HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

A Framework for Context-Aware Visitor Guide Systems

Maurizio Tidei

Konstanz, 16th April 2015

MASTER'S THESIS

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

an der

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Fakultät Informatik
Studiengang Master of Science Informatik

Thema: **A Framework for Context-Aware Visitor Guide Systems**

Masterkandidat: Maurizio Tidei, Kapellenstr. 4, 78262 Gailingen

1. Prüfer: Prof. Dr. Marko Boger
2. Prüfer: Prof. Dr. Christian Johner

Ausgabedatum: 16.10.2014
Abgabedatum: 16.04.2015

Zusammenfassung (Abstract)

Thema:	A Framework for Context-Aware Visitor Guide Systems
Masterkandidat:	Maurizio Tidei
Firma:	HTWG / contexagon GmbH
Betreuer:	Prof. Dr. Marko Boger Prof. Dr. Christian Johner
Abgabedatum:	16.04.2015
Schlagworte:	Context-Aware Computing, Mobile Computing, Indoor Navigation, Scala, Swift, Play, Couchbase

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Ehrenwörtliche Erklärung

Hiermit erkläre ich *Maurizio Tidei, geboren am 23.11.1980in Singen*, dass ich

(1) meine Masterarbeit mit dem Titel

A Framework for Context-Aware Visitor Guide Systems

bei der HTWG / contexagon GmbH unter Anleitung von Prof. Dr. Marko Boger selbständig und ohne fremde Hilfe angefertigt und keine anderen als die angeführten Hilfen benutzt habe;

(2) die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 16.04.2015

(Unterschrift)

“Moving on is hard, but inevitable if you don’t want to be the next COBOL programmer."

From the preface to "Play for Scala" written by P.Hilton, E.Bakker, F.Canedo [Hilton, 2014]

Acknowledgements

Lorem ipsum dolor

Contents

List of Figures

Abbreviations

INS	**I**nertial **N**avigation **S**ystem
MEMS	**M**icro**e**lectro**m**echanical **S**ystem
DSL	**D**omain**S**pecific**L**anguage
REPL	**R**ead-**E**val-**P**rint-**L**oop

Chapter 1

Introduction

Mobile Devices are rapidly evolving to very powerful personal devices full of sensors that provide several kind of context information, combined with great computing power in an hand held device.
This level of computing power was reserved for stationary desktop computers a few years ago, or to room filling supercomputers some decades ago. At the same time, the devices are spreading out to more and more people.

Various sensors detecting a whole number of environmental characteristics have been miniaturized

This development empowers a new kind of computing to find it's way out of research laboratories, where such systems ran on special hardware in artificial environments, into our daily lives: [cf. Scoble, 2013] *context-aware computing.*

1.1 The goal of this work

This thesis is about designing and implementing a context-aware mobile system for guiding a visitor through indoor and outdoor exhibitions like parks and a museums.

The goal is to design a flexible software platform that is easily adaptable to fit different exhibitions and runs on commodity hardware.

Several analytic functions

The system consists of two main parts:

The first one is a backend system that allows to define all relevant textual and audio information describing the exhibitions areas and single exhibits using a web interface and to process and visualize collected data.

The second one is the application running on the mobile device of the visitor. The goal is to provide relevant information at the right time to the visitor, thus enhancing his experience, and to collect anonymized sensor data for later analytics.

This thesis is also an experiment about moving on from established languages and platforms to emerging ones: From C-like languages and Java to Swift and Scala, from relational databases to NoSQL, from the Java web architecture to Play, from NFC to Bluetooth low energy beacons.

1.2 Motivation

This work is part of the foundation for the first product of the start-up contexagon GmbH in Kreuzlingen, Switzerland. The company was founded in September 2014 by the author, Maurizio Tidei, and Sascha Lorenz [1] after several conversations with museum directors in the tourist region of the Lake Constance in Summer 2014, who expressed interest in such a solution and encouraged us in pursuing the development of an affordable system based on commodity hardware and a highly configurable software platform.

The first project is a context-aware guide for the World War I exhibition at the Schloss Arenenberg in Salenstein, that is scheduled for August 2015. Contexagon will realize the technical part of the exposition in cooperation with Steiner Sarnen Schweiz AG , a company designing and realizing exhibitions and other visitor attractions since 1997 mainly in Switzerland, Germany, Austria and Italy [Steiner-Sarnen].

1.3 The Approach

Chapter 2 handles the theoretical background and fundamentals of this work. It starts with a definition, categorization and the history of context-aware computing. Since the location context is key for the system that has to be designed, different methods for providing and defining the location context are evaluated. For the evaluation of some new technologies, two technology prototypes are written and described in this chapter, too.

[1] S. Lorenz is currently writing his Master Thesis on Big Data Processing with Apache Spark, e.g. for analyzing visitor interests

The context-aware guide system overall architecture and design will then be discussed in Chapter 3.

In Chapter 4, the design and implementation of the front end, the guide's mobile application written in Swift, is described.

Chapters 5 addresses the system's back end used for modeling the indoor or outdoor site, e.g. an exhibition or park guide, and for running analytics on anonymous visitor movement paths and feedback for the optimization of visitor flows and visitor experience. This part of the system is designed as a web application in Scala and Play, with Couchbase NoSQL

Chapter 6 handles the conclusion and future work.

Chapter 2

Theoretical Background, Fundamentals and Preliminary Work

2.1 Introduction

In this section, the main terms and used technologies are discussed. Several positioning principles and their suitability for providing the mobile guide's location context are described. As Swift is a very young language[1] and thus merits closer consideration, a subchapter is dedicated to Swift and hot it compares to Scala. As preliminary work for the evaluation of the new proximity technology BLE (Bluetooth Low Energy) Beacons, a Swift-Prototype was implemented and is presented in this chapter, too.

2.2 What is Context-Aware Computing?

The term "context-aware computing" was first used by B. Schilit et. al. 1994 in the paper "Context-Aware Computing Applications" [Schilit, 1994]. They introduce this new kind of computing as a "new class of applications that are aware of the context in which they are run. Such context-aware systems adapt according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time".

[1] Swift 1.0 was released on Sep 9, 2014

2.2.1 Definition of Context

Schilit defined context by enumeration of examples that are all related to location and proximity.

Dey and Abowd provided a precise universal definition of context in their paper "Towards a Better Understanding of Context and Context-Awareness" published 1999 [Abowd, 1999]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

While being precise, this definition is too general to get an idea of which kind of context information an application could use. So the following subchapter provides a wider list of context examples and a categorization.

2.2.2 Categories of Context

There are many different types of context:

Computing Context

Some examples are network connectivity, communication costs and bandwidth or available resources like printers, displays and input devices.

User Context

- The user's *identity*, including his age, gender, education, profession, interests. In today's age of social media the user's connections to other people and their identity are important and widely available as well.
- The *location* is the most obvious kind of user context. It can be subdivided in two different sub types of location:
  Logical location, e.g. "at home" or "at work" or slightly different "in a library, restaurant, cinema"
  Absolute location, e.g. "47°40'02.0"N 9°10'19.7"E" as geographic coordinates on earth or 2nd floor room 205 in the F-Building of HTWG Konstanz.
- The *emotional state* indicating if the user is happy, angry, worried etc. At which level is the personal stress level?
- *Fisical state*: Is the user rested or tired?

- The user's *health state*, determined by measuring some vital parameters (e.g. hearth rate, blood pressure and oxygen saturation). Many dedicated low-cost hardware items for this kind of measurements were released in 2014 or are announced for this year.
- Information about other people or mobile devices nearby forms the *co-location* context.
- The current *activity*. For example walking, driving a car, riding a bicycle or jogging. Some of this activities need to be further specificated, e.g. jogging to practice sport or jogging for catching a train.

Time Context

The time context, such as the current time of the day, the day of the week, the current season or the full date.

Physical Context

Some examples for environment information are the current temperature, the weather, the intensity of ambient light, the noise level and even traffic condition.

cf. [Paulo, 2014]

2.2.3 Categories of Context-Aware Applications

Schilit and his co-authors describe the following four categories of context-aware applications:

- Proximate Selection

  Highlighting actions or information based on the current location of the user is called "Proximate Selection". While this user interface technique generally requires a user entering his location manually, context-aware systems default it automatically to the currently sensed location.
  Nowadays, we encounter this behavior in many smartphone applications like weather forecasts for the current city, searching nearby stores in digital yellow-pages or even when performing online searches on non-portable computers.

- Automatic Contextual Reconfiguration

  "Automatic Contextual Reconfiguration" means loading and activating different system configurations based on the current context of use. For example, loading a different digital whiteboard per room gives the illusion of accessing it as if it was physically mounted in that room. But the considered context information is not limited to the location. Changing the energy plan of a notebook based on the connection status of the A/C power cable or the current battery level are further examples for this category.

- Contextual Information and Commands

  This category contains systems providing the right piece of information and offering the adequate actions at the right time fitting the current context.

  Retrieving information provided as text, audio, picture and video form, fitting to the current location context of the user is obviously a key feature of the museum guide that is developed as part of this master thesis.

- Context-Triggered Actions

  Context-triggered actions are applications that execute a defined action when a specific predefined context-state is reached. In contrast to contextual information and commands, these actions are automatically executed. Combining multiple rules allows designing more complex behaviors.

Obviously, nowadays there are many applications that fit in two or more categories. A modern smartphone based navigating system for example offers functions like "Take me to a gas station", displaying a list with the nearest one already preselected (proximate selection). When light conditions change, like when entering a tunnel or when it gets dark, the display is automatically dimmed. The pedestrian mode is an example for two categories: The device senses steps and switches automatically into pedestrian mode considering paths not accessible by car (automatic contextual reconfiguration) and offering commands like "Take me back to my parking lot" (contextual commands).

2.2.4 Ubiquitous Computing

The vision and research field of ubiquitous computing was coined 1991 by Mark Weiser of Xerox PARC in his article "The Computer for the 21st Century" [Weiser, 1991]. Weiser's starts his article with these words: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

For computers, that means being integrated in common objects of daily usage, supporting humans without demanding their full attention.
As an excellent example for a technology that disappeared in the background he uses the vanishing of electric motors. At the beginning, big powerful engines powered a whole fabric, with axles and belts distributing the force to the single machines and workplaces. Later, as motor engineering improved, each machine had its own motor and today several electric motors - miniaturized and bigger ones - are integrated into a single device. Often they work in the background letting us focus on the main task. Already in 1991, at the time of his writing, a typical car had more than 20 integrated motors, nowadays even more. They are used to start the engine, lock and unlock the doors, clean the windshield, open and close the windows, align the rear-view mirrors, adjust the seats by several axis and so on.

Although Weiser didn't use the term context-awareness, he already recognized the importance of the location context: "We have found two issues of crucial importance: location and scale. Little is more basic to human perception than physical juxtaposition, and so ubiquitous computers must know where they are. [...] If a computer merely knows what room it is in, it can adapt its behavior in significant ways without requiring even a hint of artificial intelligence.".

2.3 Sensors Integrated in Mobile Devices

2.3.1 Introduction

Every type of context information has one or more sensors as direct or indirect source. Thus, context-aware computing is only possible with high quality data provided by a variety of sensors.

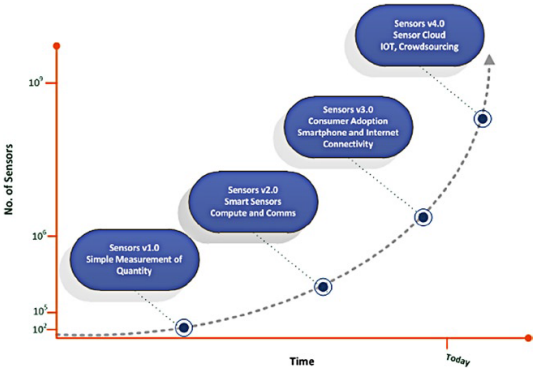

FIGURE 2.1: Evolution of sensors [McGrath, 2013, Ch. 1], with kind permission

According to [McGrath, 2013], sensors had a enormous growth over the last 30 years and, with accelerating improvement and deployment rates. The authors identify sensors as a technology nexus driven, inter alia, by public healthcare, sports and fitness, pervasive communications and the Internet of Things (IOT). In turn, sensors enable innovation in many sectors, so that one sector benefits of improvements made due the demands of another sector.

In the next subsections, the major sensors present in mobile devices are presented.

2.3.2 Micro-Electro-Mechanical Sensors

Micro-Electro-Mechanical Sensors (MEMS) are "three-dimensional, miniaturized mechanical and electrical structures, typically ranging from 1 to 100 µm" [McGrath, 2013, Ch. 2]. They use a movable proof-mass that moves relatively to a frame, electrically measuring the deviation or bending. MEMS are widely used in the automotive industry, where they are part of several systems like the antilock braking system (ABS) or the electronic stability program (ESP). The recent development towards extremely compact, low-cost and power MEMS sensors led to a growing integration in consumer electronics devices like smartphones, tablets and game console controllers.

Accelerometer

Motion sensing can be separated in five modes:

- acceleration
- vibration (periodic acceleration)
- shock (instantaneous acceleration)
- tilt (static acceleration)
- rotation

An accelerometer can measure all of them except rotation.

Combining the measurements for three orthogonal axis, a motion along any arbitrary path in a 3D-space can be reconstructed.

There is one sensed acceleration component always present on Earth: The gravitational acceleration. Near the Earth's surface, this component can be approximated by $9.81\frac{m}{s^2}$. The direction of this component allows to deduce the attitude of the device in respect to the ground using the accelerometer.

Gyroscope

MEMS Gyroscopes are able to measure rotatory motion without relying on rotating parts that would require bearings but using a vibrating proof-mass affected by Coriolis forces.

The following figure shows the three axis of a 3D gyroscope, as it is available in many mobile devices today.

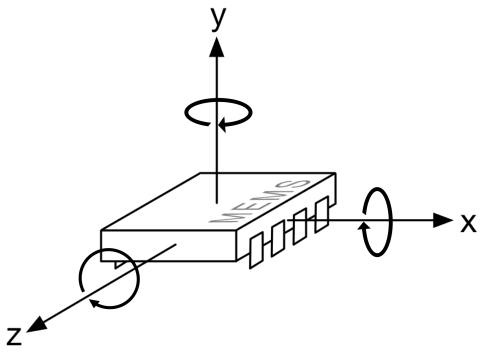

FIGURE 2.2: The rotation axes of a three dimensional MEMS Gyroscope

Barometer

Some of the latest smartphones and tablets are equipped with a MEMS barometer for measuring the air pressure. This information combined with the current weather dependent ground pressure can be used to determine the current altitude or the floor of the building the device is in.

2.3.3 Other Sensors

GPS Sensor

A GPS Sensor is integrated in many of today's mobile devices. It is a standard part of almost every smartphone and an optional part of many tablets, often bound to a cellular connectivity module. For details on the accuracy, see 2.4.2.2.

Magnetometer

The magnetometer acts mainly as digital compass.

Proximity Sensor

The proximity sensor typically measures the distance of the device's front to the next object. It is used to turn off the display when holding a mobile phone at the ear or to determine if the phone is inside a pocket.

Light Sensor

The light sensor measures the intensity of the light. It is primarily used to adapt the screen brightness to the ambient light.

2.3.4 Conclusion

Sensors are evolving rapidly and there are already a broad spectrum integrated in common consumer products like smartphones and tablets. The better the quality of the sensors and the variety of measurable quantities, the more precise and sophisticated can context-sensitive software become. Sensor data from different sensors is often combined. For example, a magnetometer and gyroscope can be combined to a fast reacting improved electronic compass without drifting.

In the next section addresses different localization techniques and discusses how they can be used for the guide system to be designed.

2.4 Localization Techniques

2.4.1 Introduction

As already emphasized, the location context is very important for many context-aware applications and especially for the visitor guide that will be designed and implemented during this work. Therefore a short overview of available positioning techniques and their classification is provided and their applicability to this work discussed.

2.4.2 Classification

According to ([Borenstein, 1997], cited by [Noureldin, 2013, Ch. 1.1]) there are two main categories of positioning techniques, that can be further split up into seven different subcategories.

2.4.2.1 Techniques Using Relative Measurements

This techniques are also known as dead reckoning (DR), and they all need an initial position estimate.

Odometry

By measuring the wheel rotations and the angle of the steer axis, the position is estimated. Due to wheel slippage and measurement errors, the position tends to drift away over time.

Inertial Navigation

Using only inertial sensors (gyroscopes and accelerometers) to measure rotation rates and acceleration forces, the new heading can be estimated integrating the gyroscope data once and the new position integrating the acceleration data twice over time. Pure inertial navigation systems are self-contained and don't need external references.

Measurement and (numerical) integration errors accumulate very fast, especially for the position due to the double integration, requiring high precision sensors for decent results. With low cost sensors, as they are used today in many mobile devices, the position error grows unbound already after a very short period of time.

2.4.2.2 Techniques Using Absolute Measurements

Electronic Compasses

Electronic compasses are not a stand-alone positioning solution, but they provide heading measurement to the magnetic north pole. When the declination angle (the angle between magnetic and geographic north pole) is known for a specific place on Earth, the actual north heading can be calculated. However, local magnetic fields can have a huge impact on the measurements. Near power lines and metallic structures like the structural steel beams in buildings have an impact on the measurement.

Active Beacons

If the environment is already known and beacons are mounted on defined positions, the position of a moving platform can be determined either by triangulation, trilateration or fingerprinting algorithms. Triangulation involves measuring the angles at which at

least two beacons are seen. For a trilateration algorithm, the position is calculated by measuring the distance to the beacons. Fingerprinting needs the preliminary creation of a database of measurements at different positions. The position is then estimated by choosing the position-measurement pair out of the database that best fits the current measurement.

Global Navigation Satellite Systems

Abbreviated GNSS, the most well-known example is GPS (Global Positioning System). The receiver can calculate it's position by performing a trilateration. The input is the difference of the travel time of the different satellite's signals and their current location included in the signal. GPS provides good absolute localization for outdoor environments. Indoor, the signal is either very poor or not available at all.

The GPS accuracy of current consumer tablets and smartphones was examined in [Musulin, 2014]. The authors compared measurements from six different smartphones and tablets with reference coordinates, both with static and moving devices. The average error in static measurements ranged from 2.18 to 4.18 meters with standard deviations of 0.82 to 2.17 meters. In case of moving devices, two devices had better results and four showed higher errors with average deviations ranging from 1.76 to 5.15 and standard deviations of 0.61 to 2.50 meters (ignoring one statistical outlier with an average error of 15.08 meters).

Landmark Navigation

Similar to active beacons, this technique requires a well known environment with distinct natural or artificial fixed objects, whose position, exact size and shape must be well known in advance. Most landmark navigation systems are based on computer vision.

Map-Based Positioning

With this technique, a local map is created using sensors like cameras and laser range finders and compared to global pre-stored map, e.g. a CAD model of the environment. If a part of the global map matches the local map, the position can be determined.

2.4.3 Discussion

A robust and computationally not very expensive technique is needed to efficiently use the limited battery power of mobile devices. Furthermore, that technique must be satisfied by the sensors currently available in common mobile devices alone.

GPS is a good choice for outdoor positioning, being technically mature and widely available on mobile devices. The substantial power drain often associated to the activation of GPS in mobile devices is mostly caused by CPU-intensive navigation software and an active screen rather than the actual sensor.

For indoor navigation, Map-based positioning and landmark navigation can be clearly excluded due to their demands.

Although inertial navigation combined with human odometry is very promising, they require the inertial measurement unit (IMU) to bee foot mounted. The reason is being able to re-calibrate the system and thus the drifting (set the current speed to zero), every time one foot rests on the ground while walking [Angermann, 2012]. The authors are confident, that "projecting the improvements in microelectromechanical system (MEMS)-based inertial sensors to the next ten or even 20 years, it is foreseeable that FootSLAM will no longer require the inertial sensors to be mounted on the foot." Only then will the sensor quality probably be high enough to provide a position estimate with an arbitrarily hold smartphone comparable to the position estimate quality achieved today with a foot mounted IMU.

So this technique is currently not suitable for providing the location for the guide, although sensing the steps and thus the pace of the visitor is a goal to keep in mind.

The most promising technique for inexpensive indoor location are iBeacons (see Ch. 2.5), that fall into the category of active beacons. If the quality of the distance measurement has a limited error, a trilateration can provide the position when signals from two or more iBeacons are received. To determine the capabilities of iBeacons, a prototype will be implemented and described in Chapter 2.5.2.

2.5 Bluetooth Low Energy Beacons - iBeacons

2.5.1 What is an iBeacon?

On Mid 2013, Apple introduced iBeacons as new technology for adding location awareness to applications. iBeacons are little devices containig a power cell and a small logic board

with a Bluetooth Low Energy (BLE) transmitter. BLE was introduced with Bluetooth 4.0 for applications with very contained power requirements that can be satisfied with 1 Mbps data transfer rates, co-existing with standard Bluetooth [BLE].

iBeacon devices broadcast BLE advertisement packets at an configurable time interval (named "advertisement interval"), mostly between 100 ms and 1 s, containing, in essence, a fixed preamble identifying the data as iBeacon advertisement, a set of three configurable device specific numbers and the measured RSSI (Received Signal Strength Indication) in dBm at 1m. The latter value is needed as reference for calculating a distance estimation using the current RSSI measured by the receiving device as input.

Based on the configured advertisement interval and the broadcasting power, that influences the distance until which the beacon ca be detected, the battery is expected to last for up to two years. The broadcasting power determines the energy of the signal and thus the maximum distance until which it can be received.

Apple provides only the specification and an API, but does not manufacture the devices itself. Instead, third-party supplies, among these a number of start-ups, began to ship iBeacons in 2014. For this thesis, different devices were ordered from estimote (New York, USA), kontakt.io (Kraków, Poland), BEACONinside (Berlin, Germany) and Accent Advanced Systems (Barcelona, Spain). This avoids the risk of concentrating on features or flaws of a specific device brand.

2.5.2 The iBeacon Prototype

2.5.2.1 Design and Implementation

To familiarize with iBeacons and the whole iOS platform, an iBeacon prototype "Beacon Scanner" was implemented as preliminary work to the guide framework.

The main screen of the prototype provides an overview of all iBeacons in range.

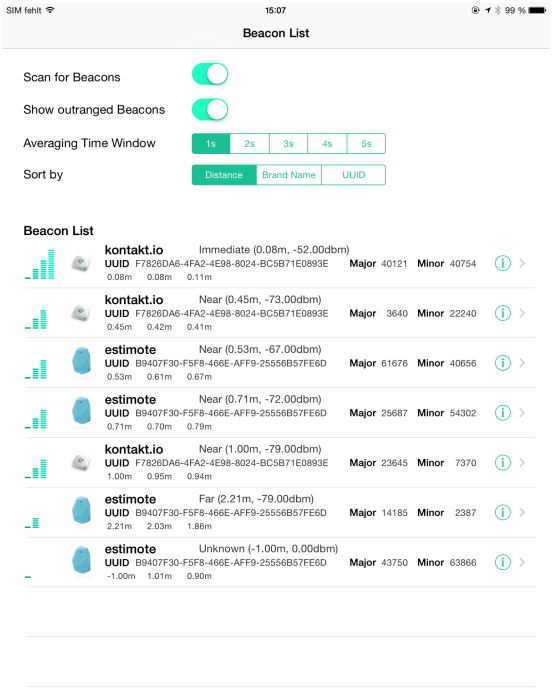

FIGURE 2.3: The main screen of the iBeacon prototype

After selecting a beacon in the list, the beacon detail view opens showing the details and the single measurements for that beacon on the first screen and some statistics on the second screen.

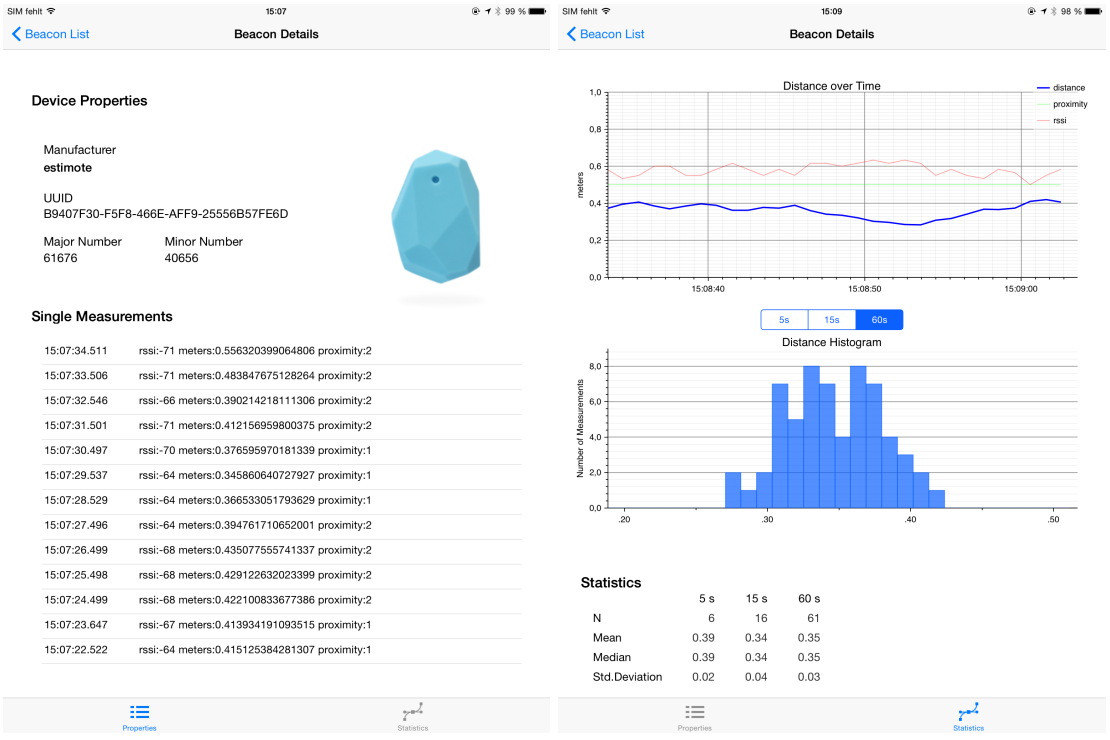

FIGURE 2.4: The iBeacon detail screens, including statistics

2.5.3 Experimental Determination of Trilateration Fitness of iBeacon Measurements

Introduction

Using the implemented iBeacon Scanner prototype, the quality of the distance measurement has to be determined in a series of measurements. The goal is being able to decide if a trilateration based on this technology can provide meaningful position for indoor guides.

Experimental Setup

To collect iBeacon measurements at different distances, the floor is prepared with ten equidistant marks at 1-10 meter distance and additional two marks at 15 and 20 meters.

Both the beacon and the measuring device, an iPad mini 2 with iOS 8.1 and the iBeacon Scanner prototype, are mounted on an tripod. There are no obstacles obstructing the line of sight between sender and receiver. The tripod with the beacon remains fixed, while the tablet's tripod is moved from floor mark to floor mark, starting at the 20 meter mark and finishing with the tablet and beacon touching. At each distance, the arithmetic mean of the measurements collected during 15 seconds is calculated.

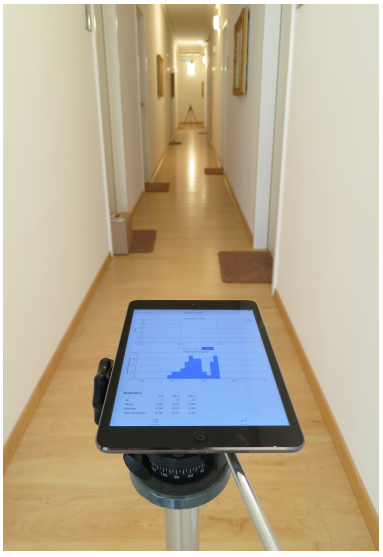

FIGURE 2.5: The tripod mounted iPad headed towards the sending beacon

This procedure is repeated with three different beacons:

- estimote beacon with 100ms advertising interval and +4dBm (highest setting) broadcasting power setting

- kontakt.io beacon with 100ms advertising interval and -12dBm (medium) broadcasting power
- estimote beacon with 950ms advertising interval and -12dBm (medium) broadcasting power

Measurements

The following table contains all the measurements for the three different beacons. For each distance, the arithmetic mean of the measured distances along with the sample's standard deviation (σ) and the median of the proxymity class (Prox.). The unit is always meters for are distance related numbers. For the proximity class, 3 stands for far, 2 for near and 1 for immediate [Proximity].

	estimote 100ms			kontakt.io 100ms			estimote 950ms		
Real Distance	Mean	σ	Prox.	Mean	σ	Prox.	Mean	σ	Prox.
20	13,29	1,01	3	5,10	0,38	3	3,16	0,00	3
15	7,65	0,91	3	4,94	0,22	3	2,89	0,19	2
10	11,65	1,00	3	3,18	0,09	3	2,27	0,18	2
9	7,64	0,13	3	1,94	0,06	2	1,64	0,18	2
8	7,77	0,72	3	2,82	0,16	3	2,91	0,08	2
7	9,13	1,85	3	2,36	0,03	3	2,51	0,03	2
6	11,95	1,18	3	2,10	0,04	2	1,50	0,19	2
5	5,11	0,52	3	2,06	0,04	2	1,14	0,14	2
4	3,05	0,17	3	1,83	0,10	2	0,68	0,12	2
3	2,68	0,25	3	2,02	0,06	3	0,67	0,06	2
2	2,20	0,21	3	0,90	0,01	2	0,86	0,16	2
1	1,44	0,17	2	1,04	0,04	2	0,11	0,02	1
0	0,07	0,03	1	0,40	0,02	2	0,04	0,01	1

Although iOS provides one data update per second, some of the 15 updates received during the 15 seconds can be null in case no signals were received. This occurs more often with higher advertisement interval or at greater distances.

FIGURE 2.6: Beacon measurements with gaps at 15m distance

The following diagram visualizes the mean distance plotted for every beacon.

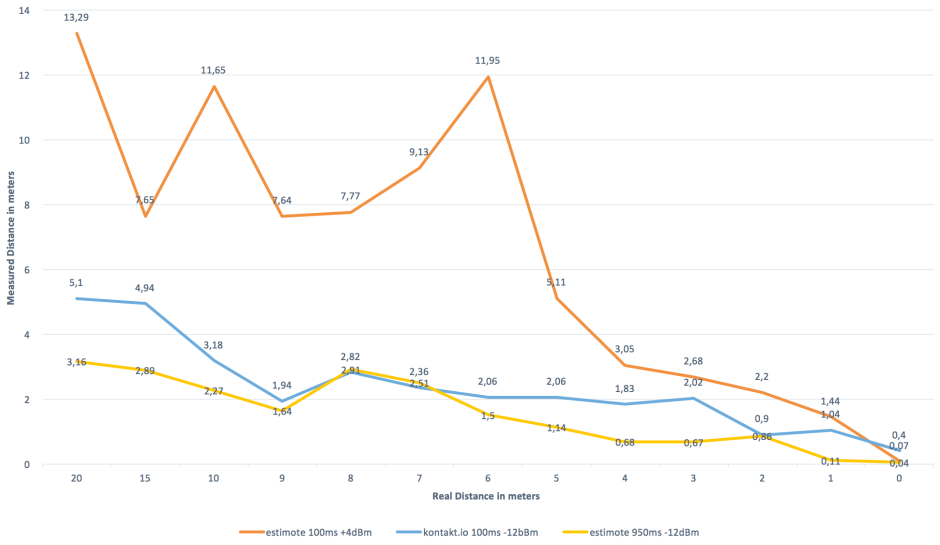

FIGURE 2.7: Plot of the beacon distance measurements

Insights and Conclusion

The data shows that the distance measurements based on the received signal strength of BLE beacons do not match the real distance. The measured distance does not even decrease continuously with decreasing actual distance. Generally, the correlation of the measured value and the real distance is too weak to serve as a basis for a precise trilateration, despite perfect conditions.

Regarding other complications like the dependency of the RSSI to the orientation of the device and signals obstructed by the human body beacons can be regarded as definitively unsuited for trilateration based positioning.

The proximity class is not consistent with all beacon configurations, neither. However, excluding the third (high advertisement interval) beacon, the measurements of distances up to two meters are lower than the ones at 3-25 meters. So we can deduce vicinity up to 2 meters to a beacon by setting a known threshold that depends on the used beacon's configuration.

Despite iBeacon trilateration is not possible, marking single areas of interest (e.g. rooms, showcases, pictures) using one or more low-cost beacons based on the area's size is a promising way for triggering guide actions indoors with a low error rate.

2.6 Geographic Coordinates

2.6.1 Introduction

Since we need to work with GPS coordinates on the mobile front end and use geographic coordinates to model sites on the back end, it is fundamental to understand how the geographic coordinate system works and what kind of problems we may encounter.

2.6.2 Standards

There are many different systems for specifying a precise point on the Earth's surface and describing them all would go far beyond this thesis. The main problem is that the world has a complex shape. Even ignoring tides and winds the sea level does not fit an ellipsoid due to gravity anomalies over the Earth's surface. The surface that most closely approximates sea level is the geoid.

But for performance and storage reasons an ellipsoid is used in many applications including GPS. One widely used standard is the WGS (World Geodetic System) defining a coordinate system and an ellipsoid. The most recent revision is WGS84, which is exactly what GPS uses and what is used in this work. The ellipsoid can be 106 meters above and 85 meters below the geoid (cf. [Geoid-Grace]). The following diagram shows the difference in meters between the geoid and the ellipsoid.

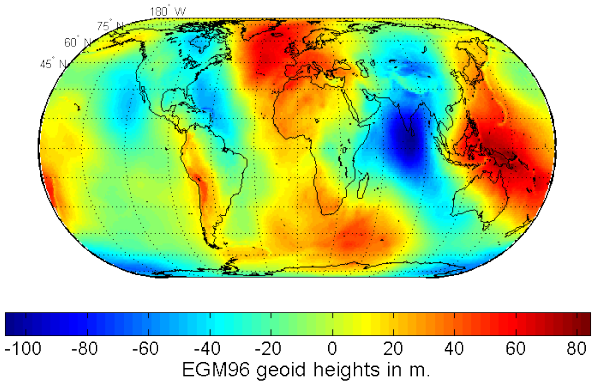

FIGURE 2.8: Difference in meters between the geoid and the ellipsoid [NASA-Geoid]

The city of Konstanz, for example, is 46.62 meters above the ellipsoid.

2.6.3 WGS84 Coordinates

A specific point on Earth's surface is defined by a Cartesian coordinate pair consisting of the latitude, that specifies the north-south position and the longitude for the east-west position.

The latitude is defined to be 0° at the equator and goes up to +90° at the geographic North Pole and down to -90° at the South Pole. The longitude, lacking a natural zero point, is measured relatively to the prime meridian, where the longitude is defined to be 0°. The prime meridian of WGS84 is the line of longitude on the geoid starting/ending at the geographic North/South Pole and passing trough a well defined point 102 meters to the east of the Royal Observatory, Greenwich, London. Eastwards the longitude goes up to 180°, to the west it goes down to -180°, meeting at the same meridian - the one opposite to the prime meridian.

For this work, longitude and latitude are represented by decimal fractions.

The maximum resolution using 6 fractional digits depends on the position on Earth, more specific on the latitude, because all meridians converge at the poles. For the WGS84 ellipsoid, the formula

$$R_\phi = \frac{a \cos \phi}{(1 - e^2 \sin^2 \phi)^{1/2}}$$

can be used to calculate the radius of the circle of latitude at a given latitude ϕ [cf. Noureldin, 2013, Ch. 2.4].

The length of one degree of the circumference of that circle of latitude is then calculated with

$$\delta_\phi^1 = \frac{2\pi}{360} * R_\phi$$

Solving with latitude $\phi = 47,6$ for Konstanz, equatorial radius $a = 6,378,137.0m$ and eccentricity $e = 0,08181919$ as defined in WGS84, the solution is 75.200 km.

So in Konstanz, 0,000001 degree of longitude corresponds to 7,52 cm. The biggest distance, and thus the lowest resolution, is on the equator, where it amounts to 11,13 cm.

The distance of 0,000001 degree of latitude is almost constant at about 11,1 cm.

2.6.4 Using different Mapping Providers

One of the most well-known provider for maps is Google Maps, offering a web interface and an API for the integration in own web applications. For iOS development, the native solution is the Apple Map Kit framework, that uses the own Apple Maps service[2].

The maps images at the closer zoom levels are in most cases taken by a camera mounted on a plane. This camera often does not look perpendicular to the ground but with a gentle angle, so that you can see the building facades. While this results in a nice isometric-like perspective, it introduces a problem one has to be aware of: Higher parts of the image like higher floors or small hills appear shifted to a certain direction.

An example: To check the consistency of Google and Apple maps, I initially used the roof window of the office in which the biggest part of this thesis was written. The coordinates of this window on both maps showed a notable discrepancy of several meters. The cause is the different camera angle of the two pictures.

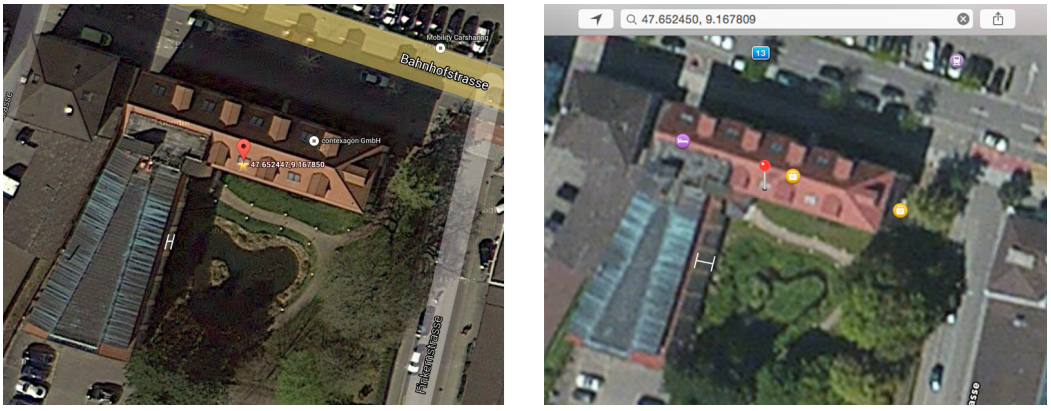

FIGURE 2.9: View angle error on Google (left) and Apple (right) maps

This results in a latitude shift from Google to Apple Maps of $+3 * 10^{-6}$ degrees and a longitude shift of $-4,1 * 10^{-5}$ degrees. Using the results of the previous section translates in a deviation to the nord of 0,33 m and a considerable deviation to the west of 3,08 meters.

To exclude this view inclination error and get the pure ground coordinate discrepancy of the two mapping solutions, the coordinates of a small distinctive fixed ground object (one of the white lighting poles on the borders of the path in the park) are measured in both mapping solutions.

The measured pure deviation amounts to not negligible $-1,4 * 10^{-5}$ degrees of longitude or 1,05 meters.

[2]Until iOS 5.1, Apple used the Google Maps service and the decided to start it's own competing product

It is important to keep in mind all this different kinds of errors discussed in this section during development and testing of the guide, to avoid searching for errors at the wrong place.

2.7 Swift

2.7.1 What is Swift?

Swift is a modern programming language released by Apple in 2014. In [Apple, 2014a] Apple introduces Swift as "a new programming language for iOS and OS X that builds on the best of C and Objective-C, without the constraints of C compatibility".

Since Swift is a cutting-edge language, this section is dedicated more attention as one would normally do for a implementation language inside a thesis.

Developers already familiar with the well designed functional programming language Scala will recognize several concepts. It has a concise Syntax avoiding big parts of boilerplate code and syntactic noise, supports functional programming and is statically typed.

The following section is an overview comparison of Scala and Swift features, created during the familiarization with Swift for this thesis. Language features marked will a * will be discussed separately in section 2.7.3.

2.7.2 Comparison Scala and Swift

Language Feature	**Scala**	**Swift**
Type inference	yes	yes
Line end separates commands (no need for semicolon)	yes	yes
Implicit type conversions *	yes	no
Default access levels (access level has to be only provided if it differs from default)	yes	yes
Functions are first class types[3]	yes	yes
Closures	yes	yes
Curried functions	yes	yes

[3]Functions can be passed to functions, returned from functions, created at runtime and assigned to variables

Language Feature	**Scala**	**Swift**
Operator functions	yes	yes
Named parameters *	yes	yesAs fixed part of the method's signature
Optionals *	via `Option[T]` class	widely used, dedicated Syntax
Switch with pattern matching	yes	yes
String interpolation	yes `"Hello $nameVar"`	yes `"Hello \(nameVar)"`
Keyword for variable definition	var radius = 1	var radius = 1
Keyword for constant definition	val pi = 3.14	let pi = 3.14
Array literal	`Array(1,2,3)`	`[1,2,3]`
Map literal	`Map(1->"a", 2->"b")`	`[1:"a", 2:"b"]` [4]
If condition must be boolean	yes	yes
Tuples	yes, but without named elements	yes
Ranges	`for i <- 0 to 4` `0 until 4`	`for i in 0...4` `0..<4`
Constructor	def this()	init()
Extended getter/setter concept *	yes, no observers	yes, very flexible concept including observers (willSet() didSet() events)
Interfaces	trait	protocol
Extension of existing types	yes	yes
Struct	no	yes
Enum	via extending the Enumeration class	yes, dedicated keyword
"Any" Type	Any, AnyVal, AnyRef	Any (instance of any type, even function types) AnyObject (instance of any class type)

[4] Maps are called dictionaries in Swift

Language Feature	**Scala**	**Swift**
Qeury an instance of a type by a key in brackets, like arrays or maps	obj(index) calls obj.apply method obj(index) = newValue calls obj.update(0, newValue)	`subscript(i:T) -> T2 { get {...} set(newValue) {...} }`
Memory Management	JVM Garbage Collection	Automatic Reference Counting
Nested Functions	yes	yes
Generics	yes	yes

2.7.3 Major Differences to Scala

In this section, the major differences between Scala and Swift that are encountered during the first Swift projects are described.

2.7.3.1 Implicit Type Conversions

In Swift, every type conversion has to be explicit. Even when using different numerical types inside an arithmetic expression, the conversion is not done automatically, in contrast to Scala and even Java. So this code yields a compile time error for example:

```
let a = 1.0
let b = 2
let c = a + b // compiler error: cannot invoke '+' with an
    argument list of type '(@lvalue Double, @lvalue Int)
```

To get an integer 3 assigned to the variable 'c', you need to cast 'a' to Int. For a decimal 3.0, the variable 'b' has to be cast to Double.

```
let c = Int(a) + b     // ok, c is 3 Int
let d = a + Double(b) // ok, d is 3.0 Double
```

Although initially it can be a bit frustrating running into compile errors of this kind, you get used to explicitly casting the values to the desired types fast. The advantage of this approach is that the developer explicitly sees the type of the resulting value without having to remember language specific rules.

In contrast, Scala has a powerful implicit type conversion system. Combined with operator overloading it enables developers to create beautiful internal DSL (Domain Specific Languages) that read more like natural language. [5] But, as M. Odersky rightly wrote in

[5] A good example for using implicit conversion to build a DSL query language can be found at [Scala-DSL-Example]

[Odersky, 2010, Chapter 6.13], "... bear in mind that with power comes responsibility. If used unartfully, both operator methods and implicit conversions can give rise to client code that is hard to read and understand.".

2.7.3.2 Optionals

Variables are non nullable by default in Swift. So the following code will not compile:

```
var name = "Swift"
name = nil // compile time error: Type 'String' does not conform
    to protocol 'NilLiteralConvertible'
```

As a consequence, the variable can safely be accessed at any time without the danger of a NullPointerException respectively a nil runtime error.

To allow a variable to assume the value of nil (the null equivalent in Swift), it's type has to be defined as optional by the '?' postfix to the type name.

```
var name:String? = "Swift"
println(name)  // prints 'Optional("Swift")' to the console
println(name!) // prints 'Swift'
name = nil     // ok
let statement = name + " is great" // compile time error: value of
     optional type 'String?' not unwrapped
println(name!) // runtime error: unexpectedly found nil while
    unwrapping an Optional value
```

The first println statement outputs 'Optional("Swift")' because the string value is wrapped inside the optional and needs to be unwrapped using the '!' postfix. Note that trying to unwrap an optional without value yields a runtime error.

A convenient way of querying properties or calling methods on optionals is called optional chaining. Imagine a circle object with an optional custom style that may have a border, which in turn has an optional custom color. To check the existence of the custom border color, instead of

```
if circle.style != nil && circle.style.border != nil && circle.
    style.border.color != nil
```

it is possible to write

```
if circle.style?.border?.color != nil
```

If any link in this chain is nil, the whole chain fails gracefully and returns nil.

Another concept in the context of optionals are failable initializers (known as constructors in other languages). When explicitly defining an initializer as failable appending a

question mark (init?), it's return type is optional variant of the type it should initialize. That can be useful for handling invalid parameter values or other initialization problems.

Optionals are widely used in Swift's iOS APIs and their syntax one of the first things noticed when looking to Swift sources as a novice. In my opinion, the usage of optionals results in beeing more aware of the presence or absence of values and coding more prudently. Of course, optionals can only add real value if not blindly unwrapped just to silence the compiler errors.

2.7.3.3 Getter and Setter

Swift has a well designed flexible property system, with a concise getter and setter syntax thanks to built-in language support.

It addresses the two main problems, encapsulation and computing bound to accessing or mutating a property, classical getter and setter methods solve, without the overhead of writing separate accessing and mutating methods for an object's properties.

Encapsulation can be archived by restricting write access to a stored property with the private(set) keyword.

```
private(set) var age = 55
```

This restricts write access to the current source file in Swift[6].

In case some computation is needed to update dependent values, Swift offers the willSet and didSet observers that are executed right before and after a stored property is mutated.

```
var age = 55 {
  didSet {
  }
  willSet {
  }
}
```

If a property is purely computed, it can be defined in a similar way.

```
var name:String = {
  get {
  }
  set(newName) {
  }
}
```

[6] It is still possible to restrict the access to instances of the class by using a separate file for the class definition.

The setter is optional. Nothing changes for the calls for accessing and mutating that property: obj.name = "newName" executes the set block, obj.name the get block.

2.7.3.4 Exception Handling

The exception concept is completely missing in Swift. Unfortunately, there is no official explanation from Swift's designers for this decision.

Using Swift's tuples it is possible to return multiple return values, for example an optional return value paired with an optional error object (cf. [**swift-error-handling**]):

```
func doSomething(param: String) -> (return: String?, error:
    NSError?)
```

Another option to consider is creating an enumeration with a success and a error member.

```
enum Result {
    case Success(res:Int)
    case Error(msg:String)
}

func next() -> Result {
    return Result.Success(res:3)
}

switch next() {
case .Error(let msg):
    println("Something went wrong: " + msg)
case .Success(let res):
    println("OK: " + String(res))
}
```

These alternatives misses the exceptions ability to retract some levels of the call stack and retry without explicitly passing an error object up the call chain.

2.7.3.5 Named Parameters

2.7.3.6 The Legacy of Objective-C

The hardest part of learning a new language is not the grammar itself, but getting familiar with the underlying standard library and special APIs for the myriad of tasks a platform has to be capable to handle nowadays. Swift uses all the existing Objective-C

libraries, which are dynamically translated to Swift using a set of rules to improve the method naming.

For example, Objective-C initializers are mapped to Swift by slicing off the init or initWith part of the first parameter name. So

```
UITableView *myTableView = [[UITableView alloc] initWithFrame:
    CGRectZero style:UITableViewStyleGrouped];
```

becomes

```
let myTableView: UITableView = UITableView(frame: CGRectZero,
    style: .Grouped)
```

in Swift (cf. [Apple, 2014b]).

For several tasks it is necessary to mark an own class or protocol with the "@objc" attribute[7], that makes it available in Objective-C code.

An example: Objective-C APIs sometimes use selectors, which are strings identifying a certain callback method by its method name and its named parameters that will be called on the passed object. When an API is used that expects a selector as parameter, the @objc attribute is needed on the passed object's class, so that that the library can find and call the method, otherwise at runtime a fatal error occurs stating no method with that name can be found. This is a clear limitation of that automatic translation of old APIs. In my opinion, it would be much safer to rewrite this APIs in Swift and replace all selectors with function parameters or closures. This way, a misspelled method name is recognized at compile time.

Because of the usage of old Objective-C standard libraries, learning Swift automatically means - at least to a certain degree - learning Objective-C, too.

2.7.3.7 Summary

Swift is currently the most modern programming language available for any mobile platform. Although it's expressive and concise syntax and it's Playground and REPL (Read-Eval-Print-Loop) feature, Swift remains a compiled language. Like Objective C, it is compiled into machine code by the LLVM compiler, which proved to be very fast. Being a new language, the tooling is not yet very stable.[8]

[7] Attributes are the counterpart of annotations in the Java world.

[8] At the time of writing, the XCode IDE doesn't support any refactoring in Swift. The compiler isn't fully finished (a few language elements cause a "Not yet implemented" error), some error messages are cryptic and misleading, the context assistance doesn't always work and the editor can get stuck after some editing with phantom errors that are hard to remove.

Despite of the problems in this early stage, Swift is - in my opinion - a great Language and a big step forward compared with it's predecessor Objective C. It will likely become even better as time passes and the language and tooling matures.

2.8 iOS and Cocoa Touch

iOS Cocoa Touch Framework Developing with XCode

2.9 Couchbase NoSQL Database

2.9.1 What is a NoSQL Database?

For a long time, designing the persistence layer of a system meant deciding which relational database to use. Former alternative approaches like object oriented databases failed to establish themselves. However, during the last few years, more and more companies and projects are rely on a new kind of database system: NoSQL.

Unfortunately, no precise definition exists for a NoSQL database system. In their Book "NoSQL Distilled" [Sadalage, 2013], which provides a good and concise introduction to NoSQL, P. Sadalage and M. Fowler try to provide five common characteristics of NoSQL databases:

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web estates
- Schemaless

The authors think relational databases will not disappear. But the innovation is seeing relational databases as an option among others, and choosing the one that best fits a project.[9]

[9]Sometimes that even means choosing different database types for several parts of a single system - what is called polyglot persistence.

2.9.2 Couchbase

Couchbase Server [Couchbase] is a modern JSON-based NoSQL database server, available as an open-source community edition used in this thesis. While JSON was initially born as "Javascript Object Notation" for data exchange between web / application servers and the javascript web gui, today it is used in more and more products independently from javascript. It's simplicity, readability and compactness helped to repress the comparatively cumbersome XML in many areas. Especially document-oriented databases like Couchbase use the JSON notation for storing the structured data of it's documents.

Couchbase offers a specially sleek mobile version - called "Couchbase Mobile" or "Couchbase Lite" - for all major mobile platforms, and so of course for iOS, too. Since especially mobile devices cannot rely on an uninterrupted networking connection, the mobile version a good choice for a network independent local cache of the application data. With it's synchronization functionality, the local mobile database can be automatically synchronized with a database server. Every time a network connection is available, the local database is updated with changes on the server, and local changes made during the offline time are propagated to the server.

To connect the mobile version to a regular Couchbase server, a sync gateway has to be set up:

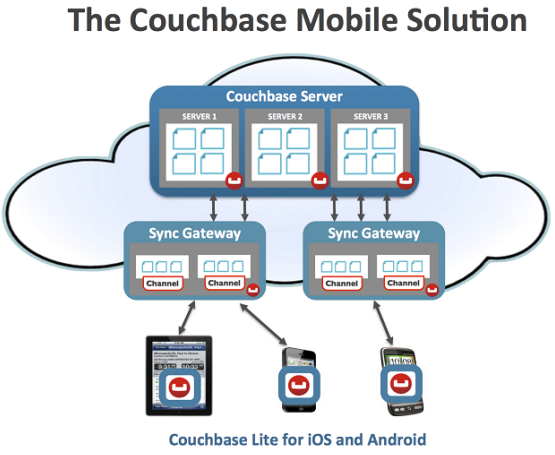

FIGURE 2.10: Couchbase Lite and Sync Gateways. From [Couchbase-Mobile]

While traditional relational database management systems (RDBMS) are typically hard to scale out[10], Couchbase offers a built-in distributed cache concept that can easily be scaled on more servers without modifying the application.

[10] also "Horizontal Scaling" - dividing a database over several servers after reaching the limit of upgrading a single server with more powerful hardware (vertical scaling)

Queries are performed using the map reduce programming model originally developed by Google Engineers, allowing massive parallel execution (cf. [Dean, 2008]). This enables couchbase to perform computation and storage heavy big data analytics.

Couchbase was chosen for this work for it's mobile variant, it's synchronization abilities over several databases and it's JSON-based documents that are a good fit for the web back end and are easy to read and create manually.

2.10 Typesafe Reactive Platform

2.10.1 Overview

In 2013, Typesafe Inc. launched the "Reactive Manifesto", that was updated in September 2014 to the Version 2.0 [React-Manifesto]. The core message is as follows: "We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems."

The Typesafe Reactive Platform consists of the Play web framework, the Akka message driven runtime, the Activator build tool and of course the Scala programming language. It focuses on providing the tools needed to build highly responsive applications, that not only are easy to scale, but even able to increase and decrease the allocated resources at runtime to handle varying workload (thus be "elastic") and stay responsive even in case of failure (what is called to be "resilient").

Since it's publication, the manifesto and the platform gained rising attention. In my opinion, this platform has a high potential of getting the de facto standard for the development of how modern software systems, including web applications and web services, during the next years.

The Typesafe Reactive Platform is used in this thesis for building the back end part of the system, as Scala application with a Play web gui and an NoSQL Couchbase driver based on Akka, that will be introduced later.

2.10.2 Activator Tool

Typesafe Activator is a dependency manager and build tool for the reactive platform built upon sbt (Simple Build Tool, sometimes Scala Build Tool). So it completely replaces sbt, understanding all commands formerly issued to sbt. In the Play framework, it replaces

the Play command, which also was a wrapper around sbt. (cf. [Typesafe-Activator]). It comes with an optional browser based ui for quickly creating some sample projects and an inspection feature for monitoring Play requests and akka actors.

2.10.3 Play Web Framework

The Play Web Framework v1.0 was released on 19th October 2009. For version 2.0, released in March 2012, it was completely rewritten in Scala. Although inspired by popular high-productivity web frameworks like Ruby on Rails or Groovy on Grails, it comes with full type safety typical for Scala but missed in the other frameworks.

Play is a full-stack web framework.

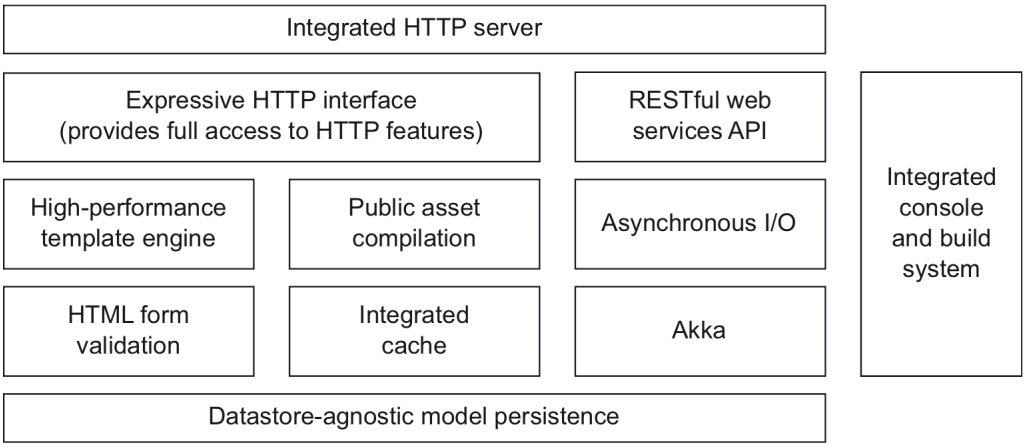

FIGURE 2.11: The Play framework stack. From [Hilton, 2014] with kind permission.

The key features that make Play a high-productivity web framework are:

- Integrated HTTP Server ("JBoss Netty")
- RESTful web services API
- Type-safe templates for defining the html user interface
- Central mapping from HTTP requests, including type-safe parameters, to the corresponding Scala controller method using a routing configuration file
- Integrated build system that automatically rebuilds when the page is reloaded in the browser
- Ability to test the web application without a browser or even a running web server and call controller methods or render templates directly from a Scala REPL with the pre-loaded project

- Support for AJAX, Comet and WebSockets
- JSON to/from Scala Objects mapping

The Play framework is not the only Scala based web framework. According to Typesafe's CTO, they decided to integrate it in it's Platform instead of Lift or similar frameworks because it is completely written in Scala, uses Akka for asynchronous processing, providing a good concurrency handling, has the better type-safety even in html templates, and because it was the one with the greater momentum in terms of ther number of open source projects using it and related discussions in the web etc., at the time the choice was made [cf. Play-Decision].

Chapter 3

System Overview

3.1 Introduction

The Context-Aware Guide (subsequently called "CA Guide" or simply "Guide") needs to be defined to fit indoor, outdoor and mixed sites, like classical museums, parks and gardens. The basic functionality must be accessible even by persons not familiar with mobile technology. The context awareness techniques can help avoiding big parts of explicit user input, allowing the visitor to focus completely on it's environment without reading and typing numbers.

This thesis, however, focuses on the technical foundation of the whole system, and thus on the system architecture on several levels of abstraction.

The following figure shows a high level view of the complete system, consisting of the visitor guide front-end running on a mobile device, a back-end for the modelling the guide and performing analytic functions. The database server is accessed by both system parts and represents the communication basis - there is no direct communication between font and back end. All data is exchanged over the database, enabling asynchronous communication.

As database server Couchbase was chosen due to it's automatic synchronization capabilities, it's mobile version and it's ability to scale out without completely redesigning the way the database is accessed by the application.[1]

[1] After actually having reached the limits of scalability of a single classical relational database server in a commercial project and painfully distributing the database on several servers of the productive system, the idea of an easy scale out using a NoSql database cluster and map-reduce queries seems very attractive to me.

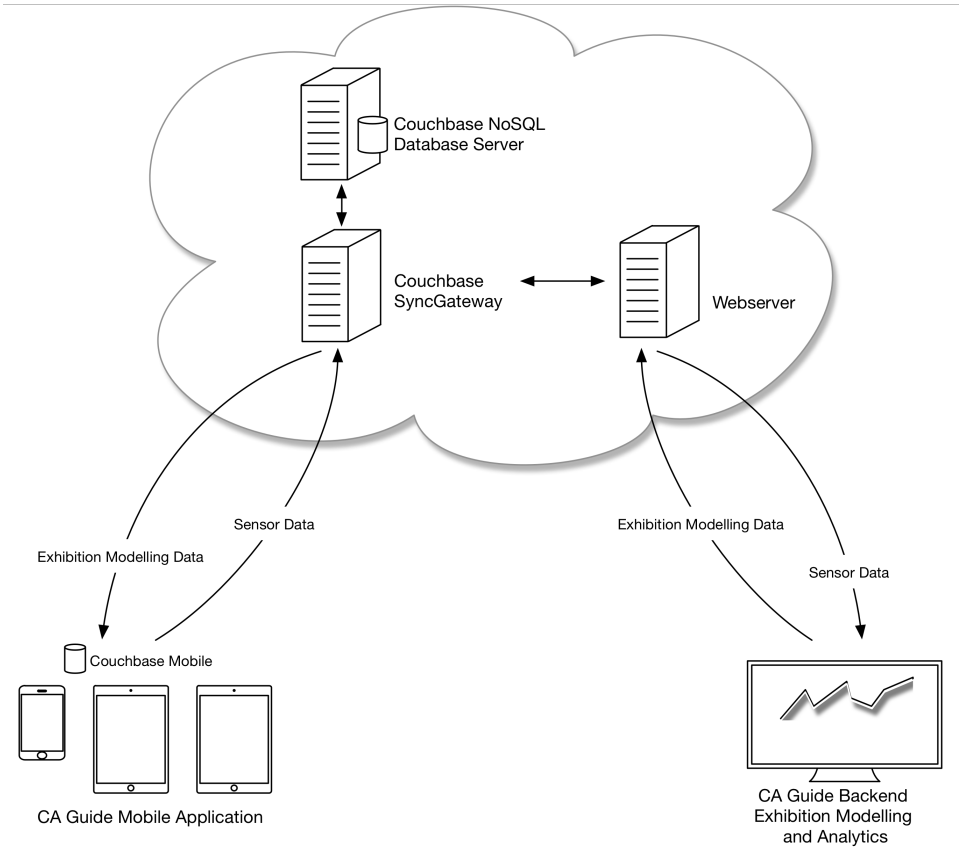

FIGURE 3.1: CA Guite Architecture Overview

The two system parts share a unique data structure for the site model containing all data needed for the mobile guide and the sensor traces, containing recorded raw or computed measurements at different time intervals. Thus, before starting with the front or back-end, it is important to reason about the common data model.

3.2 Data Model

3.2.1 Site Modelling

A single park or museum is modeled using the JSON data format. Compared to XML, JSON is more readable, easier to produce without dedicated tools in a simple text editor and more compact, mainly on account of the elimination of duplicated entity names in closing tags.

The main entities used for modeling a site are shown on the following picture.

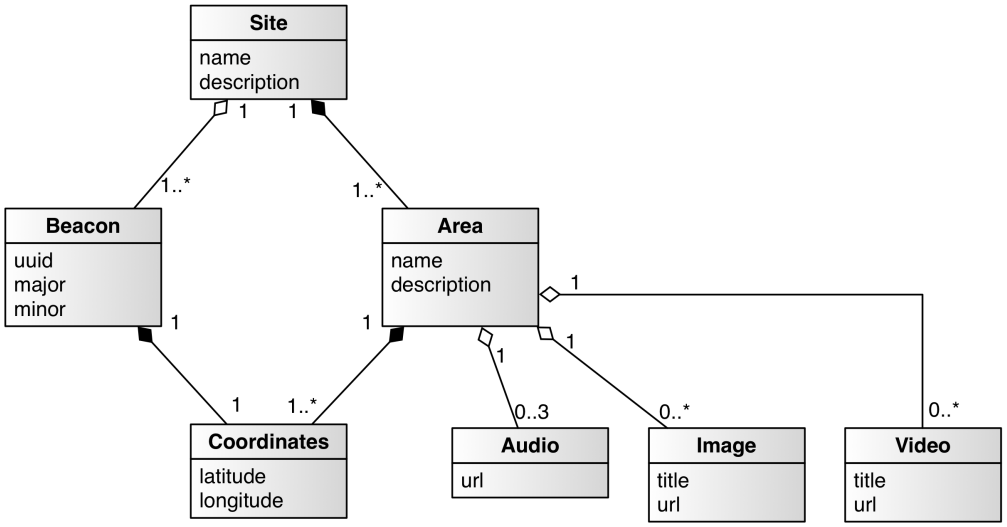

FIGURE 3.2: Data model of a site (using UML notation)

A site is composed of several areas, geographically delimited by a list of coordinates. For indoor sites, several beacons can be added with their identifying properties and the defined position. Beacons are decoupled from areas in the data model. They are a positioning help[2] that could be replaced in the future. An area has a textual description and optionally (even though in productive systems very likely) up to three audio files containing explanations as voice recordings and other fitting audio material. These three files reflect a rising level of detail. Furthermore, several images and videos can be added to the area.

With a normal relational database, pretty much of the entities would be saved in separate tables in a normalized way using primary and foreign keys to represent the relationships between the tables. In case of the one to many relation between an area and it's images, the usual way is to define a separate table "Image" and to store a title and the image or an url beside the areas id they belong to.

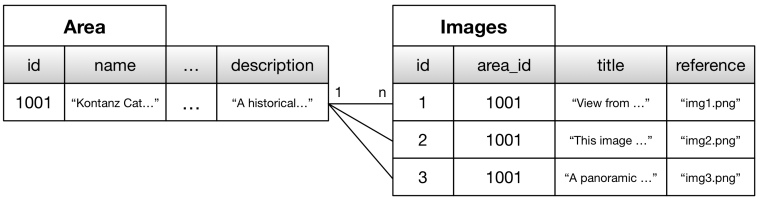

FIGURE 3.3: A 1:n relationship in a classical relational database

[2]After the mobile front-end determined it's (indoor) position based on the beacon's data, it identifies the area it is situated using geometrical computations

With a document-oriented database, data is commonly stored in a more aggregated way. In contrast to the records of a relational database, records of document-oriented database have a more complex structure allowing lists and nesting. In [Sadalage, 2013], the authors suggest the term "aggregate" for these type of records. They adopted this term from [Evans, 2003] "Domain-Driven Design". An aggregate is a set of objects used together. That means they are loaded and saved to the database as a unit and they determine the unit to be locked for the management of consistency.

So the precise level of aggregation depends mainly on how the data will be accessed later, and as is often the case, there is not only one right solution.

From a site modeling point of view at the back-end, single areas and single beacons can be viewed as aggregates, allowing more granular modifications and enabling collaborative editing of a single site. From a front-end point of view, a whole site definition can be regarded an aggregate. During initialization, the whole site is read with all areas and beacons. This can be done with a single database query. On the mobile device, sites are only read and never modified, so that there is no problem with a bigger aggregate that has to be locked.

Since having a working copy of a site for modeling in the back-end that is not deployed to the mobile devices until it is marked as finished is desirable, I decided to take a hybrid solution. On the back-end, a site is modeled with finer-grained aggregates having separate documents for sites, areas and beacons. These aggregates lies in a separate modeling database (or bucket, as they are called in Couchbase) that is not synchronized with the mobile devices.

At the point a the model is marked as finished, all these documents are merged into one big site aggregate, containing all the data. This document is written to the shared database and is automatically synchronized with the mobile databases as soon as they have networking connectivity.

The site model can the be refined and edited at the back-end without compromising the deployed version. For productive usage, an additional shared testing database synchronized only with a testing mobile device would be necessary. The site aggregate could first be written to this database for testing before deploying it to the rest of the mobile devices. This functionality can be added at a later stage and is out of scope of this thesis.

The following JSON-document shows a simplified site aggregate containing one area with the simplest possible polygon and one beacon, to get an idea of the structure.

```
key: "Site-CXN01"
value: {
 "type":"city",
```

```
"name":"Konstanz",
"description":"",
"areas": [{
  "id":"area-51",
  "type":"area",
  "name":"Konstanz Cathedral",
  "description":"A historical building in Konstanz, ...",
  "locationDefinition": {
        "type": "polygon",
        "coordinates": [
          {"lat":47.663093, "lon":9.176371},
          {"lat":47.663336, "lon":9.175215},
          {"lat":47.663673, "lon":9.176075}]},
        "images":["cathedral-panorama.png","cathedral-nw.png"],
        "audio":{
          "level1": "cathedral-1.mp3",
          "level2": "cathedral-2.mp3",
          "level3": "cathedral-3.mp3"},
        "videos":[]}],
"beacons": [{
  "id":"beacon-42",
  "uuid":"F7826DA6-4FA2-4E98-8024-BC5B71E0893E",
  "major":"40231",
  "minor":"203",
  "coordinates": {"lat":47.663278, "lon":9.176214}}]
}
```

Listing 3.1: An exemplary site aggregate in JSON representation

The binary data of images, audio files and videos can be stored outside of the document, maintaining only a reference to them inside the document (in this sample, a file name was used to represent a reference). Storing binary data outside of the document avoids wasting storage capacity due to inflation of data represented in a base64-encoded way and reduces the parsing time of the document. For this purpose, Couchbase offers the concept of attachments, that can be associated to a document but are stored separately in a binary way. Another advantage is a higher efficiency of the replication, since attachments are not retransferred when the associated document was changed but only on real modifications of the attachment itself [Attachment].

Although NoSQL databases are often referred to as being schema-less, it's important to keep in mind that there still is an implicit schema: the data structure the application relies on. The risk of malformed data can be handled with automated tests. Especially in case of manually produced JSON documents or JSON data coming from a foreign system, a JSON schema can be used for asserting a valid structure of a JSON file, similar to a DTD (Data Type Definition) in XML (http://json-schema.org).

3.2.2 Sensor Data

Sensor data measurements is saved to the Couchbase database to allow replaying it for development, testing, presentations and for analytics presented in the system's web back-end. Thus, the producer and consumer of the data is switched here, having the data produced inside the front-end, written into mobile database and later synchronized over the network with the Couchbase database for enabling read-only access by the back-end.

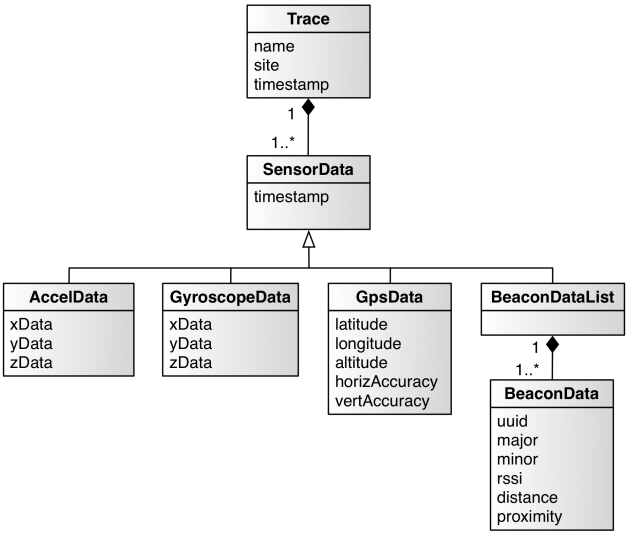

FIGURE 3.4: Data model of a sensor trace

The single measurements can be very high frequent, for example in case of accelerometer data, that is updated every 10 milliseconds. Updates on beacon measurements and GPS measurements are typically available every second. Events on a higher logical level, like "entering region a", will occur much less frequently.

The two use cases are recording data for development and testing and recording visitor movements for analytics. Low level testing sensor traces, containing 100 measurements per second for each sensor, will only be recorded over a limited time period for testing and profiling algorithms. The recording duration for analytic purposes during an actual visit can extend to hours, but at the same time only higher level data is recorded and thus the data amount grow much slower. Therefore, saving a trace into a single aggregate is considered feasible for both use cases.

The following listing shows a trace as JSON document with an exemplary sensor data measurement for each type.

```
key: Sensortrace-CXN01
value: {
  "type": "trace",
```

```
  "timestamp": 12345678.35,
  "site": "Site-CXN01",
  "data": [
    {"type":"accelerometer", "timestamp": 12345678.350,
      "data": [1.235, 0.364, 0.021]},
    {"type":"gyroscope", "timestamp": 12345678.360,
      "data": [0.031, 0.005, 0.207]},
    {"type":"gps", "timestamp": 12345678.495, "data": {
      "latitude":47.652712, "longitude":9.168142, "altitude":404.90,
      "vertAccuracy":65, "horizAccuracy":10}
    {"type":"beacons", "timestamp": 12345678.782, "data": [
      {"uuid":"B9407F30-..6D", "major":14185, "minor":2,
        "proximity":1, "rssi":-67, "distance":0.53},
      {"uuid":"B9407F30-..6D", "major":14185, "minor":3,
        "proximity":2, "rssi":-77, "distance":4.53}
    ]}
  ]
}
```

Listing 3.2: An exemplary trace aggregate in JSON representation

3.3 Setting up the Couchbase Server Infrastructure

After locally installing Couchbase Server, a web based admin console is accessible at http://localhost:8091. It offers an overview of the system status, performance details and several administration tasks. In case of having several servers attached, the whole cluster can be managed and analyzed over this web application.

For the guide platform, two Couchbase buckets are created: "guide" is the shared database and "guide-editor" the back-end only one. The following picture shows two screen shots of the cluster overview page and the data buckets page with our buckets.

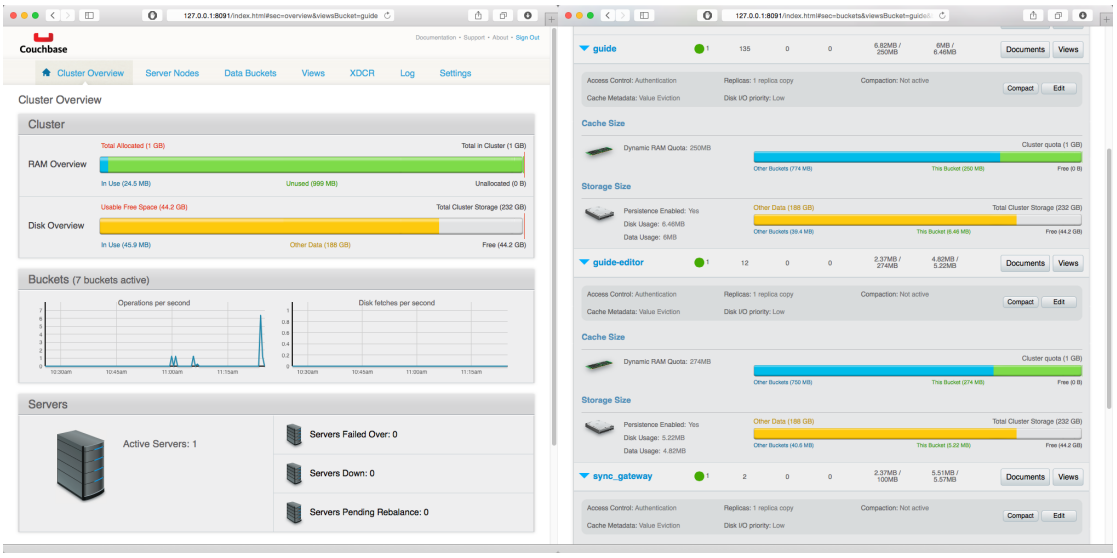

FIGURE 3.5: The Couchbase Server management console

For the automatic synchronization of the shared bucket between the server and the mobile devices, a "Couchbase Sync Gateway" has to be installed and configured via a json file, where primarily the database server url, the users and the shared bucket are defined.

```
"log": ["HTTP+"],
"databases": {
  "guide": {
    "server": "http://localhost:8091",
    "bucket": "guide",
    "users": {
      "GUEST": {"disabled": false, "admin_channels": ["*"] }
    }
  }
}
```

LISTING 3.3: Connecting the Sync Gateway to the Couchbase Server

After creating this file and a management bucket named "sync_gateway" on the server, the gateway can be started and exposes a REST sync API at port 4984 and an admin API at port 4985. A rudimentary synchronization gateway web gui is also available at "localhost:4985/_admin/db/guide".

For the shared bucket, all communication has to pass over this synchronization gateway, avoiding to communicate directly to the actual database server. The gateway keeps track of the documents revisions with the management bucket and by adding a management property called "_sync" to each document, used to save the revision number and other internal version control data.

Chapter 4

CA Guide Mobile Application

4.1 Introduction and Goal

The CA Guide mobile application has to display predefined textual descriptions of sites areas, show images and play audio files when a specific context state is reached. The trigger will often be entering a specified area at a moderate pace that signals the visitor is interested and not only passing by to reach another place. But also leaving an area can be of interest, for example to indicate single subareas that were missed. Even offering a coffee break discount coupon on the display after the visitor made 5000 steps through the park or exhibition is imaginable.

An essential aspect is the ability to record and replay the data coming from all accessed sensors. That's the only way to effectively develop a context-sensitive application and so special attention has to be given to this part of the system.

4.2 The Target Platform

The target mobile platform for the CA Guide front end application is iOS 8.1. The targeted mobile device is an iPad mini 2 with GPS sensor. This device was chosen for it's compact size and light weight, while still offering much more screen size than a normal smartphone, which is advantageous for displaying images and text in a size that is more comfortable to read.

4.3 Setting Up the Development Platform

This mobile application is written in Swift 1.2 using the native Apple XCode IDE in version 6.3, which at the time of writing is still in beta status[1]. It can be downloaded at [Xcode-beta].

For logging purposes, CocoaLumberjack [CocoaLumberjack] was used. In contrast to the logging system shipped with the iOS SDK, it offers several logging levels and has a much greater performance [CL-Benchmarks]. This is especially important when logging low level functions like the one handling accelerometer updates every 10ms.

To add extensions like CocoaLumberjack to the project, the dependency management system CocoaPods [CocoaPods] was used. It creates a XCode Workspace containing the own project paired with a managed CocoaPods project containing all the extra libraries. The wanted libraries are specified in a file named "Podfile", where you define the ids and the desired version for each build target.

```
target 'Guide' do
   pod 'CocoaLumberjack', '2.0.0-beta'
   pod 'couchbase-lite-ios', '~> 1.0'
   pod 'CorePlot', :git => 'https://github.com/core-plot/core-plot
   .git', :branch => 'release-2.0'
end

target 'GuideTests' do
   pod 'CocoaLumberjack', '2.0.0-beta'
   pod 'couchbase-lite-ios', '~> 1.0'
end
```

Listing 4.1: The CocoaPods file of the CA Guide front end XCode project

Libraries not available in the public CocoaPods repository can be added defining a custom git url and branch, as was done for Core Plot [Core-Plot], an open source plotting framework used for plotting beacon signal strength on debug screens.

Because at the time of writing many modules are written in Objective-C, a so-called bridging header file has to be added manually to the own Swift project to expose the Ojective-C headers to the Swift compiler and let it automatically create a Swift API that can be used in the own Swift code [Apple, 2014b, p. 71, chapter "Importing Objective-C into Swift"].

[1] In fact, the last "stable" version 6.1, which I used for several weeks before, crashed too quite often with the Message so I decided to use the newest beta IDE version, that proved not to be more buggy than the release version while offering the newest Swift Compiler with several optimizations in compile and runtime performance.

4.4 Architecture

4.4.1 Overview

Below the presentation layer and the application logic, a clean sensor data processing foundation that provides the context information for the application has to be designed. This foundation, hereafter referred to as "Sensorbase framework", is divided in three logical layers. At the bottom are the sensor data sources, consisting of the iOS sensor APIs and virtual sensors. Since processing of sensor data is often done in several steps, as we will see in this chapter, two layers of sensor data processing are added, the first for low-level algorithms called at higher frequency and the second for higher level ones, performing more computation-intensive tasks at a lower frequency.

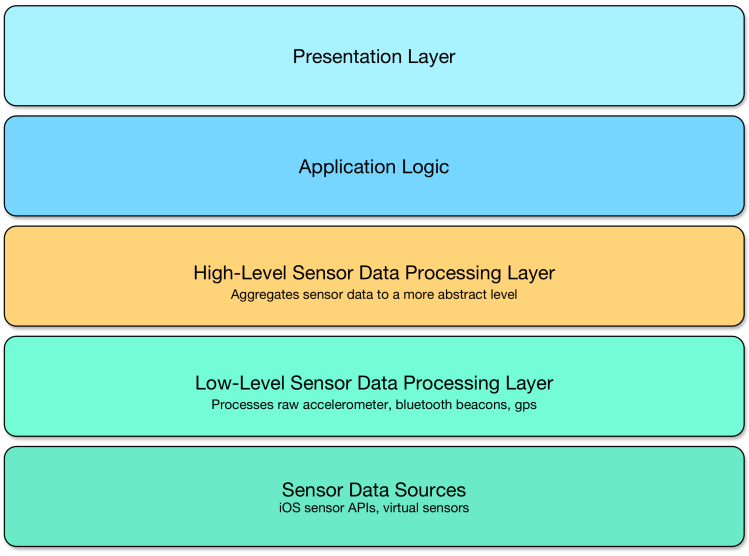

FIGURE 4.1: Layer model of the guide front-end

A fundamental requirement for the guide front-end, from a developer point of view, is the ability to record and replay recorded sensor readings of all accessed sensors and to automatically test single processing steps using recorded sensor data for comparing it with it's expected result. To achieve this goal, virtual sensors are introduced at three different levels of abstraction inside the lower layers as depicted by the following figure.

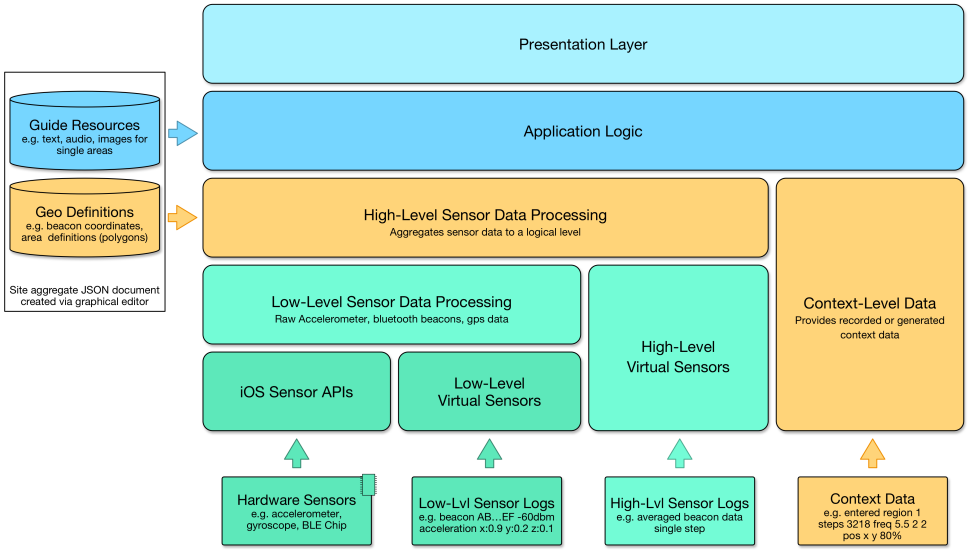

FIGURE 4.2: Layer model with virtual sensors

A virtual sensor delivers the same kind of data the processing algorithm of the corresponding layer would output, loading a previously recorded trace from the database. Also depicted are the external input data coming from the JSON site aggregate, containing the positions of all beacons with combined with their advertising data, the area geographic definitions and and the guide resources. The high-level sensor data processing layer uses the beacon related data for indoor positioning.

Figure 4.3 visualizes the different kind of data that is handled in every different layer, taking the accelerometer data processed by pedometer and statistics algorithms as an example.

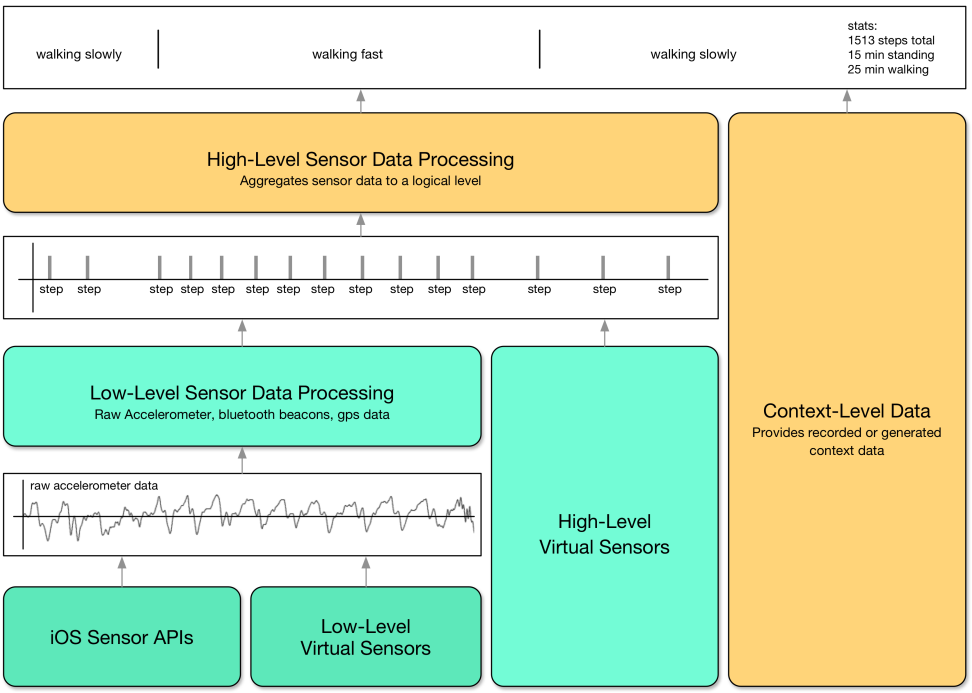

FIGURE 4.3: Data flow between the lower layers of the Sensorbase framework

The pedometer algorithm in the low-level sensor data processing layer receives raw accelerometer data as input. It analyzes the data, detects single steps and passes them to the upper layer, the high-level sensor data processing. Here the single steps are aggregated to periods of slow and fast walking and walking statistics are updated.
For most types of sensor data and information to be extracted, a processing approach using algorithms split in two layers can be found. For the rest the second layer could be omitted without affecting the other sensors. All the sensors implemented for this thesis use the two step processing approach.

The frequency of events passed to the next layer decreases from real-time (100 updates per second), demanding very fast algorithms, to some seconds or minutes. The following sequence diagram visualizes the decreasing frequency of updates from the lowest to the highest sensor related layer.

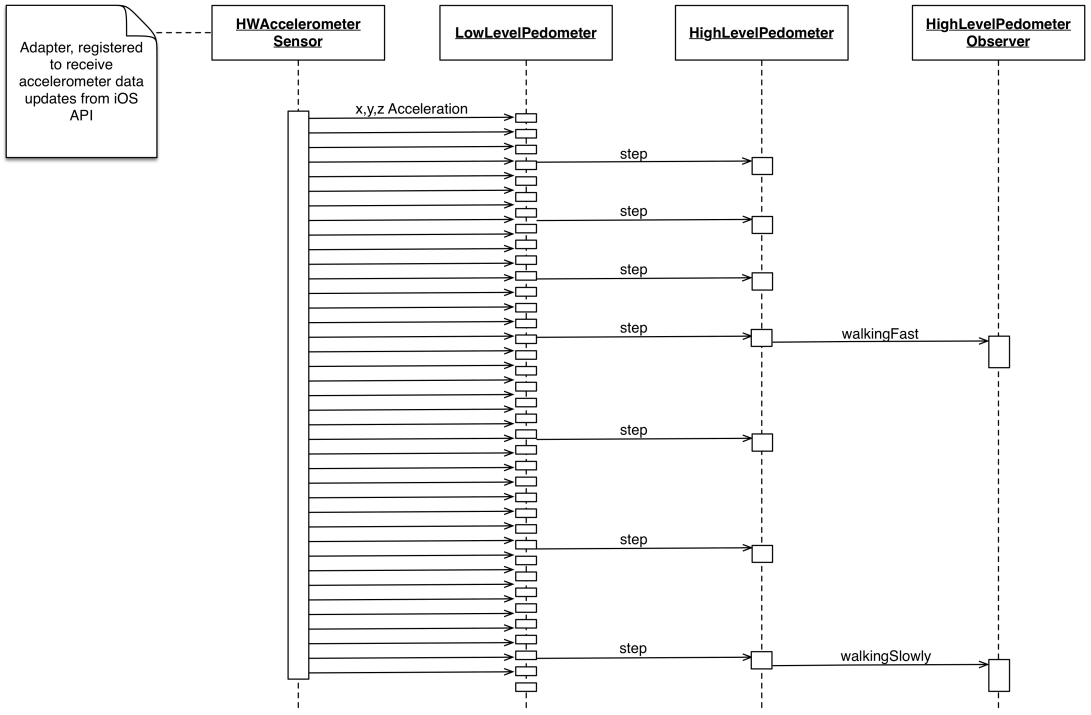

FIGURE 4.4: Sequence Diagram for the Sensor Layers

Beside enabling automated unit tests at different abstraction levels, this layered virtual sensor concept has various other benefits:

- During development, a simulation of walking through a park or museum is possible, for manually testing not only algorithms of the lower layers, but also elements of the uppermost layer like different screen transitions or other UI behavior.
- For demonstrating the guide to customers or prospects, a walk through a park or museum can be simulated at the desk on a real device running the guide, beeing much more intuitive than showing only screen shots.
- With an own class in the sensor data sources layer acting as a bridge to the iOS APIs it is easier to migrate the algorithms to a different mobile platform, by rewriting this adapter class for the target platforms sensor API and automatically translating the other sources that have no dependencies to the platform.

With the last point in mind, the value classes holding sensor data were only used in the lowest layer and converted to own iOS independent classes for further processing.

4.4.2 Implementation concepts

The following class diagram shows how the sensor related classes are organized and how they are related to one another, using the accelerometer processing classes as an example (the classes concerning other sensor types are organized the same way).

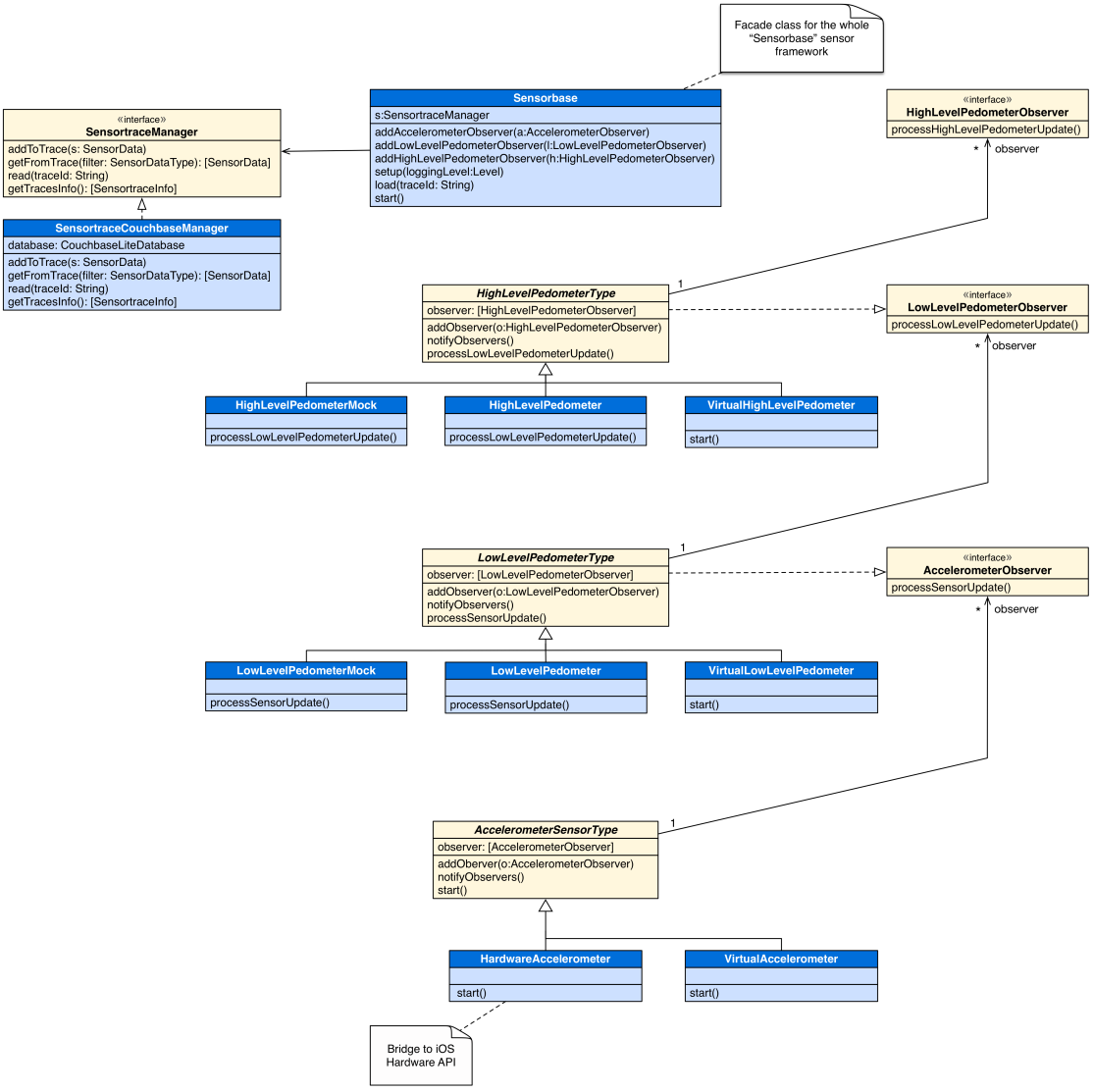

FIGURE 4.5: Simplified Class Diagram for Sensor Data Processing

Every layer observes the underlying layer, being subscribed to data updates and getting them via calls to their corresponding "processUpdate()" method. A common abstract class implements the observer pattern and is extended by two or three classes per layer and sensor type: a class implementing the actual processing algorithms needed for the processing step (e.g. HardwareAccelerometer), a class that acts as virtual sensor and forwards recorded data of that layer (e.g. VirtualAccelerometer) and a class used in automated tests for the underlying layers (e.g. HighLevelPedometerMock). For example,

to test the step detection algorithm, the system can be set up using an instance of VirtualAccelerometer with known accelerometer data, observed by an instance of LowLevelPedometer, whick in turn is observed by an instance of HighLevelPedometerMock, where the timestamps of reported steps can be compared with expected ones.

On the top lies the Sensorbase class. It is a singleton and acts as facade (cf. [Gamma, 1995, Ch.4, Facade]) for the whole framework, providing a single interface to the set of interfaces of the Sensorbase framework. It is responsible for instantiating the classes and connecting them based on the desired level of tracing. It also setups the framework for the usage with virtual sensors, loading a trace over the SensortraceManager, instantiating the involved virtual sensors, feeding them with the data concerning the corresponding sensor type and starting the playback. Inside the instances of the "Virtual..." classes, the loaded data is replayed using an independent thread to enable the concurrent execution of processing of different sensor types. More precisely, since threads cannot be directly created with Swift, nor does Swift offer a thread abstraction model of its own, the iOS/OSX "Grand Central Dispatch" (GCD) [cf. GCD-Reference] concept is used.

For external classes that want to observe the data updates of the lower layers, it offers methods accepting references to different types of observers that are passed to the proper objects of the framework for registration as observer. The debug view, for examples, uses these methods for registering itself as observer of

The SensortraceManager interface at the top left avoids binding the Sensorbase facade to the Couchbase database and so allowing to change the underlying database system more easily.

4.4.3 An encountered Swift limitation

The class diagram of figure 4.5 is not Swift specific, since it shows abstract classes not available in Swift. Moreover, interfaces are named protocols and do not accept any implementation.

When it came to expressing that for example the different LowLevelPedometer classes have a common implementation of some methods and simultaneously are AccelerometerObserver, the lack of abstract classes and abstract methods in Swift became visible.

Swift does not offer a perfectly clean way to model the described situation at the time of writing.

There are two workarounds each with own drawbacks. The first is simulating an abstract method using a regular class and implement the concerned method with an assertion to cause a failure if it was not overridden by a subclass.

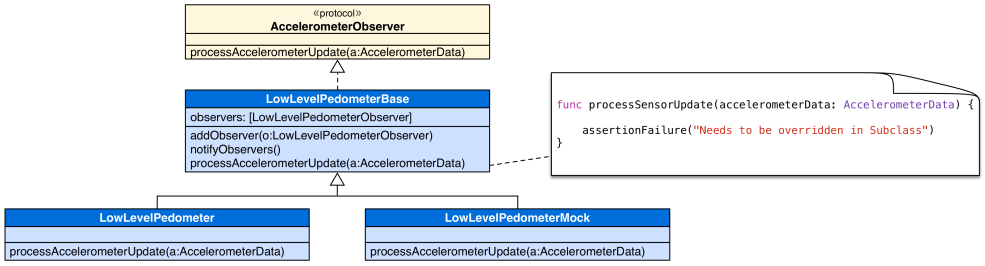

FIGURE 4.6: Simulating an abstract method using an assertion

The great drawback is that this error is detected only at run time, causing an application crash for a problem that can easily be detected at compile time respectively directly in the IDE.

The second approach consists in not letting the LowLevelPedometerBase class implement the AccelerometerObserver protocol but letting each class extend the base class and implement the protocol separately.

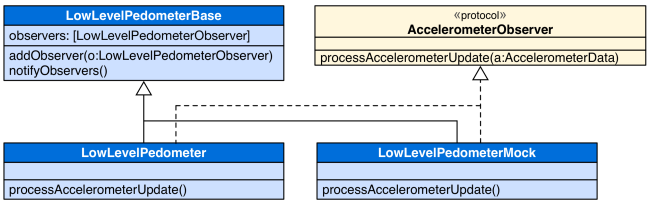

FIGURE 4.7: Separating base class and protocol

There is no implementation independent type combining AccelerometerObserver and LowLevelPedometerBase.

Both approaches share the same flaw, that LowLevelPedometerBase can be instantiated.

4.5 Querying the Mobile NoSQL Database

Since NoSQL queries differ substantially from common SQL based database drivers, this section explains the way the database is queried inside the SensorbaseCouchbaseManager class.

When the key of the document to retrieve is known, the JSON document can be retrieved by simply using a getter function on the database object. All other queries are performed quite differently from standard databases.

In the debug view, for example, a list of all recorded trace documents has to be displayed to allow the user selecting a trace for replaying it. To perform this query in Couchbase

Mobile and Swift, first a named view has to be created. A Couchbase view follows the map/reduce paradigm and thus a map and an optional reduce function have to be defined.

In the current implementation of the Couchbase driver, the map function is defined using an Objective-C block that is automatically imported as Swift closure. That closure will be called internally on every document with the document itself and an emit function as parameters. The emit function can be called 0..* times for a single document with a key and value parameter.

```
// Setup the views
let tracesView = self.database.viewNamed(self.tracesViewName)
tracesView.setMapBlock({ (doc, emit) in
  if let timestamp = doc["timestamp"] as? String {
    if let site = doc["site"] as? String {
      if let data = doc["data"] as? [AnyObject] {
        emit([site, timestamp, data.count], doc["_id"])
      }
    }
  }
}, version: "3")
```

Listing 4.2: The map function of the traces view

Being a very young language, new versions of Swift were released during this work. So one improvement of Swift 1.2, released on 9th February 2015 [Swift-1.2], was allowing multiple optional value bindings inside a single if statement, avoiding deeper nesting levels seen above.

```
// Setup the views, Swift v1.2
let tracesView = self.database.viewNamed(self.tracesViewName)
tracesView.setMapBlock({ (doc, emit) in
  if let timestamp = doc["timestamp"] as? String,
    site = doc["site"] as? String,
    data = doc["data"] as? [AnyObject] {
      emit([site, timestamp, data.count], doc["_id"])
  }
}, version: "3")
```

Listing 4.3: The map function avoiding nesting with Swift 1.2

The key can be a single value or a list of values, as seen here, useful for later querying using start and end barriers on single elements of the key list, as shown later. The emitted key for the sites view is built with the site's id, the timestamp of the trace and the number of measurements inside the document.

The view can be thought of as an index in a conventional database - not a query. Couchbase uses the view definition with it's map and reduce function to create it's internal indices. Once created, the view can then be used to instantiate a query, configure it and retrieve the desired data.

For example, to create a query returning only sensor traces for the site "Site-CXN01" that were created after 1st January 2015 (1420070400 as Unix timestamp) using the previously defined sites view, the startKey and endKey properties can be defined the following way.

```
let tracesView = database.viewNamed(tracesViewName)
let query = tracesView.createQuery()
query.startKey = ["Site-CXN01",1420070400]
query.endKey   = ["Site-CXN01",[:]]
```

Listing 4.4: Setting up the NoSQL query

The empty dictionary used as second element of the end key is Couchbase value sorted always at the end and the preferred way to define an upper bound greater than all existing elements.

A single view can be queried by defining a coherent range of the ordered keys. While it is possible to limit the timestamp value inside a single site with this view, to select a timestamp range with arbitrary sites a new view has to be defined, emitting the timestamp as first key element.

The reason is Couchbase organizes it's data holding a sorted index of the emitted keys. For multi-element keys, they are sorted primarily by the first key element, then by the second and so on.

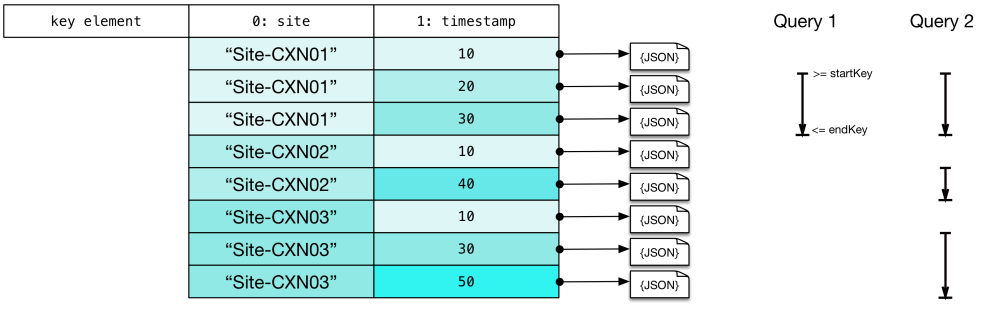

Figure 4.8: Couchbase limits for querying views

Figure 4.10 visualized the index with a simplified key consisting of two elements: the site id and the timestamp. The first query is equivalent to the on of listing 4.4. Selecting only timestamps >= 20 for arbitrary sites would result in the scattered ranges pictured

by query two. Therefore query 2 cannot be expressed by defining a start and end key on this view.

That's the price to pay for the focus on high performance and the automatic scaling capabilities of Couchbase and similar NoSQL databases. Although, compared with the benefits, this extra work seems reasonable.

Finally, running the query is similar to known relational database drivers.

```
var error: NSError?
let result = query.run(&error)
var found = false
while let row = result?.nextRow() {
   // do something with the row
}
```

Listing 4.5: Performing the NoSQL query

4.6 Application Logic

For this work, the application logic on top of the sensor layers focuses on computing the area that contains the position obtained by the lower layer. This is done using computational geometry algorithms described in

For the final product, the guide has to keep track of the audio files already played and decide if it is appropriate to play the lower levels of detail based on the current walking speed and the size of the area.

If so and some other rules apply, the content is presented to the user in the appropriate way.

If not walking fast load and play current content.

If display is on show actions that user has to opt in, if display is off play automatically.

4.7 Presentation Layer

The full design of the front-end GUI is out of scope of this thesis, which limits the presentation layer to the debugging dashboard. This view shows processed and unprocessed sensor data of different layers, and contains controls for loading and saving sensor traces. Even after adding the end-user GUI, this debugging dashboard will remain in the application for development and testing.

A mockup of the debugging dashboard was designed using the tool "Balsamiq Mockups" [Balsamiq].

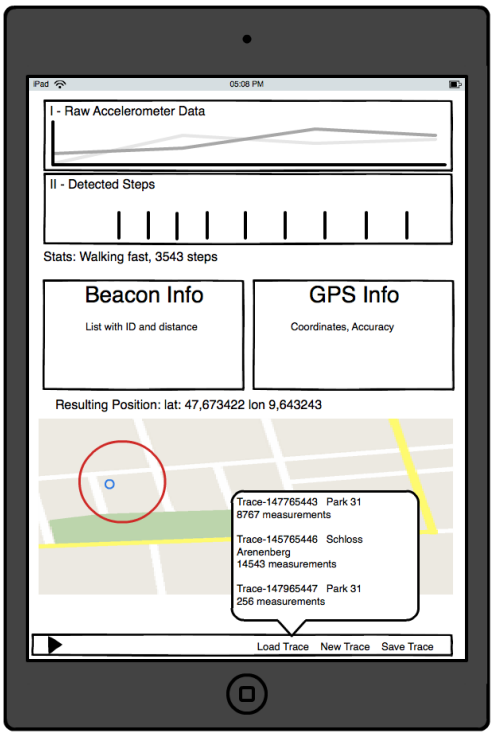

FIGURE 4.9: A mockup of the debug dashboard

On the top, the raw accelerometer data is depicted in a chart with the acceleration force on it's vertical axis and the time on it's horizontal axis. Below, a graph shows vertical peaks every time a step is detected. The statistics of the third layer are visible as text showing the walking speed and the cumulated steps. The beacon info section consists of a list of all currently sensed beacons with their properties. Beside, a GPS section shows all GPS related data. The resulting position is additionally shown on a map using a blue circle, with an optional red circle for the pure GPS position and accuracy, depicted as radius of the circle. On the bottom of the screen, a toolbar contains all controls needed for managing the sensor traces. The load trace action has to show a list of all traces (cf. section 4.5) and loads the selected trace.

FIGURE 4.10: Screenshot of the debug dashboard running on an iPad

4.8 Summary

We now have a mobile system capable of processing sensor readings in several layers and recording and replaying them at a desired level of detail. The system loads sites defined as JSON document and uses the beacons definitions to locate itself when GPS signal is missing or not accurate. Based on the current position and the loaded area definitions, the current area is computed and the content loaded.

The next chapter handles the design and implementation of the second part of the framework: The back end serving as site modeling and analytics tool.

Chapter 5

The Guide's Back End

5.1 Introduction

The guide's back end is meant to be used by the park or museum staff responsible for designing and optimizing the visiting experience. In contrast to the front end, the user interface can be more complex requiring an introductory training and support. Of course, the usability deserves special attention in spite of the training, as it has a major influence on the frequency a software is used, the attitude towards it and thus the success of the whole product.

There are two main tasks accomplished with the CA Guide back end:

- As graphical modeling tool, it is used to model the outdoor or indoor site by defining areas on a map or floor plan and adding content that will be presented later on the mobile device when entering this area (and some other rules apply). Especially for indoor sites, single Bluetooth beacons with their identifying numbers and positions must be added to the plan to enable the mobile device to locate itself.
- As analytical tool, it offers analytic functions for understanding how visitors move trough the park/museum/exhibition and thus allowing to optimize it. This is similar to the way the behavior of website visitors is tracked and used to optimize a web presence, ported to physical sites.[1].

[1] Of course, the privacy of the visitor has to be respected at any time. The data must only be collected anonymously and respecting all local privacy and data protection regulations.

5.2 The Target Platform

The back end user interface has to run in a standard web browser without specific plugins. This has several advantages: Users can start immediately to work with the product without installing any client. Having the project data in a cloud database, they can work on the same project from different locations using different computers without manually setting up a synchronization infrastructure. Computation expensive analytic functions can be performed directly on the database or web/application servers and only the results are transferred to the client, enabling it's usage on common hardware.

By allowing collaborative editing, the museum staff can easily be supported remotely in real time without having to be on site.

5.3 Architectural Decisions

5.3.1 The Client and Server Code Gap

One of the main challenges developing for the web is the client/server code barrier.

Modern web applications need to be highly responsive for providing a good user experience and thus be accepted by them. Operations like adding, editing or deleting entities have to be performed without having to do a full page reload to display some sort of HTML form and reloading the whole page again after the form was submitted to the server. This can be achieved using asynchronous calls to the server in the background, exchanging only the needed data with the server and refreshing just the needed part of the HTML DOM. Many operations like resorting a table can even be performed completely in the browser without even performing a server request.

This technique is commonly known as AJAX (Asynchronuous JavaScript and XML) and inside this acronym's phrase one can already spot the main problem. While AJAX gained popularity during the last years, it leveraged the massive usage of JavaScript, a dynamically typed scripting language originally not designed for big projects. In fact, JavaScript[2] was developed in 1995 by Brendan Eich in ten days for the Netscape browser, which is an extremely short time despite Eich's big experience in building languages [Interview-Eich]. That led him to design the language to be malleable, and there are many libraries making programming in JavaScript a little "less painful", without solving the problem of lacking static types and other flaws.

[2] In the cited interview Eich says "The name is a total lie" - it was just a marketing decision, the language is not related to Java. JavaScript has been standardized later by Ecma International to ECMAScript.

So the two main problems to be solved are:

- Some data structures and algorithms already existing in the server code have to reprogrammed for the client running in the browser, creating redundancy with all the known problems it has
- The standard programming language for client-side code lacks static typing and a class syntax, among others

The next subsection focuses on different solution attempts.

5.3.2 Solution Attempts

5.3.2.1 Improved Javascript

In the last years, several new scripting languages are emerging that compile to JavaScript.

One of this languages was even integrated in the Play framework, which comes with an built in compiler for CoffeeScript [CoffeeScript]. It is a small language that provides a nicer syntax for JavaScript, introducing even classes and inheritance, and compiles to plain JavaScript. However, it does not add any support for static typing.

Another noteworthy web-client language is TypeScript, which is maintained by Microsoft as open source [TypeScript]. As the name implies, it adds static typing to JavaScript with type inference and similar to CoffeeScript it enhances the syntax. In contrast to CoffeeScript, it is a superset of JavaScript, meaning any JavaScript code is automatically valid TypeScript code, too. For using third party libraries in a typed way, a big repository with open source type definitions for 796 libraries at the time of writing [TS-TypeRepository]. By including a reference to such a type definition, the library's API can be accessed in a statically typed way.

This screenshot demonstrates a TypeScript compilation error and an inferred numeric type.

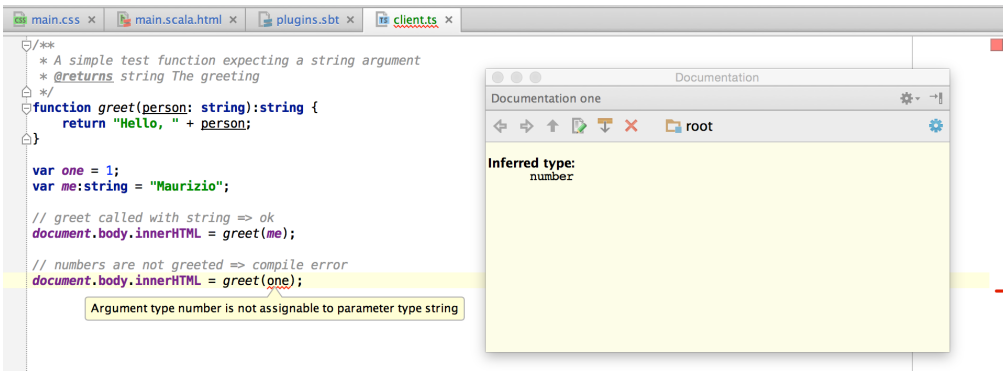

FIGURE 5.1: TypeScript in IntelliJ IDEA

While these solutions, and especially TypeScript, smooth out the main flaws of JavaScript and are being adopted by a rising number of developers, they do not solve the need to create redundancy in client/server web applications. So for this work, I continued to search for a solution to both problems.

5.3.2.2 Unified Programming Language for Server and Client Code

There are several attempts to bring JavaScript on the server, and the most popular is Node.js, which was released in 2009. While there surely are some scenarios for server-side JavaScript like screen-capturing of rendered websites, in my opinion this is the wrong way of language unification. It brings the problems connected to JavaScript to the server side, where much better designed languages exist, like Scala.

The other way around would be the perfect solution: Using a rich and well engineered language to write server and client code. So after some research I found the Scala.js project [Scala.js]. It was started on February 2013 by Sébastien Doeraene, a member of the Scala team at LAMP (Programming Methods Laboratory at EPFL), after Martin Odersky suggested him to work on a JavaScript compiler [Interview-Doeraene].

Scala.js ports most parts of Scala and it's standard library to the browser, compiling Scala to JavaScript or more precisely to ECMAScript 5.1. So, among others, it is possible to use Scala collections, classes and traits, types, pattern matching and even Futures for concurrent processing.

Beside avoiding to use ECMAScript directly, the big advantage of using Scala.js is beeing able to cross-compile parts of the same Scala code for both the server-side JVM and the browser-side ECMAScript engine. This helps reducing redundancy and is especially helpful for data transfer objects (DTO) representing entities of the respective domain, in the case of the CA Guide sites, areas and beacons, as we will see later.

On 5th February 2015, at the time of this writing and exactly two years after this project was started, the release v0.6.0 was announced on the official Scala website and the experimental flag was removed, defining Scala.js "production-ready" [Scala.js-0.6].

So I decided to use this very promising compiler for the CA Guide back end.

5.4 Setting up the Development Platform

5.4.1 Project Structure, Scala and Play

Typesafe's Play web framework (cd. section 2.10.3) serves as basis for the guide's back end. It is installed using the Activator tool, which acts as build tool and dependency manager.

The project is configured according to the "Play! application with Scala.js" skeleton [Scala.js-Crosscompiling] referenced on the Scala.js homepage, splitting it up into three subprojects:

- **editor-server** The Play project including server side Scala code and the typical Play application structure
- **editor-client** Client only source code, the Scala.js project
- **editor-shared** Shared Scala sources, accessible from both the client and server source code

The project was imported in IntelliJ IDEA 14 as sbt project, which worked very well without major problems.

After starting the activator tool in a shell on the project root directory, several commands are available. The three most important commands are "run", "test" and "testOnly", where the first starts the web server and automatically rebuilds the project on every page refresh in the browser, if files were modified. The "test" command executes all defined tests, whereas "testOnly" is used with 1..* specs2 [specs2] specification names[3] to selectively execute only desired tests.

[3]The names of the implemented test classes extending the specs2 "Specification" class

5.4.2 Adding Reactive Couchbase as Scala Database Driver

ReactiveCouchbase is an open-source database driver for connecting a Couchbase database to a Scala program [React-Couchbase]. Queries are performed using the elegant Scala Future paradigm, that will be discussed later in detail. For the integration with a Play application, a dedicated plugin is available.

Plugins can be added to Play by defining the name, version and if needed the repository url to the build.sbt file. This dependency has only to be added to the server project's dependencies.

```
libraryDependencies += "org.reactivecouchbase" %% "reactivecouchbase-play" % "0.3"
resolvers += "ReactiveCouchbase" at "https://raw.github.com/ReactiveCouchbase/repository/master/releases"
```

Listing 5.1: Adding ReactiveCouchbase to Play via build.sbt

To activate the plugin, a line has to be added to conf/play.plugins, where the first element defines the activation order and the second denotes the plugin.

```
\small\ttfamily{400:org.reactivecouchbase.play.plugins.CouchbasePlugin}
```

Listing 5.2: Activating the plugins via play.plugins

Now the connection to a previously installed Couchbase Server has to be configured. A Couchbase bucket named "guide-editor" was created using the database's admin console.

```
couchbase {
  buckets = [{
    host="127.0.0.1"
    port="8091"
    base="pools"
    bucket="guide-editor"
    user="<user>"
    pass="<pass>"
    timeout="0"
  }]
}
```

Listing 5.3: Attaching the driver to the local Couchbase Server

5.4.3 Adding µPickle as Lightweight Serialization Framework

µPickle (pronounced micro-Pickle) is a lightweight open source serialization/deserialization[4] framework developed by Li Haoyi [µPickle]. It works on the server side JVM as well as in the Scala.js browser part, making it's a good fit for transferring objects between the client and the server in this project. This is achieved by avoiding reflection in favor of performance, depending on nothing but the standard library on the Scala.js part.

[4] serialization/deserialization is also called pickling/unpickling, explaining the framework's name

For enabling it, a dependency has to be added to both the client and the server projects in the build.sbt file.

```
resolvers += "bintray/non" at "http://dl.bintray.com/non/maven"
# client
libraryDependencies += "com.lihaoyi" %%% "upickle" % "0.2.8"
# server
libraryDependencies += "com.lihaoyi" %% "upickle" % "0.2.8"
```

Listing 5.4: Adding a dependency to the client and server project

Note the new "%%%" operator, which adds the Scala.js version to the artifact name to be retrieved, extending the regular sbt "%%" operator that adds the Scala version.

5.4.4 Adding Bootstrap as UI Framework

Bootstrap is a popular open source web UI library, started by two Twitter engineers in 2011 [Bootstrap], providing a rich set of UI components like buttons, dropdowns, glyphs and modal dialogs in the browser. It consists primarily of cascading style-sheets (css), a font containing the glyphs for common actions and optional JavaScript extensions for programmed behavior.

After adding the bootstrap.css to Play's folder "public/css" and the glyph fonts to the "public/fonts" folder of the Play application, the components can be used by creating HTML elements assigned to the desired css classes and matching the needed structure of the respective UI component.

5.4.5 Activating the Less Compiler

Cascading style sheets are the common way to separate design and data of web sites and applications. Less is a meta language on top of css, adding missing concepts to write more concise and better readable css-code. It was started in 2009 as open-source project by Alexis Sellier [less].

After enabling less, assets can be placed under assets/less and are automatically compiled to regular css files with the regular 'compile' command or when an automatic rebuild is triggered by a browser page refresh after changing a file.

5.5 Architecture and Design

5.5.1 Overview

In former days, web applications consisted mainly of a collection of several html documents and forms. On every user action, a new document were created at server side, and served to the client already containing the data to display, where the browsers rendered the page.

To create a modern reactive web application, full page reloads giving the feeling of navigating a website have to be minimized and some parts of the server side code code has to be moved to the client. After an initial page request to Play's web server, the web browser get's the basic html document and loads all referenced resources, consisiting of css files, images and the compiled client side scala code now available as javascript. From now on, all interactions with the webserver are done asynchronously by the client side controllers and updating parts of the user interface when necessary.

For example, to setup the main screen, a list of sites is requested using an AJAX call from the server, which in turn performs a database request to load the data and send it to the client, where it is stored in a client side model accessed by the view to render GUI elements.

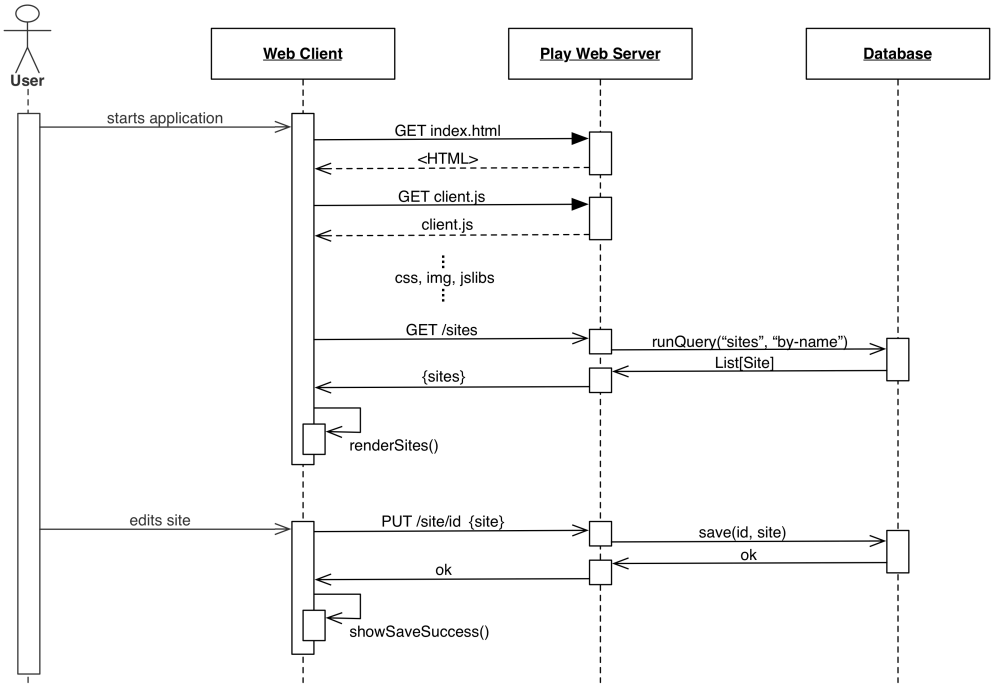

FIGURE 5.2: High level sequence diagram of initial page request and edit action

When the user edits an entity, e.g. a site, the client passes the updated site to the server for persistent storage and informs the user of the result.

In the next subsections, different involved concepts and technologies are discussed, starting from the server side HTTP interface.

5.5.2 The Controller

5.5.2.1 Designing the HTTP REST Interface

All actions initiated by the web client code involving the server are performed using a Representational State Transfer (REST) interface accessed with standard HTTP requests using common HTTP methods GET, PUT, DELETE, thus forming a so called RESTful API. The elegance of such an API is that it uses directly what HTTP provides instead of building another layer of abstraction upon it.

For all types of the guide's entities, sites, areas and beacons, four actions are defined for the following tasks:

- Get a list of all entities of that type
- Get the full details of a single entity instance
- Update a single entity instance
- Delete an entity instance

Additionally, an action is defined for publishing a site to the shared database as site aggregate.

Play has a good support for building such an interface. A file named "routing" is used to map the API calls to the corresponding controller action. The following listing is an excerpt of the guide's back-end routing file showing mainly area related routes.

```
# Home page and tests
GET         /                       controllers.Application.index

# Area related routes
GET         /site/:siteId/areas     controllers.Areas.list(siteId: String)
GET         /area/:id               controllers.Areas.getArea(id: String)
PUT         /area/:id               controllers.Areas.save(id: String)
DELETE      /area/:id               controllers.Areas.delete(id: String)

# Map static resources from the /public folder to the /assets URL path
GET     /assets/*file       controllers.Assets.at(path="/public", file)
```

Listing 5.5: Excerpt from the routing file

As stateless interface, the server does not keep track of what site is beeing edited. All needed information to process any request has to be encoded in the HTTP verb, the url and the request body. For that reason, there cannot be an action like "get all areas/beacons of the currently active site" combined with an action to set the current area. The site id is included in the api call to obtains the areas and in the call to obtain the beacons, too. One big benefit of that approach is being able to distribute requests over multiple independent servers.

To get a list of all areas from a server running locally on port 9000 (the standard for Play), a normal HTTP GET request to the URL "http://localhost:9000/site/Site-CXN01/areas" can be issued. Since that is the same request type a browser creates, the interface can be queried by pointing a web browser to that address. Play detects the first matching route, binds "Site-CXN01" to the siteId variable, checks if the type matches and calls the controllers.Areas.list("Site-CXN01"). The next subsection shows this exemplary Play Action.

5.5.2.2 Controller Actions

Play actions are, in essence, functions that take a parameter of type Request and return a Play Result, that represent a HTTP response with a header and a body. There are several helper for building results, the most important one is "Ok(body)" that creates a HTTP response "200 OK" (cf. [HTTP-Codes]) with the passed content as body.

Synchronous actions are defined using the apply method of the Action companion object, that takes a closure returning a Result. So the a very simple action can be built with ActionOk("Hello World!").

To define asynchronous actions, there is a similar helper method "Action.async", that this time takes a closure returning a future result - formally a Future[Result]. Since Scala Futures are an interesting way to model asynchronous and concurrent execution, widely used in Play and the used Couchbase database driver, section 5.5.3 contains a short excursus on Futures and Promises.

```
/**
* Action for retrieving a serialized list of all areas in the site
*/
def list(siteId: String) = Action.async {

  // Generate a Future[Result] containing a list of areas retrieved from
    db
}
```

LISTING 5.6: An exemplary asynchronous controller action

The construction of the Future[Result] containing the areas areas is discussed in section 5.5.4.3.

5.5.3 Excursus: Futures and Promises

5.5.4 The Model

5.5.4.1 Designing Classes to Cross-Compile

5.5.4.2 Creating NoSQL Design Documents

In contrast to Couchbase Mobile, views are created directly on the database server. Views are grouped by so called design documents

5.5.4.3 Accessing the Database using Scala Futures

The interface of the Reactive Couchbase Driver makes extensive use of Futures, a language feature introduced in Scala 2.10[5].

In this section the Future concept is explained using a simple query which retrieves a list of all sites stored in the guide-editor Couchbase bucket.

```
/**
* Action for retrieving a serialized list of all sites
*/
def list(siteId: String) = Action.async {

  // Get a Future on a list of areas retrieved from db
  val areaFuture = Area.getAreas(siteId)

  // On completion, generate an Ok Result containing the serialized area
    list
  areaFuture.map {
    areaList => Ok(

      // write is upickle's method for serializing the data
      write(areaList)
```

[5]The most recent Edition of [Odersky, 2010] at the time of writing doesn't cover Scala 2.10. A more detailed description of Futures can be found in the Scala Online Documentation [Scala-Futures].

```
    )
  }
}
```

Listing 5.7: An exemplary controller action

5.5.4.4 Mini DSL for Defining Buildings

Selected buildings that are part of the site have to be modelled with their position and a floorplan for every floor. The floorplans are later overlaid to the map for indoor site modelling and indoor analytics.

For defining the building with their floors, a little internal DSL was designed, letting the beauty of the Scala language emerge.

The resulting definition of a building resembles natural language.

```
// Techcenter Park 31
new Building  placedAtSouthWest Coordinates(47.652374, 9.167594)
              andNorthEast      Coordinates(47.652580, 9.168151)
              using ("park31-attic-floor"   asFloor 3)
              using ("park31-2nd-floor"     asFloor 2)
              using ("park31-1st-floor"     asFloor 1)
              using ("park31-ground-floor"  asFloor 0)
```

Listing 5.8: Definition of a building with its floors in Scala

The two methods of the Building class "placedAtSouthWest" and "andNorthEast" take a parameter of type Coordinates. They return a reference to the object to create a so called fluent interface, where method calls ca be chained. The last method "using" takes a tuple of type (Int, String) that groups a floor number with the corresponding floorplan name. An implicit conversion from String to an helper class enables calling the "asFloor" method, that creates the tuple, on a string.

Paired with Scala's infix notation, the high readability of the DSL is finally achieved. The involved code is attached in Appendix A.

5.5.5 The View

5.5.5.1 Concept

The user interface of the back-end was first conceptualized using the already mentioned mockup tool.

The first screen should present a list of all sites as bootstrap panel components with the site's name and description. Next to the list a zoomed out map shows the geographical position of the single sites. An "Add Site" button at the bottom of the list allows to create new sites.

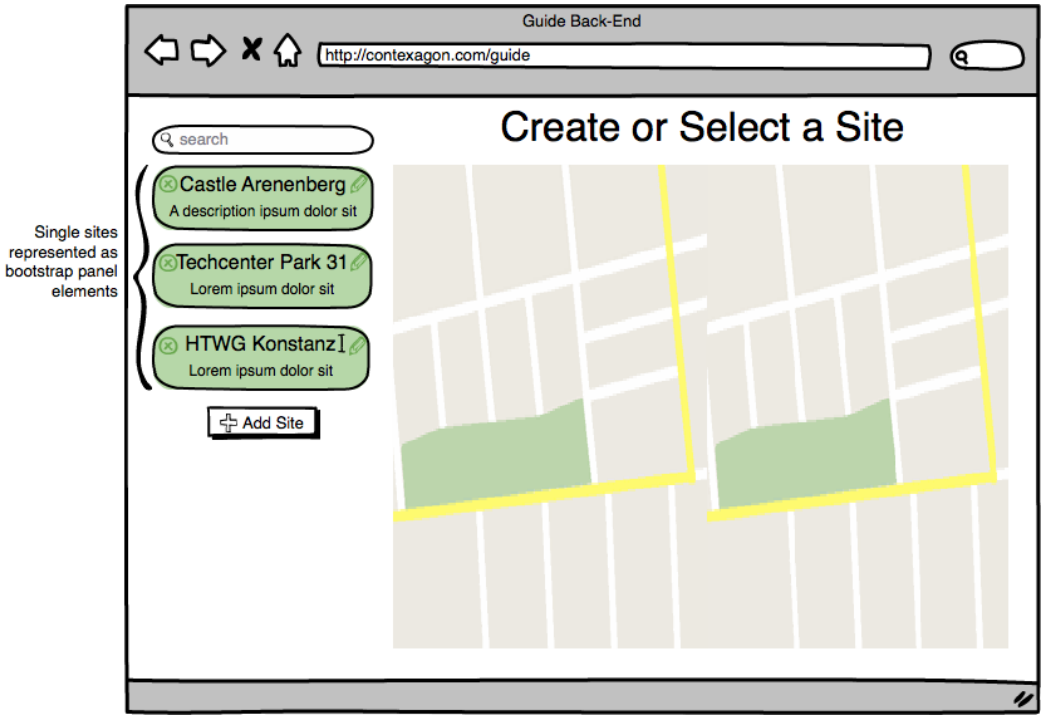

FIGURE 5.3: Concept of the site list screen

For fast and comfortable editing, an edit icon (the small pens on the concept screen) has to activate in-place editing of the name and the description when clicked. Another icon at the top left of each site panel is connected to a delete action, asking the user with a modal dialog if he is sure before actually deleting that site.

When a site panel is clicked, the next screen shows a list of areas contained in that map using again a panel for every ares. A tab control on top of the list enables the user to switch the list to the beacons. The panels can be clicked to open a bigger panel showing all details of the selected site or beacon.

The map depicts the single geographical definition of the areas using geometric shapes as circles, squares or polygons. Clicking on an area should open the corresponding site the same way as it was selected on the list. The geometric shapes should be editable directly on the map. In a similar way, the positions of single beacons have to be represented on the map by a distinctive shape or icon.

A button on the top right of the screen starts the publishing of the current site to the shared database.

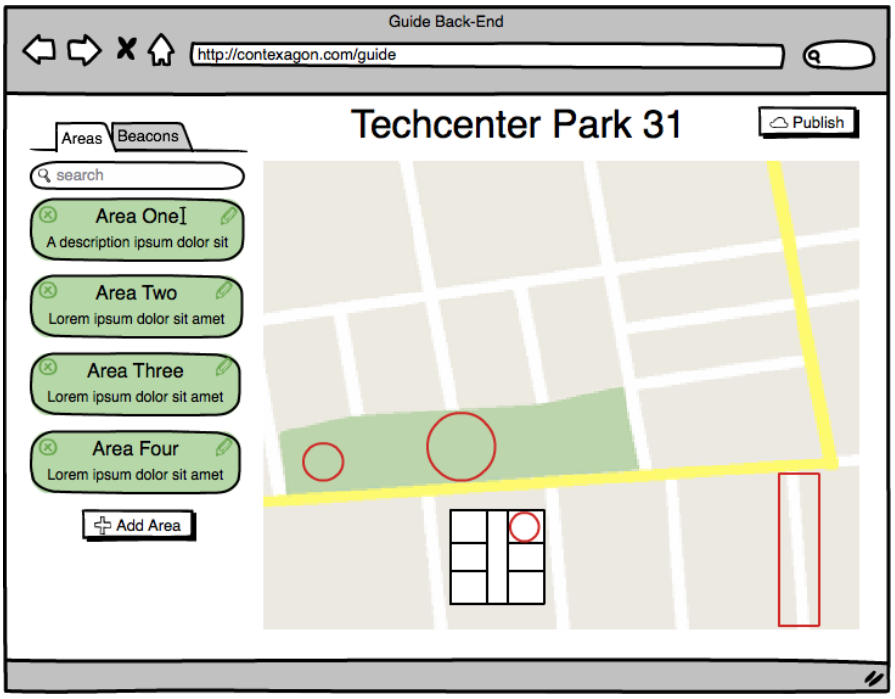

FIGURE 5.4: Concept of the modeling screen for a single site

5.5.5.2 Play HTML Templates

Play has a powerful HTML template engine supporting typed parameters and nesting. For the guide back-end, a HTML template is used to serve the initial HTML document with the basic HTML structure containing a left column for the lists and a right column for the map used by the client code. This document contains references to the compiled css files, external js libraries and, most important, the js files compiled by Scala.js.

An an universal escape character "@" allows to interweave Scala variables and code with templates. This snippet shows how a parameter named title is inserted in the template.

```
<title>@title</title>
<link rel="stylesheet" media="screen" href="@routes.Assets.at("
    stylesheets/bootstrap.css")">
```

In the second line, a helper method of a managed object is called returning a reverse route. This is useful to avoid duplicating path information already provided in the routes files.

5.5.5.3 The Web Client View in Scala.js

The Scala code inside the "editorClient" project is integrated in the main play template calling the helper method provided by Scala.js.

```
@playscalajs.html.scripts("/assets", projectName = "editorClient")
```

The web client is implemented using an own MVC system. Transferred objects are hold in a local model, a sort of local cache. This enables actions like resorting or filtering the displayed list of entities without involving the server. Controller actions interact with the server side REST interface and update the local model. The view contains own html templates representing single sites, areas, and beacons.

Even in the code targeted for to the browser, Scala Futures can be used. For example, displaying a dialog and waiting for user input is an asynchronous action that can be modeled nicely using a Future, as done for the back-end's Dialog class.

```
class Dialog {
  val promise = Promise[Int]()
  /**
   * Show a dialog
   * @param title The title to display.
   * @param message The message appearing body of the dialog.
   * @param options A Map defining the possible user choices.
   *        The key is the option index and the value the buttons label.
   * @return Future[Int] The future index of the pressed button
   */
  def show(title: String, message: String, options: Map[Int, String]) :
    Future[Int] = {

    // [...] Build the dialog with a button for each options element
    // register actions for every button with the corresponding index
    jQuery(s"button#$index").click( (event: JQueryEventObject) =>
      // complete the promise.future with the number of the user's choice
      promise success $index
      // [...] close the dialog
    )
    // [...] display the dialog and return the promise's future
    promise.future
  }
}
```

Listing 5.9: "Excerpt of the Dialog class"

After opening the dialog, a onSuccess callback is connected to the obtained future with a case block handling the possible user choices[6].

```
val futureUserChoice = Dialog.show("Confirm Deletion", "Sure to delete
    the Area?", Map(0->"Cancel", 1->"Delete"))
futureUserChoice onSuccess {
```

[6] No more Future use cases in future, I promise.

```
  case 0 => logInfo("Cancel pressed") // Nothing to do
  case 1 => logInfo("Delete pressed")
            areaController.deleteArea(id) onSuccess {
              case true => jQuery("#arealist div#" + id).fadeOut()
            }
}
```

LISTING 5.10: "Excerpt of the Dialog class"

5.5.5.4 Interaction with a Mapping Service

Being able to display a site's map and interact with it is essential for the guide back-end.

For the first development stage that is part of this thesis, Google Maps were chosen as mapping service provider with it's rich JavaScript API v3 [GoogleMaps-API].

With later development in mind, for example directly using floorplans and other site representations without google maps, the concrete map component is decoupled from the rest of the system using a MapView trait. The class GoogleMapView implements all methods defined in MapView interacting with a google.maps.Map object. Thus it serves as adapter, converting the guide back end data types to the one's used in the Google API.

To access the actual Maps JavaScript API inside the GoogleMapView class in a typed way, the "Scala-js-ts-importer" tool by Sébastien Doeraene [TS-Importer] was used, which translates TypeScript type definitions to Scala code compatible with Scala.js. As already stated in section 5.3.2.1, [TS-TypeRepository] offers type definitions of a vast amount of JavaScript APIs, including one for the Google Maps API.

After some smaller modifications[7], the type definitions could be imported and the API accessed as if it were written in Scala.

For bridging semantically equivalent types like our Coordinates class and Google's LatLng class, Scala's implicit conversions are perfectly suited.

```
// Define an implicit conversion from our coordinates to the ones used
// in Google Maps. So we can use ours where Google's LatLng are expected.
implicit def coordinatesToLatLong(coordinates: Coordinates): LatLng = {
  new LatLng(coordinates.lat, coordinates.lon)
}
// And the other way around.
```

[7] The author notes on the project's site, that "the process is not 100 % accurate, so manual editing is often needed afterwards. This can be improved, but not to perfection, because the features offered by the type systems of TypeScript and Scala.js differ in some subtle ways." [TS-Importer]

```
implicit def coordinatesToLatLong(latLng: LatLng): Coordinates = {
  new Coordinates(latLng.lat, latLng.lng)
}
```

LISTING 5.11: "Implicit conversions between two different geographic coordinates classes"

With this implicit conversions and the type definitions in scope, we can interact with Google Maps in a concise and beautiful way. As a short example, the implementation of the addMarker method of our MapView trait is included. Note that "new Marker()" creates a google.maps.Marker object and setPosition is a (LatLng) -> Unit function.

```
/**
* Add a marker to the map
*
* @param id an id allowing to reference to the marker later
* @param coordinates
*/
def addMarker(id: String, coordinates: Coordinates) = {
  val marker = new Marker()
  marker.setTitle(id)
  marker.setMap(map)
  marker.setPosition(coordinates)
  marker.setIcon(new MarkerImage(url = "/images/beacon.png", anchor = new
     Point(16,16)))
  markers(id) = marker
}
```

LISTING 5.12: "A method using JavaScript type definitions and implicit conversions"

5.6 Summary

This chapter provided an overview of architectural decisions and the technology stack used to create the back-end. Furthermore, some important design elements were discussed.

The next chapter shows how the back-end is used to model an exemplary site.

Chapter 6

Example Site Model

6.1 Introduction

6.2 Modelling the Tech Center

6.2.1 Indoor Modelling

6.2.2 Outdoor Modelling

6.3 Testing the CA Guide Front End

6.4 Analytics with the CA Guide Back End

Chapter 7

Conclusion

7.1 Summary

7.1.1 The CA Guide

7.1.2 Swift and Scala

In my opinion, the quality and the success of a programming language is tightly coupled with it's standard library. Currently, Scala has the better libraries, being better composed allowing to find familiar concepts and elements learned before and reused as part of other concepts.

For Swift, I hope Apple will redesign all libraries and APIs and implement them natively in Swift soon.

Despite that, Swift, CocoaTouch and iOS is one of the best platforms available for mobile development, with a superior runtime performance providing an excellent, highly responsive UI experience.

XCode has a long way to go before beeing really stable and providing all the features a modern IDE needs like Swift refactoring support and code navigation, but I think Apple engineers are working on archiving this goal.

7.2 Future Work

WWI Indoor Exhibition at Schloss Arenenberg Guide for the Schlosspark Arenenberg Guide for popular bike and tracking routes in the region of the lake of constance Virtual Geocaching

Appendix A

A Little Scala DSL

```
import shared.model.Coordinates

/**
 * Tiny Internal DSL for defining buildings
 * Maurizio Tidei, 2015.
 */
class Building {

  // The south-west corner
  var sw: Option[Coordinates] = None

  // The north-east corner
  var ne: Option[Coordinates] = None

  // A map floor number -> floorplan image name
  var floors = Map[Int,String]()

  /**
   * Defines the south-west corner
   * @param coord The south-west Coordindates
   */
  def placedAtSouthWest(coord: Coordinates) = {

    sw = Some(coord)
    this
  }

  /**
```

```
    * Defines the north-east corner
    * @param coord The north-east Coordinates
    * @return
    */
  def andNorthEast(coord: Coordinates) = {

    ne = Some(coord)
    this
  }

  /**
   * Adds a floor
   * @param floor a tuple combining the floor number to a
   floorplan image name
   */
  def using(floor: (Int, String)) = {

    floors += (floor._1 -> floor._2)
    this
  }
}

/**
 * Helper class for implicit conversion of Strings
 * @param floorImageName
 */
class FloorImage(floorImageName: String) {

  /**
   * Creates a tuple combining the floorImageName with the floor
   number
   * @param number The floor number
   * @return tuple for Building's using method
   */
  def asFloor(number: Int) = {

    (number, floorImageName)
  }
}

/**
 * Companion object
 */
object Building {
```

```
  // Defines the implicit conversion from String to FloorImage
  implicit def stringToFloorImage(floorImageName: String) = new
   FloorImage(floorImageName)

  // A list of buildings
  val list = List(

  // Techcenter Park 31
  new Building  placedAtSouthWest Coordinates(47.652374, 9.167594)
                andNorthEast      Coordinates(47.652580, 9.168151)
                using ("park31-attic-floor"   asFloor 3)
                using ("park31-1st-floor"     asFloor 2)
                using ("park31-1st-floor"     asFloor 1)
                using ("park31-ground-floor"  asFloor 0)

  )
}
```

Listing A.1: Building.scala

```
/**
 * A geographic coordinate pair
 *
 * @param lat The latitude
 * @param lon The longitude
 * Maurizio Tidei, 2015.
 */
case class Coordinates(var lat: Double, var lon: Double) {


  /**
   * Add two coordinates
   * @return the resulting Coordinate
   */
  def +(that: Coordinates): Coordinates = {

    return Coordinates(this.lat + that.lat, this.lon + that.lon)
  }


  /**
   * Multiply a coordinate with a tuple of two doubles
```

```
   * @return the resulting Coordinate
   */
  def *(tuple: Tuple2[Double, Double]): Coordinates = {

    return Coordinates(this.lat * tuple._1, this.lon * tuple._2)
  }

}
```

Listing A.2: Coordinates.scala

Bibliography

Books and Papers

[Abowd, 1999] Gregory D. Abowd and Anind K. Dey. "Towards a Better Understanding of Context and Context-Awareness". In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. HUC '99. Karlsruhe, Germany: Springer-Verlag, 1999, pp. 304–307. ISBN: 3-540-66550-1.

[Angermann, 2012] Michael Angermann and Patrick Robertson. "FootSLAM: Pedestrian Simultaneous Localization and Mapping Without Exteroceptive Sensors - Hitchhiking on Human Perception and Cognition." In: *Proceedings of the IEEE* 100.Centennial-Issue (2012), pp. 1840–1848.

[Apple, 2014a] Apple. *The Swift Programming Language*. 1st. Swift Programming Series. Apple Inc., 2014.

[Apple, 2014b] Apple. *Using Swift with Cocoa and Objective-C*. 1st. Swift Programming Series. Apple Inc., 2014.

[Borenstein, 1997] J. Borenstein et al. "Mobile Robot Positioning Sensors and Techniques". In: (1997).

[Dean, 2008] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782.

[Evans, 2003] Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321125215.

[Gamma, 1995] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[Hilton, 2014] Peter Hilton, Erik Bakker, and Francisco Canedo. *Play for Scala: Covers Play 2*. Manning Publications, 2014. ISBN: 1617290793.

[McGrath, 2013] Michael J. McGrath and Cliodhna Ní Scanaill. *Sensor Technologies: Healthcare, Wellness and Environmental Applications*. Apress, 2013. ISBN: 978-1-4302-6013-4.

[Musulin, 2014] Ivo Musulin, David Brčić, and Serdjo Kos. "A Study of Smartphone Satellite Positioning Performance at Sea Using GPS and GLONASS Systems". In: *In Proceedings of the 22nd International Symposium on Electronics in Transport (ISEP 2014)*. 2014.

[Noureldin, 2013] Aboelmagd Noureldin, Tashfeen B. Karamat, and Jacques Georgy. *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*. 1st. Springer, 2013. ISBN: 978-3-642-30465-1.

[Odersky, 2010] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide*. 2nd. USA: Artima Incorporation, 2010. ISBN: 0981531601.

[Paulo, 2014] Ferreira Paulo and Alves Pedro. *Distributed Context-Aware Systems*. Springer-Verlag, 2014.

[Sadalage, 2013] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1st. Third printing. Addison-Wesley Professional, 2013. ISBN: 0-321-82662-0.

[Schilit, 1994] Bill Schilit, Norman Adams, and Roy Want. "Context-Aware Computing Applications". In: *In Proceedings of the Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society, 1994, pp. 85–90.

[Scoble, 2013] Robert Scoble and Shel Israel. *Age of Context: Mobile, Sensors, Data and the Future of Privacy*. 1st. Patrick Brewster Press, 2013.

[Weiser, 1991] Mark Weiser. "The computer for the 21st century". In: *Scientific American* 265.3 (Sept. 1991), pp. 66–75.

Web Sources

[Steiner-Sarnen] Steiner Sarnen Schweiz AG. *The Company Steiner Sarnen Schweiz AG*. URL: http://www.steinersarnen.ch (visited on 03/03/2015).

[React-Couchbase] Mathieu Ancelin and contributors. *Reactive Couchbase*. URL: http://reactivecouchbase.org (visited on 03/02/2015).

[GCD-Reference] Apple. *Grand Central Dispatch (GCD) Reference*. URL: https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html (visited on 02/18/2015).

[CoffeeScript] Jeremy Ashkenas and contributors. *CoffeeScript Homepage*. URL: http://coffeescript.org (visited on 03/17/2015).

[Balsamiq] LLC Balsamiq Studios. *Balsamiq Mockups*. URL: https://balsamiq.com/products/mockups/ (visited on 04/07/2015).

[Interview-Doeraene] Elliot Bentley. *Scaling from the back to the frontend - Interview: Compiling Scala to JavaScript with Scala.js*. URL: http://jaxenter.com/interview-compiling-scala-to-javascript-with-scala-js-107247.html (visited on 03/17/2015).

[Play-Decision] Jonas Bonér. *Why did Typesafe select Play for their stack instead of Lift?* URL: http://www.quora.com/Why-did-Typesafe-select-Play-for-their-stack-instead-of-Lift (visited on 03/09/2015).

[React-Manifesto] Jonas Bonér et al. *The Reactive Manifesto 2.0*. URL: http://typesafe.com/blog/reactive-manifesto-20 (visited on 03/02/2015).

[HTTP-Codes] World Wide Web Consortium. *HTTP/1.1: Status Code Definitions*. URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html (visited on 04/10/2015).

[Scala.js-0.6] Sébastien Doeraene. *Scala.js no longer experimental*. URL: http://www.scala-lang.org/news/2015/02/05/scala-js-no-longer-experimental.html (visited on 03/14/2015).

[Scala.js] Sébastien Doeraene. *Scala.js - the Scala to JavaScript compiler*. URL: http://www.scala-js.org (visited on 03/17/2015).

[TS-Importer] Sébastien Doeraene. *TypeScript Importer for Scala.js*. URL: https://github.com/sjrd/scala-js-ts-importer (visited on 02/17/2015).

[CocoaPods] Eloy Durán and contributors. *The CocoaPods open source dependency manager for iOS and OSX projects*. URL: http://cocoapods.org/ (visited on 03/10/2015).

[Geoid-Grace] *GRACE - Gravity Recovery and Climate Experiment*. URL: http://www.csr.utexas.edu/grace/gravity/gravity_definition.html (visited on 02/23/2015).

[BLE] Bluetooth Special Interest Group. *Technical Information - The Low Energy Technology Behind Bluetooth Smart*. URL: http://www.bluetooth.com/Pages/low-energy-tech-info.aspx (visited on 03/20/2015).

[Scala-Futures] Philipp Haller et al. *Scala Documentation - Futures and Promises*. URL: http://docs.scala-lang.org/overviews/core/futures.html (visited on 03/11/2015).

[CocoaLumberjack] Robbie Hanson and contributors. *A fast & simple, yet powerful & flexible logging framework for Mac and iOS*. URL: https://github.com/CocoaLumberjack/CocoaLumberjack (visited on 03/10/2015).

[CL-Benchmarks] Robbie Hanson and contributors. *CocoaLumberjack Analysis of Performance with Benchmarks*. URL: https://github.com/CocoaLumberjack/CocoaLumberjack/blob/master/Documentation/Performance.md (visited on 03/10/2015).

[µPickle] Li Haoyi. *µPickle - a lightweight serialization library for Scala*. URL: https://github.com/lihaoyi/upickle (visited on 03/20/2015).

[Proximity] Apple Inc. *CLBeacon Class Reference: CLProximity enumeration*. URL: https://developer.apple.com/library/prerelease/ios/documentation/CoreLocation/Reference/CLBeacon_class/index.html#//apple_ref/c/tdef/CLProximity (visited on 03/30/2015).

[Swift-1.2] Apple Inc. *Swift 1.2 improvements*. URL: https://developer.apple.com/swift/blog/?id=22 (visited on 03/14/2015).

[Xcode-beta] Apple Inc. *Xcode download page, containing pre-release versions*. URL: https://developer.apple.com/xcode/downloads/ (visited on 03/10/2015).

[Attachment] Couchbase Inc. *Attachment | Couchbase - Mobile Developers*. URL: http://developer.couchbase.com/mobile/develop/guides/couchbase-lite/native-api/attachment/index.html (visited on 02/04/2015).

[Couchbase-Mobile] Couchbase Inc. *Couchbase Mobile NoSQL Database*. URL: http://www.couchbase.com/nosql-databases/couchbase-mobile (visited on 01/26/2015).

[Couchbase] Couchbase Inc. *Couchbase NoSQL Database Server*. URL: http://www.couchbase.com/nosql-databases/couchbase-server (visited on 03/11/2015).

[GoogleMaps-API] Google Inc. *Google Maps JavaScript API V3 Reference*. URL: https://developers.google.com/maps/documentation/javascript/reference (visited on 01/22/2015).

[TypeScript] Microsoft Inc. and contributors. *The TypeScript Language*. URL: http://www.typescriptlang.org (visited on 03/12/2015).

[Scala-DSL-Example] Jan Macacek. *Type-safe DSL*. 2011. URL: http://www.cakesolutions.net/teamblogs/2011/12/10/type-safe-dsl (visited on 01/05/2015).

[Scala.js-Crosscompiling] Vincent Munier. *Play with Scala.js*. URL: https://github.com/vmunier/play-with-scalajs-example (visited on 03/17/2015).

[NASA-Geoid] NASA. *The NASA GSFC and NIMA Joint Geopotential Model*. URL: http://cddis.nasa.gov/926/egm96/egm96.html (visited on 02/23/2015).

[Bootstrap] Mark Otto. *Bootstrap from Twitter*. 2011. URL: https://blog.twitter.com/2011/bootstrap-twitter (visited on 03/14/2015).

[less] Alexis Sellier and contributors. *Less history*. URL: http://lesscss.org/about/#team (visited on 03/22/2015).

[Interview-Eich] Charles Severance. *Interview with Brendan Eich, the creator of JavaScript*. 2012. URL: https://www.youtube.com/watch?v=IPxQ9kEaF8c&feature=youtu.be (visited on 03/17/2015).

[Core-Plot] Eric Skroch and contributors. *The Core Plot open source plotting framework*. URL: https://github.com/core-plot/core-plot (visited on 03/10/2015).

[specs2] Eric Torreborre and contributors. *specs2*. URL: http://etorreborre.github.io/specs2/ (visited on 04/10/2015).

[Typesafe-Activator] Dick Wall. *Typesafe Activator, What *is* it?* URL: https://typesafe.com/blog/typesafe-activator-what-is-it (visited on 02/26/2015).

[TS-TypeRepository] Boris Yankov and contricutors. *DefinitelyTyped - Repository for TypeScript type definitions*. URL: http://definitelytyped.org (visited on 03/12/2015).