another thank you to my responsible professor, Prof. dr. M.J. (Menno-Jan) Kraak. I got to know him at the beginning of my GIMA studies, having an introduction conversation. He is a very enthusiastic, bright, and energetic person, who really knows to inspire others on the geosciences field, including me.

Last persons in row I would like to thank, are Klaasjan te Voortwis and former colleague at the Province Jan van Sambeek. They helped me during my thesis work. Klaasjan with his incredible ‘nerdy’ developing skills, and intelligence, and Jan, as an expert on the subjects of spatial data, and GIS. They supported me, and motivated me to go on with the project in difficult times.

Table of Contents

Table of Figures

Table of Tables

List of abbreviations and acronyms

3G	Third Generation mobile telecommunication
A-GPS	Assisted GPS
API	Application Programming Interface
Cell-ID	Cell identification
CSS	Cascading stylesheet
DOM	Document Object Model
GIS	Geographical Information System
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Internet Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
JSONP	JSON with Padding
LAN	Local Area Network
MVC	Model View Controller
MVP	Minimal Viable Product
MVVM	Model View ViewModel
OGC	Open Geospatial Consortium
Open GL	Open Graphics Library
Open GL ES	Open GL for Embedded Systems
SDK	Software Developer Kit
SMAD	Spatial Mobile Application Development
SQL	Structured Query Language
UI	User-Interface
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
W3C	Wide Web Consortium
Wi-Fi	Wireless Fidelity
WLAN	Wireless LAN
XML	Extensible Markup Language

Terminology

To enhance reading, some terms are described, based on information from Wikipedia.org. The description, as well as links to Wikipedia.org, for further readings, are in the table below.

Term En.Wikipedia.org/wiki	Description
Boilerplate /Boilerplate_code	In computer programming, boilerplate code or boilerplate is the term used to describe sections of code that have to be included in many places with little or no alteration.
Features /Mobile_phone_features	The features of mobile phones are the set of capabilities, services and applications that they offer to their users. Low-end mobile phones are often referred to as feature phones, and offer basic telephony. Handsets with more advanced computing ability through the use of native software applications became known as smartphones.
Github /Github	GitHub is a web-based hosting service for software development projects that use the Git revision control system. GitHub is the most popular Git hosting site, and the most popular open source hosting site.
Grid /Grid_(page_layout)	A typographic grid is a two-dimensional structure made up of a series of intersecting vertical and horizontal axes used to structure content. The grid serves as an armature on which a designer can organize text and images.
Greenfield /Greenfield_project	In many disciplines a greenfield is a project that lacks any constraints imposed by prior work. The analogy is to that of construction on greenfield land where there is no need to remodel or demolish an existing structure.
Manifest / Cache_manifest_in_HTML5	The cache manifest in HTML5 is a software storage feature which provides the ability to access a web application even without a network connection.
Mockup /Mockup	In manufacturing and design, a mockup, or mock-up, is a scaled or full-size model of a design or device, used for teaching, demonstration, design evaluation, promotion, and other purposes.
Open GL /Open_GL	A standard specification defining a cross-language, multi-platform API for writing applications and simulating physics, that produce 2D and 3D computer graphics.
Open GL ES /OpenGL_ES	A subset of the OpenGL 3D graphics application programming interface (API) designed for embedded systems such as mobile phones, PDAs, and video game consoles.
Use case /Use_case	In software and systems engineering, a use case is a list of steps, typically defining interactions between an actor and a system, to achieve a goal. The actor can be a human or an external system.
WebKit engine /WebKit	WebKit is a layout engine designed to allow web browsers to render web pages. WebKit powers the Apple Safari and Google Chrome browsers.
Web service /Web_service	The W3C defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network.
Widget /Sorftware_widget	A widget is generally speaking a small application, a little bit of code, that does one simple thing but does it well. The second important things about a widget, is that they can be combined together.[1]
Wire frames /Website_wireframe	A website wireframe, also known as a page schematic or screen blueprint, is a visual guide that represents the skeletal framework of a website.
Wrapper /Wrapper_pattern	In computer programming, the adapter pattern (often referred to as the wrapper pattern or simply a wrapper) is a design pattern that translates one interface for a class into a compatible interface. An adapter allows classes to work together that normally could not because of incompatible interfaces, by providing its interface to clients while using the original interface.

[1] http://blog.landspurg.net/what-is-a-mobile-widget/

Part I. Introduction

1 Introduction

1.1 Motivation and problem statement

All over the world, more and more mobile applications, apps, are becoming available for various mobile platforms. In 2011, as predicted by market research firm Gartner, mobile app stores will have served 17.7 billion downloads, against an estimated 8.2 billion in 2010. Application downloads will soar to 185 billion by 2014. (Anthes 2011) In December 2011 the Android App Market registered 380,297 apps, and 10 billion downloads ((nl.wikipedia.org), (Distimo 2011), (Bonnington 2011)).

Many apps are developed for a specific mobile platform like Android, Apple's iOS, Blackberry OS or Symbian, and afterwards or concurrently developed on another platform. The inability to create a single application that can operate across all devices and platforms is called mobile fragmentation. Fragmentation has been labeled as the major issue facing the growth and future success of mobile technology. (mobileadvertsingbook.com: 2011) The effort put into development and maintenance of the same application for various platforms increases, as code has to be rewritten for various platforms, and updates and changes have to be rolled out to all supported platforms. For mobile apps and web development, HTML5 is nowadays often claimed to be a way to overcome the fragmentation problem. Develop only once in HTML5, and roll out the app to any one of the available mobile platforms. "Industry experts agree that HTML5 will play a prominent role in mobile development in 2012 and will be used to try to overcome fragmentation issues that the industry has consistently battled." (Johnson 2011)

In the field of spatial information, more and more applications for collecting, presenting and analyzing spatial data are launched. These applications, including mobile apps, increase awareness on the importance of location-based information, and the possibilities new technologies are bringing to a broad user base. The modern smartphones all incorporate a GPS device to determine location, and amongst others offer functions to gather more context-specific information, for example, by access to web services on current weather conditions, points of interest, and traffic information. Location can not only be determined by GPS, but also, for example, based on cell identity in the GSM/UMTS network, and within Local Area Networks. Recently, Engineers from the University of Oulu in Finland announced a new breed of indoor positioning technology that uses the accurate compass available in every iPhone and Android device, and does not require WiFi or other beacons. (Anderson 2012)
The mobile device turned in a powerful and portable means, by the access to web services combined with hardware configurations, including GPS, the full color high quality screen, one or more cameras, and increased battery capacity and processing power. It can be used for presenting and interpreting maps, satellite images, and non-spatial data associated with map elements. The smartphone, also, offers more complex functions, like mobile augmented reality, and other location-based and context-aware functions.
Already in 2001, Reichenbacher and Tumasch (Reichenbacher 2001) wrote: "The emergence of mobile computing and wireless devices has brought about a whole palette of new possibilities for cartography … they can present up-to-date spatial/non-spatial information in an individual, dynamic, and flexible way, and the user being mobile."

The development of mobile technology is an important field of business in developing countries, and may even be in an advantageous position compared to western economies. In western economies, a majority of households own one or more computers, have one or more cell phones, and have fast internet connections. In developing countries, more people are dependent on public or commercial internet facilities, and most households do not have a fixed line to their house. The ITU publication "Measuring the information society" (ITU 2009) states that fewer than five in 100 Africans owned a computer in 2007, and even fewer had connection to the internet.

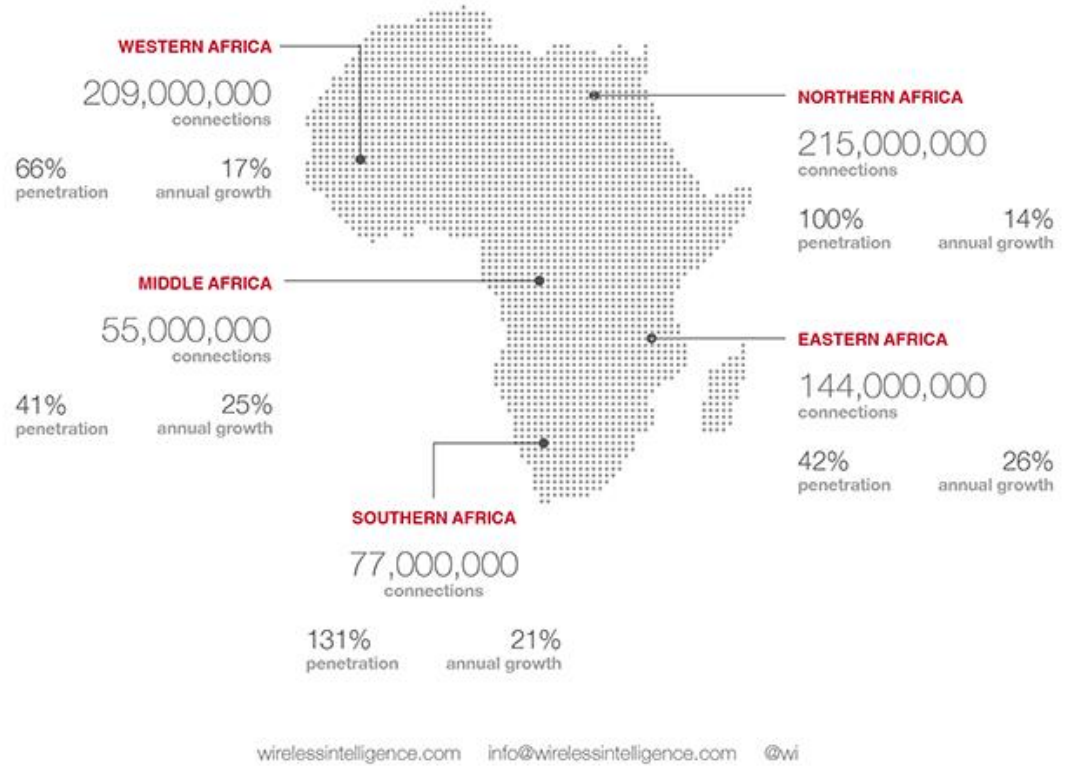

Figure 1.1 Mobile connections, penetration, and annual growth (2012) in the African regions
WirelessIntelligence.com (Croft 2012)

An interesting development is that Africa continues to have the highest mobile growth rate, 32 per cent in 2006/2007 according to ITU 2009, and 16 per cent annual growth in 2012 according to WirelessIntelligence.com (Croft 2012). Only one in 50 people owned a mobile phone at the beginning of the century (ITU 2009), as nowadays two third of the African population owns a cell phone (Croft 2012), see also Figure 1.1, which shows up-to-date figures on the number of mobile connections, the penetration rate, and the annual growth figures per region in Africa. Even more people have access to a mobile phone, as people can use a mobile owned by family of friends. In 2011, the mobile phone networks covered three-quarters of the population in many African countries. The four biggest mobile phone markets in Africa are Nigeria, South Africa, Kenya, and Ghana (Rao 2011) Last year, according to Rao, African mobile operators became serious about data availability and cost packaging for everyday Africans, and countries like Kenya and Rwanda have encouraged investment in ICT infrastructure.

Like in western societies, mobile phones are transforming life in Africa, by applications to check market prices, transferring money or simply reading the latest news, Facebook, and/or Wikipedia. Besides the common applications adapted from western countries, a whole range of apps is developed specifically aimed at the local market. Examples are:

- the Nigerian Constitution Blackberry App (Rotini 2011), which makes the Nigerian constitution accessible to Nigerians via their mobile devices.[2] Nowadays also available for other platforms besides Blackberry.
- the Airtel Card, which "Provides a developed-world service to the developing world with great use of existing and readily accessible technologies (such as MasterCard's network) to open up commerce and banking to the unbanked and underbanked" (Mobile World Congress 2011),
- M-Pesa (en.wikipedia.org 2011), which offer a mobile-phone based money transfer service for Safaricom, Kenya's largest mobile network operator, and
- M-Novels, or M4Lit, supporting teen leisure reading and writing around fictional texts in South Africa, using mobile media (Vosloo, Walton et al. 2009)

The developments described in this section, are an introduction on the thesis study, which aims to develop a framework, a development approach, for platform-independent geo-enabled mobile app development. This approach is verified by the Sakaza Muhinzi case study.
An underlying question is: Can farmers in developing countries like Rwanda benefit from the developments on smartphones, HTML5 and spatial functionality? The country is investing in its ICT infrastructure, however Rwanda is only placed 143 of 154 countries in the ITU ICT development index 2007 (ITU 2009). The question is, to what extent broadband is in place, nowadays. Further, smartphones are still expensive goods, which not everybody can afford. Nevertheless, it might be relatively easy and cheap to further roll out mobile facilities in countries like Rwanda, as the density of cell phones is already much higher than personal computers with internet access. Providing sensible information to the farmer could empower farmers and their cooperatives to make better decisions on for example: buying seeds and fertilizers, acting on current weather predictions and information on plant diseases and pests, and selling their products. Farmers, and their cooperatives, having the right information will be able to maximize turnover.
A framework on the development approach of geo-enabled mobile apps based on HTML5, is not an off-the-shelve product. This study aims to build a development approach for platform-independent geo-enabled mobile app development, based on relevant scientific studies and available development frameworks and approaches. The framework will be applied in the context of Rwanda, a developing east-African country, as the state of technology, maturity of end-users, and available resources plays an important role for eventual adaptation of a proposed solution.

Problem statement:

How to systematically develop geo-enabled mobile applications, based on HTML5 that offer different spatial functions to support professionals in their daily work and decision-making. A case study with Rwanda coffee farmers, their personalized profiles, their community network, and the production data from their fields.

[2] http://www.constitutionforall.com.ng/

1.2 Research objectives

The primary goal of this study is to understand the method of developing location-based mobile apps that are platform-independent, and create a development approach for them. This study aims to answer the questions “What kind of spatial functions are useful in mobile apps?”, “How to categorize these spatial functions and their information flows?”, and “How can these functions be implemented in these kind of applications?”. Conducting a case study of this method, and building a prototype app is the secondary goal.

The goal of the case study is to improve information sharing with and between coffee farmers, to support productivity improvements, also by means of personalized information on the farm and farmer. Information on actual weather conditions and forecasts, high resolution area maps, market prices, and information on plant diseases, are examples of important information that a farmer can act on, and thereby minimize production losses and maximize production returns.

To support coffee farmers in Rwanda to collect, share, and analyze information a mobile app is proposed as a future technical solution. A smartphone with access to a mobile network, and a specific app developed in HTML5, will enable the coffee farmer to have access to better information, bring information and data together, and support the farmer with proper functions. The objective of this research is to present a framework for mobile application development. The framework can be applied in the context, and the local conditions, of southern Rwanda. A working prototype is developed, based on HTML5, and various functions, specifically aimed at location-based information, and functions that support the farmer in accessing data and information, and in collecting and sharing local information with others.

Stakeholders of this research are all people interested in mobile application development, and specifically those interested in HTML5 and platform-independent development. Secondly, this research project aims at Rwanda farmers, who should benefit from a solution providing them with better information by mobile functions. The collection of local data aims to support scientists, who want to verify continental prediction productivity models, for example the HarvestChoice initiative; and ITC scientists specifically, who are sponsor of this research project, and want to enlarge knowledge on mobile geo-enabled applications.

1.3 Research questions

The research questions are based on the problem statement:
How to systematically develop geo-enabled mobile applications, based on HTML5 that offer different spatial functions to support professionals in their daily work and decision-making. A case study with Rwanda coffee farmers, their personalized profiles, their community network, and the production data from their fields.

1. What is HTML5 and how can it be used to determine location, present maps and associate non-spatial data with map elements?
2. What technique and development method can be used to develop a geo-enabled mobile app with HTML5? To what extent can platform-independence be achieved?
3. What specific conditions and challenges are to be dealt with developing spatial functionality in mobile applications for use in the developing world? What technical architecture is suitable?
4. What data, information and functionality are available and needed, and how can they be used and reused in the application?
5. How to verify the success of the development framework and the proposed design?

1.4 Research approach

The research approach of the study project, as described in this section, is depicted in Figure 1.2 Research approach. The research is divided in four phases, the colors refer to those used in Figure 1.2:

Phase 1. Development of a General Framework,
Phase 2. Case Study and Use Cases,
Phase 3. Prototyping,
Phase 4. Evaluation

1.4.1 Phase 1: Development of a General Framework

Research questions 1 and 2 on HTML, mobile applications, and spatial functions will be answered in Phase 1 “Development of a General Framework”. Part of the insight from Question 3 on the context of a mobile apps within developing countries will be input for the development of the framework. Key elements in this phase are to get insight in HTML5, Mobile apps, and Spatial functionality. Understanding these building blocks will lead to a development approach that offers a suitable platform, usable spatial mobile functions, and is context-aware.

1.4.2 Phase 2: Case Study and Use Cases

Research question 3, together with Question 4 on case study data, and functions are dealt with in Phase 2 “Case Study and Use Cases”. Use cases on presenting web maps, positioning, and adding personal spatial data are seen as three important fields of mobile functionality. By exploring these use cases, and describing their specifications properly, the study aims to have a solid basis for the actual prototyping phase. Both central data, as well as personal and local data will be input for the prototyping phase.

1.4.3 Phase 3: Prototyping

The prototyping phase will deal with input on Research question 4. First a basic prototype is built, based on the use case "Presenting web maps". After testing and validation, the prototype will be enhanced with positioning functionality, as well as the functionality to add personal spatial data.

1.4.4 Phase 4: Evaluation

Phase 4 "Evaluation" rounds up the study project by evaluation, providing lessons learned, and presenting next steps, and thereby provide answers to the last research question.

1.5 Related work

A wide range of topics is more or less important in conducting this research project.
From a technical perspective, subjects of interest are: HTML5 and its developments and usage, and application development and mobile application development more specifically. Unhelkar and Murugesan (Unhelkar, Murugesan 2010) present an enterprise "mobile application development framework" (MADF). This framework is based on presentation, application, middleware and binding, information, and communication, all in light of security. The framework shows a high-level technical view of what elements are to be considered building mobile business applications. The Model-View-Controller principle (Trygve 1979) has a direct relation to the elements of information, presentation, and application of this framework.
The service-oriented framework (Ennai, Bose 2008) allows mobile applications to easily interface with enterprise back ends, and be lightweight and flexible through built-in context awareness and a Web 2.0 front-end. The OpenGis® standards of the Open Geospatial Consortium (OGC[3]), and related services like WMS/WFS services, and OpenLayers (JavaScript maps) are developments that boost web based geodata dissemination.
The key topic "mobile spatial functionality" asks for experiences and studies on this specific topic. To what extent is mobile spatial functionality available in present day apps? In the field of mobile cartography, Konečný et al. ((Konečný, Staněk et al. 2007) , (Konečný, Staněk 2010)), appoint portrayal environment awareness, and portrayal homogeneity, besides task orientation and user orientation as the most important aspects of adaptive cartography. Portrayal environment awareness focuses on technological fit, and portrayal homogeneity on a uniform representation of heterogeneous data to allow quick browsing for geoinformation and easy visual comparison of spatial data.
The case study (Kagoyire, de By et al. 2011) of the developing country Rwanda asks for study materials about mobile app development in African countries, and Rwanda specific. The infrastructure (bandwidth, uptime, coverage), and also political, cultural and economical issues and stakeholders involved, will probably influence the way an app is set up and can operate, and in the end determine the future success of the mobile app. In other words, the context for the interaction between humans and computational services is created by the information in the physical and electronic environments. (Dey, Abowd et al. 2001) In their study on context-aware computing, Dey et al, define context as any information that characterizes a situation related to the interaction between users, applications, and the surrounding environment.(Dey, Abowd et al. 2001)

[3] www.opengeospatial.org

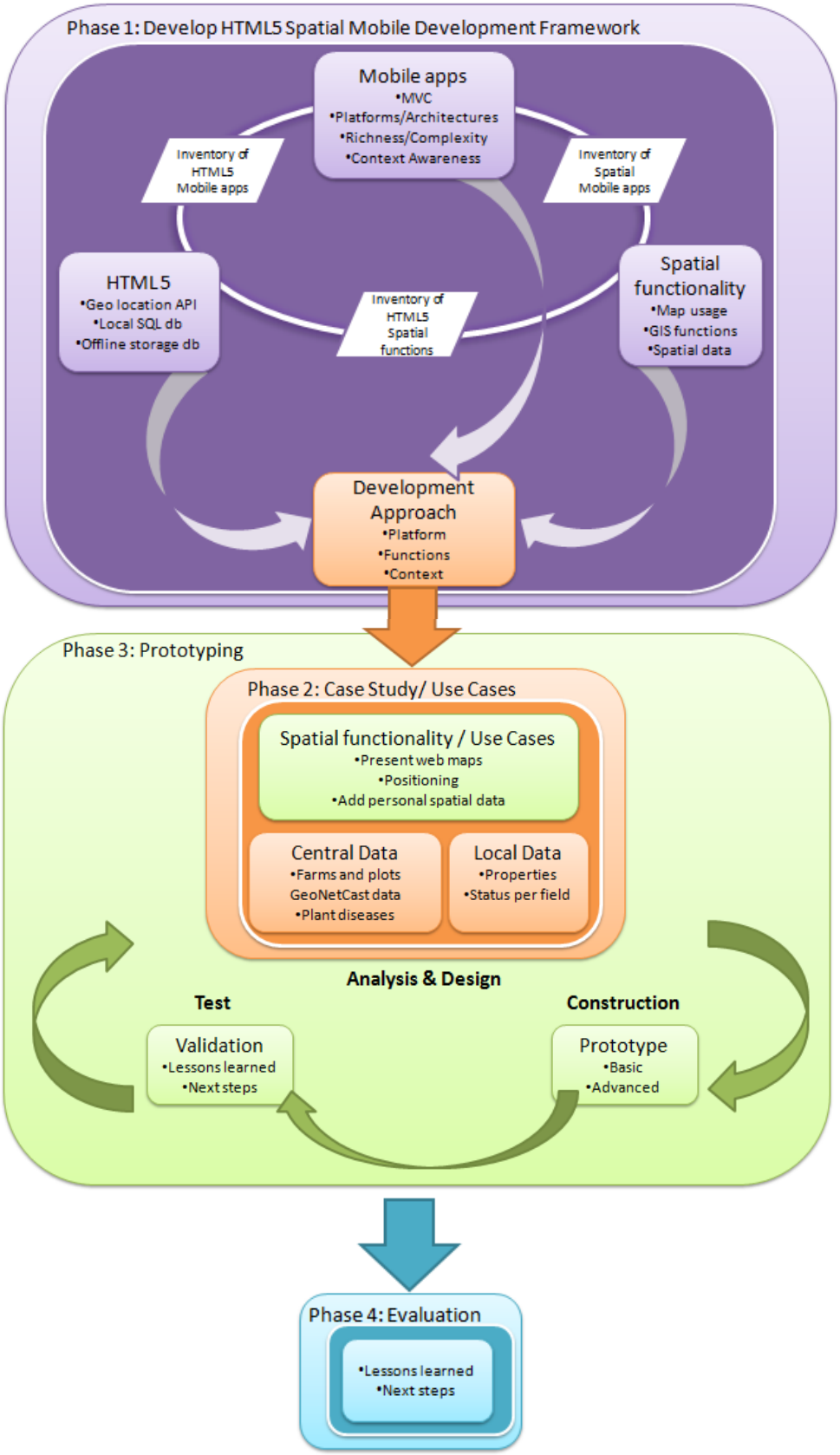

Figure 1.2 Research approach

Part II. Theoretical Framework

Phase 1: Development of a General Framework

Part II of this report aims to answer the Research questions 1 and 2 on HTML, mobile applications, and spatial functions. Part of the insight from question 3 on the context of a mobile apps within developing countries is input for the development of the framework as well. Key elements in this phase are to (Figure 1.3):

1. introduce various related frameworks, and define the framework aimed for,
2. introduce the building blocks: Mobile apps, HTML5, and Spatial functionality,
3. present the theoretical framework: development approach.

The knowledge gained from relevant frameworks and a good understanding of the three building blocks, mobile apps, HTML5, and spatial functionality, leads to a development approach that offers a suitable platform, offers usable spatial mobile functions, and is context-aware. This framework will be referred to as SMAD framework: Spatial Mobile Application Development framework.

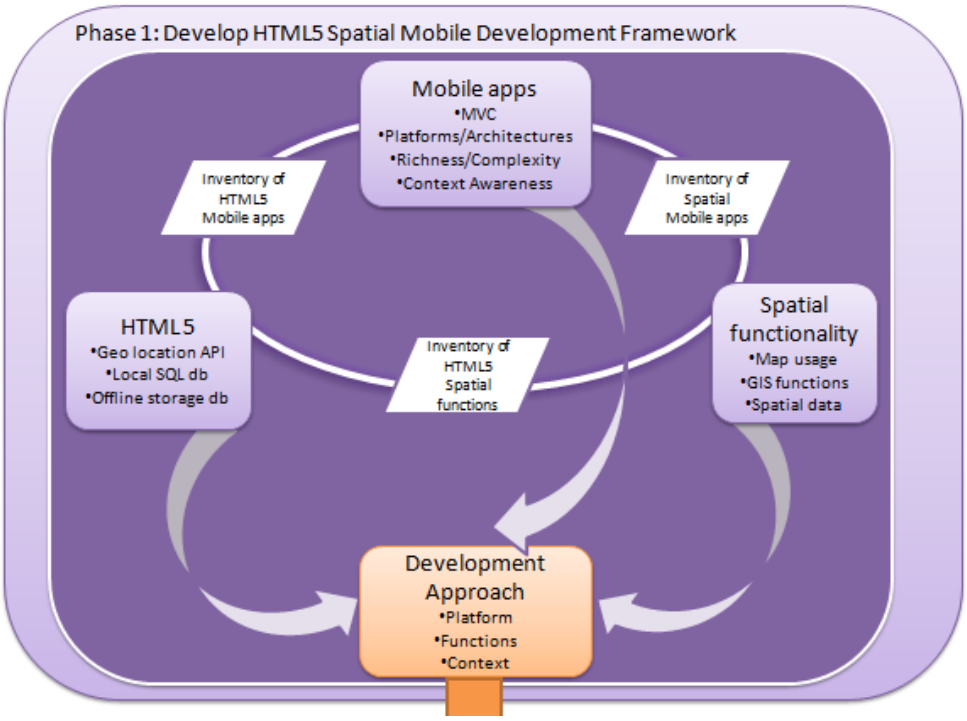

Figure 1.3 Phase 1: development of HTML5 spatial mobile development framework

2 Development of a framework

2.1 Introduction

This chapter sketches the research process in building the SMAD framework. The framework is like a coat rack, various concepts and practices are brought together to improve understanding of the real world. In this thesis the aim is to present a practical framework that offers a generic and standardized approach to develop geo-enabled mobile apps based on HTML5. By breaking up the framework in four separate focus areas, the framework aims to bring transparency in concepts and practices during the development process, and tries to keep the developer focused on developing the right functionality in a standardized manner. Section 2.2 defines the aims of the SMAD framework, and Section 2.3 describes four related frameworks to this thesis research, which were used to get better insight in building a framework, as well as deeper knowledge of concepts related to mobile applications, HTML5, and spatial functionality.

2.2 Framework definition and elements

This section starts with a definition of the SMAD framework. Based on the goals of the framework, the four core elements of the framework are presented:

- technical architecture
- categorization of functions
- project approach
- design and development

First, a proper definition of the term framework in relation to this thesis project is needed. A framework, like a model, aims to get a simplified representation of complex processes or systems, as to make operation and decisions on these processes and systems easier. In application development, the term framework often refers to the object oriented reuse technique (Johnson 1997). These kinds of framework provides a development architecture that offers classes of reusable code to a developer, which should speed up the development process. The term framework aimed for in this study refers to a conceptual framework, in which the technical architecture is a prominent element, however, is not the only concept regarded in the framework.
Smyth (Smyth 2004) refers to a definition of Reichel and Ramey (Reichel, Ramey 1987): “A conceptual framework is described as a set of broad ideas and principles taken from relevant fields of enquiry and used to structure a subsequent presentation”. The SMAD framework presented in this thesis is defined as a structure that combines reusable concepts, and practices in application development, to offer a structured method to students in GI science, and mobile application developers with an interest in GI science, to develop platform-independent mobile applications, that have prominent geo-functionality.
.

The SMAD framework (see also Figure 2.1) presented in this thesis aims to:

1. Propose a solid technical architecture to build geo-enabled, interoperable, mobile applications with HTML5.
2. Provide a systematic approach to design, develop, and test these applications based on rapid development cycles. Phases of Analysis & Design, Construction, and Testing are distinguished, and prototyping plays an important role.
3. Separate between reusable core mobile functions and services, and application specific functionality. A modular approach, in which flexibility and one-time development is strived for. The application should function online, as well as offline.
4. Provide a systematic approach based on a categorization of spatial functions, including context-awareness as a basic element of mobile spatial applications.

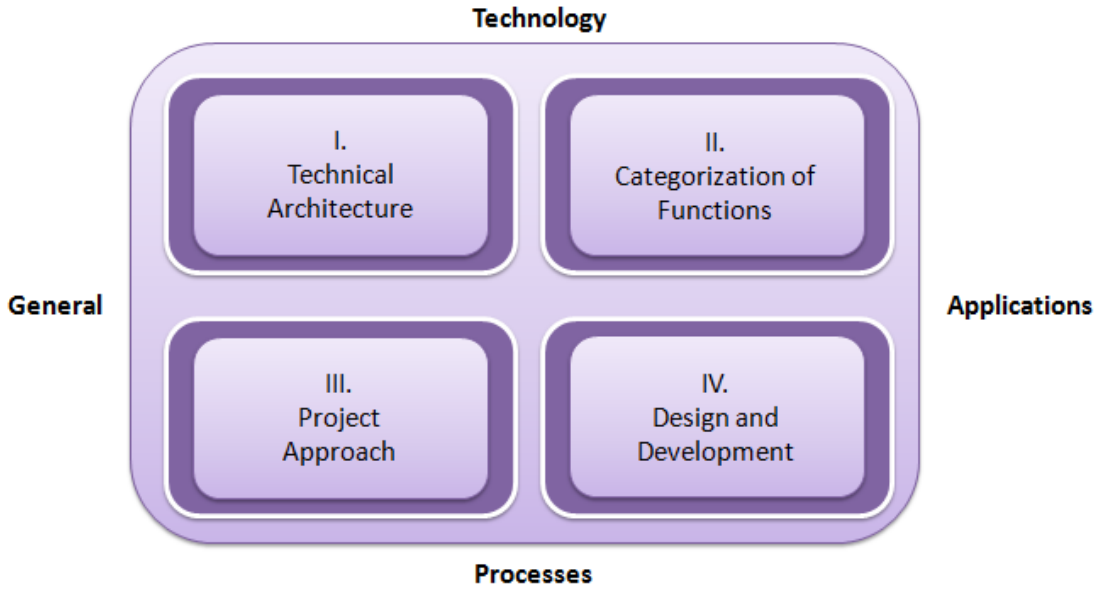

Figure 2.1 Mobile development framework quadrant

From the goals set for the SMAD framework, a framework quadrant (Figure 2.1), is derived. This quadrant contains four elements:

I. Technical architecture: The architecture presents the set up of the technical environment in the tools and techniques, and its relationships to develop the apps in a modular manner, and benefit from reusable objects and code.
II. Categorization of functions: Functions are categorized by complexity, context-awareness, technique of visualization, and security, and are related to the tools and techniques to successfully integrate these functions in the app.
III. Project approach: Presents a standard application development method to manage the project.
IV. Design and development: Offer the process and techniques/templates to design and build the apps. Techniques of web development, and mobile app development are used. Analysis and testing are also part of the design and development quadrant.

The left half of the framework quadrant provides the general approach, that is not application-specific (I and III). The right half of the quadrant, i.e., Elements II and IV, are application-specific. The approach of II and IV are the same for all projects, however the content, and thus designs, and applied functions differ, depending on the specific application that is targeted. The upper half of the quadrant is on technology (Elements I and II), as the lower part covers processes (Elements III and IV).

The technical architecture (Element I) forms the actual development environment on which the mobile apps are created, and is the foundation for how apps are constructed. The technical architecture is the enabler of the mobile app development. The categorization of functions (Element II) is a guide for the whole development process, and gives handles to the project manager, managing the project, and analysts, designers, developers, and testers, to develop the right product. The technology elements combine both in an evolving cookbook on how to develop platform-independent mobile apps with prominent geo-functionality.
The Elements III and IV are on the actual execution of a project with specific goals. The project management approach is needed for developing enterprise (professional) applications. People in different roles have to collaborate within the boundaries of available resources, time and money to deliver a successful end-product. The core process within the process of project management is the actual product development, which is represented by Element IV: Design and development. The technical elements (I and II) set the scope for application development, and offer tools to build apps, based on reusable concepts and code. The project sets the targeted product, and timelines. The analysts, designers, developers, and testers make use of these inputs to develop the right product, at the right time, within the boundaries of resources.

2.3 Mobile frameworks

The frameworks briefly introduced in this section exist in different formats, types, levels of detail, and have different purposes. All frameworks are on mobile, and have their specific focus: enterprise apps (2.3.1), service-oriented architecture (SOA, in 2.3.2), cartography (2.3.3), and context-awareness (2.3.4).

The first goal to search for scientific mobile frameworks, was to find a framework as a scientific basis for the SMAD framework. Second goal was to get insight in concepts, terminology, and techniques that are presently available in relation to mobile app development.

The four frameworks that are briefly presented in this section, are, respectively:

2.3.1: A mobile application development framework.
This section looks at three levels of mobile applications: presentation, application, and data. The section ends with a description of the related model-view-controller principle.
2.3.2: A service-oriented mobile framework
This framework is about simplicity of mobile applications, context-awareness, but above all about integration with and access to services. Here, the OpenGis® standards of the Open Geospatial Consortium (OGC[4]), and related services like WMS/WFS services, and OpenLayers (JavaScript maps) are explained.
2.3.3: A conceptual framework for mobile cartography
Important lessons deriving from this cartography framework are that in mobile cartography, user, context, and tasks are the main elements responsible for the adaptation of a visualization.
2.3.4: A conceptual framework of context-aware applications
This framework sets out an approach to simplify the applications itself, and separate certain functions and services, to be available for all applications. Mobile devices have features to gather context information, however, these are not be accessible for HTML5 code. An application that is more context-aware, is categorized as more complex, and richer.

2.3.1 A mobile application development framework

The mobile development framework is presented, because it gives a clear distinction of technical elements in a mobile development framework. It is also a framework that is applied in commercial environments. The technical architecture of the SMAD framework is derived from this framework, in which the model-view-controller concept, as described in this section, is a main component. The framework covers security, and is used in practice, however details on security and usage are not available in the paper on the framework.

Unhelkar and Murugesan (Unhelkar, Murugesan 2010) present an enterprise "mobile application development framework" (MADF). This framework is based on presentation, application, middleware and binding, information, and communication, all in light of security (Figure 2.2). The framework shows a high-level technical view of what elements are to be considered building mobile business applications.

[4] www.opengeospatial.org

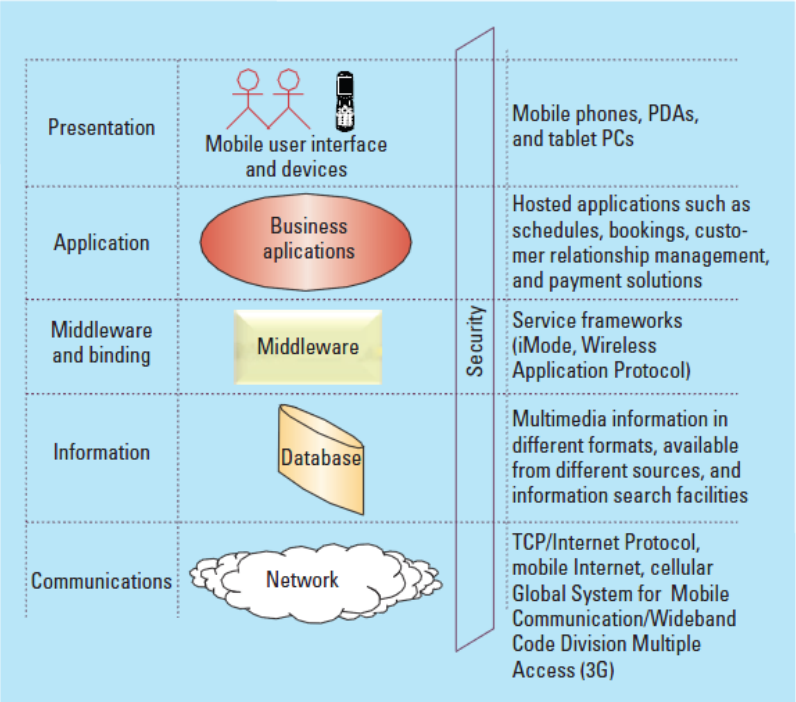

Figure 2.2 Enterprise mobile application development framework
(Unhelkar, Murugesan 2010)

The SMAD approach focuses on the elements presentation, application, data (information), and security. Here, only a short introduction of these elements is provided, as other sections will go in-depth on them. The applications must be suitable for smartphones, which sets constraints to the user interface, or presentation layer, but also offers specific functions to an application, which personal computers lack. The applications that are built according to the development approach, use HTML5 combined with JavaScript and CSS3, as basic coding ingredients. Existing applications, widgets and services, can be (re-)used within the application. Information needed within these applications is collected or is available from different sources, like web services and remote and local databases. Also local information can be gathered and stored. The local storage function must be suitable for spatial data. In presenting, gathering, and sharing data the user must be confident that private information remains private, and is thus reliant on good security.
The Model-View-Controller principle (Trygve 1979) has a direct relation to the elements of information, presentation, and application of the framework described above . The model is represented by the information element, the view by the presentation element, often regarded as the UI (User Interface), and the application element is the controller. The MVC principle can be used as a good coding practice to model interaction, and to separate the business model, and its underlying data, from the views, and the controller. This enhances reading the code, and also reusing the code (Wikipedia 2011). The MVC principle, is particularly of interest in developing larger applications, as for developing applications with only a few functions, this type of separation is surely overkill, according to Snook (Snook 2009). The SMAD framework sees the MVC principle as a basis to structure the apps, and to keep code separated, and thus more easily maintainable, and reuable.

Figure 2.3 depicts the relationship between the model, view, and controller. The solid lines indicate a direct association, while the dashed lines indicates an indirect association (observer pattern, in which state changes are notified automatically to the observers[5]). In an Android application, the GUI layout or view consists of XML files which include 'Layout' and 'View' elements, and controller and model functionality are written in Java. (Goadrich, Rogers 2011)

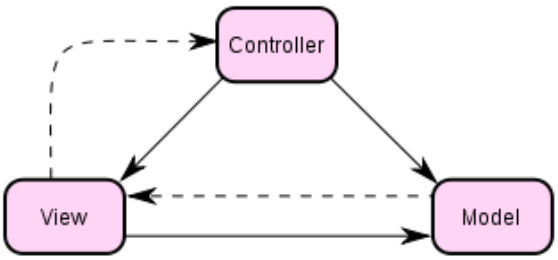

Figure 2.3 Diagram of the model-view-controller principle
(Wikipedia 2011)

Sauter et al. (Sauter, Vögler et al. 2005) present an extended MVC model, as MVC does not necessarily minimize code redundancy, in maximizing the number of supported devices, which is a primary goal of pervasive application development. Different devices, having their own screen sizes, for example, ask for new or at least adapted views and sometimes also adjusted controllers to offer the functionality. The paper offers a practical task-based implementation guideline for pervasive Java-based Web applications that separates an application's task state from its view state.

Another way to minimize code redundancy, for the screen size example, is the solution of flexible grids, as one of the best practices of responsive design. A fluid layout, which is a layout that is continuously responsive to the users display dimensions by targeting the max and min widths rather than specific device sizes or orientations, is created with CSS3 media queries. (Lara 2011)

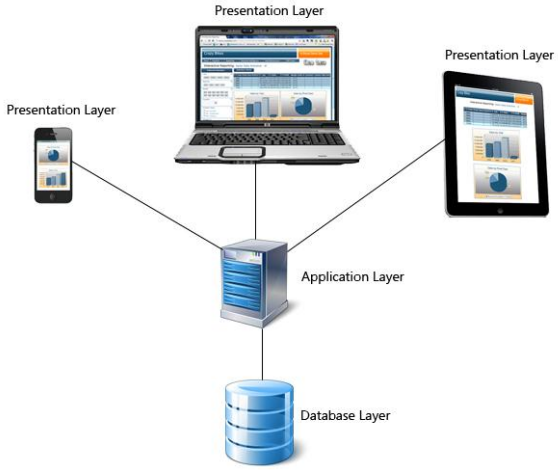

Figure 2.4 n-Tier architecture
(Stangarone 2011)

[5] http://en.wikipedia.org/wiki/Observer_pattern

An n-tier architecture (see Figure 2.4) is related to this MVC approach. The view, or presentation layer, is separated from the application layer. Data is stored in the database layer. (Stangarone 2011) In an n-tier architecture, multiple presentation layers can be created for each application, each optimized for a different device.

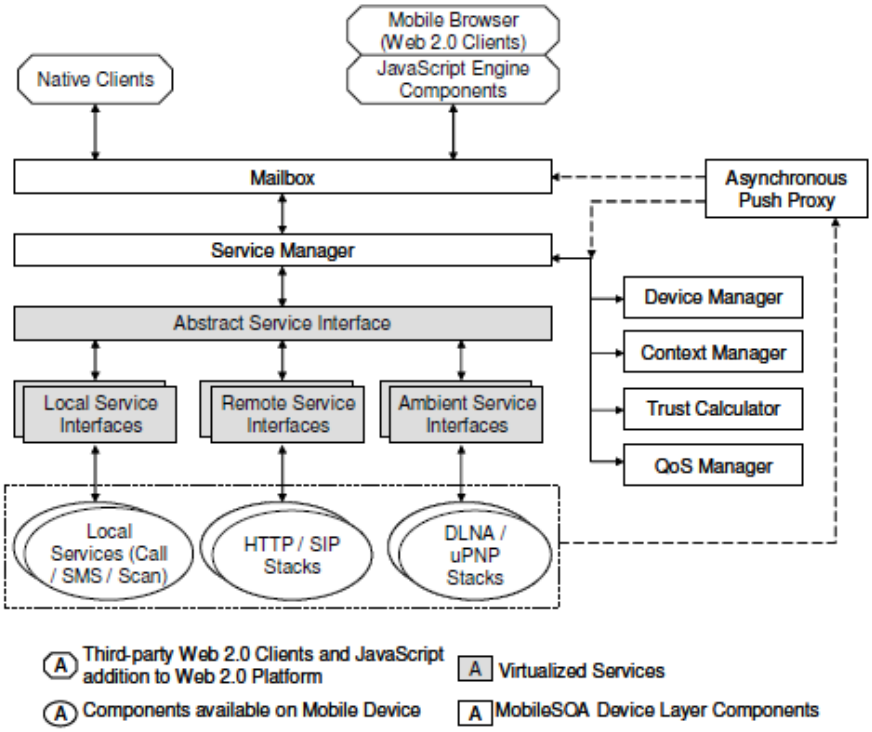

Figure 2.5 Mobile SOA System Architecture
(Ennai, Bose 2008)

2.3.2 A service-oriented mobile framework

The second framework is the SOA mobile framework (Figure 2.5), which is described at the more practical level; the mobile system architecture. Ennai and Bose (Ennai, Bose 2008) present a framework that is based on concepts such as context-awareness, services, and processing/analysis versus presentation.

The service-oriented framework allows mobile applications to easily interface with enterprise back ends, and be lightweight and flexible through built-in context-awareness and a Web 2.0 front-end. "Mobile device user interfaces are not good at supporting exploratory data search or multiple application access from different windows. Mobile users need just-in-time access to relevant applications, services and data from a unified user interface." (Ennai, Bose 2008)

This framework shows the importance of simplicity within the application, context-awareness, as well as the possibility to have access to various applications and services in a SOA architecture. The application itself could behave as a context-aware portal, giving the user the information and services s/he wants.

From a spatial point of view, access to OGC services like WMS/WFS, and OpenLayers (JavaScript maps) must not be overlooked in the SMAD framework. The OpenGis® standards, of Open Geospatial Consortium, supporting interoperable solutions that "geo-enable" the web, is a development boosting web-based geodata dissemination. An important ground for their standards is open specification access: An "open" environment must include free, public, and open access to all interface specifications. (McKee, OGC Staff 2005)

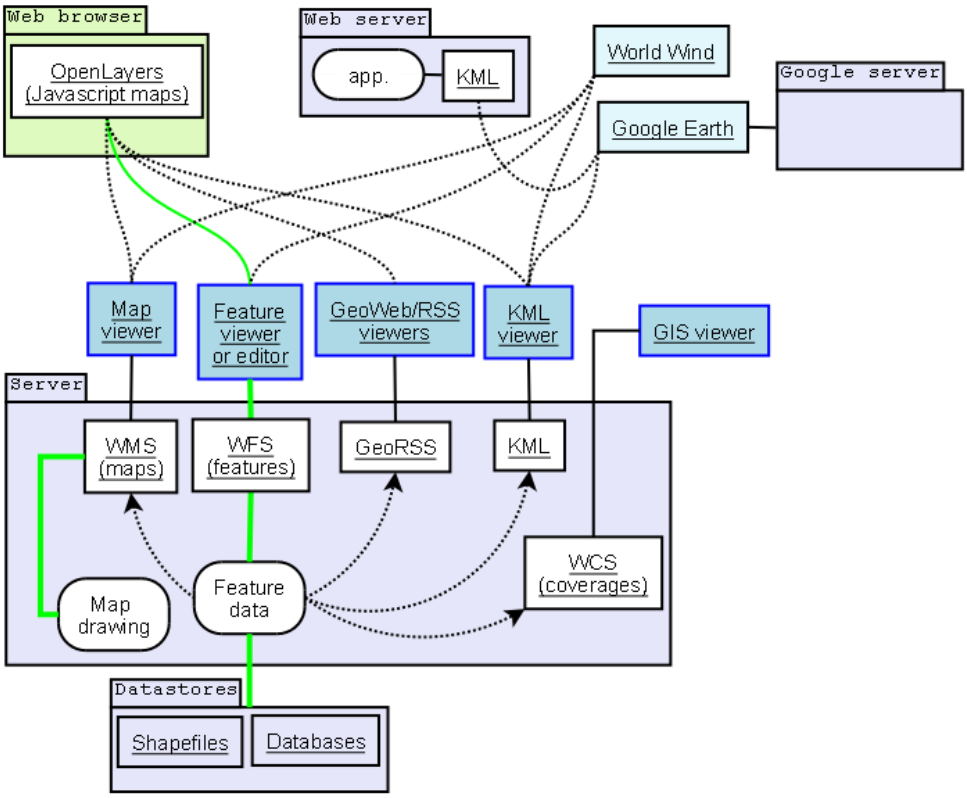

Figure 2.6 Relationship between clients/servers and some OGC protocols
(Wikipedia 2012b)

The OGC web services are depicted in their context in Figure 2.7. Figure 2.6 shows a number of OGC protocols, and their client/server relationships. Among them are:

- WMS: Web Map Service, providing map images
- WFS: Web Feature Service, for retrieving or altering feature descriptions
- WCS: Web Coverage Service, providing coverage objects from a specified region
- KML: Keyhole Markup Language, an XML-based language for geographic annotation and visualization

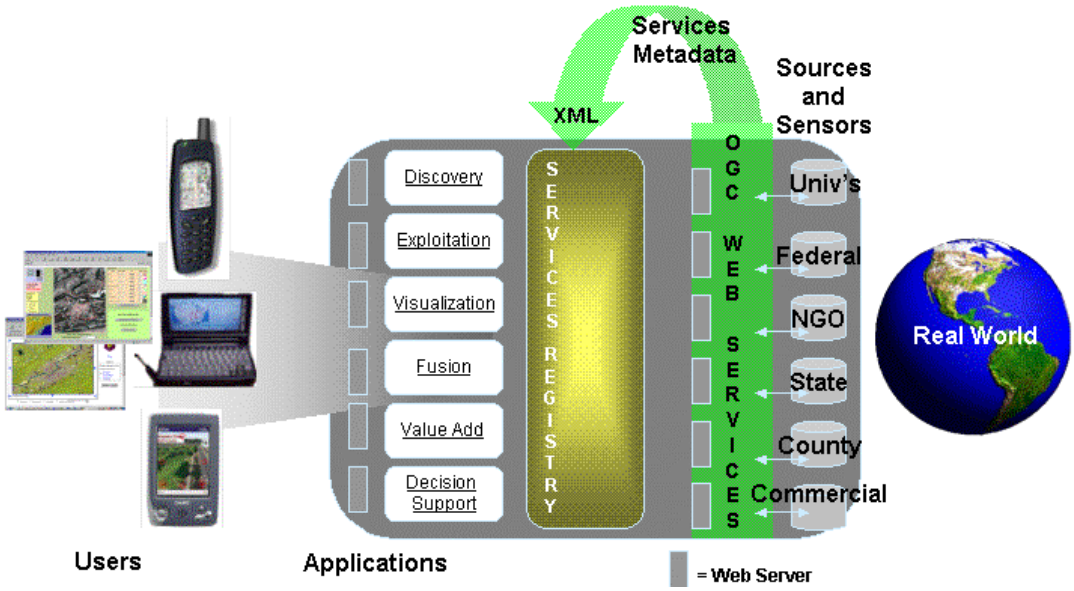

Figure 2.7 OGC Web services in its context
(Doyle, Reed 2001)

2.3.3 A conceptual framework for mobile cartography

This conceptual framework for mobile cartography (Reichenbacher 2001) is presented here, and specifically aims at spatial mobile functionality. It provides insight in presentation of information in maps and related concepts. It is a useful mindset that helped in defining the SMAD element categorization of functions. In mobile cartography, user, context, and tasks are the main elements used to adapt visualization, see the dark arrows in Figure 2.8. "The context in which a user is performing tasks has a strong connection to the prevailing information needs, and controls the content, as well as the visualization mode of the information."

Konečný et al. appoint portrayal environment awareness, and portrayal homogeneity, besides task orientation and user orientation as the most important aspects of adaptive cartography (See also Figure 2.9). Portrayal environment awareness focuses on technological fit, and portrayal homogeneity on a uniform representation of heterogeneous data to allow quick browsing for geoinformation and easy visual comparison of spatial data.

In the perspective of this thesis research, the mobile cartography framework gave input for the categorization of mobile functions. From a cartography point of view, the context is, like in the former discussed mobile framework, an important element.

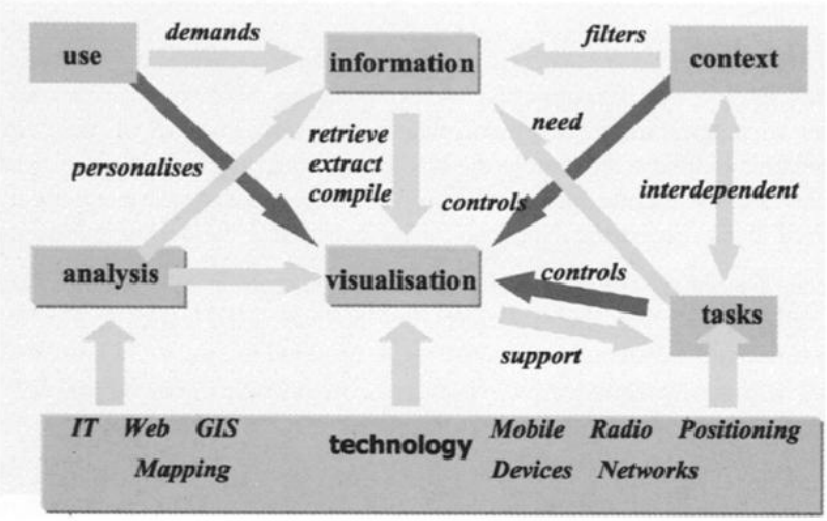

Figure 2.8 Conceptual framework of mobile cartography
(Reichenbacher 2001)

A visualization is successful when it suits the user context, and the tasks that are performed. Also, the various types of mobile device are part of this experience. Visualization should meet the smartphone user's demands in both functional and technical sense. A relatively small smartphone screen must offer a good user experience in presenting a map, and presenting other information and functions. Besides smartphones, other devices, including pc's, are considered to be supported as well. Developing an application only once, suitable for various devices, is what is strived for.

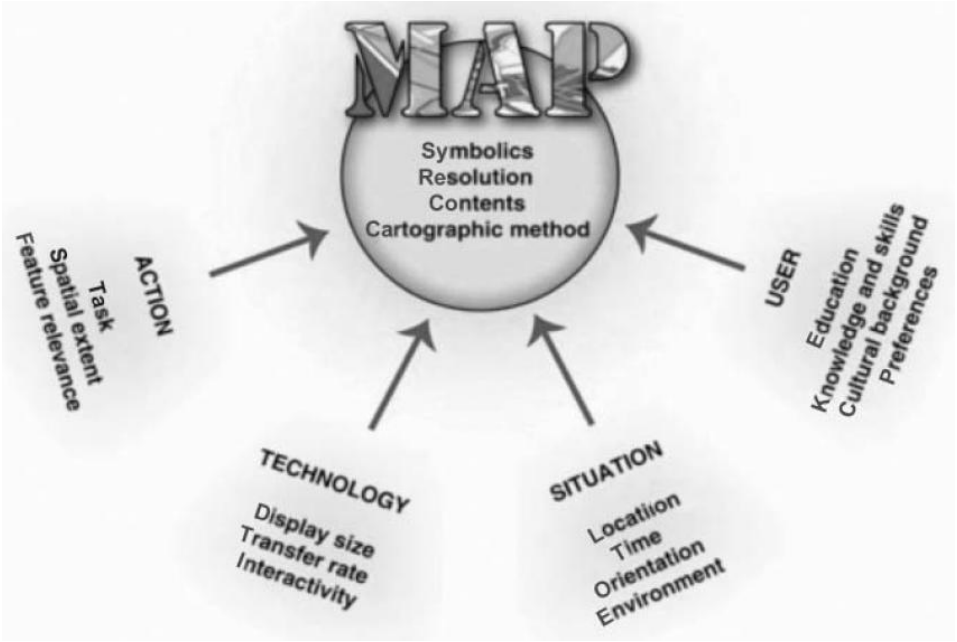

Figure 2.9 The context of adaptive cartography
((Konečný, Staněk et al. 2007) , (Konečný, Staněk 2010))

2.3.4 A conceptual framework of context-aware applications

Context-awareness is often a fundamental element in mobile applications, as it provides information on the user, the environment, activities, and access to entities. Many geo-applications present information on the direct surroundings of a user in a map or textual format. For example, the navigational apps, showing directions, and the social network apps, showing people, or places of interest in the direct surroundings. In their study on context-aware computing, Dey et al (Dey, Abowd et al. 2001), define context as any information that characterizes a situation related to the interaction between users, applications, and the surrounding environment. People (who), places (where), and things (what) are entities most relevant to context-awareness, as well as four characteristics of contextual information: identity, location, status or activity, and time.

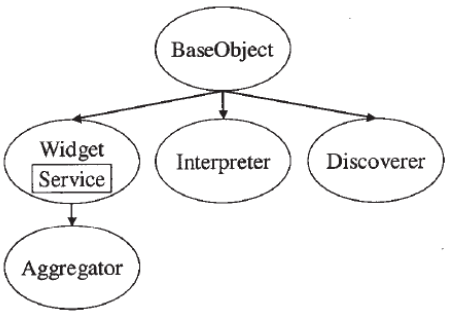

Figure 2.10 Object diagram of context toolkit abstractions
(Dey, Abowd et al. 2001)

The framework that Dey et al. present, is a conceptual framework on context-awareness of applications. They introduce four additional categories of components or abstractions, besides context widgets, to address context-specific operations: interpreters, aggregators, services, and discoverers. (Dey, Abowd et al. 2001) Context widgets are the basic sensors that provide information. Examples are a temperature widget, and presence widgets, which determine a person's presence in a location. The reusable interpreters are helpful in combining information into a higher level of context information. Aggregators collect related context information, and make it easily accessible on an aggregated manner. Services can be used to execute certain actions on behalf of the application. Discoverers are library functions that register what functions are available for the applications. Figure 2.11 shows an example configuration of these context toolkit components. Figure 2.10 shows the abstractions as subclasses from the BaseObject. This BaseObject encapsulates distributed communications via HTTP and XML, which is inherited by the subclasses.

Proper use of context information is not always simple and straightforward. Collection of context data, of many fields of interest, has become much easier, with new technologies, like the GPS sensors and internet connectivity of a smartphone. Proper synthesis and interpretation of the data can be quite a challenge still. The involvement of context-awareness in a mobile application can be an essential part of the application, at least it is an apparent added value of mobile technology. In line with the complexity and richness of an enterprise application, context-awareness makes a solution less or more complex. An application that is automatically, or manually, aware of its context, contributes to the user experience, and thus to the success of this application.

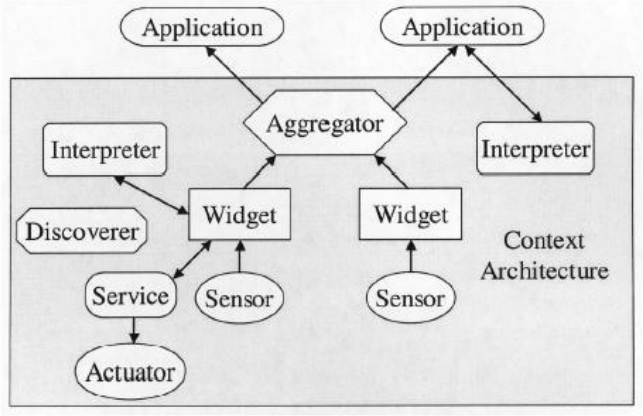

Figure 2.11 Example configuration of Context Toolkit components
(Dey, Abowd et al. 2001)

Interesting for this study is the question what mobile devices have to offer in different sensor capabilities, to automatically gather environmental data. What kind of context information is relevant to an application and its functions offered, and how can this context information be made available to the application? The use of services and context widgets helps to make the app context-aware, depending also on available sensors. Looking deeper into the subject of mobile application development teaches us that HTML5 cannot be used to enter all features, and thus sensors, on different devices, and for example plug-ins in JavaScript may be used to enable this for the various platforms separately. Certain context information can be manually made available, and other functions, such as the inclusion of pictures from the device's camera, can be scripted using a work-around. It is questionable, however, to what extent the user experience is negatively affected by such work around.

Like the second framework discussed, this framework sets out an approach to simplify the applications itself, and separate certain functions and services, to be available for all applications. In the context-aware framework, the applications can subscribe to the context functions by the discoverer.

The proposed approach in this thesis aims to reuse elements if available, and develop elements as generically as possible. Both context functions and spatial functions, must be available or accessible, as reusable objects. Metadata, a good description of the logics used, within these objects is relevant. Functions, based on different logic, will return different outputs, and one needs to know which calculations and definitions are at the basis of certain conclusions. A simple example is distance, which can be calculated in many different ways, and which can also be presented in various manners: distance as the crow flies versus distance per road/railway; and distance in minutes/hours, versus distance in kilometers.

3 Building blocks of the SMAD framework

3.1 Introduction

The three building blocks of the SMAD framework are:

- Mobile apps
- HTML5
- Spatial functionality

The frameworks discussed in Chapter 2 already introduced subjects related to mobile apps, and spatial functionality. The MVC principle was described, ideas on mobile cartography, and the usage of services, and context-awareness in a mobile environment. This chapter presents additional concepts of and backgrounds to the three building blocks, that together lead to the SMAD framework as presented in Chapter 4.

Section 3.2 starts with a general discussion of the variety of mobile platforms, current market shares, and introduces some important user interface toolkits, including the HTML5 option. Some best practices, by developers and specialists, on mobile app development are described; these are adopted in the SMAD framework. The HTML5 building block is described in Section 3.3, together with its relation to JavaScript and CSS3. We also remark that HTML5 is still a standard under development, and that web browsers support HTML5 features to a varying degree. The spatial functionality building block is introduced on the basis of aspects of spatial data and GIS functions in Section 3.4. This section explains GIS functionality, and in particular, GIS functions that are of interest in the mobile environment.

3.2 Mobile apps, platforms, devices, and best practices

Section 3.2.1 starts with a categorization of mobile applications for mobile functionality in the SMAD framework. Section 3.2.2 gives insight in the current mobile platform market, and the developments in market share, which shows an extreme growth and market share of the open source Android platform. Many mobile platforms exist, and new platforms are to be expected. HTML5 as development platform is introduced to support multi-platform development. Interoperable app development is further affected by the wide variety of mobile devices running on the diverse platforms, each with their own specifications, and more or less unique features (Section 3.2.3). This study investigates to what extent an interoperable approach to mobile app development is presently available using HTML5, and what hurdles are still to be taken. Therefore, the section ends with a discussion of best practices in mobile app development in the context of HTML5 and interoperability (Section 3.2.4)..

3.2.1 Categorization of mobile applications

Mobile applications exist in many forms, and with many different purposes. The Android apps in the Google Play Store (play.google.com/store), for example, are divided in two main groups, games and apps. And the apps are grouped and tagged functionally into categories such as financial, photography, lifestyle, and backgrounds. For this study, the focus of the SMAD framework, is on applications in an organizational setting. A categorization of mobile applications is defined, based on two categorization models: the taxonomy of mobile enterprise applications (Unhelkar, Murugesan 2010) and the map usage cube (MacEachren, Taylor 1994).

The taxonomy of mobile enterprise applications (Unhelkar, Murugesan 2010) is based on richness and complexity of mobile applications. Figure 3.1 shows this categorization of mobile applications, from simple large-scale information broadcasting towards rich and complex collaboration. M-broadcast stands for the facilitation of large-scale information broadcasts to mobile gadgets, such as advertisements and promotions. M-collaboration aims to support collaboration within and outside the enterprise.

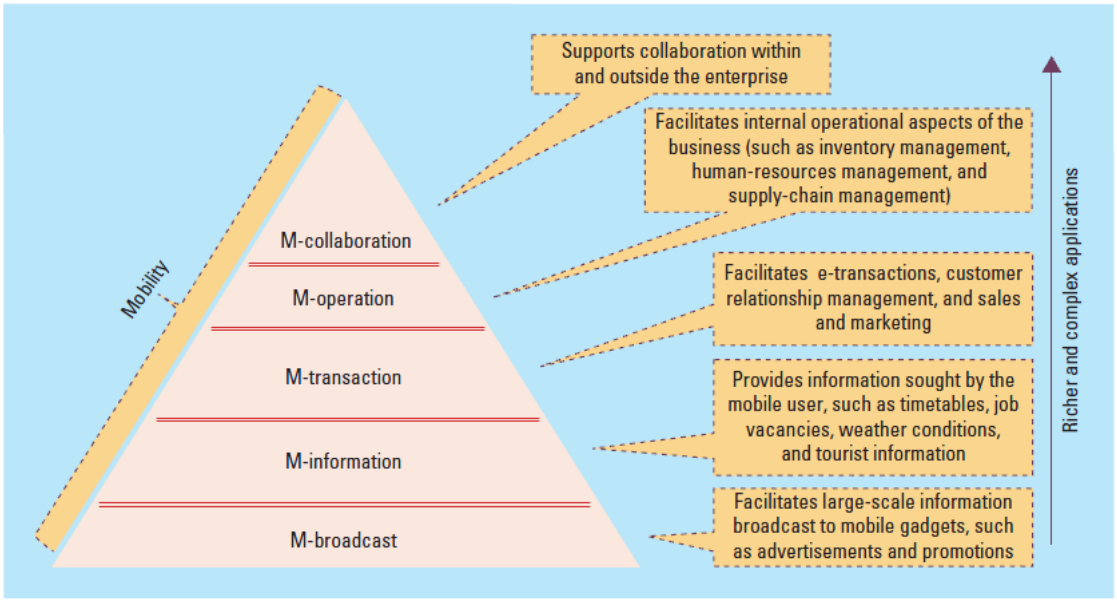

Figure 3.1A taxonomy of enterprise mobile applications
(Unhelkar, Murugesan 2010)

Based on the taxonomy of Unhelkar and Murugesan, the categorization is brought back to three levels in richness and complexity of enterprise mobile apps; Information, Operation, Collaboration. These three groups relate to the question *"what kinds of process are facilitated"*:

1. Information: Present information of large-scale information broadcasts, and information sought for.
2. Operation: Facilitate operations and transactions of the core processes, as well as supporting processes.
3. Collaboration: Support collaboration within and outside the enterprise or community.

When developing a new application, it is wise to start with basic functionality that can be categorized as less complex and less rich. One implements this solution, and continues with more complex functionality in a next release. However, the development approaches for simple and for complex applications, do not have to differ from each other. From an interview with Matt Powers of Applico (www.applicoinc.com), a mobile app development company, we learn, that they treat each client exactly the same, regardless of complexity and size of a project. "For every project, unless dictated otherwise by our client, we create wireframes, mock-ups, and detailed requirements before even starting to code. Our internal project plans are all created with the same internal goals in mind. Additionally, each project is put through the same rigorous quality assurance tests regardless of complexity or size." (Huntley 2011) This underlines the need for a standard development approach, which still can consider different paths for different kind of functionality.

Besides complexity and richness of information, the categorization of applications is based on the map usage cube (MacEachren, Taylor 1994). GIS map usage (see Figure 3.2) can be categorized on the basis of three characteristics; interaction, audience, and data relations, into four groups (1) presenting, (2) synthesizing, (3) analyzing, and (4) exploration.

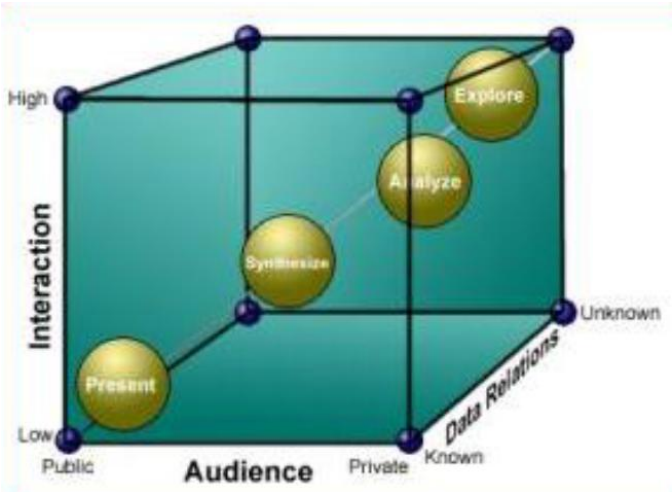

Figure 3.2 Map usage cube
(MacEachren, Taylor 1994)

Presenting information in a map is a process in which data relations are known, interaction is low, and information is public. This process is thus a known and straightforward process. Exploration on the other side of the spectrum is a more complex process, because interaction level is high, data relations are unknown, and information must be kept private.
The concept of the map usage cube is used in the categorization of mobile applications to relate to the question:*" how are data and information used"*. The dimensions 'Present', 'Synthesize', and 'Analyze', are adopted from the map usage cube:

1. Present focuses on single source information,
2. Synthesize on having multiple sources of data, and
3. Analyze is aimed to support analysis of the data.

Exploration is not taken into account, and is left out of scope in the SMAD framework. One reason is the level of complexity, combined with the idea that the screen size of a modern smartphone is seen as a blocking factor to successfully support mobile exploration.

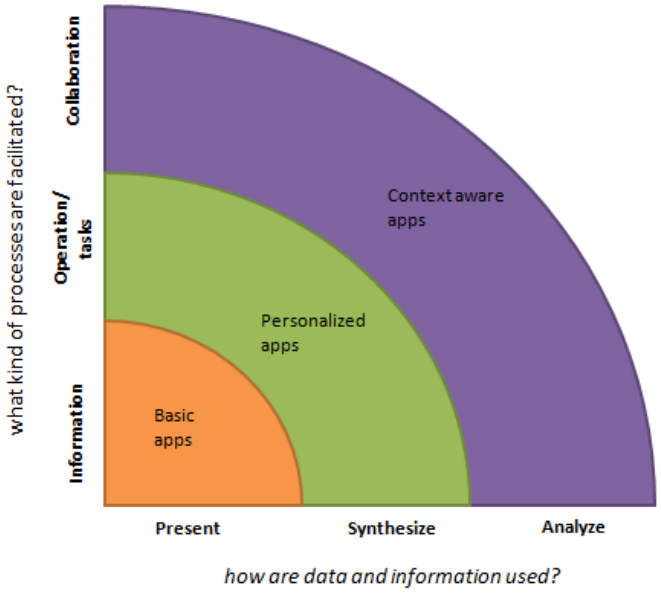

Figure 3.3 Three levels of mobile applications

Combining the models of complexity, and richness, with the map usage cube, result in three levels of mobile applications:

1. Basic applications (Present, Information)
   Single source data that is presented to a broad audience.
2. Personalized applications (Synthesize, Operational/tasks)
   Multi-source data, including web services, that are used to support single-user operational tasks.
3. Context-aware applications (Analyze, Collaborate)
   Data can be selected and combined in analysis, and can be used to support collaboration between two or more users.

3.2.2 Mobile platforms

Mobile apps, complex and less complex, are available for the different mobile platforms, and devices running these platforms. Each one of these has its own development kits, and programming language used. This section gives an overview of the different platforms, and presents HTML5, in combination with JavaScript and CSS, as a way to develop multi-platform, thus platform-independent, apps.

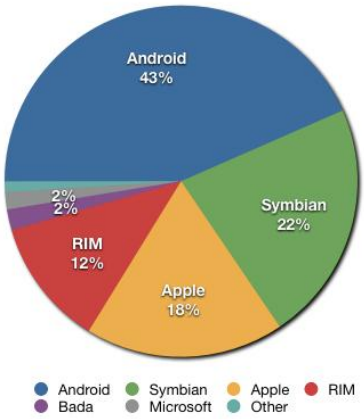

Figure 3.4 Share of worldwide 2011 Q2 smartphone sales by operating system
(Gartner 2011)

Devices running Android outnumber other platforms in smartphone market share, having 43% of total market share Q2 2011, and even 53% Q32011 (See Figure 3.5, Figure 3.4). Followed by Symbian with 22% resp. 17%, and Apple's IOs with 18% resp 15%. (Gartner 2011, Wikipedia 2012a) Number of devices running Android is growing, as share Symbian and IOs is declining. The share of Microsoft, with the Windows 7 platform, will probably increase, as Nokia is expect to sell quite some new smartphones running this platform.

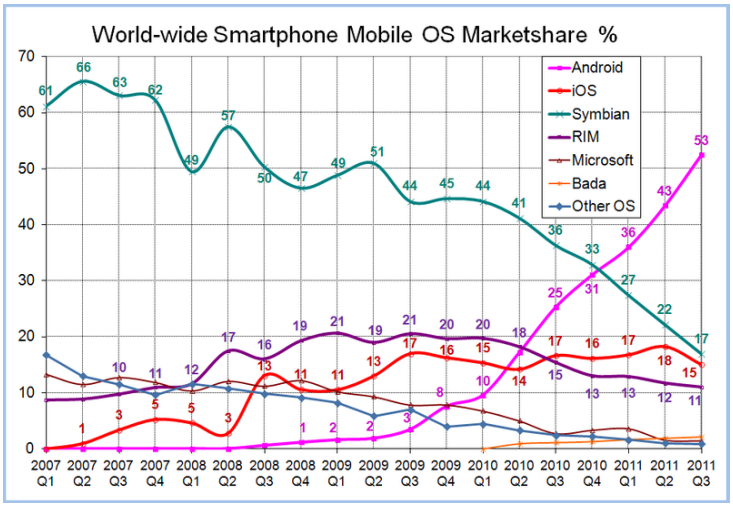

Figure 3.5 World wide Smartphone Mobile OS Marketshare %
(Gartner 2011, Wikipedia 2012a)

Dorokhova, and Amelichev (Dorokhova, Amelichev 2010) attempt to compare eight modern mobile platforms from the developer standpoint: Symbian, MeeGo, iPhone, Android, Windows Phone 7, Palm, Blackberry, and Bada. Table 3.1 shows a comparison of most popular user toolkits and their features, all supporting Open GL ES. According to the researchers Maemo/MeeGo, Symbian and Android are the most open platforms, and the most cross-platform solutions are the Qt toolkit and the HTML-based toolkits. Most web browsers are based on the WebKit engine, making cross-mobile-platform applications much simpler if developers stick to standard HTML, CSS and JavaScript.

Table 3.1 most popular user toolkits and their features
(Dorokhova, Amelichev 2010)

MOST POPULAR MOBILE INTERFACE TOOLKITS AND THEIR FEATURES

UI Toolkit	Markup	Resizing	Events	Device Features	Multi touch	3D	Video	Per-sisten-ce	Plat-forms
Qt	Declarative (XML) for static code generation; imperative (C++) for static & dynamic interface	Auto, if QMainWindow class is used for the app & QLayouts are used	signals & slots	GPS, accelero-meter, battery, memory, processor info, light sensor, compass	since 4.6	Open-GL ES 2.0	Platform-defined	On-disk file	Symbian, Maemo/ MeeGo
Silverlight	Declarative (XAML) and imperative for static & dynamic interface	Automatic, if layout other than Canvas is used	delegate (ordinary) events and routed (bubbling) events	camera, microphone, GPS, accelero-meter, power notifications	since 3	Direct X 9	Windows Media Foundation (MPEG4, H.264, etc.)	On-disk per-app isolated storage	WP7
UIKit/ CoreGra-phics	Imperative (Objective-C)	Automatic (Springs & Struts based layouts)	UIRespond er chain (bubbling)	camera, microphone, GPS, accelero-meter, power, compass	since 1	Open-GL ES 1.1, 2.0 since iPhone 3GS	MPEG4, H.264	On-disk per-app isolated storage	iPhone
Android UI	Imperative (Java)	Automatic (if not using SurfaceView for pixel-precise drawing)	event listeners & event handlers	camera, microphone, GPS, accelerome-ter, compass	since 1	Open-GL ES 2.0	H.263, MPEG4 SP, H.264 AVC	On-disk per-app isolated storage	Android
HTML5	Declarative (HTML), imperative (JavaScript UI toolkits)	Automatic (if relative CSS size points are used and no fixed-size blocks are in the page)	bubbling DOM Events	No access	iOS, Andro-id, MeeGo	Canvas and webGL (Open-GL ES in Java-Script)	HTML5 <video> (H.264 AVC, etc.)	DOM Storage (on-disk secure cache)	iPhone, Android, Palm, Maemo/ MeeGo, WP7 (limited)

3.2.3 Mobile devices

Mobile apps can be bought, or made freely available, in the app store of the specific smartphone platforms, for example Apple's App Store that offers iPhone apps, and Android Market, just recently renamed to Google Play, offering various Android apps. Although mobile applications are spreading rapidly, there are still some important limitations of the mobile devices a developer should be aware of when creating new apps.

Saunders ((Saunders, Greyling et al. 2010b), (Saunders, Greyling et al. 2010a)) distinguishes between hardware issues, stemming from a need for portability, size and weight reduction, and software issues. Hardware issues include: screen size, insufficient memory, lack of processing power, low battery life, poor storage capacity, unreliable connections, limited communication bandwidth, and high cost for bandwidth. Software issues concern navigation and browsing difficulties mainly caused by the smaller screens, visual issues like restrictions to images and icons, and content lay out, that may differ between the various size devices.

In Appendix A.1, specifications of three present day devices are presented in an overview; the popular iPhone 4s (iOS), and Samsung Galaxy SII (Android), and the new Nokia Lumnia 900 (Windows 7). These device specifications give the reader some idea of the current screen resolutions, memory, and processing power of the leading smartphones, which will be soon outdated.

3.2.4 Best practices in mobile application development

This section presents best practices in mobile app development in the context of HTML5 and interoperability.

Rhodes 3.0 is a mobile development platform from Rhomobile,[6] to write native apps once and build for all smartphones. Adam Blum, Founder and CEO of Rhomobile, comes up with the following best practices of enterprise smartphone apps, which are also largely adapted in this research (Blum 2011):

- leverage device capabilities
- be task-oriented, and context-sensitive
- avoid typing, and support all devices
- have synchronized local data, offline usage, and handle metadata

The practices ‘be task oriented’, and ‘context sensitive’,refer to context-awareness, which concepts were discussed in Chapter 2. To include ‘synchronized local data’, ‘offline usage’, and ‘handle metadata’, are practices that are part of the SMAD framework quadrant ‘categorization of functions’, discussed in Chapter 4. Offline usage and storage of data are also described using HTML5 functions in Section 3.3.
According to Blum, taking full advantage of a device’s capabilities, is most important for a successful app: Such as sight (the camera), hearing (microphone and voice phone connections), touch (as in touch screen), and a sense of location in space (GPS), and even better to be able to integrate with other capabilities as well. The HTML5 toolkits have problems accessing these device features (see also Table 3.1). Leveraging device capabilities is thus a problematic issue in developing cross platform mobile applications with HTML5. In the context of this research it is good to know that HTML5 includes a standard function, GeoLocation, to access a user’s physical location. To what extent other functions, or workarounds, are presently available, to leverage other device capabilities cross platform, was investigated during the ‘prototyping phase’, see also Chapter 6.

The remainder of this section is dedicated to the principles of responsive design, which focus is on ‘supporting all devices’. Confronted with the many different platform, and even many more different mobile devices, the development of mobile applications and websites asks for flexible designs, that are adaptive to the various media that renders them. Standard-based technologies exist that makes this flexible and more adaptive design approach, also called responsive web design, possible. (Marcotte 2010)

[6] www.rhomobile.com

Figure 3.6 www.audiovroom.com, web app loads in any HTML5-capable device
(Lara 2011)

Lara (Lara 2011) describes three best practices, essential in responsive design, to look attractive on any device:

1) Know your minimal viable product (MVP)
2) Make the MVC (Model View Controller) principle an integral part of your HTML5 application from day one
3) Design to the 'grid', and make good use of CSS3 media queries

The minimal viable product aims to simplify the application to an absolute minimum functionality required to maintain an acceptable user experience. The application should run smoothly and be user-friendly on a device with smaller screen, and less processing power, as well as for other devices that have bigger screens, and more processing power. The Model-View-Controller design pattern was discussed in Section 2.3.1. Design to the grid is a technique to be able to render the application layout-based on the device's screen sizes, see also Figure 3.6, which shows the application Audiovroom on differently sized devices, based on a flexible grid.

3.3 HTML5, JavaScript, and CSS3

HTML5 is chosen as the second building block of this study's framework. HTML5, combined with JavaScript, and CSS3, is presented, as the way to develop platform-independent mobile apps. Many web sites and mobile apps are already developed using HMTL5, even though the language is evolving still. This section briefly introduces HTML5, JavaScript, and CSS3 (Section 3.3.1). The availability of various HTML5 development frameworks is described in Section 3.3.2, as background to the chosen frameworks in the SMAD framework. Finally, three functions that were introduced in HTML5, and that are of importance in the scope of this research, are shortly introduced (Section 3.3.3): the geolocation API, the canvas element, and local and offline storage.

3.3.1 HTML5, an introduction

HTML5 is the most recent version of HTML, superseding HTML 4.01, XHTML 1.0, and XMHTML 1.1 (Pilgrim 2010) It is supported by browsers on all modern mobile platforms, with varying degree of compliance to upcoming HTML5, EcmaScript 5, CSS3 and DOM Core L3 standards. (Dorokhova, Amelichev 2010) With good coding practices, such as extensive use of JavaScript frameworks for the application interface and behavior, comes the ease of running the application across all platforms without a single modification. Mobile application development using HTML5 in combination with JavaScript, and CSS3 (P2PU 2012):

- JavaScript is the instruction code
- HTML provides the structure of a web document
- CSS (Cascading Style Sheets) are used to control the display and style of the page

The World Wide Web Consortium (W3C[7]) has a working document on HTML5 that describes current functionality. (Hickson 2011) Clark (Clark 2010) warns that HTML5 developers have to pay attention, as HTML5 is still an evolving standard. Rules and protocols are still evolving, and standard codecs need to be decided upon. It is not expected that the app store model will go away anytime soon to be displaced by the alternative HTML5, JavaScript, CSS3 development platform. HTML5 still needs work on its security limitations. None of the web browsers supports HTML5 to the full extent, and some older browsers do not support HTML5 at all. It is good to note that mobile web browsers preinstalled on iPhones, iPads, and Android phones all have excellent support for HTML5. (Pilgrim 2010) The support of different web browsers for HTML5, and CSS functionality varies a lot, and can be checked at various web sites[8].

[7] www.w3.org
[8] www.html5test.com, and www.caniuse.com

3.3.2 HTML5, JavaScript, and CSS3 development frameworks

Mobile app development with HTML5, CSS3 and JavaScript, possibly in combination with other techniques as JQuery, make the development environment suitable for multi-platform development, and less intensive coding of the backend.

Two important architectures of mobile apps can be distinguished: web-based mobile applications, and native mobile apps, in which the application is specifically coded for the platform on which the mobile device runs. The native variant, in its essence, means coding, and thus maintaining, the application for different platforms. The web-based mobile applications offer functionality via the web browser that can offer most of the functionality via the various types of mobile browsers.

Web-based mobile applications have, however, their shortcomings (Gilmore 2011):

- Several key APIs cannot be accessed, such as the current network state, file system, notifications, the address book, and camera.
- Decreased visibility due to lack of presence on the device's touch screen menu
- Lack of distribution visibility because web applications cannot be distributed through hubs such as Apple's App Store.

Distribution to the app stores can also be seen as a burden, as it takes extra time to get the application and its updates approved for distribution via the app stores. Gilmore presents the Phonegap framework to overcome the listed shortcomings. Phonegap is a platform that offers translation from your HTML5, CSS3, JavaScript application into a native application by wrapping the code.

Various mobile development platforms offer the potentials of HTML5, CSS3, and JavaScript. This study started with the 3 platforms, DHTMLX Touch, the M Project and the 52 Framework, discussed on the site HTML Goodies (Clark 2011), and the Phonegap platform. This list was extended by frameworks as Sencha Touch, Iui, JQTouch, JQuery Mobile, KendoUIMobile, RhoMobile, Skeleton, and Bootstrap 2 that all offer mobile application development from the user interface perspective. In the end, the list was also extended with Javascript frameworks, for which the focus is on developing the back end application logic: Ember.JS, BackboneJS, and KnockoutJS. Appendix A, Section A.2, briefly lists advantages, and disadvantages of the various platforms, based on the following criteria:

- HTML5-based
- Multi-platform, design once, roll out to various devices
- Collect, process, store, and present spatial data and information
- Available in offline mode
- Access to device features, GPS, camera, network, file, compass, accelerometer
- Well documented
- Popularity platform
- Licence cost free for future commercial development
- Multi user/developers.

The development frameworks selected, as part of the technical architecture of the SMAD framework, are described in Chapter 4.

3.3.3 Geolocation API, Canvas elements, and local and offline storage

The three HTML5 functions in this section are used to geo-enable the application, and enable offline usage. Location information can be captured using the Geolocation API. The canvas element is introduced, as a way to present images, and thus an ability to present web maps. The local and offline storage capabilities of HTML5 are presented, as a way to enable offline usage.

Geolocation API

A mobile application that can handle spatial information, should be able to handle location information such as longitude, and latitude. A user may manually enter location information, however a strong function that is available in HTML5, is to track a user's physical location by the Geolocation API. "Geolocation: Part of the location API maintained by W3C, it does what you would expect—allows your browser, and therefore your web app, to track a user's physical location." (Clark 2010) The API does not know the underlying information sources of the location information, which can be an IP address, a GPS waypoint, or a GSM cell ID, among others. The API is capable of both single position requests, and repeated position updates, as well as to query cached requests. (Popescu 2010)

Canvas element

In presenting spatial information on maps, the Canvas element can be used. The Canvas element is a two-dimensional grid. Using the <canvas> tag to draw elements into the browser, developers can create graphics, images, games, and visualizations, using simple programming and markup. (Clark 2010)
Boulos et al. (Boulos, Warren et al. 2010) propagate the use of Cartagen. Cartagen, written in JavaScript and using the canvas element, is a vector-based, client-side framework for rendering maps in native HTML5. It draws maps dynamically on the client side, which offers functions to move, adapt and redraw maps, and which can include as many layers of data/levels of detail as needed.

Local and offline storage

The new local and offline storage functionality enables full, persistent database-like storage within the browser. You can take your apps offline and still make edits and changes to your data. (Clark 2010) The locally stored data can be retained through offline sessions, which in turn grants developers the ability to create offline web applications and provide users with a seamless experience that is unhindered by low quality or nonexistent network connections and the like. (Harjono, Ng et al. 2010) Offline storage can be included in the application by adding a manifest file, which lists the files that are needed for the web application to work offline that are stored by the web browser. (Hickson 2011)

3.4 Spatial functionality

Spatial functionality will be introduced based on functions that are present in geographical information systems (GIS systems) in Section 3.4.1. In a mobile situation, other geo-information tasks and needs exist, compared to a traditional stationary environment. Section 3.4.2, therefore, focuses on geo-information in mobile situations. Geo-information functionality is looked at, at three levels: presentation, application, and data. These three levels were deducted from the mobile application development framework of Unhelkar and Murugesan (Unhelkar, Murugesan 2010), described in Section 2.3.1.

3.4.1 Traditional GIS and GIS functionality

The SMAD framework supports the development of geo-enabled mobile applications. It is not a goal to develop a mobile GIS. However, understanding traditional GIS and its usage, is a help in defining what spatial functionality is useful, and adds value, to the users in a mobile environment.

First a definition of GIS by Kraak and Ormeling (Kraak, Ormeling 2003), who combine a technical definition of GIS with an institutional/organizational perspective. A technical definition of GIS by Burrough (Burrough, McDonnell 1998), defines GIS as a powerful set of tools for collecting, storing, retrieving at will, transforming and displaying spatial data from the real world. Combined with Cowen's (Cowen 1988) institutional definition: a decision support system involving the interaction of geospatially referenced data in a problem solving environment. The combination of the two: A GIS is a computer-assisted information system to collect, store, manipulate and display spatial data within the context of an organization, with the purpose of functioning as a decision support system.

Above all, GIS are systems that are able to present information in maps. A GIS offers functionality to combine geospatial and non-geospatial data from different data sources in a geospatial analysis operation in order to answer all kinds of questions (Kraak, Ormeling 2003):

- Identification; What is there…?
- Location; Where is….?
- Trends; What has changed since….?
- Optimal path; What is the best route between…?
- Patterns; What relation exists between….?
- Models; What if….?

All measurements from a GIS will be an approximation, since vector data are made up of straight line segments, and all raster entities are approximated using a grid cell representation. (Heywood, Cornelius et al. 2006) Common GIS functions include:

- Overlay
- Neighborhood functions
- Spatial interpolation
- 3D analysis
- Visibility analysis
- Network analysis

At present there are two main ways in which computers can handle and display spatial entities: raster and vector. The raster spatial data model is one of a family of spatial data models described as tessellations. In the raster world the basic building block is the individual grid cell, and the shape and character of an entity is created by the grouping of cells. A vector spatial data model uses two-dimensional Cartesian (x,y) coordinates to store the shape of a spatial entity. In the vector world the point is the basic building block from which all spatial entities are constructed. (Heywood, Cornelius et al. 2006) Most GIS systems started as file-based systems, as spatial databases were used to be rare. Presently, various database systems are present that support spatial data, and functions, and GIS systems store data in a database.

In traditional GIS systems the most comon way to construct the computer models is by using layers. Each layer is thematic and reflects either a particular use or a characteristic of the landscape. Another approach to structuring geographical space is the object oriented approach, related to object-oriented (OO) programming (Cowen 1988). Features are not divided into separate layers but grouped into classes and hierarchies of objects. These objects have attributes, and behaviors defined, as well as relations to other objects.

3.4.2 Geo-information in mobile situations

In a stationary environment other geo-information tasks and needs exist, compared to the mobile situations. Table 3.2 shows four groups of user tasks, that can be distinguished in a mobile environment: locators, proximity, navigation, and events. (Reichenbacher 2001)

Table 3.2 Tasks in a mobile environment
(Reichenbacher 2001)

Tasks	Subtasks	Examples
Locators	Own position	xy coordinates, place name
	Objects	Attributes of an object
	Other persons position	Who is this?
Proximity	Objects	Next object with certain attributes
	Persons	Known people in the area?
Navigation	Routing	Way descriptions
Events	What happens at a place	Obstacles (e.g. traffic jam)?

The geo-information tasks and needs, from a user and technical perspective, are described by the mobile application elements: presentation, application, and data (see Table 3.3). These three elements of mobile applications are pulled out the EMAD framework, in Section 2.3.1, and relate to the MVC principle. The grouping of spatial functions in Table 3.3 is discussed in the remainder of this section.

Table 3.3 Spatial functions by presentation, application, and data layer

Presentation	Application	Data
Maps	Locators/events	Spatial database
OpenLayers	Proximity/navigation	Spatial reference (WGS84)
Granularity/scaling	Context widgets/discoverers	OGC web services
Context-awareness	Interpreter/aggregator	
Draw/red lining	Add, update, delete data	

Presentation

One of the most important features of GIS, is the presentation of spatial information in maps. This is also an important feature for mobile applications. In presenting maps, the user needs to be able to select different layers of geographical information, OpenLayers must be supported. Data, presented in the maps, also include spatial data from OGC web services (see also Section 2.3.2). The maps must have the ability to show legend, scale bar, and north arrow. An important feature is the ability to zoom and pan the maps, and thus change granularity of the information by scaling. The information presented in the app, should be context-aware, meaning for example, showing the map of the user's location, and dependent on the task the user is performing. Users need to be able to add spatial data by drawing and red lining, and select objects from the map and alter the information.

Application

The tasks in a mobile environment in Table 3.2 (Locators, Events, Proximity, Navigation, (Reichenbacher 2001)) are important spatial functions at the application level of the mobile app. Locators, and events are quite straight forward functions, that use geo-location as a basic input, and return non-spatial information in addition to the geo-location. Proximity, and navigation are more complex functions that analyses on geo-location, distances, and network information. For navigation purposes it is also very important that information is constantly updated.
Context widgets, discoverers, interpreters, and aggregators (Section 2.3.4, (Dey, Abowd et al. 2001)) offer automated functions to make an app context-aware. Context widgets and discovers are basic context functions, as interpreters and aggregators ask for interpretation, and logic of context, widgets, and services.
The user must be able to add, update, and delete spatial data, for example by red lining. Further, the user must also be able to add, update, and delete attribute data of spatial objects, for example claim a piece of land presented in the map.

Data

A mobile application must be able to store data. For simple applications, file-based storage can be sufficient, however, storage of the data in a database is preferred when creating more complex apps based on large data sets. This database must support spatial data, both raster and vector, and offer spatial reference, and spatial indexing. The spatial reference of spatial data must be in WGS84 which is an international standard, and for example used in GPS and Google Maps. The data layer of the mobile app must be able to process data from OGC web services.

4 Spatial mobile application development (SMAD) framework

4.1 Introduction

This chapter brings information and concepts discussed together in the SMAD framework (illustrated in Figure 4.1). The SMAD framework, as presented in this chapter, describes an approach, including the techniques and development methods, to develop geo-enabled mobile applications using HTML5.

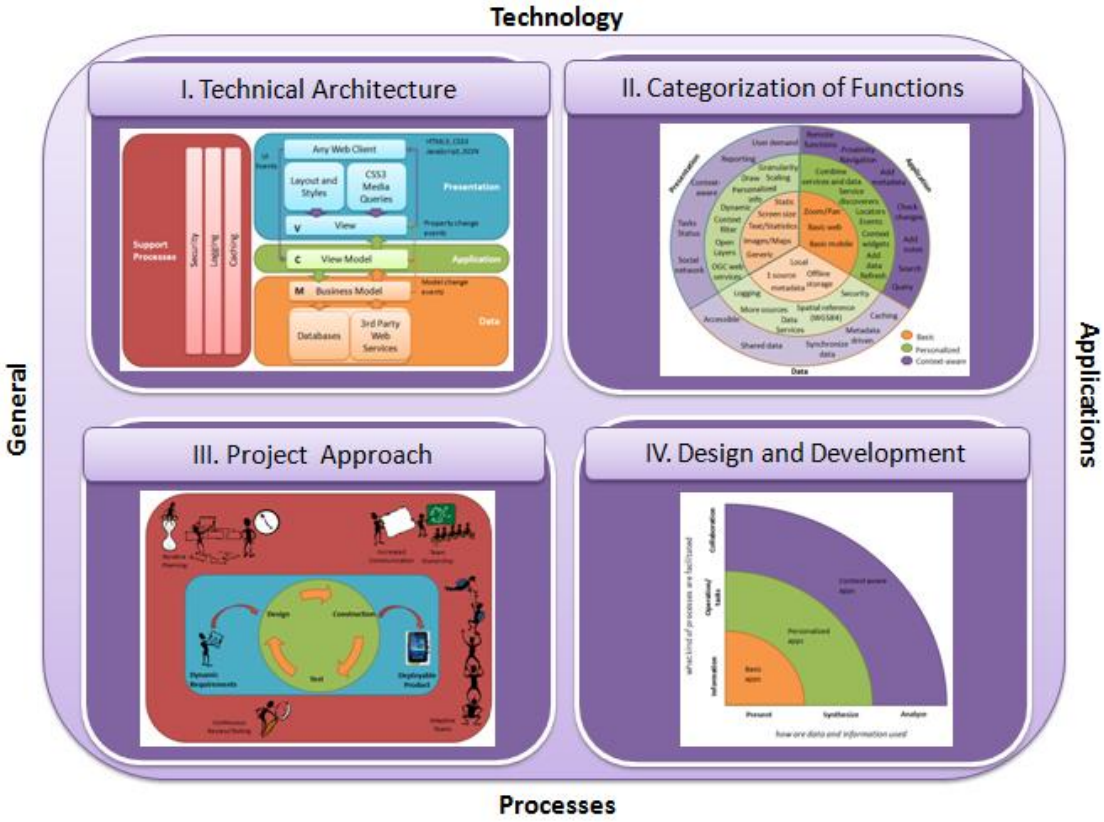

Figure 4.1 The SMAD framework illustrated

Section 4.2, Technical architecture, presents a technical architecture in-line with the MVC principle, and built up using HTML5, JavaScript, and CSS for platform-independent development. Also choices in techniques and tools are considered.

Categorization of functions is described in Section 4.3, and based on the categorization of applications, discussed in Section 3.2.1.:

1. Basic applications
2. Personalized applications
3. Context-aware applications

This categorization of functions is based on richness and complexness of mobility (Figure 3.1), the level of interaction, privacy, and know data relations (Figure 3.2), and the elements of the MVC principle (Section 2.3.1); presentation, application, and data. The categorization distinguishes between nine categories and describes the main characteristics and specific features of these categories.

The section on project approach (Section 4.4) describes a practical way of organizing a development project based on incremental development. This concept was chosen, as it is a flexible, lean, and time-based project approach that has proven itself in application development, and website development.

The design and development quadrant of the SMAD framework (4.5) is presented by practical concepts, such as use cases, and wireframes, that are used by organizations and developers in designing and building an application. The content-out approach, with the user demands and expectations as starting point, and focus on the user interface, forms the basis. The content-out approach can also be used in a supply driven development project by a single person. The concepts of UML use cases and wireframes, offer insight in the functionality to be build. These use cases, and wireframes, offer direct handles to discuss the ideas with the customer or a sparring partner, before starting the actual prototyping in HTML5, JavaScript, and CSS. The use cases and wireframes do not only lead to the user interface specification, but also translates to functioning of the controller, and design of a proper class diagram and data model.

4.2 Technical architecture

The technical architecture to develop geo-enabled mobile applications is build upon two major concepts: the MVC model, and the usage of HTML5, JavaScript, and CSS3. Based on the MVC principles, as described in Section 2.3.1, the architecture is described, in Section 4.2.1, at three levels: presentation (view), application (controller), and data (model). For each of these levels development tools are selected that supports the proposed architecture (9A.4).

4.2.1 MVVM architecture

The illustration of the technical architecture in Figure 4.2 is inspired by the MVC architecture of AgileApps (AgileApps 2012), an organization, that develops mobile applications among other services. Core of the SMAD architecture is the MVVM principle, a variant of the MVC principle, which distinguishes between the Model, View, ViewModel. Subsequently, the architecture is described by the presentation layer, the application layer, and the data layer. MVVM, optimized for stateful rich client applications, updates client-side business logic and application state through user or service interactions. (Microsoft.com 2012) General processes, like security, caching, and logging and audit, are to be considered, and implemented over the complete MVVM model.

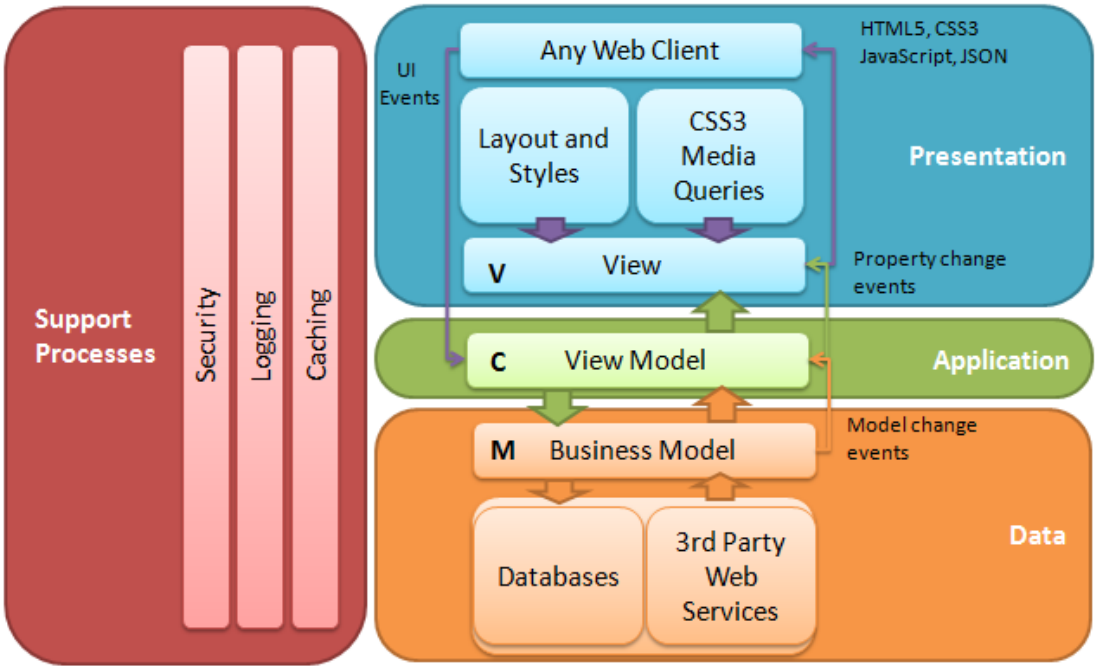

Figure 4.2 SMAD technical architecture based on MVC-principle

Presentation layer
The presentation layer exist of the user interface defined, that is build up from different views of the data. The first focus of the SMAD framework is on supporting smartphones, however other media, such as tablets and PCs can also be served via the web client. A mobile web application, based on HTML5, JavaScript, and CSS3 suits not only multi-mobile platform by the mobile web browsers, but also supports non-mobile browsers, and thus platforms. The layout and style of the user interface is separated using CSS3 style sheets, and media queries.

Application layer
The application layer, ViewModel, is the controller of the application, and contains the application logic to execute the application functionality. This layer, containing the back end functionality, is the core element of the app. It gets it tasks, via requests, from the web clients, or as changes occur in the model. Parameters are passed through to the business model, and underlying databases and web services. Changes, or responses to requests, are put through to the web client.

Data layer
The data layer consists of the business model, and the underlying databases and web services. The model is a logic representation of the data in the databases and services. The database used to store data for the application should be a spatial database to be able to store spatial data.

4.2.2 Tools

The tools that are selected to build geo-enabled platform-independent mobile applications, are:

- Presentation, or user-interface: Twitter Bootstrap[9]
  Twitter Bootstrap is a platform for simple and flexible HTML, CSS, and Javascript for popular user interface components and interactions.
- Application, or application logic: KnockOutJS[10]
  Knockout (KO) is a JavaScript library that helps you to create rich, responsive display and editor user interfaces with a clean underlying data model.
- Data, or database: CouchDB[11]
  CouchDB is a document oriented, peer based distributed database system. CouchDB hosts, servers and offline-clients, can have independent "replica copies" of the same database, where applications have full database interactivity (query, add, edit, delete).

A more detailed description of these frameworks, and the reason these were selected are in Appendix A, Section A.4.

4.3 Categorization of functions

Besides technical issues and challenges, it is important to have a good understanding which functionality needs to be developed. Functions, that vary in richness and complexity, may ask for a different approach, or at least ask for different front-end capabilities. Building an application to look up information, is much simpler than creating an application to gather, create, update, and to share information and tasks. Presenting information in a map, may ask for different manners of zooming and panning, than presenting statistical information in a table.

The categorization of functions is created based on the 3 levels of mobile applications, see also Figure 3.3 in Section 3.2.1:

1. Basic applications (Present, Information)
2. Personalized applications (Synthesize, Operation/tasks)
3. Context-aware applications (Analyze, Collaboration)

Figure 4.3 presents these three levels of applications, and the main characteristics and features in this category of applications. These characteristics and features are grouped by the three elements of mobile applications: presentation, application, and data. These labels are in line with the Model-View-Controller principles that are adapted in the SMAD framework, and are also described as the main elements from the EMAD-framework in Section 2.3.1.

[9] twitter.github.com/bootstrap/index.html
[10] knockoutjs.com
[11] www.couchdb.com

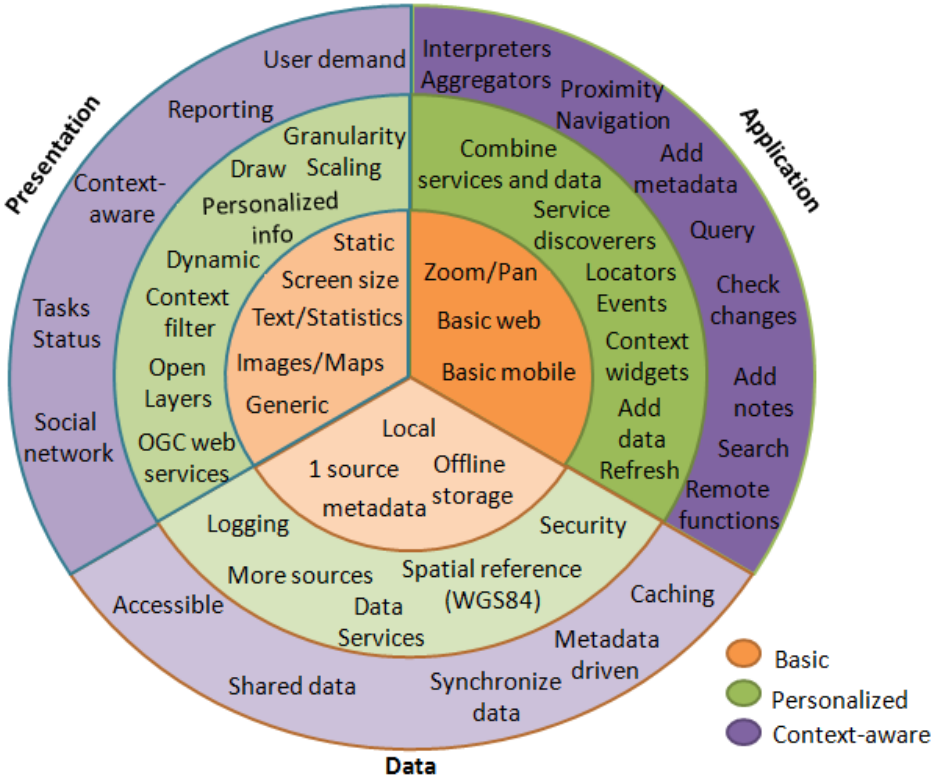

Figure 4.3 Three levels of functions by presentation, application, and data

The spatial functions, from Table 3.3 described in Section 3.4.2, as well as the more general mobile functions, are part of this categorization. These characteristics and features are partly deducted from the frameworks presented in Section 2.3. The use of local and remote services is part of the service-oriented framework. The framework on mobile cartography presented the role of user preferences, task controls, and feature controls. The framework on context-aware applications described usage of service discoverers and context widgets. The MVC-architecture of AgileApps, as presented in 4.2, endorse the need for security, caching, and logging.

Figure 4.4 is a more detailed illustration of the categorization of functions, as depicted in Figure 4.3. In the more detailed illustration, an extra categorization of functions is visible: the complexness and richness of applications with the groups information, operation/tasks, and collaboration, versus the groups present, synthesize, analyze from the map usage cube (see also Section 3.2.1).

Based on the categorization of functions, the three levels of mobile applications, are further specified, and described in the remainder of this section.

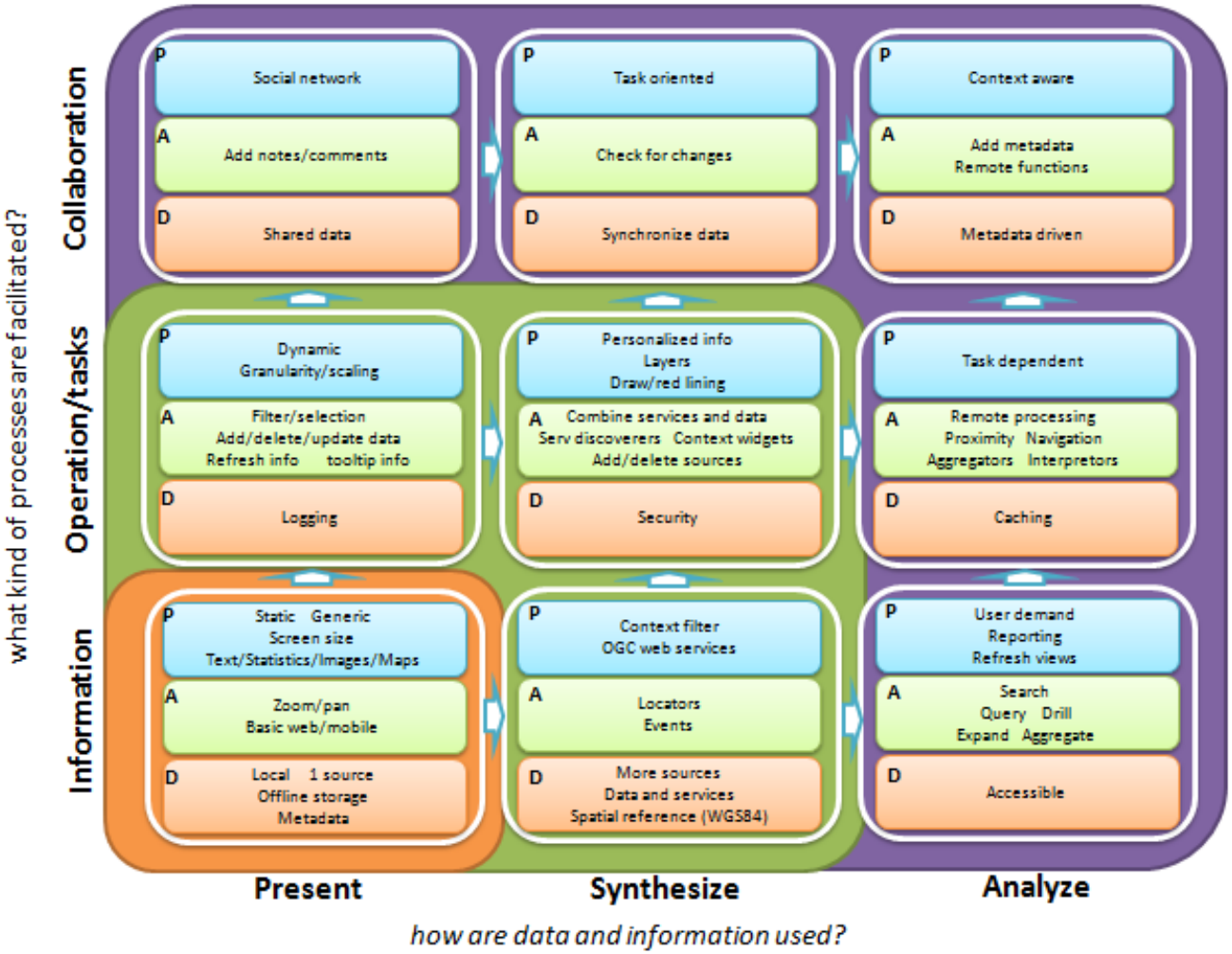

Figure 4.4 Categorization of functions; what processes vs how used

Basic applications

The basic mobile functionality aims to provide information, in a predefined format, from a single source. Many websites, for example, are based on a content management system (CMS) in which all data is captured centrally, and presented on the internet. Basic mobile applications, are structured likewise. An example is the Dutch EHBO app, that offers a databank with first aid knowledge.

The basic application scores low on richness and complexity, has low interaction, aims at a broad public, and data relations are known. Many websites are purely informative and publically available, and offer no specific functions other than an adjusted layout for different screen resolutions. Information is presented in formats such as text and figures, graphs, images, and image maps. The basic mobile functionality can be offered via a web server, or be copied onto a mobile device, and be updated as new information is available or a new version is built.
The basic application is the base of more complex applications, and therefore it is important to build the basic application with the MVC-principles in mind, and add metadata to the back end of the application.

Personalized applications

The more advanced, personalized applications, include the characteristics and features of the former discussed basic applications. The personalized applications are not dependent on one local source, and can bring data from different sources together. Further, these applications aim to support individual users that have to fulfill certain tasks and operations.
The user may have functions to filter data that is presented, and alter data in the database. Multiple sources, including third party databases and web services can be accessed, and the user must be enabled to add his own sources of data. Logging and security of the data, and thus the application, must be considered for these applications, as part of the data may be private.

Spatial functions that are supported in the personalized application are:

- OpenLayer web maps able to present different layers of spatial information, including the OGC web map services.
- The ability to add spatial data, including the option to draw on maps, for example red lining.
- Add spatial services, and context widgets, including locators and events.
- Scaling and granularity, to adjust the map to the fit the user's needs

The information presented can be based on the location of the user, or based on objects or other peoples' location. Information presented in maps are layer based, and have the ability to only present that information that is of interest for the user.

Context-aware applications

An app that aims for collaboration and analysis is more complex and richer than the levels discussed above. The application is not only used individually. Users are able to collaborate on tasks. Data has to be shared, and be synchronized, and users need functions to share information about those tasks. On the other hand, the user must be able to query the data, have access to analysis functions, or perform remote analysis and get the results returned. Metadata can be added to the data and functions to allow users control over the data contents and keep track of functions and their meaning. Metadata, or business rules, can be used to create metadata driven applications. Caching may become relevant, when large volumes of data are analyzed and captured.

The spatial functions, proximity and navigation, are part of the functions available in the context-aware apps. Presentation of information is adaptive to its full context. The application is not only adaptive to the user's preferences, knowledge, and skills, and technology features, such as display size, transfer rate, and connectivity, but also adapts to situation, and actions (Figure 2.8. Conceptual framework of mobile cartography) Implementation of fully context-aware applications asks for thorough understanding of different kind of knowledge areas, from social behavior, and business processes, to artificial intelligence and knowledge systems. The framework presented by Dey et al (Dey, Abowd et al. 2001) offer a practical environment to build context-aware applications. The context widgets and services, discoverers, interpreters, and aggregators allows the app to respond to its context. However, to what extent fully context-aware applications can be build, using any framework, is questionable, also whether building these kind of fully context-aware applications is really something to aim for. In the scope of this research, and also having existing apps in mind, app development aims at functionality that combines collaboration and synthesizing data. Some analysis functions, like proximity, may be added, however much of this functionality is developed in specific software, like navigation apps, data analysis and data mining apps, or mobile GIS.

4.4 Project approach

The scope of this thesis does not include the comparison of project management approaches for mobile app development. The basic principles of project management is to manage deliverables, time, and money. A project management approach is needed in developing enterprise applications, as people in different roles have to work together to successfully develop the application asked for. Therefore the project approach element is presented in the SMAD framework (Figure 4.5), as an important cornerstone in enterprise mobile development.

The market of mobile app development, is however full of technology pushed or supply driven applications, developed by a single, or small group of developers. The thesis case study was also performed by a supply driven approach, because of time constraints, no direct relation to the customer/end user, and the more scientific approach of this study. Still, also for these kind of development projects, managing such a project is needed.

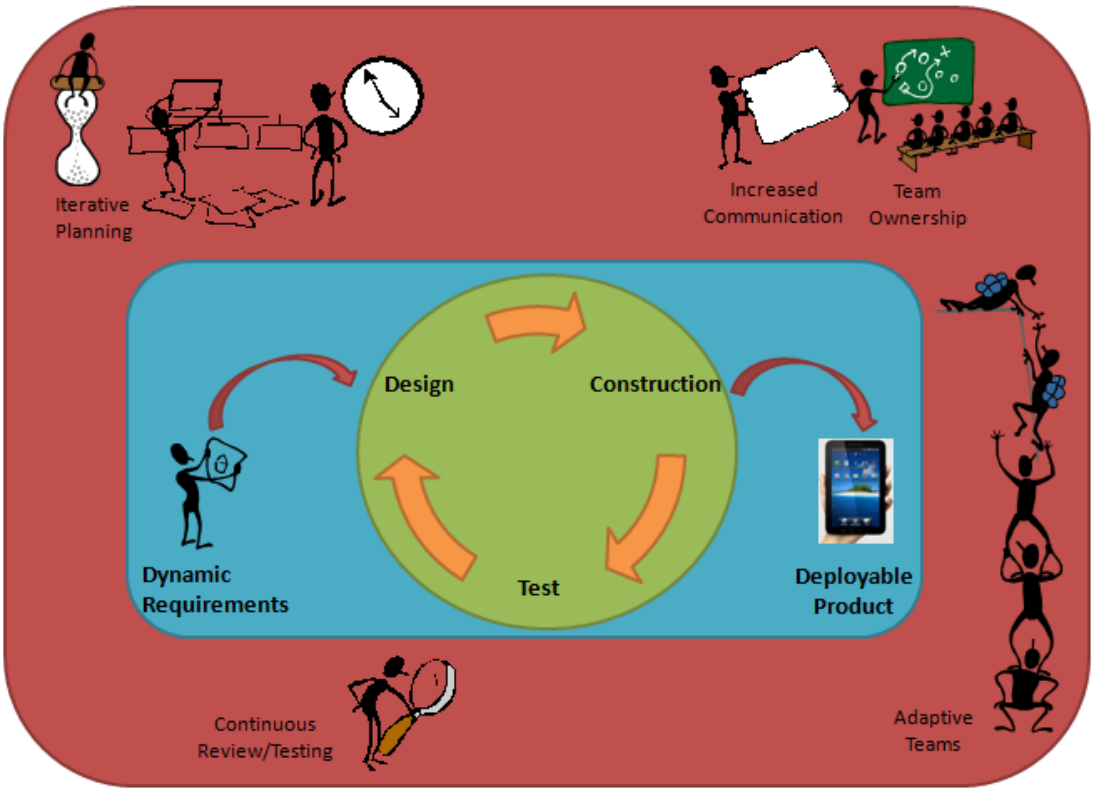

Figure 4.5 SMAD project approach: Agile project managment

Standard processes in IT development are: design, build, and test. Managing these processes can be performed in different ways, like the traditional waterfall methods, or the more incremental and evolutionary methods. The Agile project approach presented is chosen based on the three major attributes agile processes include (Alleman 2002):

- Incremental and evolutionary, which makes the development process flexible and adaptive to changes
- Modular and lean, which allows for using only those processes and parts of these processes that are relevant
- Time based, which built on iterative and concurrent work cycles

The entire team, including developers, project management, and customers, operate as a tight integrated unit, and are as a whole responsible for project management. (Tutorialspoint.com 2012) Figure 4.5 is based on the figure from Tutorialspoint that shows the agile development process, pointing out some underlying values of Agile project management: communication, simplicity, feedback, courage, and humility.

In a supply driven development project by a single person, this single person must be able to approach his project from different perspectives. Have at least one sparring partner to evaluate processes, and progress of the project, and not only focus on the development of the application itself. Solving the technical issues during development, a developer in HTML5, JavaScript, and CSS3 have access to many other developers and skilled others, via the internet, who are eager to share knowledge and skills.

4.5 Design and development

In Agile development the time to market is targeted to be much shorter than using traditional development methods (Figure 4.6, Figure 4.7). Instead of making one major adjustment after weeks of work, the Agile method makes hundreds of minor changes during the process. (Mark 2011)

Figure 4.6 Tradional web development 2000-2007
(Mark 2011)

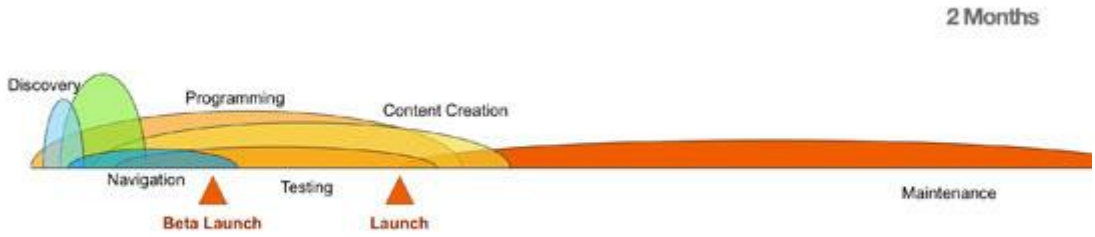

Figure 4.7 Modern Agile web development
(Mark 2011)

Picchi (Picchi 2011) uses a mobile information architecture, based on nine concepts that defines how information is structured, and how users interact in their context (Figure 4.8). An agile design process is presented, in which techniques like flowchart, site map, wireframes, and prototypes are used and reused as needed for the specific project.

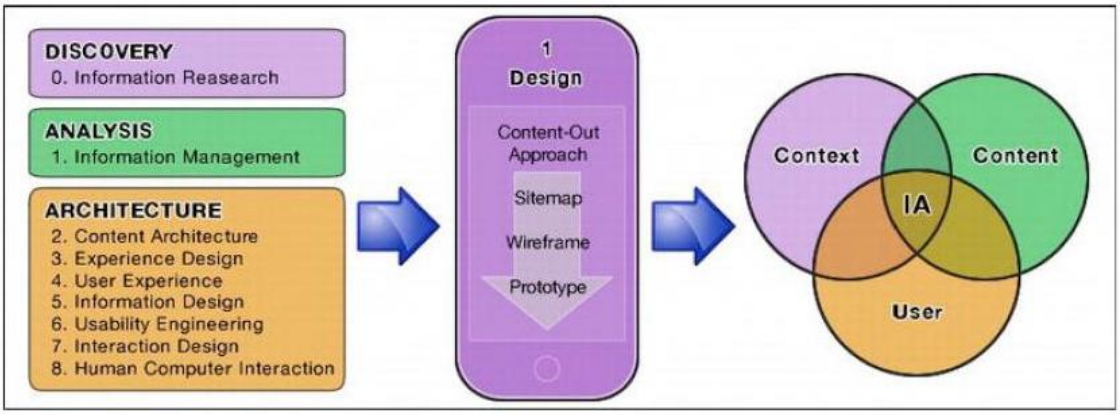

Figure 4.8 Three processes in the information architecture framework
(Picchi 2011)

The SMAD design approach presented in this thesis is based on Picchi's approach, in which the content-out approach has a central role. The content-out approach is an approach, in which the user demands and expectations, and focus on the user interface, are the starting point. Traditional application development methods are often more oriented towards the back end development. The SMAD method of app development is described by sixteen steps, as part of six main steps. Steps 1 to 12 are focused on the analysis and design, and steps 13 to 16 are on the actual prototyping cycle of development, test, and evaluate. The following steps in analysis, design, development, and test are promoted (see also the process diagram in Figure 4.9):

Analysis and Design

1. **Introduction to the app**
   A short description (1) of the application to be developed, adhere to special points of interests (2), and categorization of the application and its functions:
   - Basic functionality (3)
   - Personalized functionality (4)
   - Context-aware functionality (5)
2. **Priorities**
   Determine priorities, based on user needs and reusable objects (6) for different functions:
   - MOSCOW: set priorities, group into must, should, could, would
   - Check for reusable code available in Cookbook SMAD functions
3. **High level design**
   High level overview/Site map (7)
   - Different views to be developed (8)
4. **Detailed design**
   Detailed wireframes of all views (9, 10, 11)
   - Start with mobile portrait (Less than 479px)
   - Add mobile landscape (Between 480px and 767px), and web (960px)
   - Class Diagram (12)
   - Local and remote data involved (12)

Prototyping

5. **Prototyping**
   Set up the environment, and start prototyping: develop, test, and evaluate
   - Start with basic functionality (13)
   - Extend with personalized functionality (14)
   - Extend to context-aware functionality (15)
6. **Evaluation**
   Evaluation and update Cookbook SMAD functions (16)

The method is described in more detail in the remainder of this section. This method is illustrated by the development of the farmer prototype of the Sakaza Muhinzi case study, which is described in Chapter 5 on analysis and design, and Chapter 6 on prototyping.

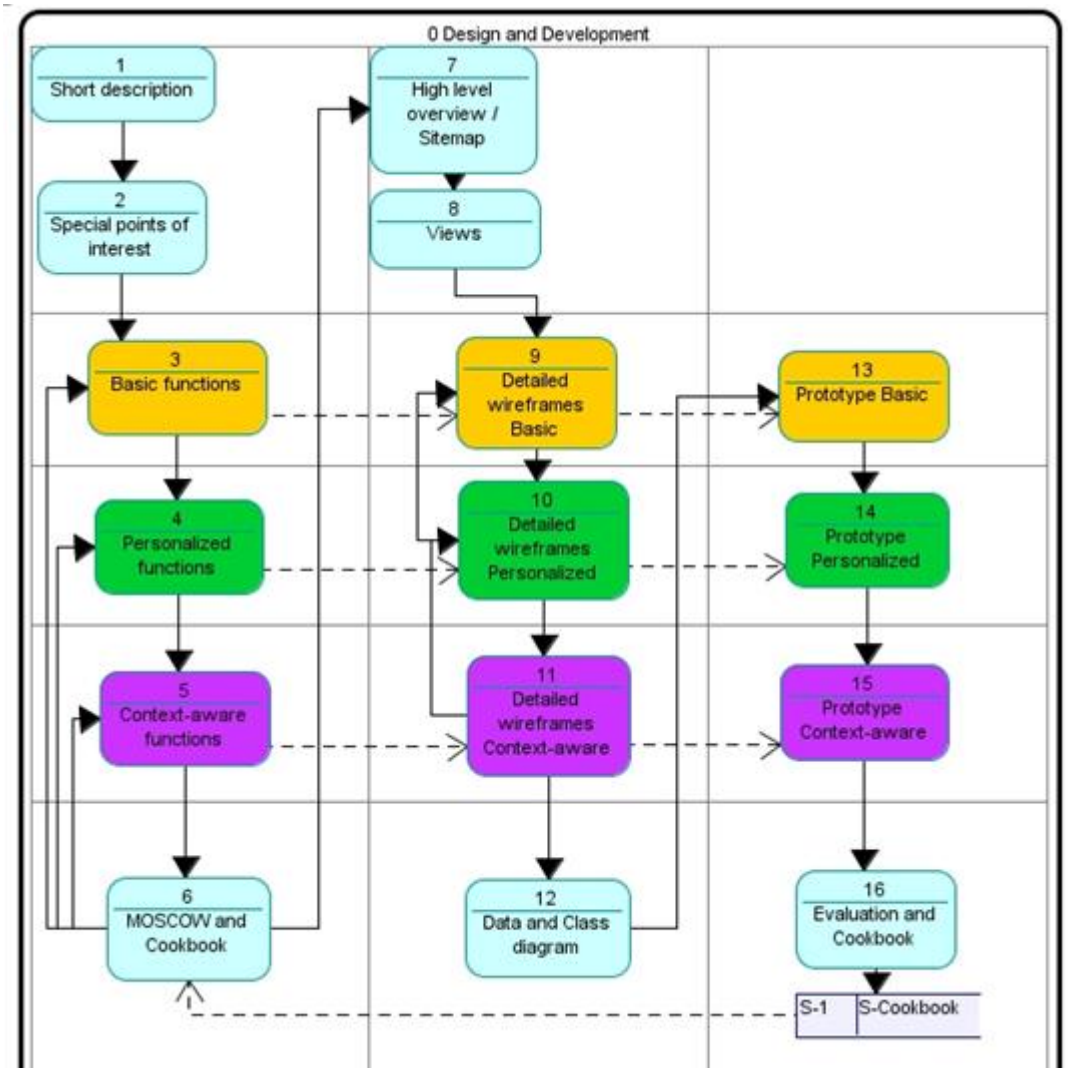

Figure 4.9 SMAD Design and development process diagram

4.5.1 Introduction to the app

The development of an app starts with an introduction to the app. This introduction gives a description of the functions, and functional areas that are to be developed, and why and for whom the app is to be developed. Mention aspects of the apps that are project specific, and need extra attention, and explain why these need extra attention in design and development. Use the SMAD quadrant "Categorization of Functions", Figure 4.3, and Figure 4.4, to categorize the functions into basic functions, personalized functions, and context-aware functions.
Define stages, increments in app development, based on the categorization of functions, and functional areas. In each stage a set of functions that can be used independently from later stages must be developed. Functions of earlier stages can be extended in a later phase. The more complex functions should be planned in a later stage than the basic and personalized functions.
The layout of the report on introduction to the app is described in the template in Table 4.1, which describes a number of items in the template, and a limitation on the length of such report.

Table 4.1 Template step 1: "Introduction to the app"

Item	Description	Pages
Step 1:	**Introduction to the app**	**1 to 5**
Project name	Name of the project, or app to be developed	-
Background	Background information on the project, and app. Answer question like: Who is the sponsor of the project, in what context the project is executed? Who is the supposed user of the app, and what are his/her information needs?	max 1
Short description	A general introduction on the app, and its functions. What is aimed for, and for whom?	max 1
Points of attention	Mention aspects of the apps that are project specific, and need extra attention, and explain why these need extra attention in design and development.	max 1
Basic functions	List all functions that can be categorized as basic functions	-
Personalized functions	List all functions that can be categorized as personalized functions	-
Context-aware functions	List all functions that can be categorized as context-aware functions	-
Stages	Define stages, increments, in app development, based on the categorization of functions. And give a short description of the stages, and functions within the stage.	max 1
Functions per stage	Group the categorized functions (step 3, 4, and 5) by stage in a table	max 1

4.5.2 Priorities

Based on the overview of "Functions per stage" from the step 1 template, the priorities are reported. The user needs, available reusable objects, and advanced functions hands criteria to determine a function's priority based on the MOSCOW prioritization. The functions that must be developed are prioritized as a "M" for "Must". The "S" stands for should, the "C" for could, and the "W" for would. During the prototyping phase, these priorities are evaluated weekly, and can be changed according to the situation, and the lessons learnt. After each stage, before continuing with the next stage, the priorities, as well as the functions in the list can be altered. For each function check for reusable code available in the SMAD Cookbook. The priorities can be described using template step 2: "Priorities", see Table 4.2.

Table 4.2 Template step 2: "Priorities"

Item	Description	Pages
Step 2:	**Priorities**	**1 to 2**
Project name	Name of the project, or app to be developed	-
Background	Background information on the prioritization, and references to the Cookbook.	max 1
Priority overview:	Priorities, and references to the Cookbook reported in a table, with items listed:	1 to 2
Function number	A unique code/number to refer to the specific function	
Stage	Stage of project in which the function is introduced	
App level	Is the function categorized as basic, personalized, or context-aware?	
Function	Function name	
Priority	Priorities are set following the MOSCOW priorities: M: must, S: should, C: could, and W: would	
Cookbook reference	A reference to available reusable code from the Cookbook	

4.5.3 High level design

Create a high level overview, or site map, of the app. In this site map the different functional parts of the app are defined as views. Determine the logical views that are related to the functional areas, and one or more functions, as defined in Step 1. Start with the homepage of the app, and continue with the functions that are prioritized "M", in Step 2. Design the homepage, and its logical information views, as a page with information that is publicly available, and offer access to personal functions to users that log in, via a view on this same homepage.

Table 4.3 Template step 2: "High level design"

Item	Description	Pages
Step 3:	**High level design**	**1 to 3**
Project name	Name of the project, or app to be developed	-
Site map	Design the site map of the app, as in Figure 4.10	max 1
Short description	Add a short description of all views	1 to 2

Group functions into groups of three or four views that are accessible from a parent view. The levels of views in the mobile app should not exceed four levels, to have an acceptable user experience. The template in Figure 4.10, created with Open ModelSphere[12], is a starting point for a site map design. Relations between the views are not only hierarchical, top-down. Include important cross-view relations in your site map. Add a description of the defined views.

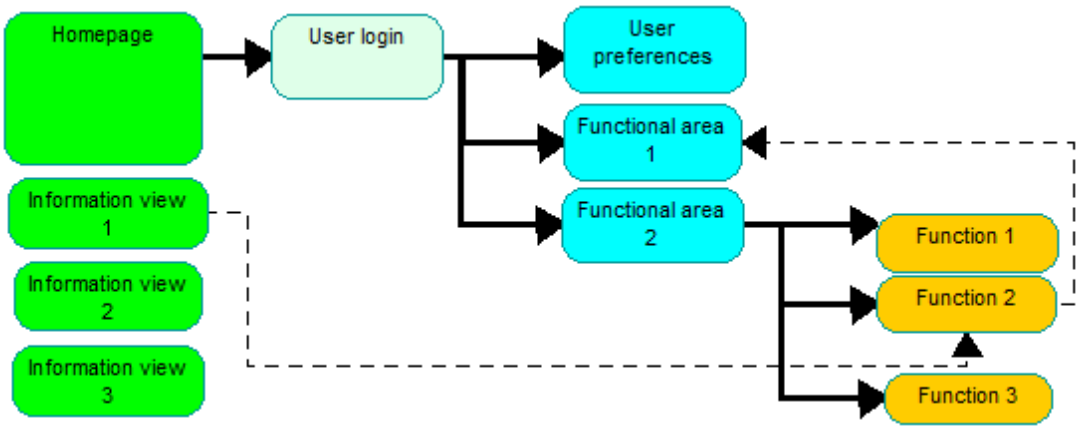

Figure 4.10 Example step 3: "High level design", a site map design

[12] http://www.modelsphere.org/

4.5.4 Detailed design

From the high level design site map, select the views that are related to the functions in the first stage, defined in Step 1. For the first stage, design wireframes for the applicable views. Wireframes are designs of the pages in the app without actual application logic. They can be drawings on paper, but to create a more detailed wireframe a digital version is preferred, see Figure 4.11.

Table 4.4 Template step 3: "Detailed design"

Item	Description	Pages
Step 4:	**Detailed design**	**1 to 5**
Project name	Name of the project, or app to be developed	-
Wireframes	Design wireframes per stage, see in Figure 4.11	max 1
Short description	Add a short description of all wireframes	max 2
Data	List the data that are part of the app, with a short description of its usage	max 1
Objects	List the object items that are part of the app	-
Class diagram	Create a conceptual class diagram of the objects and its relations	max 1

The add-on in Firefox Pencil[13] can be used to create these detailed wireframes. This tool offers default items for app design, such as radio buttons, and drop down menu items, and one can include images to make the design more attractive.
Create a conceptual class diagram in which all objects are defined, and the major data items. The data relations are included, as well as actions that the objects can perform. See Table 4.4, for a template on the detailed design.

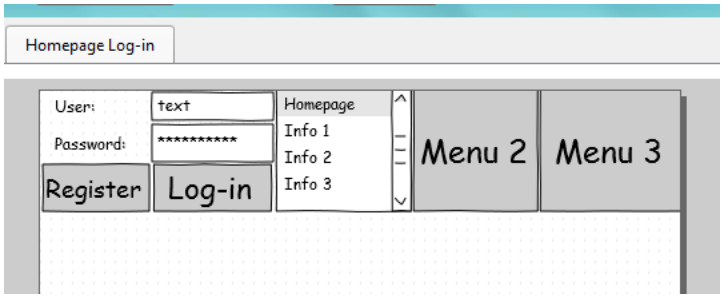

Figure 4.11 Example step 4: "Detailed design"

[13] http://pencil.evolus.vn/en-US/Home.aspx

4.5.5 Prototyping

The prototyping environment must be set up in-line with the technical architecture described in the first SMAD quadrant. The set up of the environment is described in Appendix B, Section B2. A quick-start in setting up the environment is to only set up CouchDB, and CouchApp, and install Prototype 3, as described in Section B5.
The following stepwise approach helps in building web apps, with web maps:

1. Choose a HTML template to start with
2. Create the basic public content
3. Realize the page navigation
4. Choose an appropriate lay-out, and styling
5. Include widgets, and other third party services
6. Add simple data functions
7. Add web maps
8. Add more complex data functions

Ad 1. Choose a HTML template to start with

The CouchApp controls what is shown on the screen. The app pages, and its functions are served by CouchApp from the Couchdb. The HTML page, or view, determines what content, and logic is on the page. The prototype's user-interface is based on the fluid Bootstrap example,[14] Plots.html of Prototype 3 (see Section B5, and the attached CD) can be used as a template for development.

The structure of the HTML file is as follows:

- The file start to state it is a HTML file:
  `<!DOCTYPE html>`
- The header section (`<head> … </head>`) starts with some meta information on the file.
- Update the title of the page (`<title> … </title>`), and the content of `meta name="description"`, and `meta name="author"`.
- The second part of the header section contains the links to the attached stylesheets. The link to bootstrap.css (`<link href="style/bootstrap.css" rel="stylesheet">`) is needed for the fluid behavior, and styling of the page, according to the Bootstrap framework. If OpenLayers is included in the app, the style.css stylesheet of OpenLayers should be linked to. An additional project stylesheet can be created, and the link to this CSS stylesheet should be in the header as well.
- The HTML files of the prototype contain additional CSS styling in the header section. These lines of style code, are read after the stylesheets are read, and therefore will overwrite styles from these files. As the type of code is CSS, and not HTML, the entry `<style type="text/css">` is included.
- The header can also contain links to the JavaScript, however if these links are placed at the end of the document the page loads faster.

[14] http://twitter.github.com/bootstrap/examples/fluid.html

- The body section (<body>…</body>) is the main section after the header. In the body is defined what is visible on the screen. It contains the navigation bars, sections that include logic, and data fetched via JavaScript, or information set up in the HTML file itself.
- The body contains a footer section (<footer>…</footer>) that shows at the bottom of the page.
- At the end of the HTML file, between the closing tags </body> and </html>, the JavaScripts are loaded. In Couchapp a loader file is defined that loads all necessary JavaScript. In Plots.html the loader file is loaded from the directory vendor/couchapp/_attachments: <script src="vendor/couchapp/loader_SM3PlotsSM3.js"></script> <script type="text/javascript" charset="utf-8">
- CouchApp loads default JavaScript files from CouchDB, and CouchApp to function. Additional JavaScript frameworks are loaded: bootstrap.js, OpenLayers.js, and knockout-2.1.0beta.js. These JavaScript files must be loaded to be able to use functions from these frameworks.

Ad 2. Create the basic public content

In the body section, the content between the navigation bars, and the footer, are defined in a Bootstrap "span9" class . The side bar navigation uses 3/12 of the screen, and the body part besides this navigation bar is 9/12 of the screen. In this part of the body, information can be added using a "row-fluid" , which makes the information be shown correctly on both wider, and smaller screens. Each row can be divided in columns. To create three columns in a row, the "span4" class ( ) is used, which means 4/12 of the fluid row is used for the section.

For each information item in the homepage (see Figure 4.10 in Step 3), create a section with the id of the section (div id=), which is a short name of the information item. Add text and images that should show in the section. Standard layout tags for headers (<h1> <h2> <h3>), and text in paragraphs (<p>) can be used, as well as tables, list items, and so on.

Ad 3. Realize the page navigation

Adjust the navigation bars to the needs of the app. The template html contains three navigation bars: a top bar for the smartphone, and a top bar, and side bar for tablet and PC.

The two top navigation bars have both more button groups that have dropdown button logic attached. The button group has a main button, tagged Home , and list items in the "dropdown-menu" class. Alter the links (href=), and names of the buttons, into navigation links relevant to the app. To navigate within a HTML page the section ids, as implemented in the former step can be used. In the links, include the section id after the hash (#), for example: Coffee Plots
Button groups can be added, and deleted.

The sidebar navigation bar can be altered by editing the links in the section "well sidebar-nav" ( )

Ad 4. Choose an appropriate lay-out, and styling

Bootstrap offers a customized Bootstrap configuration file[15]. By means of setting some of the variables to have certain coloring, styling, or font, the total looks of the page/app can be changed.

Ad 5. Include widgets, and other third party services

The HTML, JavaScript, CSS app structure is suitable to include different kind of web services and widgets. Information based on logic, and data that resides anywhere, is included in the app page by including the widget or service.

Ad 6. Add simple data functions

The former steps were mainly focused on the HTML pages. To add simple data functions one should develop on all levels of the MVC model. Data is fetched from, and stored to the database, JavaScript logic is developed to control the information stream between the HTML view and the database model, and the HTML must be adjusted to show the proper information. The KnockOutJs framework is used to develop the application logic. The MVVM (Model-View-ViewModel) model of KnockOutJs is derived from the MVC (Model-View-Controller) model.

The ViewModel is a central element in KnockOut. The ViewModel defines the structure of the data, and functions on that data. To control the data flow from and to the database, functions are defined to get the data from the database, and to save the updated data.

Appendix C.3 contains the JavaScript (Section C.3.2) and HTML code (Section C.3.1) in which JSON data (Section C.3.4) is fetched from the CouchDB database via a CouchDB view (Section C.3.3), part of these data are feature attribute data, which are shown in the app by the HTML logic. In the app the user can update the attribute data, and send the updated data back to the database, by logic defined in the JavaScript, and attached to a button in the HTML.

Field	Value
_id	"c8f066c94084e9cfe131ff59cd0007c8"
_rev	"14-e3d9595e70a1e74689efb5527acd286a"
EPGS	"4326"
doc_type	"farmerPlots"
features	⊟ 0 type "Feature" ⊞ geometry ⊞ properties ⊞ 1 ⊞ 2 ⊞ 3
owner	"FarmerA"
type	"FeatureCollection"

Figure 4.12 CouchDB JSON data Farmer plots

[15] http://twitter.github.com/bootstrap/download.html

The data

The structure of the data in the database is depicted in Figure 4.12.
The example is on fetching the farmer plot attribute data, displaying these data, and being able to alter these data, and sending the data back to the database again.

The JavaScript

The ViewModel (viewModelAtt()) is defined by two KnockOut observables, ko.observable(), and a KnockOut observable array, ko.observableArray([]):

```
function viewModelAtt() {
    var self = this;
    self.CdbDoc = ko.observable();
    self.AttF = ko.observableArray([]);
    self.DocID = ko.observable();
```

The array (AttF) is the actual attribute data of the farmer plots that are to be displayed in the app. The other items are variables to log the document id from CouchDB to be able to save the data back to the database, creating a new revision of the farmer plot document.

Fetch the data from the database by the JQuery function $.getJSON. The URL points to the farmer plots view in the CouchDB. This view returns the farmer pots of the user:

```
// initial data

    var CDBfarmerPlots = jQuery.parseJSON(
        $.getJSON(
'http://127.0.0.1:5984/data_sm/_design/pny_sm/_view/farmerplots',
        function(data){
```

The response from the database view is in JSON format:

```
{"total_rows":1,"offset":0,"rows":[
{"id":"c8f066c94084e9cfe131ff59cd0007c8","key":"c8f066c94084e9cfe131f
f59cd0007c8","value":{"_id":
```

The response from the database is linked to the defined observables in the ViewModel:

```
        var mappedAtts = data.rows[0].value.features;
        self.AttF(mappedAtts);
        var mappeddocId = data.rows[0].value._id;
        self.DocID(mappeddocId);
```

The function to save data uses the document id from CouchDB to update data in the database by creating a new revision id for the original document. A put request is formulated with the JQuery function $.ajax with JSON formatted data. The URL points to the database in which the data is stored (data_sm), and the variable for document id, docId, is concatenated to the URL string:

```
     // ... Save data

    self.save = function() {
    var docId = self.CdbDoc._id;
           $.ajax("http://127.0.0.1:5984/data_sm/"+docId, {
                data: ko.toJSON(self.CdbDoc),
                type: "PUT", contentType: "application/json",
                success: function(result) { alert(result) }
            });
        };
```

The ViewModel and functions in the JavaScript file, are linked to a section (DataAttF) in the HTML file by the ko.applyBindings assignment:

```
ko.applyBindings(new viewModelAtt, $("#DataAttF")[0]);
```

The HTML

Within the HTML file a section DataAttF is defined:

```
<section id="DataAttF">
```

In this example information on the number of plots is shown, and for testing purposes the CouchDB document id is shown. The KnockOut observable array AttF contains the information displayed:

```
    <p>Number of plots: <span data-bind='text:
AttF().length'> </span> </p>
    <p>Couch document id: <span data-bind='text: DocID'></span> </p>
```

The table with editable attribute data is shown as the number of plots is one or more:

```
   <table data-bind='visible: AttF().length > 0'>
```

Per plot the attribute items are shown in a table row (tr) with table details (td), as updatable fields (input):

```
<tbody data-bind="foreach: AttF(), visible: AttF().length > 0">
  <tr>
  <td><span data-bind='text: properties.id_plot'></span></td>
  <td><input data-bind='value: properties.owner' /></td>
  <td><input data-bind='value: properties.plotname' /></td>
  <td><input data-bind='value: properties.coffeeType' /></td>
  <td><input data-bind='value: properties.status' /></td>
  <td><input data-bind='value: properties.description' /></td>
  </tr>
```

In the HTML only the attribute data is shown, however KnockOut will on the background keep the other farmer plot information from the CouchDB document in memory. A save button is defined to save the updated data to the database. This button is linked to the save function defined in the KnockOut ViewModel:

```
<!--    button click: save -->
    <button data-bind='click: save'>Save</button>
```

Ad 7. Add web maps

To add web maps to the app, the OpenLayers development examples are a starting point. The HTML file only needs minor changes, as the logic to present data in a web map can be separated in JavaScript.

The HTML

In the HTML file, add the following sections, and classes:

```
<div id="map" class="smallmap"></div>
```

The link to the OpenLayers framework should be included in the CouchApp loader, as described in step 1, as well as the link to the stylesheets:

```
    <link rel="stylesheet" href="OL/PNY_OL_style_SM2.css"
type="text/css">
    <link rel="stylesheet" href="OL/theme/default/style.css"
type="text/css">
```

The JavaScript

The OpenLayers JavaScript must be loaded, and when using the Google Maps API version 2 base layers, the API must be loaded in the CouchApp loader file. Use the URL to Google Maps API with the key to use for local development:

```
"vendor/couchapp/assets/js/OL/OpenLayers.js",

"http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAAjpkAC9e
PGem0lIq5XcMiuhR_wWLPFku8Ix9i2SXYRVK3e45q1BQUd_beF8dtzKET_EteAjPdGDwq
pQ"
```

In the JavaScript file, start defining variables to zoom into the map at a certain level, and center to a certain longitude and latitude:

```
        var lon = 29.6739; //5;
        var lat = -2.53963; //40;
        var zoom = 10;    //5;

        var map, layer;
```

Create a new OpenLayers map, linked to the map section in the HTML, and add the control to be able to switch layers:

```
function init(){

        map = new OpenLayers.Map('map');
        map.addControl(new OpenLayers.Control.LayerSwitcher());
```

Define the Google layers:

```
        var gphy = new OpenLayers.Layer.Google(
            "Google Physical",
            {type: G_PHYSICAL_MAP});
        var gmap = new OpenLayers.Layer.Google(
            "Google Streets",
            {numZoomLevels: 20});
        var ghyb = new OpenLayers.Layer.Google(
            "Google Hybrid",
            {type: G_HYBRID_MAP, numZoomLevels: 20});
        var gsat = new OpenLayers.Layer.Google(
            "Google Satellite",
            {type: G_SATELLITE_MAP, numZoomLevels: 22});
```

Add the Google layers to the map:

```
    map.addLayers([ghyb, gsat, gphy, gmap]);
```

Set center of the map to variables defined:

```
    map.setCenter(new OpenLayers.LonLat(lon, lat), zoom);
```

Additional layers can be added, for example, the farmer plots as in Figure 4.12. See Appendix C, Section C.1 for JavaScript on adding farmer plot feature data. The code starts with defining styles of how the features are displayed on the map by default, and when selected (styleMapFarmer). A vector layer is added with the name “Farmer coffeeplots”, using the farmer style map.
OpenLayers contains functions to formulate HTTP requests. The farmer plot data can be retrieved via the farmer plots view in CouchDB, and then be added to the farmer vector layer:

```
OpenLayers.Request.GET({
     url:
"http://127.0.0.1:5984/data_sm/_design/pny_sm/_view/farmerplots",
     headers: {'Accept':'application/json'},
     success: function (req)
             {
             var g = new OpenLayers.Format.GeoJSON();
             var data = jQuery.parseJSON(req.responseText);
             console.log(data);
             var features = data.rows[0].value;
             var feature_collection = g.read(features);
             vector_layer_farmer.addFeatures(feature_collection);
             }
```

Ad 8. Add more complex data functions

After having completed Steps 1 to 7 one has created a CouchApp, with Bootstrap lay out, KnockOutJs functions, and with OpenLayers web map functions. Additional functions can be added. The context-aware functions, with the focus on collaboration, or analysis of data, as defined the third level of mobile apps, can be such more complex data functions.

The example code in Appendix C, Section C2, is showing how to combine a KnockOutJs ViewModel, with the OpenLayers framework. The data that are fetched to show in the OpenLayers map, are also used to display, and being able to alter, the attribute data.
The user can select a plot, and attribute data of the plot is displayed.

The HTML file should contain sections for status, and DataAttSelectFarmer.

4.5.6 Evaluation

Last step in design and development is the evaluation. Evaluate the prototype by the four quadrants of the SMAD framework (Table 4.5): (I) technical architecture, (II) categorization of functions, (III) project approach, and (IV) design and development (See also Figure 2.1 in Section 2.2). The evaluation can help to evolve the SMAD framework, by feedback and contributions to the Cookbook. Above all, an evaluation is input for a next app development project, in which lessons learnt from an earlier project are an important knowledge base to the project team.

Table 4.5 Template step 6: "Evaluation"

Item	Description	Pages
Step 6:	**Evaluation**	**1 to 5**
Project name	Name of the project, or app to be developed	-
Technical architecture	Are the tools, and techniques proposed by the SMAD framework sufficient to build the app? Is the code structured and transparent, and is there a clear distinction between HTML, JavaScript, and CSS components?	max 1
Categorization of functions	Did the categorization of functions help in prioritization of the development work? Were functions missing in the categorization, or not categorized correctly?	max 1
Project approach	Is the Agile project approach helping in developing the app? Set the right priorities, develop within a certain time span, plan the work properly, work with more people on the project.	max 1
Design and development	To what extent the app includes the functions that were defined in the analysis and design phase? What lessons can be learnt from the prototyping stages, and the problems run into. Is the app platform-independent?	max 1
Cookbook contributions	What code has been developed that contributes to the SMAD framework? Describe in short what functions has been added, and contribute the code separately.	max 1

Part III. Case Study and Use Cases

Phase 2: Case Study and Use Cases

Part III describes Phase 2 of the research "Case Study and Use Cases " (Figure 4.13). The Research questions 3, and 4 are answered during this phase. Question 3 is on specific conditions to be dealt with developing an app for use in the developing world, and Question 4 is on case study data, and functions. Use cases on presenting web maps, positioning, and adding personal spatial data are three important fields of mobile functionality. By exploring these use cases, and describing their specifications, the study aims to provide a solid basis for the actual prototyping phase. Both central data, as well as personal and local data is used as input for the prototyping phase.

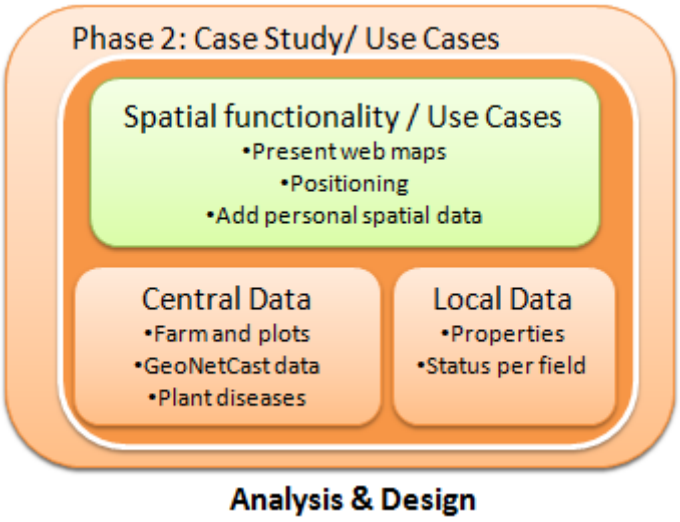

Figure 4.13 Phase 2: case study and use cases

5 Analysis and design

5.1 Introduction

The SMAD framework is put into practice by the Sakaza Muhinzi case study. Performing this case study, the framework is evaluated. Does the framework indeed offer a structured method to develop geo-enabled mobile apps? The prototype that is built aims to offer benefits to Rwanda coffee farmers by providing good access to information, both external and personal information, and facilities to share information. On return, the Sakaza Muhinzi system wants to obtain information from farmers about their coffee farming practices, for example information on farm activities. The app should help the farmer in doing this.
Providing sensible information to the farmer could empower farmers and their cooperatives, among others, to make better decisions on for example: buying seeds and fertilizers, acting on current weather predictions and information on plant diseases and pests, and selling their products. Farmers, and their cooperatives, having the right information will be able to maximize turnover. The case study aims to contribute to the PHD study of Clarissa Kagoyire, and the Sakaza Muhinzi project (Kagoyire, de By et al. 2011). Inputs, such as, user needs and demands, available data sets, and cultural aspects are gained from this study, project members, and when possible from the farmers themselves.

This thesis work is performed within a geo-science context of the MSC-program "Geographical Information Management, and Applications", therefore the case study is built around three subjects related to following spatial functions:

- Present web maps
- Positioning
- Add personal spatial data

Translated to a mobile app for coffee farmers in Rwanda:

1. Farmers get mobile access to information on the coffee plots
2. Farmers claim plots they own
3. Farmers inform the Sakaza system when they perform activities on the plots

The Sakaza Muhinzi project is introduced in Section 5.2 The Sakaza Muhinzi project. The remainder of the chapter describes the design of the application and its functions to be developed, following the stepwise SMAD approach presented in Section 4.5.

5.2 The Sakaza Muhinzi project

This section is an abstraction from the project description document of Sakaza Muhinzi (Kagoyire, de By et al. 2011):

Sakaza Muhinzi wants to improve information access for farmers in domains that help them to step up the quality and quantity of their produce. Relevant information does not always reach farmers or farmer cooperatives, and also they often hold knowledge on farming practices that is inaccessible for other farmers. Sakaza Muhinzi attempts to open the communication channels between farmers and farmer cooperatives themselves, but it also attempts to bring the body of agricultural knowledge closer to their doorstep. The project uses appropriate, modern technology as found in ICT and in Geoinformation Technology.

At present, Sakaza Muhinzi aims to support farmers that grow coffee as their most important (income) crop. The project started to collaborate with the Maraba cooperative in S Rwanda, and with some 20+ of its farmers for an initial, experimental set-up. The project work at present aims to identify the most urgently needed information exchanges between the stakeholders.

The project platform consists of a centralized information system, a single experimental telecentre, operated by the Maraba cooperative, and farmers equipped with simple cell phones. The central system is currently hosted in Butare, with a typical software stack of database server, image server, web map engine and web server, as well as an SMS gateway. The telecentre is a computerized meeting point for community members, with a small number of computers that have an internet connection. The telecentre is based in Maraba village, district of Huye, NW of Butare (2°36′S 29°45′E). Though 3G coverage is available in and around the village, farmers do not own (yet?) 3G phones, and so the platform aims to support standard cell phones at present, for its communication with individual farmers.

The alpha version of the platform is currently under development on the basis of an extended version of Ushahidi,[i] the web platform for 'activist mapping', see www.ushahidi.com. Extensions to that system are twofold: (1) the database back-end system is changed from MySQL to PostGIS/Postgresql, and (2) the data model is extenden to account for specific farmer, cooperative and farming information.

SakazaOnline will, over time, accumulate a number of information resources, to be shared amongst its users:

- A farming good-practice knowledge base
- Farm and farmer profiles reflecting on-farm activities
- Cooperative profiles reflecting cooperative activities, including shared resources usages
- Spatial base registrations in support of the above.

The following information exchange between platform stakeholders is indicated by the project:

Information to farmers

- Coffee knowledge base (plant diseases, photos, symptoms, remedies, later to be extended)
- Printed and web map of their property, fields, where available crops associated
- Derived information such as elevation, slope, aspect, soils
- Relayed information from OCIR-Café bulletins
- 5-day weather forecasts
- Information on farm inputs (where/how much)
- Information on logistics (washing station availability, dry mill availability, trucks)

Information to cooperatives

- Regular report on number of farmers in the Sakaza Muhinzi community in their area of operation
- Sakaza Muhinzi member farmer statistics; status quo and trends
- Inputs to value chain planning

Information from farmers

A gradual build-up of the information on their farm, farm plots, crops, cropping system, production:

- Administrative cell to which farm belongs
- Per farm: number of plots, last year's coffee production
- Per coffee plot: number of plants, age of plants
- Per non-coffee plot: max. three most important crops, per crop: size occupation percentage
- Over time: updates to any of the above
- Requests for capacity at washing station
- Requests for transport
- Reports on disease occurrences
- General observation reports and requests to community
- OSM inputs in form of roads, tracks, paths

Information from cooperatives

- Price developments for farm inputs
- Price developments of coffee along the value chain
- Production statistics

Information from government boards

- Advisories specific to crops, farm inputs and/or farm logistics
- Alerts and warnings on climatic conditions, pests and diseases, and their remedies

5.3 Step 1: Introduction to the app

From the Sakaza Muhinzi project document, and in consultation and agreement with the thesis' supervisor, the aim of the case study mobile app is defined, and described in this section. The sections starts with some general issues that have to be taken into account developing an app for Rwanda coffee farmers. The section continues with a short description of the functionality on basic level, personalized level, and on context-aware level, as defined in the SMAD framework.

The three general issues that are taken into account designing and developing the prototype are: offline access, modularity of language, and modularity of symbols. A 3G network is available around Maraba village, however a farmer is not expected to be online all the time. Besides, in other parts of Rwanda 3G network availability may not be in place. The prototype that is built during this thesis work is in English, and the farmers must be able to use the app in their own language. The Rwanda language is built up from rather long words, which can cause problems in layout when translating from English to Rwandese. For the symbol set used, it is important that the symbol reflects the proper illustration for certain actions and activities. Some symbols will be interpreted completely different from the Dutch interpretation.

Looking at the more specific functions of the app, the app is built up around three main spatial functions that are part of the SMAD quadrant "Categorization of Functions". These include: present web maps, positioning, and add personal spatial data. Translated to a mobile app for coffee farmers in Rwanda:

1. Farmers access information on plots, weather statistics, satellite images
2. Farmers claim plots they own
3. Farmers inform the communal social system when they perform activities on their plots

Ad 1. Farmers get mobile access to information on plots, weather statistics, satellite images, and other data. This information is related to the farmer and his plots (see also point 2). Positioning can be used to show data of the specific location and lands. The farmer can select which information he is interested in, and can add web services.

Ad 2. Farmers claim plots they own. Information on plots is available and shown as polygons in a web map, a satellite image is used as base map (see also point 1). The farmer can select the plots (polygons) s/he owns, and claims ownership of these parcels. A more advanced function that could be implemented is to cluster or split up polygons, and administer the whole cluster at once, or part of the original land. Also adding polygons, changing the size and position of the farmer's polygons, and red lining might be an interesting add-on.

Ad 3. Farmers inform the communal social system when they perform activities on the land, and are enabled to use the app to insert and share this information. This information can be plot-specific, and includes spatial data. Drop down lists should contain most common operations and statuses, as to make information capturing easy accessible. Farmers can exchange information with other farmers, as well as with the central system. The knowledge system of Sakaza Muhinzi returns information based on the activities of the farmers, and status of the plots, and its crops.

The functions related to the three subject areas are grouped by stage, and application category in Table 5.1.

Table 5.1 The functions of the Sakaza Muhinzi farmer app by category and stage

	Basic	Personalized	Context-aware
Plots	Scope: Rwanda, plot information ITC • Static information on plots, weather statistics, satellite images, and other data • Offline use • Metadata	Scope: Farmer • Identity of farmer • Information related to the farmer's preferences, and his land • Show plot polygons in a web map, a satellite image is used as base map • Positioning • Select information of interest • Add and delete web services	
Claim		Scope: Farmer • Select the polygons and claim • Cluster or split up polygon • Add polygons • Alter polygons (size, position, administrative) • Red lining	
Activities		Scope: Farmer • Add activities by type and status to polygons	Scope: Farmers' cooperative • Share information with farmers • Share information with ITC • Query on land use and status

5.4 Step 2: Priorities

This paragraph describes steps six and seven from the "Design and Development" SMAD approach: usage of reusable objects and setting priorities (Section 4.5).

No reusable objects were available when designing this first mobile application. The cookbook was still to be set up, and to be filled with code objects and lessons learnt from the prototyping phase. During the prototyping phase knowledge and example codes are collected from the Web. These can be code examples from the used development frameworks, however can also be from other HTML5/JavaScript platforms. These example codes are used as a base to develop the proper functionality, which result in reusable objects in the Cookbook for the functions that are part of the SMAD framework.

Table 5.2 describes the functions that are to be developed in the prototyping phase. The first column is a unique code to refer to the specific function. The first character refers to the stage in Column 2, and the second character refers to the level of application in Column 3. The priorities are set following the MOSCOW priorities: M: must, S: should, C: could, and W: would.

Table 5.2 Functions to be developed, and their priorities

Nr	Stage	Level of App	Function	Priority
PB1	Plots	Basic	Static information on parcels, weather statistics, satellite images, and other data	M
PB2	Plots	Basic	Offline use	S
PB3	Plots	Basic	Metadata	C
PP1	Plots	Person	Identity of farmer	S
PP2	Plots	Person	Information related to the farmer's preferences, and his land	C
PP3	Plots	Person	Show land polygons in a web map, a satellite image is used as base map	M
PP4	Plots	Person	Positioning	M
PP5	Plots	Person	Select information of interests	C
PP6	Plots	Person	Add and delete web services	C
CP1	Claim	Person	Select the polygons and claim	M
CP2	Claim	Person	Cluster or split up polygon	M
CP3	Claim	Person	Add polygons	C
CP4	Claim	Person	Alter polygons (size, position, admin)	C
CP5	Claim	Person	Red lining	S
AP1	Activ	Person	Add activities to polygons	M
AC1	Activ	Contxt	Share information with farmers	S
AC2	Activ	Contxt	Share information with ITC	S
AC3	Activ	Contxt	Query on land use and status	C

5.5 Step 3: High level design

From the analysis, and categorization of functions that are proposed for the farmer app, the sitemap is created, and depicted in Figure 5.1.

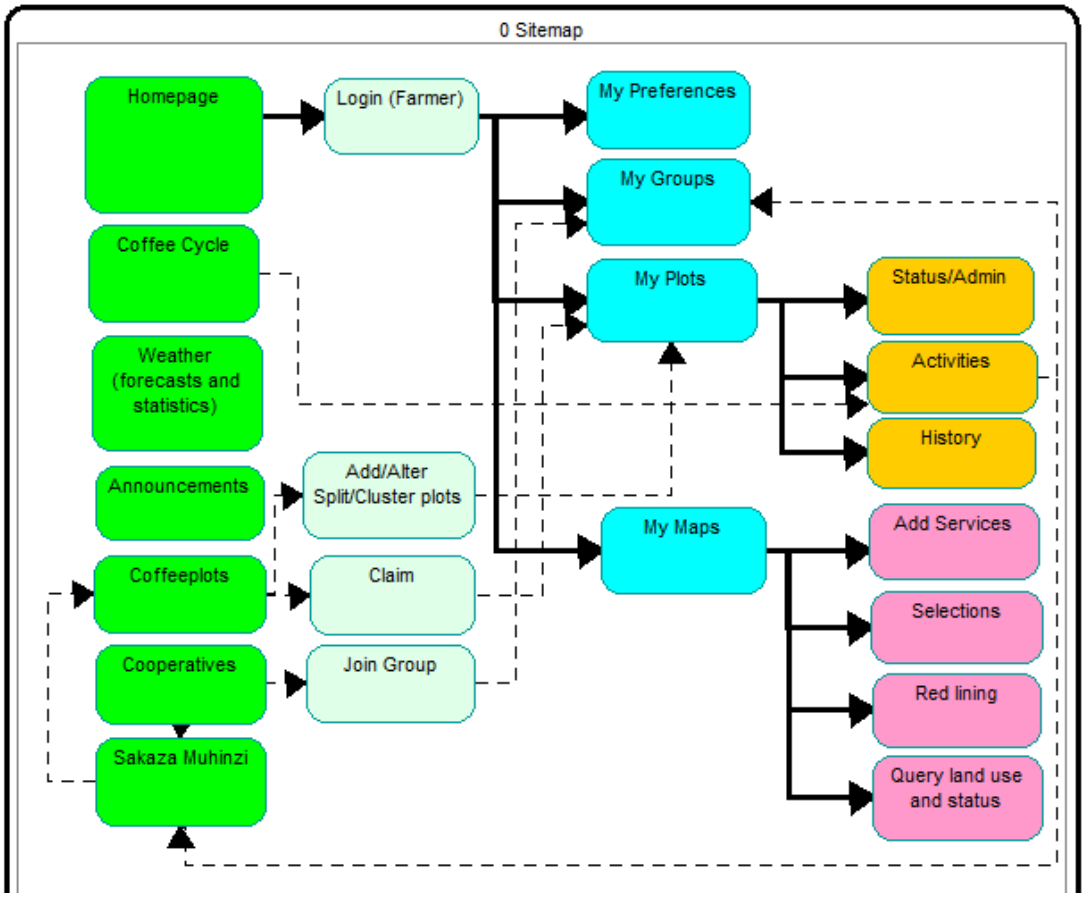

Figure 5.1 Sitemap of the Sakaza Muhinzi farmer app

Based on the sitemap, the views are grouped in two main groups, general (green), and farmer specific (blue) views. These main groups give access to specific actions and functionality, such as login, claim plots, update status/administration, and add activities.

The general views give access to the following information:

- Coffee grow cycle: Information on the various stages of coffee growing, as well as activities related to the various stages, and on current status of the growing cycle. Information from the coffee knowledge base on plant diseases, photos, and remedies.
- Weather forecasts: five-day weather forecasts, current weather, and weather statistics of previous years. Use local weather forecasts, if available, and present the information in maps, as well as numerical.
- Announcements: General information/messages to all farmers, or at least a major part of the farmer group, from the government, cooperatives, and other stakeholders, as well as relayed information from , OCIR-Café bullitens. The government can advice on crop, farm inputs, and farm logistics, and alerts on climatic conditions, pests, and diseases.
- Coffee plots: Information on the various coffee plots in the area, with additional information on elevation, slope, aspect, and soils. Farmers might share historical production figures, and other information such as crops that were grown in previous seasons.
- Cooperatives: The member cooperatives can introduce themselves, and the activities they perform. They can share information on their size, specialties, available farmer inputs, and area they operate. They might share contact information, price developments for farmer inputs, and coffee along the value chain, as well as production statistics.
- Sakaza Muhinzi: Background information on the Sakaza Muhinzi project, and the ideas behind the app, and the Sakaza Muhinze system. Information on people working on the system/maintaining the system.

The farmer views in the sitemap are divided in four subgroups; my preferences, my groups, my plots, and my maps:

- My preferences: Farmers can set some preferences, such as language, offline/online usage, remember login, stay logged-in, update frequency, and preferred map view. The farmer can also set security level, and determine, which information can be shared with whom.
- My groups: A farmer can join the cooperative he belongs to, and thereby getting information from the cooperative that is specific for the cooperative members. The farmer will be able to share information to the cooperative. Within the cooperative means, or farmer inputs, like washing station, and dry mill, or trucks, and knowledge on availability of these means, can be shared.
- My plots: The information on the plots is threefold:
    - Status and administration of the plots: The coffee species that grows on the plot, perhaps combined with other crops. The number of, percentage of, number of square meters with, and number per square meter of coffee trees. Current status of the coffee plants, and last activities performed. The farmer is offered information on elevation, slope, aspect, and soils, and relevant best practices and information on coffee species that grow best in those conditions.
    - Activities: What activity has been performed, when, how much, and on what plot. What activities are coming up, based on the coffee growing cycles, and remedy activities for plant diseases, and pests. Warnings, and announcements related to the activities.
    - History: Production figures of previous years, and information, such as coffee species, and crops that were grown in previous seasons.

- My maps: The farmer can have additional information, and perform actions, in a web map via my maps:
    - Add services: The farmer can add web services, and display these in a map
    - Selections: The farmer can make selections, set filters, in his maps, such as a specific plot, or for a specific coffee specie, or activity.
    - Red lining: The farmer can perform red lining on his maps.
    - Query land use and status: The farmer can perform analysis on the data, based on land use, and status.

The general views, and thus its content, are accessible by anyone. To get access to the farmer specific functions and information, the farmer must be identified, and

- log in to the system.

The farmer, who is logged-in, can

- join his cooperative,
- claim his plots, and
- alter his plots.

5.6 Step 4: Detailed design

The views in the sitemap, as described in the former section, are grouped per stage in Figure 5.3:

1. Webmaps, or Plots
2. Claim
3. Activities

These three stages are described in more detail in this section, in which detailed wireframes must give a good view on the apps' layout for the major views.

Figure 5.2 Wireframes Homepage, My plots, and My maps

5.6.1 Wireframes

Web maps

The first stage focuses on the basics of the mobile app, like: offline access, modularity of language, and modularity of symbols. The grand design, homepage, menus, and buttons are created. Some static content is added to the app.

The main goal of the first stage is to add web maps to the app. The web apps are added using OpenLayers in combination with the GeoJSON format. The base map is a satellite image, or Open Street Map service. Feature data, plot polygons, are added to the map, and will be set as the basic map version in My Maps. Users can zoom, and pan via OpenLayers functionality. The OpenLayers permalink control will be implemented to store the latest view of the map, see Figure 5.4 footnote. The geolocation function is implemented to update the map view to the current location, with a 10 km bounding box.

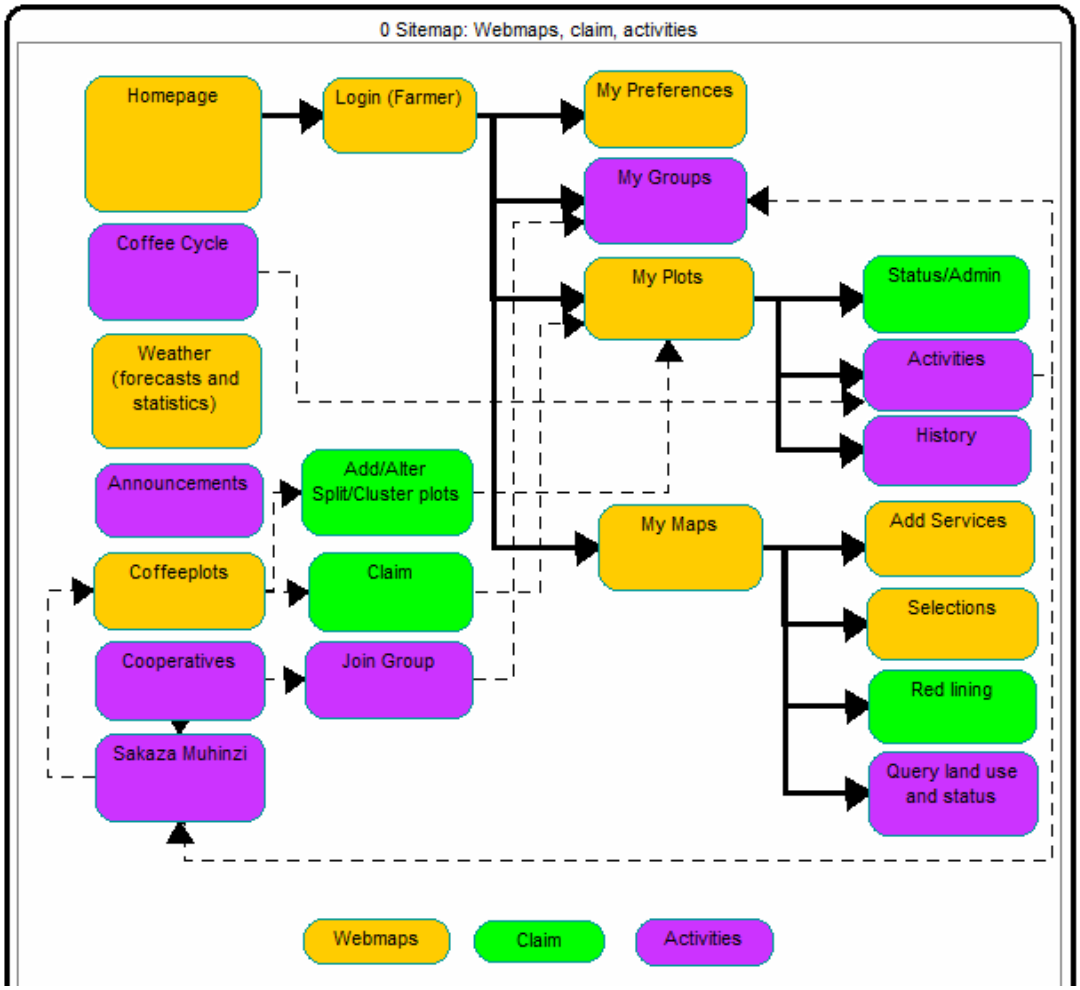

Figure 5.3 The sitemap views grouped by stage

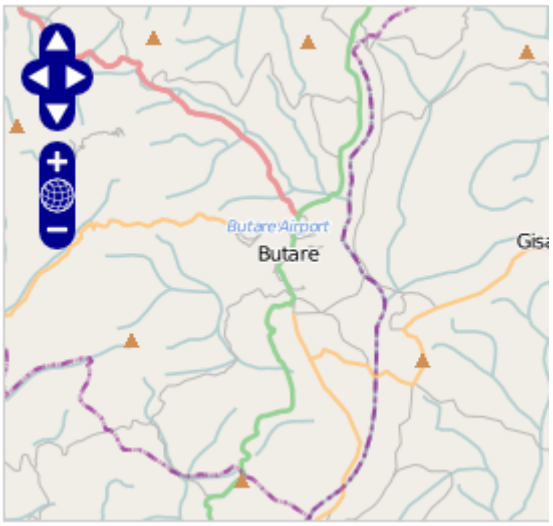

Figure 5.4 The Butare area in Open Street Map using OpenLayers Permalink[16]

The app can be personalized further by enabling login (see Figure 5.5), and store preferences, like language, and offline/online preference. Standard the language will be English.

Figure 5.5 Wireframe Homepage Log-in

The wireframes in Figure 5.2 are created before actual building of the first stage prototype. All three wireframes show personalized functions, including web maps with layers, and dynamic weather information in the homepage. The first stage prototype includes just the homepage with text, images, buttons, and links, and is extended with more advanced functions, during the prototyping phase.

[16] http://dev.openlayers.org/releases/OpenLayers-2.11/examples/anchor-permalink.html#zoom=11&lat=-2.60853&lon=29.81923&layers=B

Claim

The second stage prototype focuses on the following functions:

- Claim a plot
- Alter plot information, admin, size, position
- Red lining
- Alter polygons, cluster or split polygon, add polygons

Activities

The third stage prototype focuses on:

- Process model activities, coffee grow cycle, announcements
- Add activities, and share activities
- Periodically update background satellite image with growing cycle
- Query land use and status of plots

5.6.2 Data

The following data sets are considered in the prototyping phase:

Qualitative data on

- Plant diseases and pests
- Satellite maps, aerial photography
- Farms, plots and ownership

Quantitative data on

- Meteorological data
- Status per field

The products and services of GEONETCast will be considerd. “GEONETCast is a near real time, global network of satellite-based data dissemination systems designed to distribute space-based, air-borne and in situ data, metadata and products to diverse communities. “ (Geonetcast 2012)

The prototype can be created, based on fictive data, however real live data, especially Rwanda plot data, is nice to have, to build a more attractive prototype. Usage of frequently updated satellite image base maps may be attractive, because the background changes during growing seasons, however it might be too heavy data traffic for the Rwandan coffee farmers. Basically, the farmer only needs to download the base layer satellite image of his area once, and can use this locally stored data whenever s/he uses the app.

5.6.3 Class diagram

Eight objects are defined based on the functions of the farmer app:

- Farmer
- Cooperatives
- Plots
- Maps
- Services
- Crops (CoffeeTrees)
- (Coffee) Activities
- Announcements

A conceptual class diagram is designed (Figure 5.6) to improve the feeling with the data needed for the app, and its functions. During development the data model will evolve in a for the app applicable format.

The central web server and database are set up to host all farmers data, and also the content of the general pages. For offline access parts of this information has to be available. The content of the general pages, which will update when a farmer is connected. A farmer can see the own plot information, including administrative information and activity logging, and information on the coffee grow cycle and its activities. The farmer is able to add data offline, and synchronizes with the central data, as soon as the farmer gets connected.

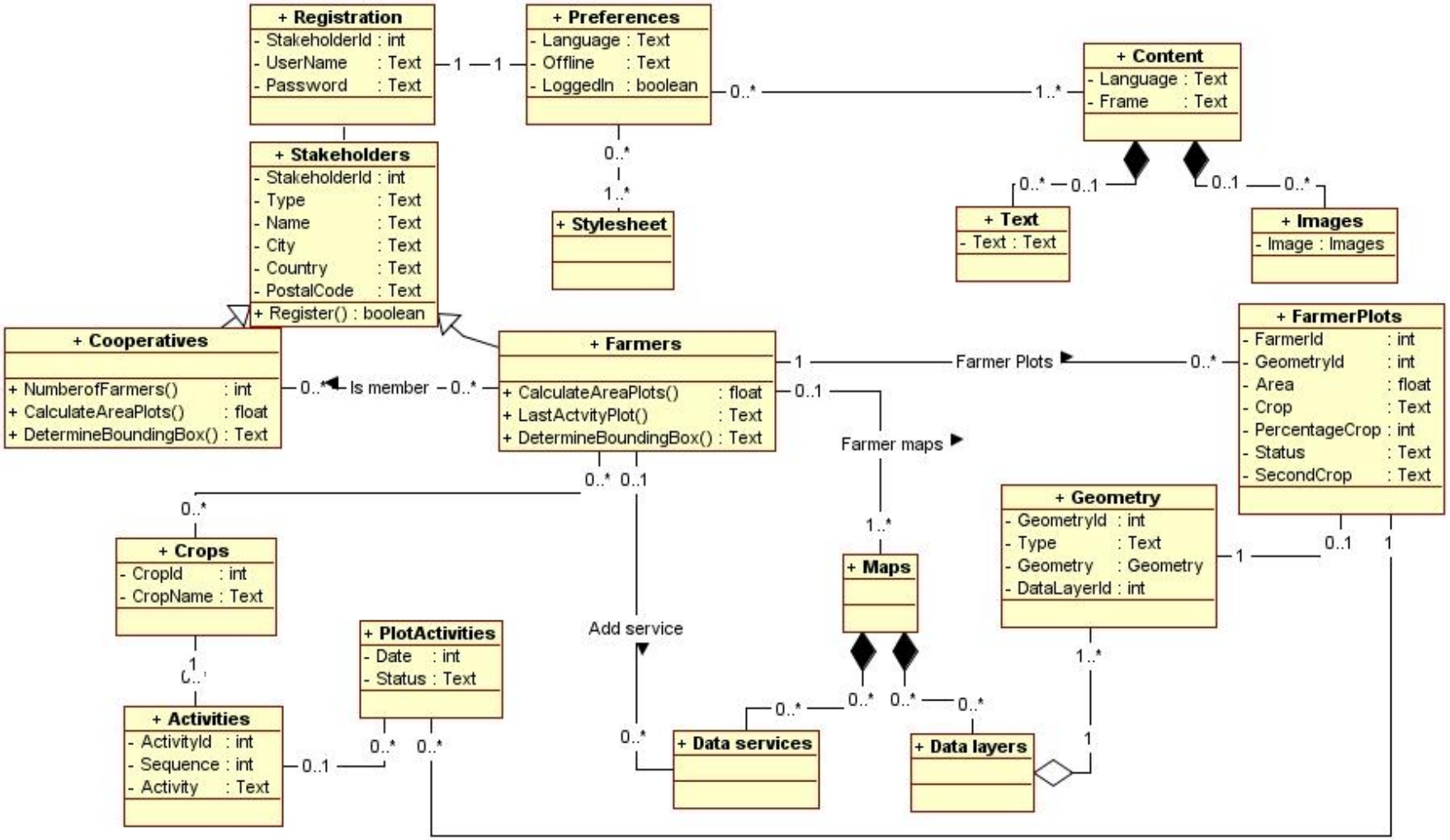

Figure 5.6 Conceptual class diagram of Rwanda coffee app

Part IV. Prototyping

Phase 3: Prototyping

The prototyping phase, see Figure 5.7, deals with input on research question 4: What data, information and functionality are available and needed, and how can these be used and reused in the application?. First a basic prototype is built, based on the use case "Presenting web maps". After testing and validation, the prototype is enhanced with positioning function, as well as the possibility to add personal spatial data.

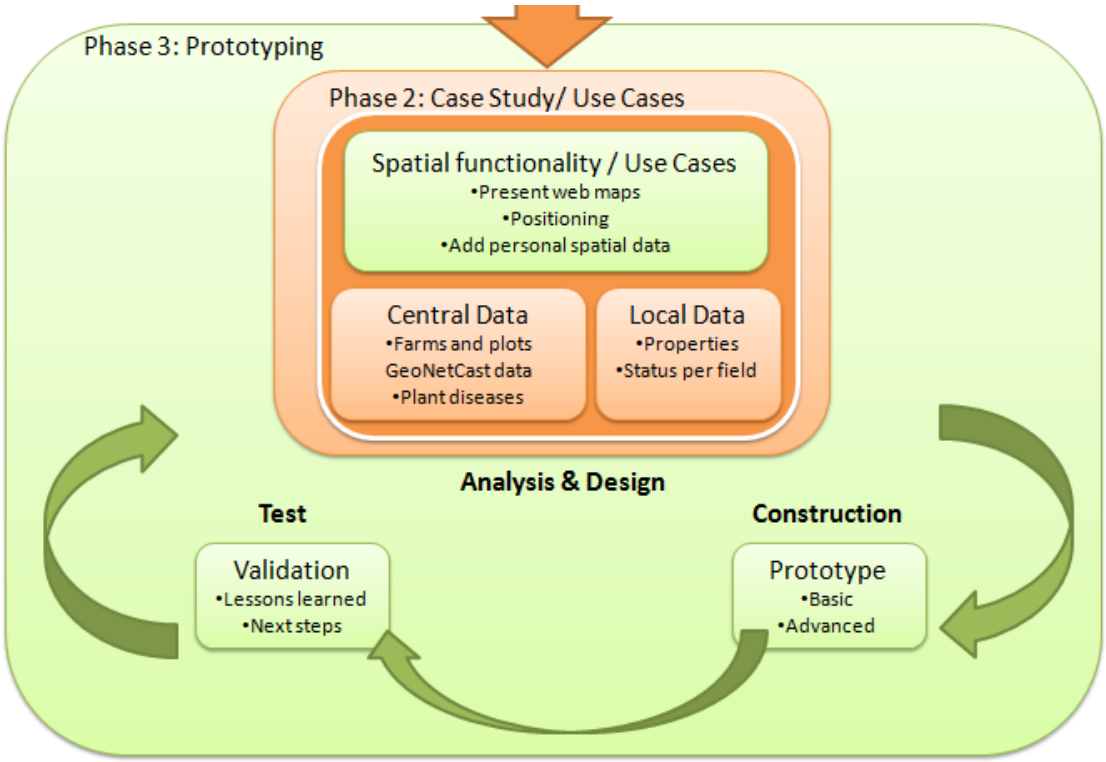

Figure 5.7 Phase 3: prototyping

6 Prototyping; the lessons learnt

6.1 Introduction

The analysis and design of the Rwanda coffee farmer app, as described in Chapter 5, are input for the prototyping phase. This chapter describes the actual prototyping phase, from initial learning and setting up the environment, to the evaluation of the developed Sakaza Muhinzi prototype. The prototyping involved three major stages, starting with implementation of some basic functions to more complex personalized, and context-aware functions:

1. Plots: Farmers access information on plots, weather statistics, and satellite images.
2. Claims: Farmers claim plots that they own within the system. No legal status or action is attached to this action.
3. Activities: Farmers inform the communal social system which activities they perform on the plots, and when.

Section 6.2 provides background information, and training materials that were used to get started to learn building mobile web apps with HTML5, JavaScript, and CSS. How to set up the prototyping environment is described in Section 6.3. Section 6.4 describes the user interface, and associated functions that are implemented in the prototype by a stepwise approach. In Section 6.5, the test results, and evaluation of the prototype are discussed in relation to the four SMAD quadrants.
In Appendix B on Prototyping, detailed information can be found on the prototyping phase.

6.2 Background and training materials

The SMAD framework aims to offer a structured method, to students in geo-science, and mobile application developers with interest for geo science, to develop platform-independent geo-enabled mobile applications. This section targets the students that are without much application development background. In this section, some of the training and background materials that can be used to learn about building HTML5, JavaScript, CSS webapps, are introduced. The proposed training starts with learning JavaScript, HTML5, and CSS basics, afterwards Twitter Bootstrap, KnockoutJS, and CouchDB are studied. A first prototype is built to get familiar with the different environments, and the scripting languages.

In the prototyping Appendix (Appendix B) a list of URLs is included that are used as training and background materials on building HTLM5 web apps using Bootstrap, KnockOutJs, CSS, CouchDB, and Couchapp. With no background in these scripting languages, and tooling, it can help to learn some of the basics. Especially knowledge on JavaScript is essential to be able to develop, reproduce, and use application logic.
Learning by doing is a friendly way to build your knowledge, as many examples can be found on the web. One starts with development of a simple website, and continues to add functions to it. Running into certain problems makes one search for information to fix it, or post one's questions on a web forum, and exchange knowledge with other developers.

P2PU university offers a study environment to learn the basics on HTML, JavaScript, and CSS. A module on "Learning HTML, JavaScript, and CSS" [17] is available. Many exercises, as well as good background information, and discussions are in the P2PU environment. Links to relevant websites are provided, for example, to the video series "JavaScript from null"[18], which get you started on JavaScript.

After learning some basic JavaScript, HTML, and CSS, concentrate on Twitter Bootstrap. The Bootstrap site[19] offers both an amount of information on the possibilities, and good examples of its functions. The Bootstrap package includes some start templates, and on Tutsplus[20], some interesting video lectures can be found, that give one a quick start building Bootstrap pages.
Bootstrap offers great user-interface possibilities using HTML, CSS, Less, and JavaScript. Learning Bootstrap is mainly about HTML, and using the right functions, and tags to build the layout that one aims for. The content-out approach of the SMAD framework, as described in Section 4.5, focuses on the user-interface, and that is exactly what can be build using Bootstrap. The logic behind the site does not have to be developed, before starting the development of one's user-interface.

The file structure of the first prototype (prototype 0), starts with only Bootstrap files (Table 6.1)

Table 6.1 File structure of prototype 0, Bootstrap only

Directory	Description
/assets	Container with all necessary assets
*** /css**	The Bootstrap CSS files
*** /ico**	Bootrap icon
*** /img**	Some images from the examples
***/examples**	Example pages
***/example-sites**	Example sites
***/glyphicons**	Icons used in examples
*** /js**	Bootstrap JavaScript, JQuery plugin functions
***/Google-code-prettify**	Prettify Google code
/less	LESS variables that can be set
/templates	Mustache layout template
*** /pages**	Mustache files

The next step is to add some application logic, and learn more about KnockOutJS. The KnockOut tutorials[21] are a good starting point. After going through the tutorials, some example functions can be included in the prototype. First function to add is a to-do list, or task list. This example shows how to fetch stored data from file, or database, update, add, or remove data, and store the changed data. The tasks example includes importing task data in JSON format from a server. In the Sakaza Muhinzi farmer app, this function, in an adjusted format, is used to alter plot attribute data, and add activities to one or more plots.

[17] http://p2pu.org/en/groups/javascript-101
[18] http://net.tutsplus.com/tutorials/javascript-ajax/javascript-from-null-video-series/
[19] http://twitter.github.com/bootstrap/
[20] http://webdesign.tutsplus.com/tutorials
[21] http://learn.knockoutjs.com/

KnockOutJs is build on the Model-View-ViewModel principle, see also Section 9A.4 on tooling. The ViewModels, which are JavaScript files, contain the background function that prepares data from the database or Model, in such a format, it can be easily presented in one's front-end user-interface or View. The SMAD framework aims to offer reusable modular pieces of code that can be implemented in different apps. Therefore, it is necessary that more ViewModels can be included in one View, or HTML page. This can be tested by the test case to have multiple ViewModels in one page.
Having multiple functions available from one HTML-page, has the advantage of combining information, and logic, but above all that certain scripts, and data, is only loaded once, entering the specific page. The user is not confronted with unnecessary waiting times switching to another function within the app.
All JavaScript files are put in the " /Assets/js" directory (Table 6.2).

Final element in the app structure is the CouchDB combined with CouchApp. Tutorials and information are found at Tutsplus[22], and Custardbelly[23]. CouchDB is not a relational database which you query with SQL. CouchDB is a NoSQL databases, having data stored in JSON documents. CouchApp serves the web app from the CouchDB, and can fire HTTP requests to the database.

The final Prototype 0 file structure, which is a list of resources to compile a CouchApp, including Bootstrap and KnockOutJs, is depicted in Table 6.2. This Prototype 0 structure, extended with the OpenLayers framework, forms the basis for building the Sakaza Muhinzi prototype. In the next section, as well as in Appendix B, is described, which steps need to be taken to set up the environment, and build the prototype.

Table 6.2 The CouchApp file structure, including Bootstrap, and KnockOutJs

Directory	Description
/_attachments	HTML files/pages
***/img**	Images used in app
***/style**	Bootstrap CSS files
/evently + /sub directories	Track changes, data.js, mustache.js, query.json
/lists	CouchDB lists used in app
/shows	CouchDB shows used in app
/updates	CouchDB updates used in app
/vendor	Container vendor
*** /couchapp**	Container Couchapp
***/_attachments**	Couchapp JQuery JavaScript
***/assets/js**	Bootstrap, and KnockOut JavaScript
***/evently**	Log in/out functions
***/lib**	Couchtapp/CouchDB JavaScript
/views	CouchDB views used in app

[22] http://net.tutsplus.com/tutorials/getting-started-with-couchdb
[23] http://custardbelly.com/blog/2010/12/08/jquery-mobile-couchdb-part-1-getting-started/

6.3 Setting up the environment

This section gives a summary of steps to set up the environment. A detailed description of setting up of the prototype environment is in the prototype appendix (Section B.2).

The prototype was realized on a Windows 7, 64 bits machine. In short the following steps must be completed:

- Install a text editor, such as CoffeeCup Free, and/or Notepad ++
- Use Firefox as main browser to test with, and include
    - Add on Firebug
    - Add on Web Developer
- Set up TwitterBootstrap
    - Include example HTMLs, and Responsive layout
- Set up KnockOutjs
- Set up CouchDB and CouchApp
    - Get GeoCouch via IrisCouch
- Set up OpenLayers
    - Include Google Maps base layers

This environment forms the basis to start the Sakaza Muhinzi case study prototyping, in which the following three function areas are aimed for:

1. Plots: Farmers access information on plots, weather statistics, and satellite images.
2. Claims: Farmers claim plots that they own within the system. No legal status or action is attached to this action.
3. Activities: Farmers inform the communal social system which activities they perform on the plots, and when.

Section 6.4 describes the user interface, and associated functions that are implemented in the Sakaza Muhinzi farmer prototype. In Section 6.5 the test results, and evaluation of the prototype are discussed.

6.4 The prototype, user interface, and implemented functions

Development of an app, by a prototyping approach, as proposed in the SMAD framework, helps to get fast results. A commonly heard saying in Business Intelligence is to “think big, start small”, which is also appropriate in building a web app. A basic app is more or less a simple website that is available in a mobile format. Having the right content, structure, navigation, and lay-out, is the first step. In the design and analysis phase, the ideas for the more advanced app functions, are described, and offer guidelines during the first steps in the actual prototyping.

The SMAD strategy to build the app is based on the content-out approach. Design the front-end/ user-interface of the app, and continue to add functions, and add certain behaviors to the app by application logic in the back-end.

The following stepwise approach can help building a web app, with geo-functions:

1. Choose a Bootstrap template to start with
2. Create the basic public content
3. Realize the page navigation
4. Choose an appropriate lay-out, and styling
5. Include widgets, and other third party services
6. Add simple data functions
7. Add web maps
8. Add more complex data functions

This stepwise approach does not take into account the general processes as defined in the technical quadrant of the SMAD framework (Section 4.2): security, logging, and caching. These are basic functions which are to be implemented by a to be defined strategy. Different mechanisms are available, however these are not investigated during this thesis research. These general functions are categorized in the "Categorization of functions" quadrant, described in Section 4.3. At the level of the personalized application, part of the data may be private, and log in facilities will be offered. By logging the activities in the database, one can always trace, which user has performed activities on the data. Caching of data becomes necessary if one want to perform analysis on data, and larger amounts of data are fetched. A basic form of caching is the offline storage function, which can make the app work in offline modus. This function is classified as part of the basic app level.

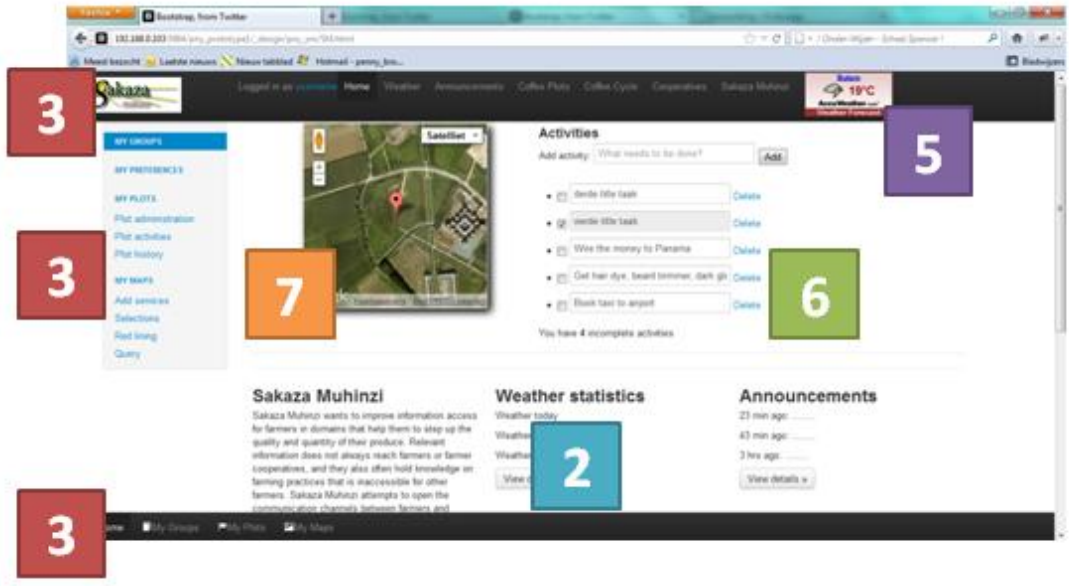

Figure 6.1 The user-interface elements relating to the stepwise prototyping approach

The following sections describe the stepwise prototyping approach, illustrated by the Sakaza Muhinzi farmer app (See Figure 6.1).

6.4.1 Choose a Bootstrap template to start with

The Twitter Bootstrap framework offers a number of templates to start building a web page using Bootstrap. The Bootstrap framework is not a mobile HTML framework; it is a general HTML framework offering mobile functions. The framework offers functions to serve your information to both pc, tablet, and smartphones. The same page logic is used to present the information on different screen sizes.

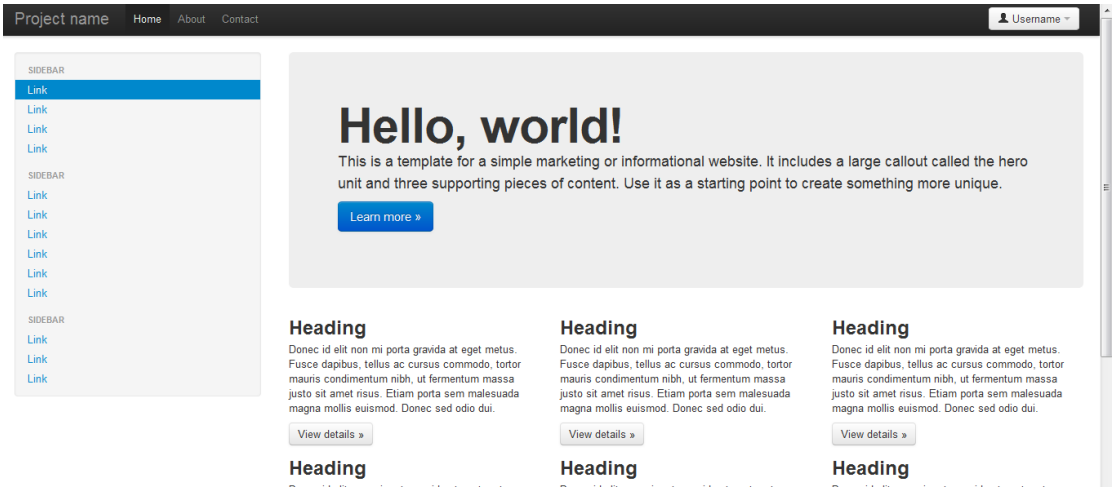

Figure 6.2 The Bootstrap fluid example template on laptop screen

The prototype's user-interface is based on the fluid Bootstrap example,[24] see Figure 6.2, and Figure 6.3. This example forms the basis, because it includes the so-called fluid, or responsive behavior to serve the information to the different devices. See also Section 3.2.4, and the Audiovroom.com example.

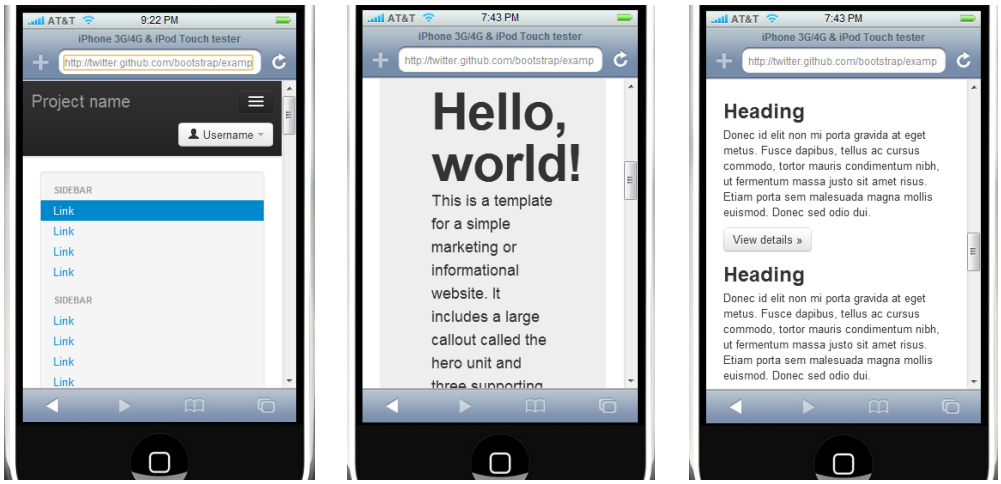

Figure 6.3 The Bootstrap fluid example on mobile (Iphonetester.com)

[24] http://twitter.github.com/bootstrap/examples/fluid.html

An application should run on different devices, including bigger, and smaller screens. Using the Bootstrap functions one can make the application adapt the lay-out to all kind of screens. In Figure 6.2, and Figure 6.3, see for example the "Hello world" section. The "Hello world" section takes almost the complete width of the larger screen, and shows the text in only two lines, as on the smaller smartphone screen the whole width is taken by the section, as well as the height of the screen, with fourteen lines of text.

The fluid template offers page navigation by menu bars, and different content blocks by which the own content can be made available.

6.4.2 Create the basic public content

Part of the information within an app is (semi-)static information, or content, as some other parts of the data are updated frequently. The update frequency depends on the information, and its usage. Many web sites are based on a Content Management System, to publish its information to the public. Different parts of the information can be maintained by different people. Information with a high update frequency will be automated content, and is not maintained by hand.
For the Sakaza Muhinzi app, the content is not developed. Based on the case study, and input from the project, a design of the app's homepage was created, containing sections for weather, announcements, and coffee cycle, among others (see Figure 6.4). Such sections will contain relevant information, and links, on these subjects. Some information can be hidden for mobile users, if the information is thought not to be relevant for the app's users. For example, the prototype includes an introduction of the Sakaza Muhinzi project that is hidden on the mobile. The mobile users, are the farmers that subscribed to the app, and they are already informed about the Sakaza Muhinzi project.

Figure 6.4 Sections of the Sakaza Muhinzi app's homepage

Weather statistics and announcements are examples of information that is updated frequently. Application logic is needed to have up-to-date information included. Weather information can be retained from weather sites, such as AccuWeather[25], for example, and announcements can be included by, for example, having Twitter[26] or other social media functions. Within a project it can also be decided to build one's own information services, which can be included in the app.

[25] http://www.accuweather.com/
[26] http://twitter.com/

6.4.3 Realize the page navigation

As with a normal web page, also in a web app, page navigation is available to navigate through the app, and access the available functions. During the analysis and design phase the grand design for the navigation is described in a site map, see Section 5.5. This site map can be implemented by creating the navigation menus.

The Bootstrap fluid template has two navigation bars included: a top bar, and a side bar, which is at the left side of the pc/laptop screen, and below the top bar on a mobile screen (Figure 6.3).

The Bootstrap fluid navigation bars are sensible navigation bars for page navigation on pc or laptop, however for the mobile a third navigation bar at the bottom of the mobile screen is introduced. The standard top bar gives a collapsed menu icon (see Figure 6.5) on the smartphone, which makes a user have to click an extra time going through the menu's. The smartphone user will only have the third navigation bar available, which offers comparable navigation as the standard top bar navigation. Navigation through the page can be performed by using the touchscreen function instead of via the sidebar or extended top bar menu.

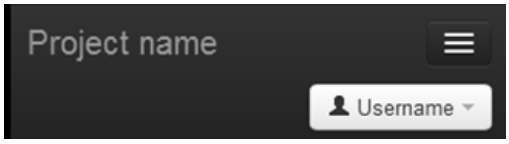

Figure 6.5 The standard Bootrap top toolbar on a smartphone

The web app is designed to run on all devices, and therefore no specific platform code is included to have access to the various smartphone menus. All navigation functions will be available on screen. In line with the design, a four button drop-down(/drop-up) bar is developed, and added (see Figure 6.6). These drop-down buttons aims to improve the user's navigation experience: functions and information is on-screen with a simple click, without having to browse through more than one navigation dropdown layer.

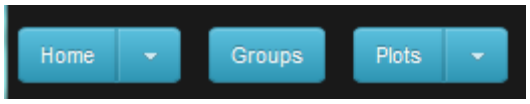

Figure 6.6 Smartphone navigation via drop-down buttons

6.4.4 Choose an appropriate lay- out, and styling

A specifically developed lay-out and styling is important to give the app its own identity. The styling can be in-line with the organization style, the company logo, or have its own specific styling. The Bootstrap basic styling (Figure 6.2) is neat, and functional, however it does not add anything special to the app's looks, and is not too appealing for the potential users.

Bootstrap offers a customized Bootstrap configuration file[27]. By means of setting some of the variables to have certain coloring, styling, or font, the total looks of the page/app can be changed.

[27] http://twitter.github.com/bootstrap/download.html

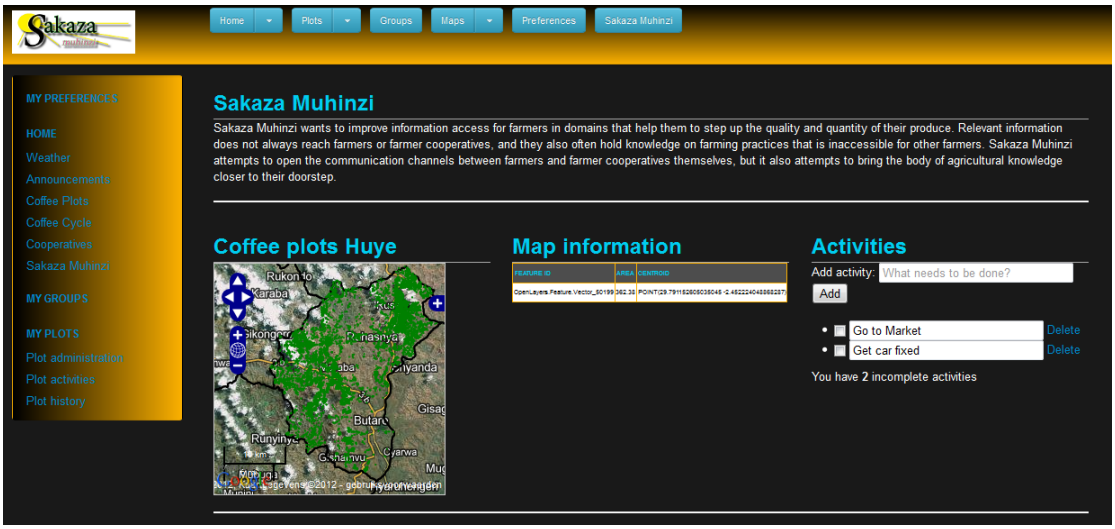

Figure 6.7 Prototype; Screendump SM2.html on laptop

An important change in prototype style is the black background, and white main text, see Figure 6.7. The reason to change this styling is two-fold: (1) a white background means more intense light on a bigger part of the screen, and thus a battery that is exhausted more quickly, and (2) a black background is felt to be more stylish, however that is a personal view, on which others may disagree. In the end the customer that asks for the app, should have a saying in the final styling. The app is built to be used in the Huye district in Rwanda, and shoud have good contrast to read the screen in a sunny environment. User tests can be used to choose against white text on a black background, or black text on a white background.

6.4.5 Include widgets, and other third party services

The HTML, JavaScript, CSS app structure is suitable to include different kind of web services and widgets. Logic and data that resides anywhere, can be made available within the app page. During the analysis and design phase, it must become clear what kind of information can be included using widgets. Also information from the own organization can be made available via widgets. Widgets and third party services can remotely process the data, and in that way enlighten the app, and the smartphone's processor. The app only shows the resulting information from the widget, according to the settings.

In the Sakaza Muhinzi prototype an AccuWeather widget is included. This widget shows the current temperature, and weather conditions in Butare in Rwanda, and provides a link[28] to the AccuWeather page for more detailed information, and forecasts. The widget should be configured to provide the weather information close to the farmer's coffee plots.

[28] http://www.accuweather.com/en/rw/butare/295612/weather-forecast/295612?partner=netweather&unit=c

6.4.6 Add simple data functions

Before including web maps in the app, it is recommended, to start with some basic data functions. These functions are at the level of the personal app. In the technology architecture of the SMAD framework (Section A.4 on tooling), the KnockOutJS library is suggested for development of the application logic.

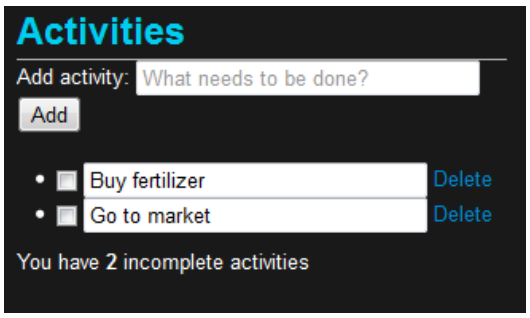

Figure 6.8 The KnockOutJS task list example

The task list example of KnockOutJS[29], see Figure 6.8, is an interesting case to include in the app. It is not the most simple example, however it shows functions that are often used in apps. Stored data is read into the page, and after making the adjustments, data is stored again. The data is a list of items (tasks to be completed), with only two attributes: name, and isDone. This basic application logic can be used entering activities to a plot, changing attribute data of a plot, and actually claiming a plot.
The code to change attribute data of a plot is provided in Appendix C, and explained in Section 4.5.5. To enter activities to a plot, the model has to be extended with the applicable attributes. To minimize the actual typing drop down lists can be created for items, such as status, and type of activity.

6.4.7 Add web maps

An important geo-function, is the presentation of information in web maps. The web map is the basis on which other geo functions rely. The SMAD framework presents OpenLayers as the framework to show information on maps, and provide functions like zooming, panning, and selection of items. If one is used to OpenLayers, the integration of OpenLayers in the web app is nothing special. In case the background in OpenLayers is missing, the website[30] helps out with many examples, on different kinds of function, such as: geolocation, accelerator, geoJSON data, select feature, and many others.
The OpenLayers examples that are of interest for the Sakaza Muhinzi app, are listed in the Appendix on prototyping. In the design and analysis phase of the project (see Section 5.2), the need for web maps is indicated. Farmers should be able to view their property on a web map. In general the web map can be used as a powerful means to present information. In a later phase when more data is available, and presented, a farmer can learn from other farmers, and information available. A web map can be one of the means to properly present the information, in such a way a farmer can deduct the information s/he is looking for.

[29] http://learn.knockoutjs.com/#/?tutorial=loadingsaving
[30] http://openlayers.org/dev/examples/

For applications, such as the Sakaza Muhinzi farmer app, a high detail satellite image base map, is preferred above base layers with not much detail. The coffee plot information is presented as a start to have the farmer claim the plots, add attribute data to the plots, and add activities to the plot. For these type of functions the base layer does not need to contain much information. The attractiveness of a high detail satellite image helps to recognize the own environment, and plots, and gives the farmer an idea of the surroundings other coffee farmers grow their coffee.

Google Maps offers detail satellite images via the Google Maps API. The Google Maps API is not available for free, depending on the type of usage, and amount of usage. For local prototyping, and testing, the Google Maps API is available to use without any costs.
The Google Maps layers are chosen as base maps, in OpenLayers, as other satellite images were not available during the prototyping phase. The following Google layers are included in the prototype: Google Satellite, Google Streets, Google Hybrid, and Google Physics. These layers offer detailed satellite images of the Rwanda Southern District, and are available online. Any other map service serving this kind of information can also be incorporated, for example a service created for this project based on available images. Within each project that include web maps in an app, one should examine the requirements of the map, including the base layer.

OpenLayers is offered in a standard style, with controls for zooming, and panning, a scale line, and an OpenLayers control is added to select the base maps, and other map layers available. Figure 6.9 shows the available layers in prototype file SM2.html. The hybrid layer is selected as standard base layer. In Figure 6.10 the physical layer is selected as base layer, and is zoomed to the Maraba area.

Figure 6.9 OpenLayers with Google Maps v2 base layers

In the SM2.html map, two additional layers, overlays, were added, besides the Google Maps base layers: Huye coffeeplots, and South district (see Figure 6.9). The South district lines are depicted in black, and the coffee plot polygons are displayed in green.

Both the plot data, and the district data are spatially referenced in EPSG:4326, the same projection Google Maps API v2 uses. OpenLayers can also combine data with other spatial references. The data is in GeoJSON format. The GeoJSON format is a format in which GeoCouch stores the data, and which can be served up by the CouchDB over HTTP.

Figure 6.10 Huye plots in OpenLayers with Google Physical base layer

The district lines (available as polygons) are in the following format:

```
{"type":"FeatureCollection",
   "features":  [
      {"type":"Feature",
        "geometry":
            {"type":"Polygon",
                 "coordinates":[[[29.6660058732595,-
                 2.41859726382248],[29.6664286436893,-2.41850291120367],
                 [……], [29.6660058732595,-2.41859726382248]]]},
                 "properties":{"PROV":"SUD","NOMDISTR":"HUYE","AREA_KM_":581
                 .52703544,"DISTR_ID":204,"POP_02":265446,"DISTR_NAME":"Huye
                 ","PERIMETER":0}}
]};
```

The Huye plots are in the same format as the district lines, except for the properties of the geometry:

```
"properties":{"distr_name":"Huye","id_plot":"shuye5582"}
```

Both maps in SM2.html and Plots.html, have selectable features. To allow editing of attribute data of a plot, the first step is to select the plot, and fetch the available information. The OpenLayers map in Plots.html is zoomed to the plots of the current user, Farmer A. Her/his four plots are indicated in blue in Figure 6.12, and when selected they turn white. The coffee plot layer has green polygons, which turn red when selected (Figure 6.11).

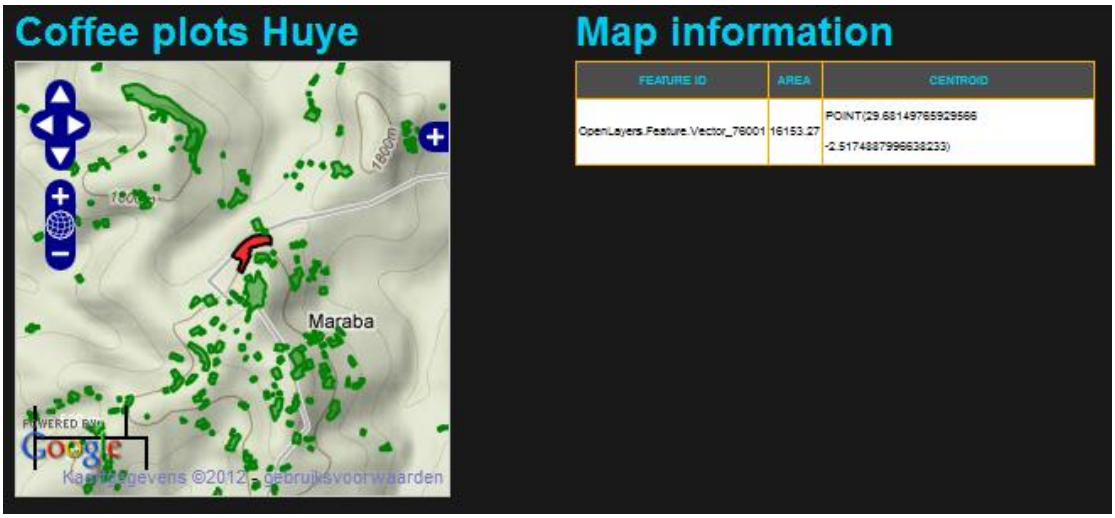

Figure 6.11 Map information selected plot on Google physical base layers

Selection of a plot of the Coffee plots layer returns feature information, as in Figure 6.11, under "Map information". It displays one the feature ids assigned by OpenLayers, and the calculated area size (m^2), and centroid of the selected polygon. The coordinates of the centroid are shown to show the OpenLayers function to calculate the centroid of a polygon.

A farmer who selects her/his own plot in the Plots page, receives additional information on the selected feature, as in Figure 6.12 under Map information. Plot data in Figure 6.12 was a first version of the plots data set, which showed the plots not properly projected on the Google Maps layers. As a test case in prototype Stage 2, the properties Plotname, Owner, and a fixed string Attribute are displayed, which were added to the properties of the farmer plots. Application logic is needed for the farmer to be able to edit his coffee plot properties. A latter example shows a more mature version of the map information section.

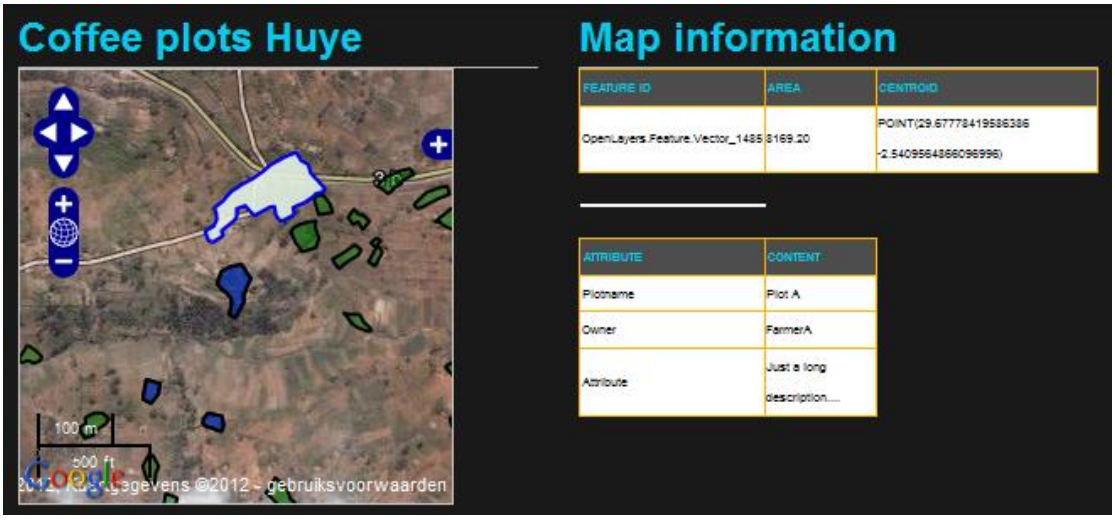

Figure 6.12 Map information of the farmer A plots in Plots.html

6.4.8 Add more complex data functions

Besides the simple data functions added in Step 6, no real application logic is developed to change or add data within the app. Most functions aim to present certain information. The last step in the development of the app is to add more advanced logic. The context-aware functions to support collaboration, or add analysis functions to the app, ask for more analysis and development skills. It may ask to combine different frameworks, and understanding new technologies. However, the process to implement more complex functions, is the same as implementing the basic functions.

In Plots.html, a piece of KnockOut script is included to have the farmer plot attribute data in a table, in which the values of the properties can be changed (see Figure 6.13). The feature data is read from the database, and attribute data is displayed per plot in a table row on-screen. Except for the plot identifier, all fields are editable.

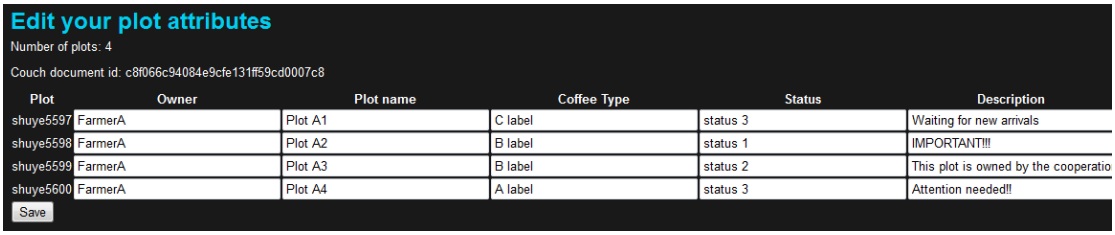

Figure 6.13 Edit plot data in Plots.html

The "Save" button contains logic to save the updated data back to the database, see Section 4.5.5, "Ad 6. Add simple data functions", for details. Next time the page is loaded, the last saved data is fetched, and shown.

The "Edit your plot attributes" function on a smartphone looks like the screendump in Figure 6.14. Depending on browser and phone, scrollbars are shown, or navigation must take place without scrollbars using the touch screen functions. A keyboard is served from the smartphone operating system.

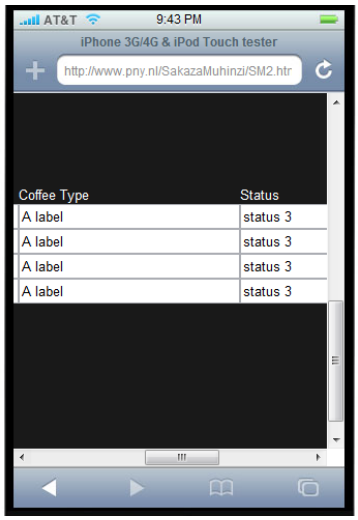

Figure 6.14 Edit the attribute data on a phone screen

Other more advanced logic that is developed, is to fetch the attribute data of the selected plot, and be able to change the attribute data of that specific plot (Figure 6.15). The OpenLayers plot id, is changed into the unique plot id from the GeoJSON feature data.

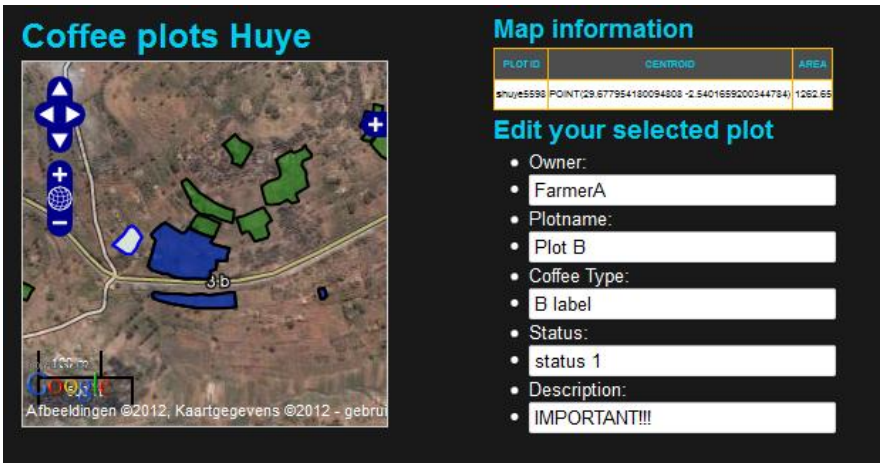

Figure 6.15 Edit attribute data of selected farmer plot

6.5 Testing, and evaluation of the prototype

This chapter on prototyping ends with lessons learned, based on the testing, and evaluation of the prototype. The lessons learned are grouped by the quadrants of the SMAD framework: (I) technical architecture, (II) categorization of functions, (III) project approach, and (IV) design and development (See Figure 2.1in Section 2.2)

6.5.1 Technical architecture findings

The architecture, as defined in Section 2.2, presents the tools and techniques, and its relationships, to be able to develop the app in a modular manner, and to profit of reusable objects and code. In Section 4.2 the MVVM architecture is explained, and tooling to build the geo-enabled apps were presented.

MVVM

The main reason for development with the MVVM architecture, is the clear overview of the different parts of the app logic. Distinction between the front-end coding, and the back-end coding helps to keep the overall picture, as well as a clear distinction between the HTML, JavaScript, and CSS components.

Creation of new functions, within an app, let one start in a greenfield. No scripting or coding from other functions that needs to be incorporated, or adjusted. A new script can be created with a clear distinction between the HTML structure of the page, the CSS styling, and the JavaScript logic. However, working with functions, based on other frameworks, other applications, and examples on functions, do not always make this distinction. In the OpenLayers JavaScript logic, for example, the styling of the selected plots, is included in the JavaScript, and not separated in a CSS style file. This does not necessary mean it is not possible to separate styling, and application logic, however it does mean additional programming is necessary to accomplish this. For various OpenLayers examples that are used in the prototype, and includes JavaScript logic in the HTML file, the JavaScript logic is put in separate files. This is done to get the distinction between back-end application logic, and the front-end layout and styling.

Combining frameworks

In combining functions from different frameworks, other CSS stylesheets are introduced in one's environment. Bootstrap offers CSS styling for the more standard layout such as headers, and paragraphs, and its own specific styling elements, such as the navigation bar, and fluid containers. The OpenLayers styling offers styling elements specific for the OpenLayers maps and functions, and the OpenLayers examples have additional styling for general elements such as headers, and paragraphs. Developing an app, one should be aware of these overlaps, and make sure unnecessary style elements from other frameworks are left out in the development environment. Besides the framework specific styling CSS files, one project specific CSS file can be created to add additional styling that is not in the framework CSS files. To overwrite styling within the framework styling elements, it is possible to have the element styling in the header of the HTML file. This is not in-line with the principle of the SMAD framework to split HTML, JavaScript, and CSS, however it is a work-around to have the layout in-line with one's design, as it overwrites styling from the CSS stylesheets.

Tools

The second part in the technology architecture quadrant is the tooling to present information, create the application logic, store and use data via the data model, and set up the support processes. The main tools used in the framework are TwitterBootstrap, KnockOutJS, CouchDB with CouchApp, and OpenLayers for the basic geo-functions.

Bootstrap

Bootstrap, in the current version, offers some interesting styling functions to make the page responsive, and attractive on both smartphone, and pc screen. However, if the need for styling is not available in Bootstrap, it is wise to search for alternatives, or accept the discrepancies of the design with the actual product delivered. Bootstrap's focus seems to lie in serving web pages to pc screen format, and secondly, making these pages also suitable for smaller screen sizes. If the main focus is on mobile web apps, which is the case in this thesis, it is difficult to have the mobile lay-out fit completely with the ideas for the mobile app's structure and navigation. Nonetheless, looking at the case study, it is a great advantage, to have the web app available for different devices. Some functions, such as updating the plot attribute information, is much friendlier on a pc having a keyboard available, and the information displayed without having to scroll. Typing on a smartphone is possible with the keyboard function provided, however it is merely to enter short texts. The prototype gives farmers having an internet pc the possibility to use it. And, another big advantage, is, there is no need to maintain two or more user interfaces, for the various screen sizes.

KnockOutJS

The selected back-end logic tooling is KnockOutJS, a JavaScript framework. Besides KnockOutJS other JavaScript frameworks are used during prototyping. Some functions are from the JQuery library, and part of the functions were implemented by pure JavaScript. KnockOutJS is a framework that offers JavaScript functions, in a shorter format. The number of lines of code needed to accomplish certain logic using KnockOut is much lower, than having to write the same code using plain JavaScript. KnockOut has predefined functions available that can be called, without having to write the whole function logic oneself.

The function to update the farmer plot attribute data is worked out with KnockOutJS. The ViewModel, in this function, is defined as the farmer plot information served by CouchDB, including the document id from the database. In HTML, code is added to show the information to the user, and to serve the data in input fields that allow for updating. The changes in the attributes are registered, and logged in the background without additional coding, and the save button logic sends the latest status back to the server.
The attribute ViewModel was successfully integrated in an OpenLayers JavaScript to serve GeoJSON features in a web map. The attribute data from the database are only fetched once, to serve both the OpenLayers Map, and the update of attribute data function.

When building applications in which other frameworks, such as OpenLayers, are integrated, the knowledge of JavaScript becomes more relevant. One should start with the basis of JavaScript, and understand the behaviors of JavaScript elements. CouchApp and Bootstrap use some JQuery functions, besides plain JavaScript. To fetch JSON data from the server, by JQuery for example, a nice function is offered to structure one's HTTP request correctly, by the function: $.getJSON.

CouchDB, and CouchApp

As mentioned above, the data is stored using CouchDB. CouchApp is introduced to serve the data, and the app pages. CouchDB is a NoSQL database, in which data is stored in JSON documents. Each document has a revision id, and if a change in the document is saved to the database, the old status is still in the database under the former revision number. This process thus, is a kind of logging process. At all times, a former status of the data can be retrieved. Creating the data model based on CouchDB, is quite different from creating a data model for a relational SQL database. During the "analysis and design" phase the conceptual data model was more or less a model based on relational databases, following normalization rules. During prototyping the data model evolved into a format more suitable for the app in development. The final data model contained more combined data in one document. Farmer information is not combined in one document for all farmers, however per farmer a document is created with farmer information, including information on his plots.

CouchDB offers functions to set up users, and their restrictions. This part of security is not implemented in the prototype. It is necessary to use security, and, for example, give users only access to update their own plot information. Depending on the data, and the level of privacy asked for, certain data should be available for all, and other data should be restricted to an individual. In the prototype, the farmer will only retrieve his own plots to edit.

CouchApp is an add-on for CouchDB that can be used to serve your application, and have easy access to the database. A CouchApp has its own structure of files that are generated into a CouchApp. This CouchApp is a design document in one's CouchDB, having attached all different files in the CouchApp structure. A CouchApp can easily be exchanged between different CouchDBs. Integration of different frameworks within the CouchApp is possible. Bootstrap, KnockOutJS, and OpenLayers were included in the prototype's CouchApp structure.

OpenLayers

Last tool, discussed in the evaluation, is the OpenLayers framework. OpenLayers is an open framework based on pure JavaScript functions, which offers web map functionality. It offers functions to show ones data on a map, but also have data from other sources. To have the map in a mobile app, the canvas on which the map is published, should fit the height and width of the smartphone screen, and thus shows up relatively small on a larger screen.

During the prototyping phase, the satellite images, of the Huye district, were not yet available. At first the plot data was published on OpenStreetMaps, and OpenLayers wms base layers. A satellite image, however is preferred, as it is much prettier and contains additional information, not available in those other two layers.

Both Google API v2 and v3 are investigated. The version 3 uses EPSG:900913 as projection system, in which the plot and district data is available, which is however also available in EPSG: 4326 projection, the standard of the v2 API. The coordinates in EPSG:900913, however did not show coordinates in Rwanda, but somewhere in the middle of the ocean. This could be due to the data being transformed, or the script, and projections, not being correctly implemented, or a reported bug with the OpenLayers version used in combination with the version 3 API.
The v3 API has alerts popping up, showing the source of the map data, with every action performed on the map, including panning, and zooming.
The Google Maps Javascript API Version 2 has been officially deprecated as of May 19, 2010, and is under Google's depreciation policy, which means the code must be updated to the Version 3, if the maps are used as base layer in the future.

6.5.2 Categorization of functions

In the "Categorization of functions" quadrant of the SMAD framework, functions are categorized on how the data is used, and what kinds of process are facilitated. The functions are related to the tools and techniques to successfully integrate these functions in the app.

Based on this categorization, the phases of prototyping were defined during analysis and design (Section 5.3). Start with the basis, a simple app, continue with more personalized functions, and finally develop the content-aware functions, see Table 5.2 in Section 5.4. The stepwise approach, described in Section 6.4, follows this same pattern; start with the simple functions, and subsequently develop the more complex functions.

Working this way, is an instructive way of development. The learning curve of the different frameworks is quite steep, and working from simple to complex leverages this. The simple functions are more general ones, which can be implemented by default functions, such as security, logging, and offline usage. The first stage is more or less comparable with the development of a basic web page, in which static information is published. The true coding comes with adding personalized functions, such as web apps, and positioning, and the more complex context-aware functions.

6.5.3 Project approach

The "project approach" quadrant presents a method to manage a SMAD project, building geo-enabled, platform-independent apps. This approach is appropriate for developers with experience in this field of development, and with knowledge of the different scripting languages: HTML, JavaScript, and CSS. Planning a relative complex project without the needed experience and background is difficult.

The prototyping phase showed that the case study includes many functions that could be implemented within the prototype. Working on such an app would preferably be accomplished by more than one person. One developer should focus more on the application layer and logic, as the other developer should focus on the user-interface, and the right requirements.

The project approach includes mechanisms to steer the project, based on the temporary results: continuous testing, and iterative planning. However in a one-person project, with certain functions in an app as the deliverable, the possibilities to steer the project are few. Or the deadlines are postponed, or the final product does not contain the functions aimed for.

6.5.4 Design and development

The last quadrant, the "Design and development" quadrant, offers the process and techniques/templates to design and build the apps. Techniques of web development, and mobile app development are used. Analysis and testing are also part of the design and development quadrant.

The SMAD design and development process diagram in Section 4.5 shows the different stages in development of an app. The general introduction on the Sakaza Muhinzi project, followed by the points of attention of the project, categorization of functions, priority list, high level overview, detailed wireframes, and conceptual class diagram form the basis to build the app. During prototyping the details can be refined, and adjustments can be made where necessary. Although the data model in the conceptual class diagram is based on relational SQL databases, and data driven applications, it still gives handles to the developers to understand what data is to be included in the app, and how certain data relates to other data. The diagram, thus, does not translate into tables for the various entities. For example, farmer data of all farmers is not stored in one Couch document, but for each farmer a document exists with "document type = farmer" on which his preferences can be stored, and which contains the geometry and attributes of his plots.

Testing, and debugging

During the prototyping phase, testing, and debugging, is a means to have the code running, and realize the necessary functions, to run without generating errors. Firefox, and the add-ons Web developer, and Firebug, offer the necessary tooling to debug, and test. HTML, CSS, and JavaScript can be evaluated separately or together. HTTP requests can be analyzed, as well as the DOM structure. The styling of an element on the page, for example, can be analyzed, or inspected in Firebug, which shows the CSS styling on the element. In JavaScript, the assignment "console.log()" can be used to display the value of a certain object in Firebug, while running the app. This is of great help when debugging the code.
During prototyping the errors logged in Firebug can help to find problems in the code. In the Sakaza Muhinzi prototype, for example, a problem was solved by upgrading the JQuery version, as CouchApp used an older library of JQuery than the KnockOutJS script.

Sakaza Muhinzi prototype

Some of the more technical findings of the Sakaza Muhinzi prototype are listed under the subjects; Bootstrap navigation, GoogleMaps geolocation, KnockOutJs functions, and OpenLayers maps. These findings are listed in Appendix B, Section B.5, to provide an idea of issues that can arise during the development of a mobile app.

Cookbook

A last step in the SMAD design, and development quadrant, is the modification of the Cookbook, with the pieces of code that can be reused in other projects. The main contributions to the Cookbook, besides the setting up of the environment are:

- Fetch JSON features in an OpenLayers layer (OpenLayers)
- Select plots, and display attribute values (OpenLayers)
- Fetch, edit, and save the farmer attribute data (KnockOutJS)

The code snippets are in Appendix C, an exemplification of the code is in Section 4.5.5.

Part V. Evaluation

Phase 4: Evaluation

Phase 4 “Evaluation”(Figure 6.16) rounds up the study project by evaluation, providing lessons learned, and presenting next steps, and thereby provide answers to the last research question.: How to verify the success of the development framework and the proposed design?

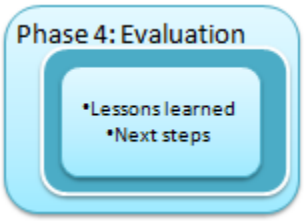

Figure 6.16 Phase 4: Evaluation

7 Evaluation of the SMAD framework

7.1 Introduction

The Spatial Mobile Application Development framework, is based on three building blocks: mobile apps, HTML5, and spatial functionality. Based on related frameworks, and these building blocks, the development approach is presented in Part II of this report. The development approach should offer a suitable platform, and usable spatial mobile functions, and includes context-awareness. The Sakaza Muhinzi case study is performed to verify the framework.

In this chapter, the question is answered to what extent the theoretical approach on the definition of the SMAD framework is successful. Section 7.2 starts with the theory on which the framework is built, evaluating the three building blocks, and the elements incorporated from related frameworks. The contributions of these elements, and building blocks to the success of the SMAD framework are identified. Sections 7.3 to 7.6, discuss the four quadrants of the SMAD framework, and to what extent they contribute to the success of the framework.

7.2 Definition of the SMAD framework

The goal of the SMAD framework is to present a development approach that offers a suitable platform, offer useful spatial mobile functions, and be context-aware. Is the framework successful in offering the platform, spatial mobile functions, and context-awareness?

The evaluation of the platform's success is discussed in Section 7.3, the mobile functions, and context-awareness in Section 7.4. The section on quadrant "Project approach" describes the usefulness of the project management method. The section on "Design and development" describes the added value of the approach proposed on analysis and design, and development.

This section focuses on the background of these four quadrants defined in the framework: three building blocks (See Chapter 3 on mobile apps, HTML5, and spatial functionality), and four mobile frameworks:

1. enterprise apps (Section 2.3.1),
2. service-oriented architecture (Section 2.3.2),
3. cartography (Section 2.3.3), and
4. context-awareness (Section 2.3.4).

7.2.1 Contributions of the reference frameworks

The reference frameworks led to the following contributions to the SMAD framework that support its goals:

- the MVC principle
- mobile cartography
- the usage of services
- context-awareness in a mobile environment

The MVC principle

The MVC principle shows to be of added value in keeping the code structured and accessible. Different blocks of JavaScript, containing application logic, are kept in separate JavaScript files. This JavaScript logic is the controller between the data model, and the HTML view. The JavaScript, combined with the data model, and the HTML structure can be reused in other applications, see Appendix C. The MVC principle translates into the different parts of mobile applications: presentation, application, and data. These three parts offer not only structure and accessibility of code, but also offer transparency in designing, and describing the app, and its logic.

Mobile cartography

Mobile cartography is probably one of the most valuable spatial functions in the mobile environment. Presentation of information is a basic function of mobile apps. Presenting spatial data on a web map is a strong means to give meaning to the data: spatial data turns into information by presenting the data the right way in a map.

The usage of services

The usage of services helps to develop an app, without the need to build every function within the project. The available services make functions and data available, which only need to be set up in the right way within the app. The service can be a map service, for example, such as the Google Maps API, or an information widget, such as the AccuWeather widget. A service can also provide some remote functions that are too complex, or too heavy, or time-consuming to run locally on the mobile device.

Context-awareness in a mobile environment

Context-awareness is, like mobile cartography, an important opportunity of mobile devices, in which location is an important element. A central part of context-awareness is directly related to location: it provides information on the user, her/his environment, activities, and the access to entities. The location of a mobile user is just one of the sensor data sources that can be made available by the mobile phone.

7.2.2 Contributions of the three building blocks

The building blocks provide, in particular, practical contributions to the framework. It shows the necessity of building platform-independent apps, as various platforms exists, and even more devices are present. The smartphone market, and related app markets, are growing rapidly.

Mobile apps

Best practices in mobile development helped to define the elements to include in the SMAD framework, and to focus on particular areas in app development. The inclusion of MVC, as described in the former section is one of such best practices. Another best practice adopted is to design to the grid, and render the application lay-out, based on the device's screen sizes. The AudioVroom web app shows that this can be realized perfectly using the Skeleton[31] framework. The prototyping with Bootstrap as attempt to realize this behavior was not a complete success, as the smartphone lay-out did not turn out as in the wireframes. Bootstrap cannot be fully blamed for this, as my experience to build a mobile app was minimal. Bootstrap needs to evolve further on the responsive grid, as for example standard navigation on the mobile environment should be more user-friendly. Default behavior of the responsive navigation shows the sidebar navigation menu in the homepage, and only after scrolling the actual content is shown. This is not an acceptable user-experience. Also the menu buttons in the top bar that collapses into one single menu button, is an extra burden in the smartphone lay-out, as users have to click through an extra level of navigation.
As more developers discover the added value of responsive design, and adopt the responsive design in mobile app development, the better the templates will become. The potential of responsive development is available, and the Bootstrap framework is relatively new, and still evolving.

The categorization of enterprise mobile applications by Unhelkar and Murugesan, were used to group mobile applications and their functions into three categories: information, operation, and collaboration. These categories relate to the question: what kind of processes are facilitated. The map usage cube (MacEachren, Taylor 1994), on the other hand, led to the grouping of how data and information are used: present, synthesize, and analyze. Based on these two dimensions, the three levels of mobile applications were defined: basic applications, personalized applications, and context-aware applications.
The categorization of applications is further elaborated in the SMAD quadrant "categorization of functions". It is a great help in development of an app, and the planning of the development project.

[31] www.getskeleton.com/

HTML, JavaScript, and CSS

The HTML language has been replaced by HTML5, JavaScript, and CSS3, as it is their combination that makes a successful platform-independent app. Web browsers understand these three coding languages. HTML provides the structure of the page, CSS the styling and display of the page, and JavaScript is instruction code to add application logic. Because the support of different web browsers for HTML5, and CSS varies a lot, a piece of code is not always interpreted in the same way. This can lead to different behavior of the app in different web browsers. Such was the case with the developed mobile navigation bar, which did not stay in place at the bottom of the page, during scrolling, in the standard web browser on the Android phone. And this behavior was not applicable in Google's Chrome web browser using IphoneTester either. Several attempts were made to change the code, to solve the problem, however this did not lead to any results. The standard behavior of Bootstrap on the smartphone seemed to overwrite the intended behavior. A solution is to built the navigation bar with HTML and CSS, leaving out the code from the Bootstrap framework.

The shortcomings of web-based mobile applications, such as several inaccessible key APIs, and the lack of an app icon on the device, did not lead to any problems in the case study. However, for certain applications one might want to use APIs such as for access to the file system, address book, and camera, and one must decide how to deal with this. A framework like Phonegap[32] offers a solution here, wrapping the HTML, CSS and JavaScript into a native app for the various platforms. The app can be made available via the app stores, and downloaded onto the smartphone. However to access APIs specific to a platform in such an app, platform specific code must be written, to be able to function on the various platforms

Spatial functions

The goal of the SMAD framework is to help develop and realize geo-enabled, platform-independent apps. The location of the user of an app is a means to geo-enable an app, for example by showing information related to the user's location. Another strong means to geo-enable the app, and in many cases a base to geo-enable the app, is the presentation of spatial information in a map. OpenLayers was selected as tools to accomplish this. The implementation of web maps with OpenLayers is quite straightforward, as it is designed to run in web browsers, and functions can be added using JavaScript.
The geolocation function is a well developed API, and is implemented using plain JavaScript, without much problems. This is also the case with adding, updating, and delete spatial data in GeoJSON format. Other spatial functions, such as WHICH, included in the SMAD quadrant on functions were not used in the case study. The framework of Dey, Abowd et al, offers automated functions to make an app context-aware. In theory this can be implemented within the SMAD framework, however no actions were taken in that direction to prove this statement in practice.

[32] www.phonegap.com/

7.3 Technical architecture

In the section above, the contributions of the reference frameworks and three building blocks were described. In this section, the first quadrant is evaluated. The technical architecture is built upon two concepts: the MVC model, actual implemented by the MVVM model, and the usage of the three languages HTML5, JavaScript, and CSS3. These two concepts are evaluated earlier in this chapter. This section discusses the selected tools for the presentation layer, the application layer, and the data layer.

7.3.1 Presentation layer

The presentation layer is built with Bootstrap, in combination with the OpenLayers framework to enable web mapping. To build a proper mobile user interface, using the responsive functions of Bootstrap, is quite a challenge. Standard functions exist to have pages nicely displayed on both large screen sizes, tablets, and smartphones. However, to have the mobile layout exactly as designed is not a straightforward process. One needs to understand standard CSS, and HTML behavior, to better understand the Bootstrap behavior. If Bootstrap gains in popularity, the support for smartphones is expected to improve, and more examples, and improved templates will be available. If such a template fits the needs of the app, the developer needs less knowledge on, and experience with CSS, as the necessary code is provided.

Inclusion of different spatial functions with OpenLayers is a straightforward process. OpenLayers does not offer the same functions as well-known GIS systems, however, it is a mature toolset to publish spatial information in web maps. Functions included are the display of datasets with different projections, the import from GeoJSON, the display of feature information, inclusion of WMS services, calculation of area size and perimeter, and the modification of spatial data. Not really mature, is the combination of the spatial information with administrative information. The attribute information attached to a feature can be viewed using OpenLayers, however, the possibility to link this information with other information is not elaborated upon. Here, the developer must develop specific code from scratch.

7.3.2 Application layer

The choice for KnockOutJS as a backbone JavaScript framework is positively evaluated. The framework offers a structured method to develop application logic, in which user-interface logic, and back-end JavaScript logic are separated. KnockOut, using the MVVM model, distinguishes between two major parts: the HTML View, and the JavaScript ViewModel. The ViewModel defines the data and functions that must be available in the app, and contain the link to the data. The View presents the information, and makes the functions available in the app by HTML. Less code has to be written using KnockOutJS as compared to plain JavaScript.

In projects in which other tools, like OpenLayers, and CouchApp, are integrated, the KnockOutJS libraries alone are not sufficient. Plain JavaScript and KnockOut functions are combined with JQuery functions. It is much easier to integrate code based on examples that work, than to newly build the code in KnockOut and plain JavaScript. JQuery is a framework that is widely used, and offers, for example, useful functions to define a proper HTTP requests to communicate with the database. Bootstrap offers JQuery functions, and JQuery is also part of the CouchApp tools.

A thorough understanding of JavaScript is a basic condition in developing platform-independent geo-enabled mobile apps.

7.3.3 Data layer

The CouchDB database offers flexibility, and ease of access to the JSON documents by HTTP. The data model does not have to be normalized, as is often the case with relational databases. For all documents, by default all revisions are kept, and these are thus available at any time. An application based on a relational data technology, is said to be data-driven, the CouchDB, on the other hand, supports an application-driven approach. Not the data and data model determine how an app is developed, but the developed functions determine what data is needed. How the data is stored in the database is of less importance, and this may evolve during development.

A point of attention is the updating of documents. The documents are often combined, not normalized, information, with possibly also hierarchical information. An example is a document with a GeoJSON feature collection, containing more features with geometry, and properties. Updating one item in such a collection, implies that the whole collection is to be stored as a new file with a new revision number.

7.4 Categorization of functions

An application that only presents information, is much simpler than one that gathers, creates, updates, and shares information and tasks. Three levels of application are defined. First, the basic applications that provide information in a predefined format. Secondly, the personalized app that combines sources of data, and aims to support individuals in performing operations and tasks. And thirdly, the context-aware applications that can be used by multiple users to collaborate, while functions are available to combine data, and perform analyses.

7.4.1 The basic app

The basic app can be compared with websites in which a content management system captures the data centrally. The data is available for the public, so security is basically having your server, from which the pages are served, secured against hackers, for example, by a firewall.
A cache manifest can be implemented to realize offline storage to provide offline access to the app, and OpenLayers logic can be built to have the spatial data in an OpenLayers map stored offline. These functions were not tested during the case study, as priority was given to other features. Metadata was added to the scripts to clarify what certain sections of code are meant to do. This way the maintenance of the code is simplified, and also the reusability is enlarged. New developers on the project, can built on this metadata without having to see through the whole code.

7.4.2 The personalized app

Three examples of personalized functions in the case study are: (1) the farmer that can select, and alter his own plots, and information, (2) the various layers that can be included and viewed in the web map, and (3) the activities that can be added to a plot. The first two examples were implemented in more detail than the third example. Only a simple form with an activitiy list is implemented to show how it should work. Drop down lists should be added that contain most common operations and statuses, as to make information capturing easy accessible. See also Section 5.5.

The spatial functions, the personalized app supports, are, in a simple form, implemented or tested during the prototyping phase:

- OpenLayer web maps with different layers, including web map services.
- The ability to add spatial data
- Add spatial services, and context widgets, including locators and events.
- Scaling and granularity, to adjust the map to the meet the user needs

On many of these functions, example code can be found on the internet, and largely also within the OpenLayers development framework. To alter plot attribute information, new code is developed, using KnockOutJS.

The Google Maps API offers base layers with various zooming levels. It is easy to integrate this API in OpenLayers. To have the plot information better positioned on the base layer, it is preferred to develop a base layer, based on available images. This service must contain logic to have the base layer in various granularities to enable zooming.

7.4.3 The context-aware app

Collaboration is the most important added value in the context-aware app. People who share information via the app, and actually work together. The context-aware functions are not implemented during the case study. The ability to save data to a central database, makes updated data available to whoever is granted access to that data. In such a way, sharing of information is realized. Another way to realize sharing of information, and collaboration is to include social networks in the app. A nice example is provided by KnockOutJS[33], on including Twitter accounts in an app, and as such sharing information with users.

[33] http://knockoutjs.com/examples/twitter.html

7.5 Project approach

The mechanisms provided with the AGILE project approach, such as continuous testing, and iterative planning, prove their added value in AGILE development. A realistic overall plan is needed upfront, however only for the short term a detailed plan is needed. Iterative planning , and continuous testing offers the mechanism to steer the project during execution.

7.6 Design and development

The SMAD framework offers an approach for the design and development of an app. The goals set for the case study afterwards turned out to be unrealistic for a starter app developer, as partly was foreseen in the project risk list. However, to show the type of application logic aimed for, a more detailed design was worked out for the complete app, at the beginning of the project.

The main contributions of the case study to the SMAD cookbook (See Appendix C, and Section 4.5.5), and thus general approach, are:

- Fetch, edit, and save the farmer attribute data (KnockOutJS)
- Select plots, and display attribute values (OpenLayers)
- Fetch JSON features in an OpenLayers layer (OpenLayers)

The content-out approach in the SMAD strategy translates into a stepwise approach of development, which makes the development of an app a straightforward process:

1. Choose a Bootstrap template to start with
2. Create the basic public content
3. Realize the page navigation
4. Choose an appropriate lay-out, and styling
5. Include widgets, and other third party services
6. Add simple data functions
7. Add web maps
8. Add more complex data functions

7.7 Suggestions for further research

Further research on development of platform-independent mobile apps is needed on a wide area of subjects. Scientific research on HTML5, JavaScript, and CSS mobile apps is scarce, as the topic is relatively new. However, mobile apps are a fast growing market, which creates new business, and adds value to those organizations offering, and using mobile apps. Research on cost/benefit of mobile apps, linking mobile apps to the organization ICT architecture, and future developments in mobile apps, are some of the more general subjects that needs further research.

In-line with this thesis research, the following research questions, and subjects, are proposed for further research:

- Responsive design versus mobile design: To what extent can responsive grid functions be used to develop the mobile user-interface to be as attractive as a dedicated mobile design? Does the advantage of having the front-end logic for different screen sizes in a single file compensate the shortcomings of the responsive grid? The case study showed problems with some of the default responsive functions in Bootstrap. It is not totally clear that this is a shortcoming of Bootstrap, or a lack of proper development experience.
- Survey on mobile app development projects: What can be learnt from mobile app development projects? What are average lead times, what is the number of project members working on the project? Such a survey will provide insight in app development, and can lead to new classifications of mobile apps.
- Use of Geocouch plugin for CouchDB: What functions are offered by the Geocouch plugin for CouchDB, and how can these be used in combination with OpenLayers? During prototyping GeoJSON format data has been disclosed in the app. This GeoJSON was fetched form the CouchDB without the Geocouch plugin.
- NoSQL database versus relational database: Is a NoSQL database the most flexible solution for mobile apps, or do situations exist in which a relational database is the better choice? The data that is disclosed in operational applications is rather structured, kept in relational databases, and is accessible via queries. The NoSQL database supports also unstructured data. The communication between an app and the database via HTTP should be compared in complexity, and performance.
- How to set up the data model in a NoSQL CouchDB: Is a data model needed in an application-oriented development approach, and if so, how can a good data model for a NoSQL database be defined? Different developer logs make one believe no data model has to be designed upfront, still somehow the data must be structured.

8 Conclusions

In this final chapter, the conclusions are presented in relation to the problem statement of this research, and the research questions (Section 1.1, and Section 1.3).

The problem statement:

How to systematically develop geo-enabled mobile applications, based on HTML5 that offer different spatial functions to support professionals in their daily work and decision-making. A case study with Rwanda coffee farmers, their personalized profiles, their community network, and the production data from their fields.

This study discusses HTML5, in combination with JavaScript, and CSS. Frameworks, and tools are presented to develop geo-enabled mobile applications. To geo-enable the mobile app, OpenLayers is included in the app. OpenLayers offers many functions, of which web mapping is the core function. OpenLayers can be combined with the HTML5 geolocation function to determine an app user's current position, and other data, and web services can be included, such as Google Maps, and Open Street Map, to combine spatial data and have the data displayed on third party base layer maps.'

The combination of HTML, JavaScript, and CSS, makes platform-independence is achieved. These three types of codes can all be understood by web browsers. A note to make is that older web browsers only have little, or no support for certain functions, and also the newest browsers have varying support for HTML5, and CSS3.
It is not a necessity to develop the back end application logic with JavaScript, as long the front end mechanisms are based on the combination of HTML, JavaScript, and CSS. The SMAD framework is build on JavaScript in the back end, which offer the advantage that less different coding practices and experiences are needed for the development.
Pure HTML, JavaScript, and CSS apps, also offer the advantage that frameworks such as Phonegap can wrap up the code into app packages for the various mobile platforms available. In such a way, the app can be distributed as a native app on these platforms.

The CouchDB database is not available for every mobile platform. Only Android, and iOS, are supported at the time of this study. Platform-independence is achieved having the CouchDB running as a central server to serve the web app, and store the app data. For offline-usage the HTML5 offline storage function, and manifest function can be used, to have the app available without any network connection. OpenLayers maps can also be kept in local storage. For mobile usage, the amount of data stored locally, should be limited, as storage capacity often is limited. This study did not get into detail on local storage, however.

The research question on technical architecture, which must be suitable for use in the developing world, is only partly answered. The SMAD framework is build on open source, and freely available tools, and frameworks, to limit the costs of the app. The platform-independency strived for, makes users are not dependent on one certain platform, and thus limited number of devices. The user can use the device at hand, and has the same functionality available, as another user on a different device.

The case study on Sakaza Muhinzi coffee farmers shows, how a development project of geo-enable apps is executed, following the SMAD framework. This case study included not only basic functions, but also personalized, and context-aware functions. Focus was set on having geo-functions available, and thus merely on personalized functions. The support processes, such as local storage, and security were under exposed. The idea is that these support processes must be received by standard available mechanisms, however this research did not get into details on this subject.
The approach to start with the basics, and work towards more complex functions, is evaluated positively. In many cases an app will not strive to offer the same full-fledge functions, which are available in an desktop environment. The Sakaza Muhinzi app does strive for functions to be available in the mobile environment, because many farmers do not own, or have direct access to an internet PC. Building such more complex apps in a modular way, as proposed in the SMAD framework, offers structure and support, to build the functions properly, and realize an app that fulfills the needs of the users.

So, can be stated that the SMAD framework is a success? From the goal to systematically develop geo-enabled mobile applications, the answer is yes. The SMAD framework offers an approach based on other scientific research, and on best practices shared by different specialists from the field.
Does the framework provide the best way to systematically develop a mobile app? This second question cannot be answered based on this study. Probably not, is the answer. Many ways exist, and approaches can and will be developed, which offer a systematic approach to develop mobile applications, including geo-functions. It would be interesting to compare these approaches, and do a survey in the market on how mobile apps are developed. This survey should answers questions like, how many people are involved in the project, what are the lead times, and what approach is followed.

Personally, I hope, my work contributes to a better integration of geo-functions in mobile applications. In my opinion spatial data are just a special type of data, like a date type. It should be possible to have spatial data and related functions available, without the restrictions of a traditional GIS system, or mapping application, such as OpenLayers, in which the central entities are the spatial data, and all other data is referred to as attribute data.

9 References

AGILEAPPS, 2012-last update, AgileApps - Platform - MVC Architecture . Available: http://www.agileapps.co.uk/platform/mvc-architecture.html [2/12/2012, 2012].

ALLEMAN, G.B., 2002. AGILE project management methods for IT projects. In: E.G. CARAYANNIS and Y.H. KWAK, eds, *The story of managing projects: a global, cross-disciplinary collection of perspectives.* Greenwood Press, pp. 324.

ANDERSON, G., 2012/07/09, 2012-last update, New Indoor Navigation Startup Uses Disruption Of Earth's Geomagnetic Field From Buildings [Homepage of ArcticStartup], [Online]. Available: http://www.arcticstartup.com/2012/07/09/new-indoor-navigation-startup-uses-disruption-of-geomagnetic-field-from-buildings [07/20, 2012].

ANTHES, G., 2011. Invasion of the Mobile Apps. *Communications of the ACM,* **54**(9), pp. 16-18.

BÄR, W. and BREUNIG, M., 2005. *Usage of mobile databases for mobile geoscientific applications.* Osnabruck: program Geotechnologien of BMBF and DFG.

BLUM, A., 2011. *Best Practices in Enterprise Smartphone Development- Rhomobile White Paper* . 2012.

BONNINGTON, C., 12/8/2011, 2011-last update, Google's 10 Billion Android App Downloads: By the Numbers [Homepage of Wired.com/gadgetlab], [Online]. Available: http://www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/ [12/18, 2011].

BOULOS, M.N.K., WARREN, J., GONG, J. and YUE, P., 2010. Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. *International Journal of Health Geographics,* **9**, pp. 14.

BURROUGH, P.A. and MCDONNELL, R.A., 1998. *Principles of Geographical Information Systems.* reprinted with corrections 2006 edn. New York, United States: Oxford University Press.

CLARK, S., 2011-last update, HTML5 and CSS3 Frameworks for Mobile Website and Application Development . Available: http://www.htmlgoodies.com/html5/tutorials/html5-and-css3-frameworks-for-mobile-website-and-application-development.html#fbid=C1ew15jS3WQ [1/23/2012, 2012].

CLARK, J.A., 2010. HTML5 Changing How You Use the Web. *Online,* **34**(6), pp. 12-14.

COUCHDB, 2012-last update, Apache CouchDB: Introduction . Available: http://couchdb.apache.org/docs/intro.html [3/12/2012, 2012].

COWEN, D.J., 1988. Gis Versus Cad Versus Dbms - what are the Differences. *Photogrammetric Engineering and Remote Sensing,* **54**(11), pp. 1551-1555.

CROFT, W., 2012/04/01, 2012-last update, Wireless Intelligence: Dashboard, Africa 2012 [Homepage of WirelessIntelligence], [Online]. Available: http://www.wirelessintelligence.com/analysis/2012/04/dashboard-africa-2012/ [2012/07/20, 2012].

DEY, A.K., ABOWD, G.D. and SALBER, D., 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction,* **16**(2-4), pp. 97-+.

DISTIMO, 2011/12/9, 2011-last update, Google Android Market [Homepage of Wikipedia], [Online]. Available: http://www.distimo.com/appstores/app-store/19-Google_Android_Market [12/18, 2011].

DOROKHOVA, R. and AMELICHEV, N., 2010. *Comparison of Modern Mobile Platforms from the Developer Standpoint.* St. Petersburg Electrotechnical University.

DOYLE, A. and REED, C., 2001. *Introduction to OGC Web Services, An OGC White Paper.*

EN.WIKIPEDIA.ORG, 2011/12/18, 2011-last update, M-Pesa [Homepage of Wikipedia], [Online]. Available: http://en.wikipedia.org/wiki/M-Pesa [12/28, 2011].

ENNAI, A. and BOSE, S., 2008. MobileSOA: A Service Oriented Web 2.0 Framework for Context-Aware, Lightweight and Flexible Mobile Applications, *Enterprise Distributed Object Computing Conference Workshops, 2008 12th* 2008, pp. 345-352.

GARTNER, 2011/08/11, 2011-last update, Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent [Homepage of Gartner], [Online]. Available: http://www.gartner.com/it/page.jsp?id=1764714 [2012/01/19, 2012].

GEONETCAST, 2., 2012-last update, GEONETCast; delivering operational services to users. Available: http://www.earthobservations.org/geonetcast.shtml [06/28, 2012].

GILMORE, J., 9/12/2011, 2011-last update, PhoneGap: Building Native Mobile Apps with HTML, JavaScript and CSS - Developer.com . Available: http://www.developer.com/ws/phonegap-building-native-mobile-apps-with-html-javascript-and-css.html [1/23/2012, 2012].

GOADRICH, M.H. and ROGERS, M.P., 2011. *Smart Smartphone Development: iOS versus Android.*

HARJONO, J., NG, G., KONG, D. and LO, J., 2010. Building smarter web applications with HTML5, *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research* 2010, IBM Corp, pp. 402-403.

HEYWOOD, I., CORNELIUS, S. and CARVER, S., 2006. *An Introduction To Geographical Information Systems.* third edition edn. Pearson Education (US).

HICKSON, I.(., 2011. *HTML5; A vocabulary and associated APIs for HTML and XHTML.* http://www.w3.org/TR/html5/ edn. W3C.

HUNTLEY, C.L., 2011. Onshore Mobile App Development: Successes and Challenges. *Computer,* **44**(9), pp. 102-104.

ITU, 2009. *Measuring the information society: The ICT development index.* Geneva, Switzerland: International Telecommunication Union.

JOHNSON, L., 12/8/2011, 2011-last update, Will HTML5 help battle mobile fragmentation in 2012? [Homepage of Mobilemarketer.com], [Online]. Available: http://www.mobilemarketer.com/cms/news/software-technology/11643.html [12/18, 2011].

JOHNSON, R.E., 1997. Components, frameworks, patterns. *SIGSOFT Softw.Eng.Notes,* **22**(3), pp. 10-17.

KAGOYIRE, C., DE BY, R.A. and DAMASCENE, J., 2011. *Sakaza Muhinzi 2011; Project description.* ITC/Universiteit Twente & CGIS-NUR.

KONEČNÝ, M. and STANĚK, K., 2010. Adaptive cartography and geographical education. *International Research in Geographical and Environmental Education,* **19**(1), pp. 75-78.

KONEČNÝ, M., STANĚK, K. and FRIEDMANNOVÁ, L., 2007. Adaptabilní mapy pro krizový management [Adaptive maps for crises management]. *Kartografická spoločnost' Slovenskej repu,* (15), pp. 41.

KRAAK, M. and ORMELING, F.J., 2003. *Cartography, Visualization of Geospatial Data.* Second Edition edn. Pearson Education Limited.

LARA, M., 11-18-2011, 2011-last update, Responsive Design Essentials: Look Great on any Device - Facebook developers. Available: http://developers.facebook.com/blog/post/599/ [2/12/2012, 2012].

MACEACHREN, A.M. and TAYLOR, F.D.R., 1994. *Visualization in modern cartography.* New York: Pergamon.

MARCOTTE, E., 5-25-2010, 2010-last update, A List Apart: Articles: Responsive Web Design . Available: http://www.alistapart.com/articles/responsive-web-design/ [2/12/2012, 2012].

MARK, J., 6/23/2011, 2011-last update, Agile Web Development That Works . Available: http://sixrevisions.com/web-development/agile/ [2/28/2012, 2012].

MCKEE, L. and OGC STAFF, 2005. *The Importance of Going "Open", An Open Geospatial Consortium (OGC) white paper.*

MICROSOFT.COM, 2012-last update, Implementing the Model-View-ViewModel Pattern . Available: http://msdn.microsoft.com/en-us/library/ff798384.aspx [3/26/2012, 2012].

MOBILE WORLD CONGRESS, 2011-last update, Global mobile awards: winners 2011. Available: http://www.globalmobileawards.com/awards/winners_2011.php [12/28, 2011].

MOBILEADVERTSINGBOOK.COM:, 3/5/2011, 2011-last update, What Is Mobile Fragmentation? [Homepage of Mobile Advertising Book], [Online]. Available: http://www.mobileadvertisingbook.com/mobile-advertising/what-is-mobile-fragmentation [12/18, 2011].

NL.WIKIPEDIA.ORG, , Android Market
[Homepage of Wikipedia], [Online]. Available: http://en.wikipedia.org/wiki/Android_Market [12/18, 2011].

P2PU, 2012-last update, P2PU | Javascript | What is JavaScript . Available: http://p2pu.org/en/groups/javascript-101/content/week1/ [2/13/2012, 2012].

PICCHI, A., 2011. Chapter 3: Web development for iOS devices. *Pro Iphone and Ipad Web Design and Development: Html5, Css3, and Javascript With Safari.* Apress, pp. 51.

PILGRIM, M., 2010. *HTML5: Up and Running.* First edn. Sebastopol: O'Reilly.

POPESCU, A.(., 2010. *Geolocation API Specification.*

RAO, M., 2011. *Mobile Africa Report 2011; Regional Hubs of Excellence and Innovation.* Mobile Monday.

REICHEL, M. and RAMEY, M.A.(.)., 1987. *Conceptual Frameworks for Bibliographic Education - Theory into Practice.* Littleton Colorado: Libraries Unlimited Inc.

REICHENBACHER, T., 2001. *Adaptive concepts for a mobile cartography.* Science China Press, co-published with Springer.

ROTINI, 2011/10/17, 2011-last update, Zubair Abubakar: How i Built One of Nigeria's Most Successful Mobile Applications. Available: http://youthhubafrica.org/2011/10/17/zubair-abubakar-a-social-enterpreneur-and-developer-of-the-most-successful-app-ever-created-in-nigeria/ [12/28, 2011].

SAUNDERS, E., GREYLING, J. and COWLEY, L., 2010a. Pragmatics Regarding Compression of Files for Delivery to Mobile Phones, , accepted to SATNAC 2010 2010a.

SAUNDERS, E., GREYLING, J. and COWLEY, L., 2010b. An intelligent framework for mobile devices, *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists* 2010b, ACM, pp. 413-416.

SAUTER, P., VÖGLER, G., SPECHT, G. and FLOR, T., 2005. A Model, View, Controller extension for pervasive multi-client user interfaces. *Personal Ubiquitous Comput.*, **9**(2), pp. 100-107.

SMYTH, R., 2004. Exploring the usefulness of a conceptual framework as a research tool: A researcher's reflections. *Issues in Educational Research,* (14), pp. 167-167-180.

SNOOK, J., 8-18-2009, 2009-last update, A List Apart: Articles: JavaScript MVC . Available: http://www.alistapart.com/articles/javascript-mvc/ [2/13/2012, 2012].

STANGARONE, J., 1-24-2011, 2011-last update, Mobile applications: Why architecture matters | mrc's Cup of Joe Blog . Available: http://www.mrc-productivity.com/blog/2011/01/mobile-applications-why-architecture-matters/ [2/12/2012, 2012].

TRYGVE, M.H., 1979-last update, Trygve/MVC [Homepage of XEROX PARC], [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html [12/29, 2011].

TUTORIALSPOINT.COM, 2012-last update, Agile Project Management . Available: http://www.tutorialspoint.com/management_concepts/agile_project_management.htm [2/27/2012, 2012].

UNHELKAR, B. and MURUGESAN, S., 2010. The enterprise mobile applications development framework. *IT Professional,* **12**(3), pp. 33-39.

VOSLOO, S., WALTON, M. and DEUMERT, A., 2009. *m4Lit: A Teen M-novel Project in South Africa.* wordpress.com.

WIKIPEDIA, 2012a. Mobile operating system --- Wikipedia, The Free Encyclopedia.

WIKIPEDIA, 2012b. Open Geospatial Consortium --- Wikipedia, The Free Encyclopedia.

WIKIPEDIA, 2011. Model-view-controller --- Wikipedia, The Free Encyclopedia.

Part VI. Appendices

Appendix A Theoretical framework

A.1 Features of three popular smartphones

The features of three popular smartphones are listed in Table 9.1, from the Mobiledia site[34].

Table 9.1 Features of the iPhone 4S, Lumnia 900, and Galaxy S2

			
	Apple iPhone 4S	**Nokia Lumia 900**	**Samsung Galaxy S2**
Release Date:	Q4 2011	Q1 2012	Q3 2011
Technical			
Network:	CDMA 800 / 900 / GSM 850 / 900 / 1800 / 1900 / UMTS 850 / 900 / 1900 / 2100	GSM 850 / 900 / 1800 / 1900 / LTE 700	GSM 850 / 900 / 1800 / 1900 / UMTS 850 / 900 / 1900 / 2100
Form Factor:	Block / Apple iPhone OS 5.0	Block / Windows Phone 7.5	Block / Google Android OS v2.3
Dimensions:	115 x 59 x 9 mm	128 x 69 x 11 mm	125 x 66 x 8 mm
Weight:	140 g	159 g	116 g
Antenna:	Internal	Internal	Internal
Navigation:	Touch Screen	Touch Screen	Touch Screen (TouchWiz 4.0)
Battery Type:	Li-Ion	1830 mAh Li-Ion	1650 mAh Li-Ion
Talk Time:	14 hours	7 hours	
Standby Time:	8.3 days	12.5 days	
Memory:	16.0 GB or 32.0 GB or 64.0 GB	14.5 GB	
Expandable Memory:			microSD / TransFlash
Safety			
Radiation (SAR):	(1.17 W/kg)		(0.96 W/kg)
Imaging			
Main Screen:	TFT (Retina Display / Gyroscope / Accelerometer / Proximity Sensor / Ambient Light Sensor) 1,670,000 colors (640 x 960 px)	AMOLED (ClearBlack / Accelerometer / Proximity Sensor / Ambient Light Sensor) 16,700,000 colors (480 x 800 px)	Super AMOLED Plus (Gyroscope / Accelerometer / Compass / Proximity Sensor / Ambient Light Sensor) 16,700,000 colors (480 x 800 px)
External Screen:			

[34] http://www.mobiledia.com/phones/compare/compare.php

Camera:	8.0 MP / Flash / Auto-Focus / Geotagging / HD Video Recorder / 0.3 MP / FaceTime Video Chat	8.0 MP / Carl Zeiss / Dual LED Flash / 3X Zoom / Face Detection / Red-Eye Reduction / Auto-Focus / HD Video Recorder / 1.0 MP / Video Chat	8.0 MP / LED Flash / Zoom / Auto-Focus / HD Video Recorder / 2.0 MP / Video Calling
Audio			
MP3 Player:	AT&T Mobile Music / iCloud (MP3 / AAC / WAV)	Nokia Music / Dolby Digital Plus (MP3 / AAC / WMA)	MP3 / AAC / AAC+ / eAAC+ / WMA
FM Radio:	☒	☑	RDS Radio
Speakerphone:	Dual Microphone	Noise Cancellation	☑
Push-To-Talk:	☒	☒	☒
Multimedia			
Wallpapers:	640 x 960 px	480 x 800 px	480 x 800 px
Screen Savers:	640 x 960 px	480 x 800 px	480 x 800 px
Ring Tones:	MP3	AT&T Music	MP3
Themes:	☑	☑	☑
Games:	☑	Xbox Live / Windows Marketplace for Mobile	Android Market
Streaming Multimedia:	AT&T Mobile TV / AT&T Video / iTunes (MPEG-4 / H.264 / MOV)	AT&T Video (MPEG-4 / AVI / H.263 / H.264 / WMV9)	DivX / MPEG-4 / H.263 / WMV3 / YouTube
Messaging			
SMS:	☑	☑	☑
EMS:	☑	☑	☑
MMS:	☑	☑	☑
Email:	POP3 / IMAP4 / SMTP / Push Email / MobileMe	POP3 / IMAP4 / SMTP / Exchange / Outlook	POP3 / IMAP4 / SMTP / Gmail
Chat:	AOL / Windows Live / Yahoo	Windows Live	AOL / Google / Windows Live / Yahoo
Predictive Text:	☑	Handwriting Recognition	SWYPE / Handwriting Recognition
Applications			
Phonebook Capacity:	?	?	?
Calendar:	☑	Office Mobile	Google Calendar
To-Do List:	☑	☑	☑
WAP:	2.0 (Safari Browser / Google Search)	2.0 (Internet Explorer 9)	2.0 (Webkit / Google Search / Flash 10.1)
Voice Commands:	Siri	☑	☑
Calculator:	☑	☑	☑
Connectivity			
Bluetooth:	4.0 (A2DP / DUN / HFP / HSP)	2.1	3.0 (A2DP / AVRCP / HFP / HSP / OPP)
Infrared Port:	☒	☒	☒
High-Speed Data:	cdma2000 1xEV-DO Rev. A / HSDPA	LTE	HSPA-Plus
Wi-Fi:	802.11 b/g/n	802.11 b/g/n	802.11 a/b/g/n / DLNA / NFC
GPS:	Compass (AT&T Navigator)	AT&T Navigator (Nokia Maps / Nokia Drive / Bing Ma	Compass (Google Maps)
PC Sync:	USB 2.0	USB 2.0 (My Phone)	USB 2.0

A.2 HTML5 mobile application development frameworks

This section lists development platforms in Table 9.2, and per platform, a link to information on the platform. The advantages, and disadvantages, in column two and three, are regarding the development of platform-independent geo-enabled apps.

Table 9.2 HTML5, CSS3, JavaScript mobile development platforms

Platform	Advantages	Disadvantages
PhoneGap phonegap.com/	Access to device features by JavaScript APIs, app stores, well documented, acces to OpenLayers, include native code for heavy weight apps	Separate SDKs per platform, Phonegap Build not for free multiple users/developers
DHTMLX Touch dhtmlx.com/touch/	User interface widgets, Google Maps, free of cost, user interface design, compatible with Phonegap/access app stores	DHTML/no HTML5, Native for Android and IoS only
The M Project www.the-m-project.net/	MVC, Native packaging, no knowledge of HTML required, Google Maps, PhoneGap device feature APIs, built in web server	Use of Expresso/Node JS
The 52 Framework www.52framework.com/	Many examples	Few documentation, website development
Sencha Touch www.sencha.com/products/touch/	native look style interfaces, compatible with Phonegap/access app stores	Commercial
iUI www.iui-js.org/	No use or knowledge of JavaScript required, iphone-style interfaces	Complex functions
JQTouch jqtouch.com/	Easy set up, emulate mobile platforms, offline usage	JavaScript plug-in mobile browsers, documentation
JQuery Mobile jquerymobile.com	Easy to learn, modular architecture, GoogleMaps examples, PhoneGap compatible	Kennis van JQuery
KendoUIMobile www.kendoui.com/mobile.aspx	JQuery-based, many functions included, supports IE7	Focus on Android and IOs
RhoMobile Rhomobile.com	Supports MVC, native apps	Needs Ruby programming, Not HTML5 based
Bootstrap 2 twitter.github.com/bootstrap/	Flexible grid, responsive design	
Ember.JS emberjs.com/ **(SproutCore** sproutcore.com/**)**	Supports MVC, scalability, data-centric applications	Specific coding

Platform	Advantages	Disadvantages
Backbone.JS documentcloud.github.com /backbone/	Model-View support, many examples	
KnockoutJS knockoutjs.com/	MVVM principle, automatic UI refresh, declarative bindings	
Skeleton www.getskeleton.com/	Resonsive grid, rapid development	Webpage, complex functions

The remainder of this section, lists some interesting URLs that were gathered during the first phase of the thesis, setting up the framework.

- Free Multiple Phone Web Based Application Framework
  http://open-tube.com/10-free-multiple-phone-web-based-application-framework-framework-for-building-native-looking-web-apps-for-smart-phones-and-tablet/
- 33 useful mobile development framework to create mobile App
  http://blog.dreamcss.com/frameworks/mobile-development-framework/
- Create offline Web applications on mobile devices with HTML5
  http://www.ibm.com/developerworks/web/library/wa-offlineweb/index.html
- 22 Excellent HTML5 Tutorials
  http://blogfreakz.com/web-design/html5-tutorial/#mobile
- 15 Useful CSS3 and HTML5 Templates and Frameworks
  http://speckyboy.com/2010/04/16/15-useful-css3-and-html3-templates-and-frameworks/
- Tutorial: Your First Mobile HTML5 App - Basics, Forms, and Geolocation (Part 1)
  http://www.mobilehtml5.com/post/371921120/tutorial-your-first-mobile-html5-app-the-basics
- Examples of HTML5
  http://slides.html5rocks.com

A.3 HTML5 and Spatial functionality

The second step in setting up the development framework was to investigate what type of HTML5 mobile apps can be found on internet and the Android app market, what spatial mobile apps are present here, and what HTML5 offers on spatial functions, and its limitations. In this sections some of the URLs are listed that offer interested examples of functions with HTML5, mobile, and spatial elements.

- OpenLayers and HTML5-Canvas:
  http://mobilegeo.wordpress.com/2010/03/01/integrating-openlayers-and-html5-canvas/
  http://mab.edina.ac.uk/canvasosm/testolcanvasosm.html
- HTML5 Local storage and OpenLayers:
  http://mobilegeo.wordpress.com/2010/03/03/html5s-local-sql-database-openlayers/
- 3D Planet Viewer by Chris Adams renders flat images of our planet from the NASA WorldWind WMS servers onto a sphere, allows for rotation/panning and zooming. Raytracing computes the colour of each pixel, the canvas is used to set each pixel colour accordingly.
  http://creativefan.com/20-shockingly-cool-html5-canvas-applications/
  http://www.canvasdemos.com/2010/11/05/3d-planet-viewer/ (not available)
- Creating mobile Web applications with HTML 5, Part 1: Combine HTML 5, geolocation APIs, and Web services to create mobile mashups
  http://www.ibm.com/developerworks/library/x-html5mobile1/index.html#resources
- HTML5 demos with geo functions
  http://html5demos.com/geo
- HTML5 Apps: Positioning with Geolocation
  http://mobile.tutsplus.com/tutorials/mobile-web-apps/html5-geolocation/
- TileMill: an open-source map design studio with Backbone.js. TileMill lets you manage map layers based on shapefiles and rasters, and edit their appearance directly in the browser
  http://mapbox.com/tilemill/
- Leaflet: Leaflet is a modern, lightweight open-source JavaScript library for mobile-friendly interactive maps.
  http://leaflet.cloudmade.com/

A.4 Selected tools in SMAD approach

In this study, web application development is chosen as a basis to develop geo-enabled mobile applications. Starting point is to write code in HTML5, JavaScript, and CSS3, as to make logic interpretable for the modern web browsers, specifically the mobile web browsers. The development of a mobile web app can be supported by various tooling. This section proposes tools, and tool sets that is used developing a geo-enabled mobile app in line with the SMAD approach.

HTML5, JavaScript, and CSS3 can be written in a simple text editor, created within the development environment of a web browser, or created using one of the many frameworks available on the net. These programming frameworks offer reusable functionality. A framework offers support for development of dynamic web sites, or web applications, offers control of the overall program's flow, has a default behavior, and is extensible. (Picchi 2011) Frameworks that were compared in this study are in Section A.2, HTML5 mobile application development frameworks. Most of these frameworks are aimed to create UIs, such as Sencha, JQuery Mobile, and Bootstrap. However, also JavaScript frameworks are included, aimed to support building the application logic, these are EmberJS, KnockoutJS, and BackboneJS. Many of these code libraries can be used together.
The following tools were selected for the implementation of the SMAD framework:

A.4.1 Presentation - User Interface: Bootstrap 2[35]

Bootstrap claims to be a platform for simple and flexible HTML, CSS, and Javascript for popular user interface components and interactions. It includes functions for responsive design, and access to basic mobile website functionality, like buttons, toolbars, and icons. Bootstrap is licensed under the Apache License, Version 2.0[36].

Bootstrap is selected for development of the user interface, for the following reasons.

- HTML5, JavaScript, and CSS3: Bootstrap is open source, and offers reusable components to support basic mobile functionality. It is a styleguide to document not only the features, but best practices and living, coded examples.
- Responsive design and flexible grid: built-in grid-classes are offered, and can be extended by own grid-classes. The components are scaled according to a range of resolutions and devices.
- For all skill levels: this framework aims to support not only skilled and experienced developers, but is also suitable for beginners.
- Browser support: Bootstrap has evolved to include support for all major browsers, even Internet Explorer 7.
- JQuery plugins: adds easy-to-use and extensible interactions.
- OpenLayers and GoogleMaps: example code is available on the internet to include OpenLayers web maps, and GoogleMaps.

As Bootstrap is a web app development tool, and access to device features, like camera, compass, and accelerometer is not part of this framework. As it is a HTML5 framework, the geolocation API can be used to determine a user's location. To have the application work offline, an application manifest can be added. Access to the device features can be added using the PhoneGap APIs.

Other UI frameworks that are investigated, and listed in Table 9.2 HTML5, CSS3, JavaScript mobile development platforms in appendix Appendix A, offer much of the same functionality. PhoneGap offers a platform to wrap up web apps, build in HTML5, JavaScript, and CSS3, into packages that can be distributed to the different app stores. DHTMLX Touch, the M Project, JQuery Mobile, and RhoMobile were not selected, as they are not HTML5 based, or asked for other programming language knowledge as well. The 52 framework, Sencha Touch, iUI, JQTouch, KendoUIMobile were set aside, because the lack of documentation and examples, their main focus on iPhone and Android, or the more commercial focus of the organization behind the platform.

[35] twitter.github.com/bootstrap/index.html
[36] www.apache.org/licenses/LICENSE-2.0

A.4.2 Application - Application Logic: KnockoutJS[37]

Knockout (KO) is a JavaScript library that helps you to create rich, responsive display and editor user interfaces with a clean underlying data model. Any time you have sections of UI that update dynamically (e.g., changing depending on the user's actions or when an external data source changes), KO can help you implement it more simply and maintainably.

KnockoutJS headline features:

- Elegant dependency tracking - automatically updates the right parts of your UI whenever your data model changes.
- Declarative bindings - a simple and obvious way to connect parts of your UI to your data model. You can construct a complex dynamic UIs easily using arbitrarily nested binding contexts.
- Trivially extensible - implement custom behaviors as new declarative bindings for easy reuse in just a few lines of code.

The KnockoutJS framework is selected over BackboneJS and Ember.JS. Main reason is the easiness of use, the clear examples, and support for MVC (in KO: MVVM) model. Ember.JS is the only one of these three frameworks that fully supports MVC development, as BackboneJS only supports model-view, and KO is build on the Model-View-ViewModel principle. The view of KO is the actual HTLM document, that is created using Bootstrap 2 in the SMAD approach. The view is linked to the viewmodel using declarative bindings. The viewmodel can be seen as a form of controller in the MVC model. It is a pure-code representation of the data and operations on a UI. The model in MVVM, is your application's stored data, completely independent of the UI. This data represents objects and operations in your business domain. This MVVM model is thus in line with the MVC model, and because the Ember.JS framework asks for specific coding with so-called handlebars, and lacks good documentation, KO was selected over Ember.JS. BackboneJS was not selected, because KO offered a friendly tutorial to start learning the basic application development principles with a JavaScript framework, and it hands good support for automatically updates of the UI initiated by user interactions, or changes in the data.

In case, specific device hardware features are not accessible via the web application method, the apps can always be extended by platform specific code. The PhoneGap framework[38], for example, can wrap up the HTML5, JavaScript, CSS3 code into native packages for the specific platforms, and offers plug-ins to access the different features of these platforms.

[37] knockoutjs.com
[38] http://phonegap.com/

A.4.3 Data – Database: IrisCouch[39]

Iris Couch claims to provide easy hosted CouchDB: It offers a web server, which serves up a database back-end, accessible via a RESTful JSON API. The database includes support for spatial data, and can be queried by JavaScript. Functionality is offered to synchronize the data to the server and the mobile devices.

CouchDB[40]
CouchDB is a peer based distributed database system. CouchDB hosts, servers and offline-clients, can have independent "replica copies" of the same database, where applications have full database interactivity (query, add, edit, delete). When back online or on a schedule, database changes are replicated bi-directionally.(Couchdb 2012) CouchDB is not a relational database, nor an object oriented database. It is document oriented, CouchDB simplifies the development of document oriented applications, which make up the bulk of collaborative web applications.

Bär (Bär, Breunig 2005) proposes the use of mobile databases as the storage backend of mobile applications. The mobile application has to provide offline update capabilities on local replicated geodata, which can be automatically incorporated back into the enterprise database. Three spatial databases were looked at: PostGIS, SpatiaLite, and CouchDB with GeoCouch. The database must above all be spatially enabled, which all of these three databases are. A local replication database on the mobile device would simplify synchronization of data with the server side database. CouchDB with GeoCouch was selected, because it is a NoSQL database that offers access to the data via JavaScript, directly from the browser, or underlying viewmodel in the KnockoutJS framework. As the CouchDB site explains: "NoSQL database technologies have emerged to enable the cost-effective management of data behind modern web and mobile applications". An replication database can be implemented on Android devices, and only recently also an iOS version of CouchDB is available.

[39] www.iriscouch.com/
[40] www.couchdb.com

Appendix B Prototyping

This appendix contains detailed information on the prototyping phase. The appendix starts with an overview of web pages that were consulted during training, and as background information developing the prototypes (Section B.1). Section B.2 describes the set up of the initial prototyping environment, and Section B.3 describes how to install the first version of the prototype, prototype Stage 1. Section B.4 contains a list of OpenLayers examples, and an URL to these examples, and the local installation of the second version of the prototype, which is a version without CouchApp and CouchDB. The structure of prototype Stage 3 is described in Section B.5.

B.1 Background and training materials

This section of the prototyping appendix gives an overview of web pages that were consulted learning basics, and more, of web development with HTML5, JavaScript, CSS. This overview also includes introductions to Twitter Bootstrap, KnockoutJS, Couchdb, GoogleMaps API, and Openlayers.

- Learning HTML, JavaScript, and CSS
  http://p2pu.org/en/groups/javascript-101
- Google: HTML, CSS, and Javascript from the Ground Up
  http://code.google.com/intl/nl/edu/submissions/html-css-javascript/
- Introduction JavaScript (and JQuery)
  http://jqfundamentals.com/book/index.html#chapter-2
- ABC of web development
  http://net.tutsplus.com/articles/the-abcs-of-web-development/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+nettuts+%28Nettuts%2B%29
- Video JavaScript from null
  http://net.tutsplus.com/tutorials/javascript-ajax/javascript-from-null-video-series/
  - JavaScript from Null video Series: Chapter 1; Hello World
    Text editor, Firefox add-ons; Firebug, and Web developer
  - JavaScript from Null: Chapter 2; Data Types
  - JavaScript from Null: Chapter 3; Conditional Statements
  - JavaScript from Null: Chapter 4; Arrays, Functions, and your First Animation
    Push, pop, unshift, shift
  - JavaScript from Null: Chapter 5 – Events
  - JavaScript from Null: Cross-Browser Event Binding
  - JavaScript from Null: Utility Functions and Debugging

- JavaScript for Newbies
  http://www.coursesweb.net/javascript/lessons
  http://www.w3schools.com/js/js_intro.asp

- Bootstrap Quick start
  http://webdesign.tutsplus.com/tutorials/complete-websites/twitter-bootstrap-101-introduction/
  http://webdesign.tutsplus.com/tutorials/complete-websites/twitter-bootstrap-101-the-grid/
  http://webdesign.tutsplus.com/tutorials/htmlcss-tutorials/twitter-bootstrap-101-introducing-2-0/
  http://webdesign.tutsplus.com/tutorials/htmlcss-tutorials/twitter-bootstrap-101-the-navbar/
  http://webdesign.tutsplus.com/tutorials/workflow-tutorials/twitter-bootstrap-101-tabs-and-pills/
- KnockOutjs Tutorials and example Multiple ViewModels
  http://learn.knockoutjs.com/
  http://blog.yojimbocorp.com/2012/02/14/multiple-view-models-with-knockout/#comment-62
- CouchDB
  http://net.tutsplus.com/tutorials/getting-started-with-couchdb/
  http://wiki.apache.org/couchdb/
  http://guide.couchdb.org/draft/tour.html
  http://wiki.apache.org/couchdb/EntityRelationship
  http://wiki.apache.org/couchdb/Configurationfile_couch.ini
- JSON
  http://www.json.org/fatfree.html
  http://geojson.org/geojson-spec.html
  http://api.jquery.com/jQuery.getJSON/
- CouchApp
  http://www.couchapp.org/page/getting-started
  http://custardbelly.com/blog/2010/12/08/jquery-mobile-couchdb-part-1-getting-started/
- Test
  http://iphonetester.com/
- Google Maps
  http://atomicrobotdesign.com/blog/htmlcss/using-html5-geolocation-and-the-google-maps-api/
  https://developers.google.com/maps/documentation/javascript/maptypes
- OpenLayers information and examples
  http://openlayers.org/
  http://openlayers.org/dev/examples/

B.2 Setting up the environment

The prototypes were worked out on a Windows 7, 64 bits machine. The environmental set up, describes how to set up the environment on a Windows machine. The installation files needed for other platforms can be found on the internet. In short the following steps were completed:

- Install a text editor: CoffeeCup Free, and Notepad ++
- Firefox as main browser to test
  - Add on; Firebug
  - Add on; Web Developer
- Set up Bootstrap2 (version 2.0.2)
  - Copy Docs folder to prototype directory
  - Include example HTMLs and edit links to JS, CSS, and icons
  - Responsive layout changes version 2.0.3
- Set up KnockOutjs (version 2.1.0beta)
  - Copy .js to /assets/js/
  - Include link to knockout-2.1.0beta.js in HTML
- GeoCouch
  - Apache CouchDB installation Windows
  - Set bind_address = 127.0.0.1 to 0.0.0.0 in CouchDB configuration file
  - JQuery version 1.7.2
  - CouchApp install
  - GeoCouch

B.2.1 The text editor

Development of a web app with HTML, CSS, and JavaScript can be done with a simple text editor. A simple notepad application will do, however a text editor that is a bit more powerful can be of great help. For this thesis work, I choose to work with Coffee Cup Free HTML editor, and Notepad++. Using Coffee Cup mainly for the HTML and CSS, and Notepad ++ for the JavaScript. These editors understand tags within HTML, CSS, and JS, and adds color highlighting to distinct between plain text, variables, and other tags.

Coffee Cup Free HTML editor

- Downloaded from: http://www.coffeecup.com/free-editor/
- Newest version 9.7, file: CoffeeFreeHTML9.7.exe
- Save locally, and execute

Notepad ++

- Downloaded from http://notepad-plus-plus.org/download/v6.1.2.html
- Newest version 6.1.2, file: npp.6.1.2.Installer.exe
- Save locally, and execute

B.2.2 Firefox web browser

Firefox is the main web browser used during development of the Sakaza Muhinzi farmer app. It has two add-ons to make the development, and testing of the web app a bit easier: Firebug, and Web Developer. Firebug is a handy toolset within Firefox to see what is happening loading a web page. It shows the HTML, CSS, and JavaScript, shows all HTTP requests, and allows for debugging by stop points, for example. The Web Developer toolbar gives, among others, easy access to functions to enable or disable HTML, CSS, and/or JavaScript, and to resize the page, which is a nice feature in building responsive layout that should adapt to the screen size of the device used.

- Firefox can be downloaded from: http://www.mozilla.org/nl/firefox/fx/
- Version 12 was used during the project, however newer versions are available.
- The add-ons Firebug, and Web Developer can be found within Firefox, from the 'Firefox -> Add-ons' menu. The extensions can be searched for, and activated:
    - Firebug 1.9.1 (with Firefox 12)
    - Web Developer 1.1.9

B.2.3 Twitter Bootstrap for UI development

Various frameworks are available to develop a User-interface for HTML, JS, CSS web apps, and web sites. The framework, Twitter Bootstrap, was selected, because of the overall functions available, the extensive examples, and above all the responsive layout functions. Bootstrap is a library of files with a few example starting templates, in which some of the functions are used. The Bootstrap site itself (http://twitter.github.com/bootstrap/) is a guide to use the various functions inside the library.

- On the Bootstrap site is a button to download the latest version. Version 2.0.2 was downloaded by downloading the Bootrap.zip
- Unzip the file to your prototype directory. The Bootstrap directory consists of the sub folders:
    - CSS
    - IMG
    - JS
- Further a less, a build, a templates, and an ICO directory were available. According to the video: http://webdesign.tutsplus.com/tutorials/complete-websites/twitter-bootstrap-101-introduction/, the directory structure was altered. An assets directory was added
    - Assets
        - CSS
        - IMG
        - ICO
        - JS

- Include example HTMLs and edit links to JS, CSS, and icons within the examples to be able to view the examples working in your browser, eg:
  - `<link href="assets/css/bootstrap.css" rel="stylesheet">`
  - `<link href="assets/css/bootstrap-responsive.css" rel="stylesheet">`
  - `<link rel="shortcut icon" href="assets/ico/favicon.ico">`
  - `<script src="assets/js/jquery.js"></script>`
  - And more
- Responsive layout changes version 2.0.3, was necessary to have responsive layout function properly. At time of writing (dd 13th of May 2012) Bootstrap 2.0.3 is the current version.
- Bootstrap offers functionality to download and customize your Bootstrap environment. This way you can include only those functions that you need for your app/site, and also set some variables, like colors, font size, and font weights. http://twitter.github.com/bootstrap/download.html.
- The first prototype phase, the full Bootstrap version was downloaded, whit standard settings for CSS variables, and all JQuery plugins available. For the next prototype phases, lay out variables were changed to personalize the user experience, and only necessary functions will be loaded.

B.2.4 Knockout JS, a JavaScript framework

The KnockoutJS framework was selected for its MVVM principle. The back end application logic is written in JS, and splits the logic, from the view in the HTML.

- Save file knockout-2.1.0beta.js from Github via http://knockoutjs.com/ to your local pc
- Copy the .js to your prototype directory: /assets/js/
- During development, include link to knockout-2.1.0beta.js in HTML, and the ViewModel .js files you want to include in your prototype. The HTML should contain your KnockoutJS View logic.
  - `<script type="text/javascript" src="assets/js/knockout-2.1.0beta.js"></script>`
  - `<script type="text/javascript" src="assets/js/ViewModelMaster.js"></script>`
- The tutorials on http://learn.knockoutjs.com/ are nice examples to get started with KnockOutJs.

B.2.5 Couchdb, a NoSQL database, with GeoCouch

IsisCouch offers hosting of Couchdb with GeoCouch. For this project it was chosen to install a local Couch that would make developing and testing more transparent, and also available offline.

- Installation file was downloaded from the CouchDB site on Github via link http://wiki.apache.org/couchdb/Installation . File named setup-couchdb-1.2.0_otp_R15B.exe, available via https://people.apache.org/~dch/dist/1.2.0/ was downloaded.
- Run the executable, and choose to set some of the parameters. I choose to install on "D:\program files" as my Windows did not allow to create new directories in the

"C:\program files" directory. The port chosen is 5984. A local service is created for the Couch.

- The database can be accessed with Futon in the web browser via http://127.0.0.1:5984/_utils/ .
- In D:\Program Files\CouchDB\etc\couchdb\local.ini the configuration of a few parameters were changed in local.ini during prototyping:
  - bind_address = 0.0.0.0 . To let the database listen to not only the localhost URL (127.0.0.1), but also the Ethernet address, in my case 192.168.0.103.
  - allow_jsonp = true . To allow Jsonp in getting data from the database, or from the file system
- No admins and users were yet defined in prototype phase 1, which allows anyone on the local network to work with the database.
- Update the JQuery library to 1.7.2 in "D:\Program Files\CouchDB\share\couchdb\www\script"

To handle HTTP requests to the database, such as GET and POST, the command line tool cURL was installed.

- cURL site: http://curl.haxx.se/
- From http://curl.haxx.se/gknw.net/win32/ the file curl-7.23.1-win64-ssl-sspi.zip was downloaded and unpacked in C:\curl. C:\curl was added to the path environmental variable.
- Examples of commando's:
  - curl -v GET http://127.0.0.1:5984/pny_sm/_design/tasks/_list/tasks/farmertasks
  - curl -X PUT http://127.0.0.1:5984/albums/6e1295ed6c29495e54cc05947f18c8af -d "{\"title\":\"There is Nothing Left to Lose\",\"artist\":\"Foo Fighters\"}"
- The first example asks for data in database pny_sm, in application tasks, all documents in the view farmertasks.
- The second example puts a new album (title, artist) to the albums database

To be able to properly work with the json data from the database in a web app on the local Windows machine, Couchapp was installed. Couchapp can serve the web app, and offers functions to easily update your web app, and clone it to other databases.

- File couchapp-0.8.1-win.exe was downloaded via http://couchapp.org/page/installing from Github.
- Run the executable, and choose a location for your Couchapps. I installed in D:\Program Files\Couchapp .
- The site http://custardbelly.com/blog/2010/12/08/jquery-mobile-couchdb-part-1-getting-started/ is a good dexcribed example to start working with CouchDB and CouchApp. As well as CouchDB starter guide on http://guide.couchdb.org/draft/ .

B.3 Prototype Stage 1: The files and local implementation

B.3.1 Install the prototype(s)

To have the prototypes running on your local machine, you need a CouchDB, and CouchApp, as described in Section B.2.5. The prototype files, including HTML, JavaScript, and CSS, are in the ZIP file: pny_prototype1.zip. The structure of the files of the CouchApp is described in Table 9.3. In bold the adjusted/added files of prototype Stage 1.

Table 9.3 The CouchApp file structure of prototype Stage 1

_attachments	HTML files/pages
	SM.html
***img**	Images used in app
***style**	Bootstrap CSS files
evently + sub directories	Track changes, data.js, mustache.js, query.json
Lists	CouchDB lists used in app
Shows	CouchDB shows used in app
Updates	CouchDB updates used in app
Vendor	
***couchapp**	
***_attachments**	Couchapp JQuery JavaScript
	loader.js
***assets\js**	Bootstrap, and KnockOut JavaScript
	Geolocation.js
	ViewModelDataKO.js
	ViewModelMulti.js
	ViewModelMulti2.js
***evently**	Log in/out functions
***lib**	Couchtapp/CouchDB JavaScript
Views	CouchDB views used in app
***tasks**	**map.js**

The procedure to install the prototype in your own CouchDB:

- Unzip pny_prototype.zip to your CouchApp location. For me: D:\Program Files\Couchapp
- Start your command line, and go to your CouchApp directory
- Type in the following command:
  Couchapp push pny_prototype1 http://127.0.0.1:5984/pny_prototype1
  This will create the database pny_prototype1 in your Couch, with a design document "_design/pny_sm" (See Figure 9.1).

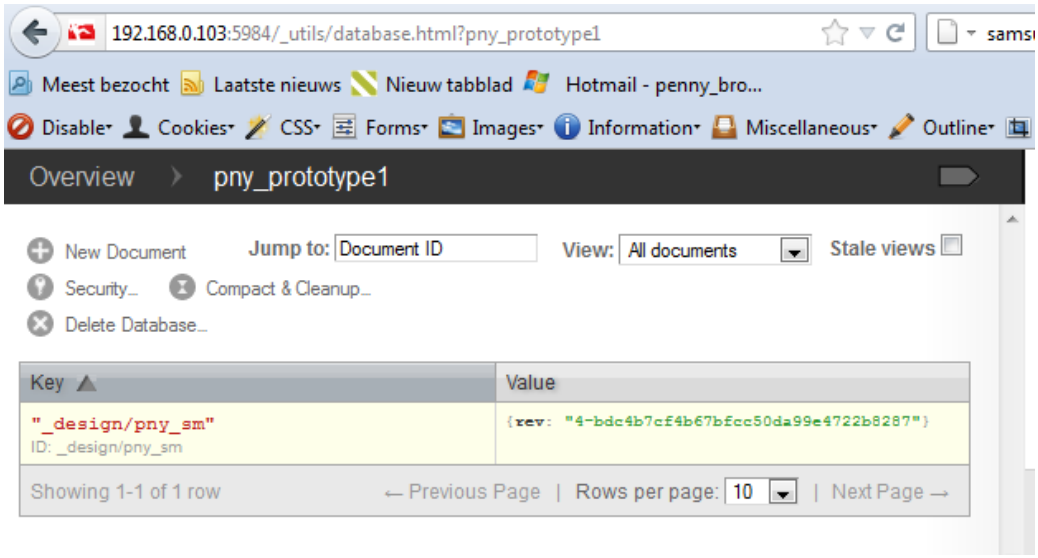

Figure 9.1 Screendump Futon of the database pny_prototype1

- Open your web browser and browse to: http://127.0.0.1:5984/pny_prototype1/_design/pny_sm/SM.html
  NOTE: The URL should match your local CouchDB URL.
- The page should open, and function. No initial activities will be served, as de data/documents in the database are not copied with the app, and the URL in the ViewModel JavaScript, to get the data, may be incorrect.

B.4 Prototype Stage 2: Backgrounds, files and local implementation

Prototype in Stage 2 is mainly aimed to incorporate the OpenLayers functionality in the web app, and use this functionality to show plots, and plot attribute data. The attribute data must be editable to be able to claim a plot, and to update plot characteristics.

The first section (Section B.4.1) contains a list of OpenLayer developer examples, and the specific usage of the examples. The implementation of the Stage 2 prototype on a local environment is described in Section B.4.2.

B.4.1 OpenLayer examples

This section lists a number of OpenLayer example from the OpenLayers site that are of interest for the case study on mobile applications. The application logic in the examples can be used as a starting point in one's one app. The last item mentioned hands some training exercises/course materials to learn the basics of OpenLayers, as well as some more advanced functionality.

- Manually added some GeoJSON polygons
  http://openlayers.org/dev/examples/vector-formats.html
- Offer among others functions to alter size of polygon
  http://openlayers.org/dev/examples/modify-feature.html
  http://openlayers.org/dev/examples/snapping.html
- Styling attribute info
  http://openlayers.org/dev/examples/utfgrid_twogrids.html
- Mobile navigation (padding, zooming)
  http://openlayers.org/dev/examples/mobile-navigation.html
- Add custom controls, eg. drawing/altering polygons
  http://openlayers.org/dev/examples/panel.html
- Rule based styling of features
  http://openlayers.org/dev/examples/style-rules.html
- Offline storage example
  http://openlayers.org/dev/examples/offline-storage.html
- geolocation
  http://openlayers.org/dev/examples/geolocation.html
- Learn OpenLayers workshops
  http://workshops.opengeo.org/openlayers-intro/index.html

B.4.2 Local installation of prototype Stage 2

OpenLayers is successfully incorporated in the CouchApp. However, to be able to better share status on the protyping, and make easy implementation possible, the second prototype was developed without CouchDB, and Couchapp. The GeoJSON data, and other data are read from file, instead of from the database. The files are made accessible via my site www.pny.nl/SakazaMuhinzi/ , secured with a password. This domain does not support the use of JSON, therefore the logic to read JSON from separate files on the server, was displaced with logic to read the incorporated JSON data within the JavaScript files.

Table 9.4 describes the directory structure of the prototype after Stage 2. The following adjustments are made to the environment:

- JQuery updated to version 1.7.2
- Openlayers 2.11: Included "OpenLayers.js", and maps "img" and "theme" in the prototype directory

Table 9.4 Prototype Stage 2 directory structure

DIRECTORY	CONTENTS
	HTML files/pages
	index.html
	SM2(lokaal).html
	Maps(lokaal).html
	Plots(lokaal).html
Bootstrap	The new bootstrap files
/css	bootstrap.css
/img	glyphicons-halflings(-white).png
/js	bootsrap.css
Data	Two datasets of Sakaza Muhinzi
	huyecoffeeplotsEPSG4326.json
	south_districtEPSG4326.json
Img	Three Sakaza Muhinzi logos
	Sakaza2(_smaller/_Xsmall).PNG
Js	JQuery, OpenLayers, and KnockOut JavaScript
	geolocation3B.js
	jquery.js
	OL_geojson3B.js
	OL_geojson4B.js
	ViewModelAtt.js
	ViewModelDataKO.js
	ViewModelMulti.js
	ViewModelMulti2.js
Knockoutjs	The knockout JS library
	knockout-2.1.0beta.js
openlayers	Openlayers files
/css	**PNY_OL_style.css**
/js/OL	Openlayers.js
/img	OpenLayer images, eg controls
/theme/default	OpenLayers style files
/theme/default/img	Openlayers default images, eg control buttons

Actions to implement prototype Stage 2 locally:

- Unpack the file "SM2 Prototype 2 20120610.rar" into any directory on your pc
- In the directory SM/Sakaza Muhinzi/ you will find the HTML files. Open the SM2lokaal, or Mapslokaal/Plotslokaal file to run prototype Stage 2.
  NOTE: the "lokaal"-files use the Google Map API for your localhost, and can thus be used from any pc. The other files, except index.html, which is without API, use a Google Map API only available for www.pny.nl.
  NOTE2: the SM2(lokaal) files read about 7,500 plots from the JavaScript file "OL_geojson3B.js". As these plots are not from the database, no filter logic is applied by running the file, and it can take a while to finish. A warning that "Openlayers.js" not react can be ignored, by pushing the continue button.

B.5 Prototype Stage 3, the files

In Stage 3, the prototype was enhanced with the loop to fetch "Farmer A" plot data from the database, alter the farmer plot attribute data in the app, and save the updated data to database. The code logic to change the attribute data from the selected plot only, was not completed, as was also the case with code to claim a plot. The structure of the files of the CouchApp is described in Table 9.3. In bold the adjusted/added files of prototype Stage 3. Implementation, as described in Section B.3.1. The prototype files, including HTML, JavaScript, and CSS, are in the ZIP file: pny_prototype3.zip.

Table 9.5 The CouchApp file structure of prototype Stage 3

_attachments	HTML files/pages
	SM2.html
	Plots.html
***img**	Images used in app
***style**	Bootstrap CSS files
evently + sub directories	Track changes, data.js, mustache.js, query.json
Lists	CouchDB lists used in app
Shows	CouchDB shows used in app
Updates	CouchDB updates used in app
Vendor	
***couchapp**	
***_attachments**	Couchapp JQuery JavaScript
	Loader_SM2.js
	Loader_SM3PlotsSM3.js
***assets\js**	Bootstrap, and KnockOut JavaScript
	OL_geojson3B.js
	ViewModelDataKOsave_CA.js
	OL_geojson4BSM3.js
	ViewModelAttFSM3.js
	bootstrap.js
***OL**	OpenLayers files
	OpenLayers.js
/img	**OpenLayers images**
/theme	**OpenLayers themes**
***evently**	Log in/out functions
***lib**	Couchtapp/CouchDB JavaScript
Views	CouchDB views used in app

B.6 Evaluation Sakaza Muhinzi prototype

Some of the more technical findings of the Sakaza Muhinzi prototype are listed under the subjects; Bootstrap navigation, GoogleMaps geolocation, KnockOutJs functions, and OpenLayers maps. These findings are listed here to provide an overview of issues that may arise during the development of a mobile app.

Bootstrap navigation

- The approach to develop a HTML5, JavaScript, CSS3 web app leads to the app being available in any web browser. Web browsers, however, have different support for HTML5 and CSS. The same page can act differently running in different web browsers. The prototype was tested with Firefox on a Windows 7 64-bits laptop, and an Asus Transformer tablet (Android 4.0.3). Firefox Beta and the pre-installed internet browser were used to test the prototype on a Samsung Galaxy SII (Android 2.3.5) smartphone. The Iphone iOS results were obtained using Iphonetester[41] in the Chrome web browser on a Windows7 laptop.
  The mobile navigation bar at the bottom was moving up during scrolling in my Samsung Galaxy SII Firefox Beta browser, as well as in the preinstalled internet browser. The navigation bar is placed correctly, after stopping the scrolling. The iPhone in the Chrome web browser does not show this behavior, neither Firefox on my laptop, nor the web browser on the Asus Android tablet.
- Responsive Bootstrap navigation did not entirely fit my idea of navigation. The navigation bar collapsed for smaller screens, and I would have liked to see four fixed buttons in a row at the top or the bottom of the phone screen.
- The HTML can be divided using sections with ids. As such, it is possible to have the app look like a multipage website, as in fact the logic is concentrated into one HTML file. An example of a one-page URL is SM.html#Weather, which, when pushing the navigation button "Weather", should activate the Weather section.
- The AccuWeather widget is not visible on phones, as the navigation bar is hidden. A second widget could be placed somewhere in the app for phone users, however, the widget only worked for one of the widgets, namely, the first widget loaded.
- The sidebar is hidden for phones, as normal behavior of Bootstrap puts the sidebar on top of the page just after the top navigation bar in case this top bar is not hidden. This standard behavior is not attractive for smartphone users.
- The buttons of the phone navigation are not placed in a row of four, but are placed two in a row, in the first version, and three in a row, plus the fourth in the second row in the second version. Additional logic should be implemented to have the smartphone navigation in an acceptable style.

Google Maps geolocation

- Both on my phone and the tablet, the location information pop-up screen, to allow for passing this location information, did not pop up automatically. A white map appeared. Phone and tablet browser settings had to be adjusted to allow sharing of location information. In the browser settings, the activate location checkbox had to be checked. The telephone and tablet settings for "my location" must also be activated, i.e., at least one of the different ways to determine the location.
- The time to load the page on the laptop is more than five seconds. An important part of this duration is the loading of Google Maps, and AccuWeather information from external sources.

[41] http://iphonetester.com/

KnockOutJs functions

- Working with KnockoutJS asks for good understanding of JavaScript. Most examples on the KnockOut-framework are with simple list data, and one ViewModel. Logic to have more ViewModels in one page, and work with hierarchical data, is included in the prototype. The solution is in the binding to a section. The separate ViewModels are all binded to a certain section:

```
ko.applyBindings(new firstViewModel, $("#firstSection")[0]);
ko.applyBindings(new SecondViewModel, $("#secondSection")[0]);
```

- To read data from CouchDB into the web app, views in CouchDB are needed, to allow picking only those documents that are needed for the purpose, and for a specific user.
- Working with a NoSQL database (CouchDB) is quite different from working with a relational database. Flexibility and storage in JSON format are nice advantages. Difficulties are querying, and updating hierarchical data. KnockOutJS seems to overcome the difficulties of updating hierarchical data.
- The GET-statement to fetch task data from the database is hardcoded in the ViewModel, and needs to be associated with a hardcoded IP address and port number. This should work with some variable, so it only has to be set once, and then be available throughout the app. The same is true with userid, name, and log-in information, at the local level.

OpenLayers maps

- The standard OpenLayers navigation controls work fine for pc or laptop. On the mobile or tablet, the touchscreen navigation is available, and preferable over the relative small OpenLayer controls.
- The navigation controls of OpenLayers and the Google Map logo, and text, are positioned on top, see Figure 9.2 The navigation bar is at a lower level. In Firefox on the laptop, only the Google logo, and text are above the navigation bar. Using the dropdown button in the telephone shows also the dropdown menu at a lower level than the OL controls.

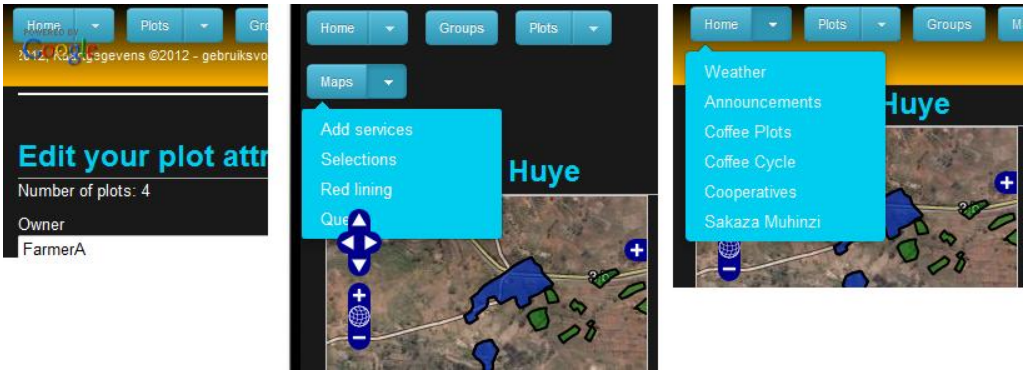

Figure 9.2 Navigation controls and Google logo on top of navigation

- The EPSG 4326 data sets can easily be plotted on Open Street Map, OpenLayers WMS, as well as on Google APIv2. The polygons, however, do not seem to be properly projected on the Google Maps layer. This may be caused by Google using a pseudo-Mercator projection, which causes systematic drift. A second dataset shows less drift, however the plots are still not exactly in the right place (see Figure 6.15). The OpenLayers WMS, and Open Street Map layer do not offer any added value to the information on the map, as they offer not much detail for the Huye district. The Google Maps layers are still preferred, despite the systematic drift.
- Attribute data of the geometry features can be displayed using a "Select Feature" control. The attribute data must be attended to the geometry using the properties list. Logic must be added to make the selected polygon data editable. The attribute data of the selected plot shown is the OpenLayers version of the data, and not the GeoJSON monitored in the back by KnockOut observables. A link must be created based on the plot id, that replaces the OpenLayers plot data with the selected attribute data monitored by KnockOut. The table with attribute data must be filtered based on the plot id, and only show the information of the selected plot.
- The properties of the GeoJSON features can be displayed, for example, in an editable table format, as in Figure 6.13. Tables do not seem to be the best format on a small smartphone screen. On the other hand, an editable table is an acceptable manner for updating attribute information on farmer plots.

Appendix C Cookbook contributions

A last step in the SMAD design, and development quadrant, is the modification of the Cookbook, with the pieces of code that can be reused in other projects. The main contributions to the Cookbook, besides the setting up of the environment are:

1. Fetch JSON features in an OpenLayers layer (OpenLayers)
2. Select plots, and display attribute values (OpenLayers)
3. Fetch, edit, and save the farmer attribute data (KnockOutJS)

This code is part of Plots.html of the Stage 3 prototype. The first two contributions are JavaScript snippets from OL_geojson4BSM3.html, and are described in Sections C.1 and C.2. The example code to fetch, alter, and save data to CouchDB consists of HTML, and JavaScript, see Section C3. This JavaScript is from file: ViewModelAttFSM3.js.

In Section 4.5.5 an exemplification is given on the Cookbook contributions.

C.1 Fetch geoJSON feature collection from CouchDB into OpenLayers

```
// Fetch farmer plots from CouchDB, and add layer with StyleMapFarmer

            var defaultStyleFarmer = new OpenLayers.Style(
            {fillColor: "blue", fillOpacity: 0.5, strokeColor: "black"});
            var selectStyleFarmer = new OpenLayers.Style(
            {fillColor: "white", fillOpacity: 0.8, strokeColor: "blue"});
            var styleMapFarmer = new OpenLayers.StyleMap({"default": defaultStyleFarmer,
                                    "select": selectStyleFarmer});


         var vector_layer_farmer = new OpenLayers.Layer.Vector("Farmer coffeeplots"
            , {styleMap: styleMapFarmer}
            );
         map.addLayer(vector_layer_farmer);
         //console.log(vector_layer_farmer);

         OpenLayers.Request.GET({
                url: "http://127.0.0.1:5984/data_sm/_design/pny_sm/_view/farmerplots",
                headers: {'Accept':'application/json'},
                success: function (req)
                         {
                         var g = new OpenLayers.Format.GeoJSON();
                         //console.log(req.responseText);
                         var data = jQuery.parseJSON(req.responseText);
                         console.log(data);
                         var features = data.rows[0].value;
                         var feature_collection = g.read(features);
                         //console.log(req.responseText);
                         //var feature_collection = data.rows[0].value;
                         //vector_layer_farmer.destroyFeatures();
                         vector_layer_farmer.addFeatures(feature_collection);
                         }
            });
```

C.2 Select a plot, and display the attributes of the plots

```
// select Control
            selectControl = new OpenLayers.Control.SelectFeature(
                [vector_layer, vector_layer_farmer],
                {
                    clickout: true, toggle: false,
                    multiple: false, hover: false,
                    toggleKey: "ctrlKey", // ctrl key removes from selection
                    multipleKey: "shiftKey" // shift key adds to selection
                }
            );

            map.addControl(selectControl);
            selectControl.activate();

            vector_layer_farmer.events.on({
                "featureselected": function(e) {
                var area = e.feature.geometry.getGeodesicArea();
                var centroid = e.feature.geometry.getCentroid();
                var vertices = e.feature.geometry.getVertices(e.feature.geometry.nodes);
                showStatus(
            "<table> <thead> <tr>"+
            "<th>plot id</th>"+
            "<th>centroid</th> <th>area</th>"+
            "</tr> </thead>"+
            "<tbody> <tr>"+
            "<td>"+e.feature.attributes.id_plot+ "</td>"+
            "<td>"+centroid+"</td> <td>"+area.toFixed(2)+"</td>"+
            "</tr> </tbody> </table>"
                        );

            /* KO Select feature and claim/edit attributes of selected feature */
            var Selected = e.feature;
            //console.log(Selected);
```

```
//console.log(Selected.data);

var AttModelSelectFarmer = function(){
    this.owner = ko.observable(Selected.data.owner);
    this.plotname = ko.observable(Selected.data.plotname);
    this.coffeeType = ko.observable(Selected.data.coffeeType);
    this.status = ko.observable(Selected.data.status);
    this.description = ko.observable(Selected.data.description);
    //console.log(this.owner);
};


        // KO uses selected feature data!!
    var viewModelAttSelectFarmer = new AttModelSelectFarmer();
    ko.applyBindings(viewModelAttSelectFarmer, $("#DataAttSelectFarmer")[0]);

    },
    "featureunselected": function(e) {
        showStatus("unselected feature "+e.feature.attributes.id_plot);
    }
});
```

C.3 Fetch farmer plot attribute data from CouchDB, alter, and save to CouchDB

C.3.1 HTML

```
<!-- PNY: DataAtt is section that binds data from ViewModel JS: ViewModelAttFSM3.js  -->

<section id="DataAttF">

    <h2> Edit your plot attributes </h2>
    <p>Number of plots: <span data-bind='text: AttF().length'> </span> </p>
    <p>Couch document id: <span data-bind='text: DocID'></span> </p>

    <table data-bind='visible: AttF().length > 0'>

        <thead>
            <tr>
                <th>Plot</th>
                <th>Owner</th>
                <th>Plot name</th>
                <th>Coffee Type</th>
                <th>Status</th>
                <th>Description</th>
            </tr>
        </thead>
```

```
<!--        <tbody data-bind='foreach: attsF'>   -->
            <tbody data-bind="foreach: AttF(), visible: AttF().length > 0">
            <tr>
                <td><span data-bind='text: properties.id_plot'></span></td>
                <td><input data-bind='value: properties.owner' /></td>
                <td><input data-bind='value: properties.plotname' /></td>
                <td><input data-bind='value: properties.coffeeType' /></td>
                <td><input data-bind='value: properties.status' /></td>
                <td><input data-bind='value: properties.description' /></td>
            </tr>
        </tbody>
    </table>

<!--    button click: save -->
    <button data-bind='click: save'>Save</button>

</section>
```

C.3.2 JavaScript

```
function viewModelAtt() {
    var self = this;
    self.CdbDoc = ko.observable();
    self.AttF = ko.observableArray([]);
    self.DocID = ko.observable();

// initial data

    var CDBfarmerPlots = jQuery.parseJSON(
        $.getJSON('http://127.0.0.1:5984/data_sm/_design/pny_sm/_view/farmerplots',
        function(data){
//      console.log(data);
        CDBfarmerPlots = data;
//      console.log(CDBfarmerPlots);
        var mappedAtts = data.rows[0].value.features;
//      console.log(mappedAtts);
        self.AttF(mappedAtts);
        var mappeddocId = data.rows[0].value._id;
//      console.log(mappeddocId);
        self.DocID(mappeddocId);
        var mappedCdbDoc = data.rows[0].value;
//      console.log(mappedCdbDoc);
        self.CdbDoc = mappedCdbDoc;

}
));
```

```
     // ... Save data

    self.save = function() {
    //console.log(viewModelAtt);
    var docId = self.CdbDoc._id;
    console.log("http://127.0.0.1:5984/data_sm/"+docId);
    console.log(ko.toJSON({CdbDoc: self.CdbDoc}) );
           $.ajax("http://127.0.0.1:5984/data_sm/"+docId, {
                data: ko.toJSON(self.CdbDoc),
                type: "PUT", contentType: "application/json",
                success: function(result) { alert(result) }
            });
        };
    }

ko.applyBindings(new viewModelAtt, $("#DataAttF")[0]);
```

C.3.3 CouchDB view definition

```
function(doc) {
if (doc.doc_type == "farmerPlots"
   && doc.owner == "FarmerA")
  emit(doc._id, doc);
}
```

C.3.4 Data in database


```
{
   "_id": "c8f066c94084e9cfe131ff59cd0007c8",
   "_rev": "14-e3d9595e70a1e74689efb5527acd286a",
   "doc_type": "farmerPlots",
   "EPGS": "4326",
   "type": "FeatureCollection",
   "features": [
       {
           "type": "Feature",
           "geometry": {
               "type": "MultiPolygon",
               "coordinates":
[[[[29.6791195957939,-2.54008193665849],[29.6791283246092,-2.54008552242252],[29.6791328426185,-
2.54008622264312],[29.6792343198385,-2.54011905539722],[29.6791858453129,-2.54018301029325],[29.6792403523828,-
2.54021957646401],[29.6793826916876,-2.54025311849735],[29.6794735450819,-2.54028055558297],[29.6794371747328,-
2.54038715817058],[29.6794341272834,-2.54046330859455],[29.6792736084067,-2.54046022250068],[29.6792826839028,-
2.54050286942779],[29.6792644893231,-2.54059424630655],[29.6791796804793,-2.54061859351669],[29.6791857190308,-
2.54069474620604],[29.6791402769102,-2.54074347161742],[29.6789494735336,-2.54073123977547],[29.6788010670654,-
2.54073729478586],[29.6786254115809,-2.54070679039959],[29.6785224495198,-2.54065498184714],[29.6784812573147,-
2.54066410968353],[29.6782995520463,-2.54060314327126],[29.6782420223525,-2.54054220792935],[29.6782844576828,-
2.54040514643682],[29.6783087036387,-2.54033813947093],[29.6783208349059,-2.54027112947233],[29.6783572075987,-
2.5401553887868],[29.6783117888351,-2.54010968671536],[29.6782905979352,-2.5400700828005],[29.6783633063835,-
2.53998785772461],[29.6784481272453,-2.53991477381434],[29.6785026493427,-2.53989041907411],[29.678651030869,-
2.53998488363569],[29.6787630758897,-2.54004887863087],[29.6788175987413,-2.54002147783027],[29.6788569865112,-
2.53996056672672],[29.6788799991444,-2.53998189480605],[29.6790647277754,-2.54006418421498],[29.6791195957939,-
2.54008193665849]]]]
           },
           "properties": {
               "distr_name": "Huye",
               "id_plot": "shuye5597",
               "owner": "FarmerA",
               "coffeeType": "C label",
               "status": "status 3",
               "description": "Waiting for new arrivals",
```

```
                "admin": {
                    "size": 400
                },
                "plotname": "Plot A1"
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "MultiPolygon",
                "coordinates":
[[[[29.6781482405972,-2.54010964577567],[29.6780937011397,-2.54020405957483],[29.6780603668996,-
2.54028020240028],[29.6779906863328,-2.54036547425965],[29.6779058857924,-2.54035631487078],[29.6777120832839,-
2.54022528627491],[29.6777999343748,-2.54014611108034],[29.6777878196907,-2.54014610804362],[29.6778574927091,-
2.54009129666232],[29.6778968790136,-2.54003647768775],[29.6779544433743,-2.53995729489014],[29.6780089639716,-
2.53993903226582],[29.6780574128971,-2.53997864301086],[29.6781119108724,-2.54005176178603],[29.6781482405972,-
2.54010964577567]]]]
            },
            "properties": {
                "distr_name": "Huye",
                "id_plot": "shuye5598",
                "plotname": "Plot A2",
                "owner": "FarmerA",
                "coffeeType": "C label",
                "status": "status 1",
                "description": "IMPORTANT!!!",
                "admin": {
                    "size": 800
                }
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "MultiPolygon",
                "coordinates":
```

```
[[[[29.6783358139525,-2.54093517147264],[29.6785023841894,-2.54096262758133],[29.6787174199846,-
2.54096268136394],[29.6788627917839,-2.54098099398548],[29.6790536027159,-2.54096276537449],[29.6791626296922,-
2.54098411493175],[29.6793413274354,-2.54095979115847],[29.6794806509188,-2.54094154963365],[29.6795503104068,-
2.54094156700695],[29.6795593663883,-2.54106341116017],[29.6795744872277,-2.54115479635207],[29.6794563636872,-
2.54117608922058],[29.6792231550461,-2.54117907706761],[29.6790020618568,-2.5411790218518],[29.6787900629745,-
2.54114546235297],[29.6786053214158,-2.54111495568576],[29.6784357262506,-2.54107226859551],[29.6783206472174,-
2.54102959513312],[29.6783085483564,-2.54096562511721],[29.6783358139525,-2.54093517147264]]]]
            },
            "properties": {
                "distr_name": "Huye",
                "id_plot": "shuye5599",
                "plotname": "Plot A3",
                "owner": "FarmerA",
                "coffeeType": "B label",
                "status": "status 2",
                "description": "This plot is owned by the cooperation, and in use by FarmerA",
                "admin": {
                    "size": 200
                }
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "MultiPolygon",
                "coordinates":
[[[[29.6808992591954,-2.54103328412764],[29.6808538275893,-2.54103936494009],[29.6808235647853,-
2.54094188389535],[29.6808144877401,-2.54090532906774],[29.680823578249,-2.54088705503914],[29.6809023260077,-
2.54087793645274],[29.6809386416783,-2.54099369528595],[29.6808992591954,-2.54103328412764]]]]
            },
            "properties": {
                "distr_name": "Huye",
                "id_plot": "shuye5600",
                "plotname": "Plot A4",
                "owner": "FarmerA",
                "coffeeType": "A label",
```

```
                "status": "status 3",
                "description": "Attention needed!!",
                "admin": {
                    "size": 200
                }
            }
        }
    ],
    "owner": "FarmerA"
}
```

C.4 CD with prototypes and thesis report

The CD attached contains the following files:

File	Description
pny_prototype1.zip	See Appendix B, Section B.3.1
pny_prototype2.zip	See Appendix B, Section B.4.2
pny_prototype3.zip	See Appendix B, Section B.5
SM3 html.zip	As pny_prototype2.zip, easy accessible: • SM2lokaal.html* • Plotslokaal.html** Unpack zip on PC, and open the html files to test. Resize the internet browser to see mobile layout
ThesisReport_PennyBroman v1.0.pdf	This report in PDF

*
Open SM2lokaal.html, to see the OpenLayer maps with the Huye District plots, and the District feature layer on Google Maps layers. Including:

- Bootstraps responsive design, and styling
- Activities: KnockOutJS task list example, to add and delete activities
- Map information: Show selected feature information in a table

**
Open Plotslokaal.html to see the OpenLayer maps with part of the Huye District plots , and 4 victional farmer plots. Including:

- Editing attribute data example with KnockOutJS. Function to save is not implemented in this version, but available in the CouchApp in pny_prototype3.zip.
- Show selected feature information in updateable input fields