



HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG  
UNIVERSITY OF APPLIED SCIENCES

# Design and Implementation of a Platform for Context-Aware Mobile Guides

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:

**Design and Implementation of a Platform for  
Context-Aware Mobile Guides**

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: Design and Implementation of a Platform for Context-Aware  
Mobile Guides

Masterkandidat: Maurizio Tidei

Firma: HTWG / contexagon GmbH

Betreuer: Prof. Dr. Marko Boger  
Prof. Dr. Christian Johner

Abgabedatum: 16.04.2015

Schlagworte: Contextual Computing, Indoor Navigation, Scala, Swift

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 23.11.1980 in Singen*, dass ich

- (1) meine Masterarbeit mit dem Titel

**Design and Implementation of a Platform for Context-Aware  
Mobile Guides**

bei der HTWG / contexagon GmbH unter Anleitung von Prof. Dr. Marko Boger selbstständig 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

<b>Abstract</b>	ii
<b>Ehrenwörtliche Erklärung</b>	iii
<b>Acknowledgements</b>	v
<b>Contents</b>	vi
<b>List of Figures</b>	ix
<b>Abbreviations</b>	x
<b>1 Introduction</b>	1
1.1 The goal of this work . . . . .	1
1.2 Motivation . . . . .	2
1.3 The Approach . . . . .	2
<b>2 Theoretical Background and Fundamentals</b>	4
2.1 Introduction . . . . .	4
2.2 What is Context-Aware Computing? . . . . .	4
2.2.1 Definition of Context . . . . .	5
2.2.2 Categories of Context . . . . .	5
2.2.3 Categories of Context-Aware Applications . . . . .	6
2.2.4 Ubiquitous Computing . . . . .	7
2.3 Sensors of Mobile Devices . . . . .	8
2.3.1 Introduction . . . . .	8
2.4 Localization Techniques . . . . .	9
2.4.1 Introduction . . . . .	9
2.4.2 Classification . . . . .	9
2.4.2.1 Techniques Using Relative Measurements . . . . .	9
2.4.2.2 Techniques Using Absolute Measurements . . . . .	10
2.4.3 Discussion . . . . .	11
2.5 Geographic Coordinates . . . . .	12
2.5.1 Introduction . . . . .	12
2.5.2 Standards . . . . .	12
2.5.3 WGS84 Coordinates . . . . .	13

2.5.4	Using different Mapping Providers . . . . .	14
2.6	Bluetooth Low Energy Beacons - iBeacons . . . . .	15
2.6.1	What is an iBeacon? . . . . .	15
2.6.2	The iBeacon Prototype . . . . .	16
2.6.2.1	Design and Implementation . . . . .	16
2.6.2.2	Measurements and Insights . . . . .	17
2.7	Swift . . . . .	17
2.7.1	What is Swift? . . . . .	17
2.7.2	Comparison Scala and Swift . . . . .	18
2.7.3	Major Differences to Scala . . . . .	19
2.7.3.1	Implicit Type Conversions . . . . .	19
2.7.3.2	Optionals . . . . .	20
2.7.3.3	Getter and Setter . . . . .	21
2.7.3.4	Exception Handling . . . . .	22
2.7.3.5	Named Parameters . . . . .	23
2.7.3.6	The Legacy of Objective-C . . . . .	23
2.7.3.7	Summary . . . . .	24
2.8	iOS and Cocoa Touch . . . . .	24
2.9	Couchbase NoSQL Database . . . . .	24
2.9.1	What is a NoSQL Database? . . . . .	24
2.9.2	Couchbase . . . . .	25
2.10	Typesafe Reactive Platform . . . . .	26
2.10.1	Overview . . . . .	26
2.10.2	Activator Tool . . . . .	27
2.10.3	Play Web Framework . . . . .	27
<b>3</b>	<b>System Overview</b> . . . . .	<b>29</b>
3.1	Introduction . . . . .	29
3.2	Data Structure . . . . .	30
3.2.1	Exhibition Modelling . . . . .	30
3.2.2	Sensor Data . . . . .	31
3.3	Setting up the Couchbase Server Infrastructure . . . . .	32
<b>4</b>	<b>CA Guide Mobile Application</b> . . . . .	<b>33</b>
4.1	Introduction and Goal . . . . .	33
4.2	The Target Platform . . . . .	33
4.3	Setting Up the Development Platform . . . . .	34
4.4	Architecture . . . . .	35
4.4.1	Setup of Couchbase Server and Sync Gateway . . . . .	39
4.5	Sensors Layers . . . . .	39
4.6	Application Logic . . . . .	40
4.7	Presentation Layer . . . . .	40
<b>5</b>	<b>CA Guide Back End</b> . . . . .	<b>41</b>
5.1	Introduction . . . . .	41
5.2	The Target Platform . . . . .	42
5.3	Architectural Decisions . . . . .	42

5.3.1	The Client and Server Code Gap . . . . .	42
5.3.2	Solution Attempts . . . . .	43
5.3.2.1	Improved Javascript . . . . .	43
5.3.2.2	Unified Programming Language for Server and Client Code	44
5.4	Setting up the Development Platform . . . . .	45
5.4.1	Project Structure, Scala and Play . . . . .	45
5.4.2	Adding Reactive Couchbase as Scala Database Driver . . . . .	45
5.4.3	Adding µPickle as Lightweight Serialization Framework . . . . .	46
5.4.4	Adding Bootstrap as UI Framework . . . . .	47
5.4.5	Activating the Less Compiler . . . . .	47
5.5	Architecture and Design . . . . .	48
5.5.1	Overview . . . . .	48
5.5.2	The Controller . . . . .	48
5.5.2.1	Designing the REST HTTP Interface . . . . .	48
5.5.2.2	Controller Actions . . . . .	48
5.5.3	The Model . . . . .	48
5.5.3.1	Designing Classes to Cross-Compile . . . . .	48
5.5.3.2	Creating NoSQL Design Documents . . . . .	48
5.5.3.3	Accessing the Database using Scala Futures . . . . .	48
5.5.4	The View . . . . .	48
5.5.4.1	Play HTML Templates . . . . .	48
5.5.4.2	Mockups . . . . .	48
5.5.4.3	The Client code in Scala.js . . . . .	48
<b>6</b>	<b>System Review</b>	<b>49</b>
6.1	Introduction . . . . .	49
6.2	Modelling the Tech Center . . . . .	49
6.2.1	Indoor Modelling . . . . .	49
6.2.2	Outdoor Modelling . . . . .	49
6.3	Testing the CA Guide Front End . . . . .	49
6.4	Analytics with the CA Guide Back End . . . . .	49
<b>7</b>	<b>Conclusion</b>	<b>50</b>
7.1	Summary . . . . .	50
7.1.1	The CA Guide . . . . .	50
7.1.2	Swift and Scala . . . . .	50
7.2	Future Work . . . . .	51
<b>A</b>	<b>JSON Site Modelling File</b>	<b>52</b>
<b>Bibliography</b>		<b>53</b>

# List of Figures

2.1	Difference in meters between the geoid and the ellipsoid [NASA-Geoid] . . . . .	13
2.2	View angle error on Google (left) and Apple (right) maps . . . . .	15
2.3	The main screen of the iBeacon prototype . . . . .	16
2.4	The iBeacon detail screens, including statistics . . . . .	17
2.5	Couchbase Lite and Sync Gateways. From [Couchbase-Mobile] . . . . .	26
2.6	The Play framework stack. From [Hilton, 2014] with kind permission. . . . .	28
3.1	CA Guite Architecture Overview . . . . .	30
4.1	Layer Model of the Museum Guide Application . . . . .	35
4.2	Layer model with virtual sensors . . . . .	36
4.3	Data flow between the lower layers of the Sensorbase framework . . . . .	37
4.4	Sequence Diagram for the Sensor Layers . . . . .	38
4.5	Simplified Class Diagram for Sensor Data Processing . . . . .	39
5.1	TypeScript in IntelliJ IDEA . . . . .	44

# Abbreviations

<b>INS</b>	Inertial Navigation System
<b>MEMS</b>	Microelectromechanical System
<b>DSL</b>	DomainSpecificLanguage
<b>REPL</b>	Read-Eval-Print-Loop

# 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 for measuring the 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<sup>1</sup> 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.

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

---

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

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 and Fundamentals

### 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<sup>1</sup> and thus merits closer consideration, a subchapter is dedicated to Swift and how 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".

---

<sup>1</sup>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 specified, 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 of Mobile Devices

### 2.3.1 Introduction

#### Accelerometer

#### Gyroscope

Pedometer ref.

#### Magnetometer

#### Barometer

Some of the latest smartphones and tablets are equipped with a 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.

## 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.

## 2.4 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 its 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.

### **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 run on a battery driven mobile device. Furthermore, that technique must get along with the sensors currently offered by common mobile devices. This excludes map-based positioning and landmark navigation.

The third requirement is that the system has to work with an arbitrary held mobile device, even when it is inside a pocket.

GPS is a good choice for outdoor positioning, being technically mature and widely available on mobile devices.

Although inertial navigation combined with human odometry is very promising for indoor navigation, they require the inertial measurement unit (IMU) to be foot mounted. The reason is being able to reset the system and thus the drifting (set the current speed to zero), when the foot rests on the ground.

The authors of [Angermann, 2012] wrote "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 of the same quality as today with a foot mounted IMU.

So this technique is not suitable for providing the location inside 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 (2.6), that fall into the category of active beacons. If the quality of the distance measurement has a contained error, a trilateration can provide the position when signals from two or more iBeacons are received. In this case the beacons could be placed independently from the can be placed at the corners of a room. Otherwise

## 2.5 Geographic Coordinates

### 2.5.1 Introduction

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

### 2.5.2 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.

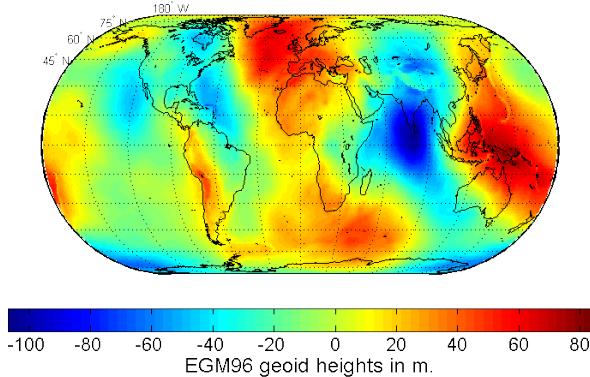


FIGURE 2.1: Difference in meters between the geoid and the ellipsoid [NASA-Geoid]

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

### 2.5.3 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  $0^\circ$  at the equator and goes up to  $+90^\circ$  at the geographic North Pole and down to  $-90^\circ$  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  $0^\circ$ . The prime meridian of WGS84 is the line of longitude on the geoid starting/ending at the geographic North/South Pole and passing through a well defined point 102 meters to the east of the Royal Observatory, Greenwich, London. Eastwards the longitude goes up to  $180^\circ$ , to the west it goes down to  $-180^\circ$ , 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  $\phi$  [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.

#### 2.5.4 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<sup>2</sup>.

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.

---

<sup>2</sup>Until iOS 5.1, Apple used the Google Maps service and decided to start its own competing product



FIGURE 2.2: 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 north 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.

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.

## 2.6 Bluetooth Low Energy Beacons - iBeacons

### 2.6.1 What is an iBeacon?

On Mid 2013, Apple introduced iBeacons as new technology for adding location awareness to applications. iBeacons are little devices containing 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 a configurable time 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 can be detected, the battery is expected to last for up to two years.

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.

## 2.6.2 The iBeacon Prototype

### 2.6.2.1 Design and Implementation

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

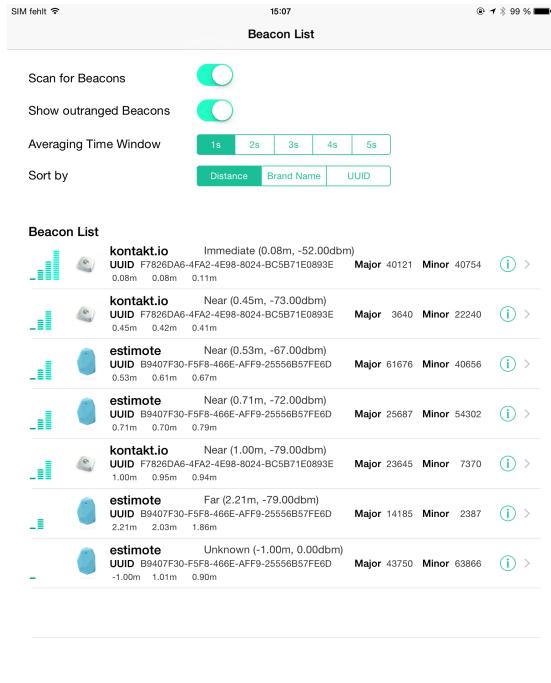


FIGURE 2.3: The main screen of the iBeacon prototype

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.

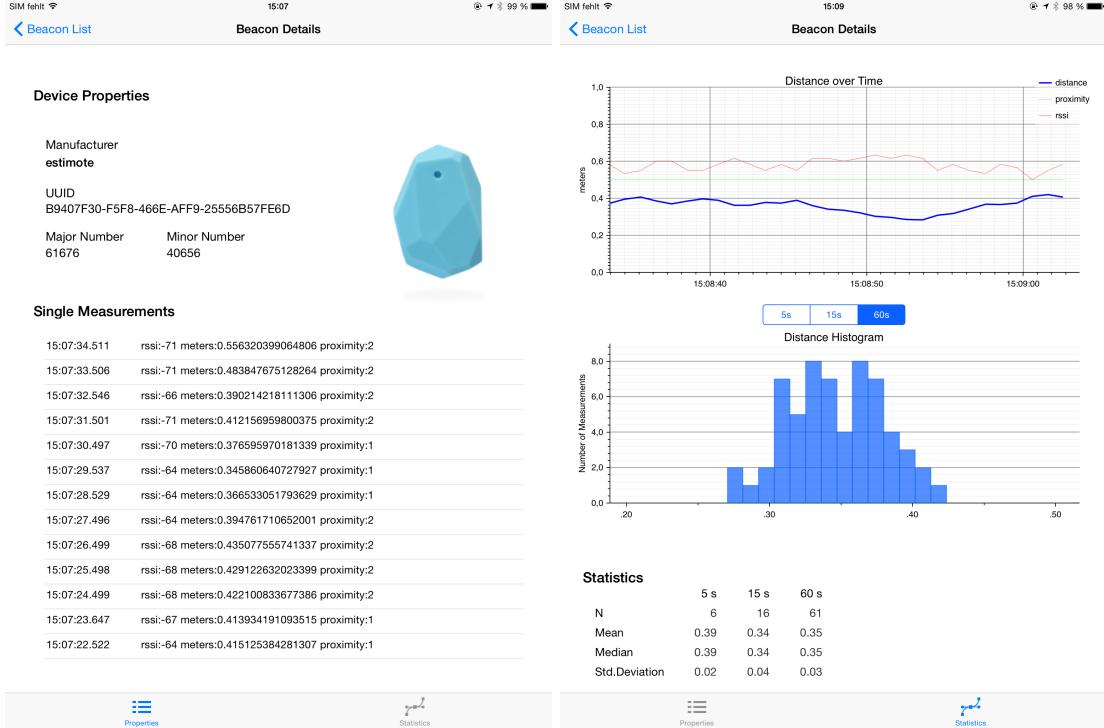


FIGURE 2.4: The iBeacon detail screens, including statistics

### 2.6.2.2 Measurements and Insights

## 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 with 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 <sup>3</sup>	yes	yes
Closures	yes	yes
Curried functions	yes	yes
Operator functions	yes	yes
Named parameters *	yes	yesAs fixed part of the method's signature
Optionals *	via <code>Option[T]</code> class	widely used, dedicated Syntax
Switch with pattern matching	yes	yes
String interpolation	yes <code>"Hello \$nameVar"</code>	yes <code>"Hello \(nameVar)"</code>
Keyword for variable definition	<code>var radius = 1</code>	<code>var radius = 1</code>
Keyword for constant definition	<code>val pi = 3.14</code>	<code>let pi = 3.14</code>
Array literal	<code>Array(1,2,3)</code>	<code>[1,2,3]</code>
Map literal	<code>Map(1-&gt;"a", 2-&gt;"b")</code>	<code>[1:"a", 2:"b"]</code> <sup>4</sup>
If condition must be boolean	yes	yes
Tuples	yes, but without named elements	yes
Ranges	<code>for i &lt;- 0 to 4</code> <code>0 until 4</code>	<code>for i in 0...4</code> <code>0..&lt;4</code>
Constructor	<code>def this()</code>	<code>init()</code>

<sup>3</sup>Functions can be passed to functions, returned from functions, created at runtime and assigned to variables

<sup>4</sup>Maps are called dictionaries in Swift

Language Feature	Scala	Swift
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)
Query 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.<sup>5</sup> But, as M. Odersky rightly wrote in [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
```

---

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

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?`), its 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 being 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 achieved 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<sup>6</sup>.

---

<sup>6</sup>It is still possible to restrict the access to instances of the class by using a separate file for the class definition.

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) {
    }
}
```

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<sup>7</sup>, 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 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

---

<sup>7</sup>Attributes are the counterpart of annotations in the Java world.

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.<sup>8</sup>

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:

---

<sup>8</sup>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.

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

As they emphasize in their book, they 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.<sup>9</sup>

### 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 its 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 its 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:

---

<sup>9</sup>Sometimes that even means choosing different database types for several parts of a single system - what is called polyglot persistence.

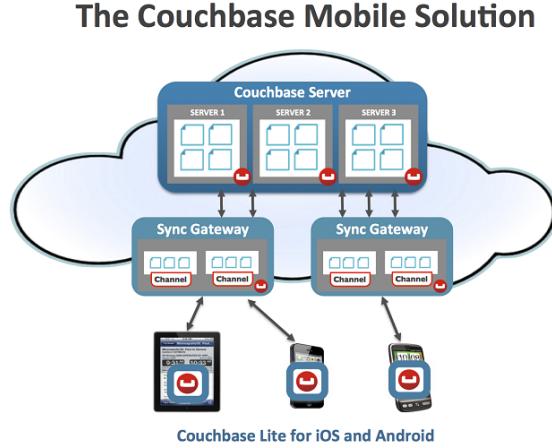


FIGURE 2.5: Couchbase Lite and Sync Gateways. From [[Couchbase-Mobile](#)]

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

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

---

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

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.

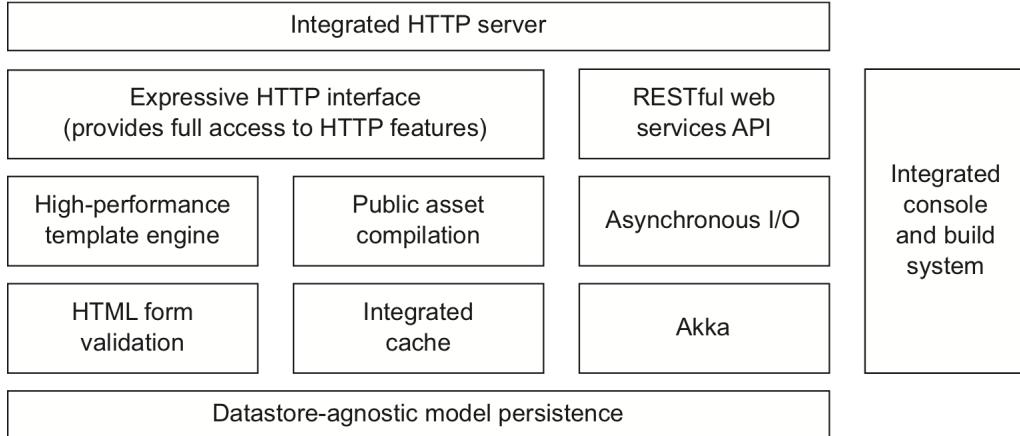


FIGURE 2.6: The Play framework stack. From [Hilton, 2014] with kind permission.

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
- 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 mapping

The Play framework is not the only Scala based web framework. According to Typesafe's CTO, they decided to integrate it in its 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 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") needs to be defined to fit indoor and outdoor exhibitions, 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 avoid big parts of explicit user input.

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

As database server Couchbase was chosen due to its automatic synchronization capabilities, its mobile version and its ability to scale out without completely redesigning the way the database is accessed by the application.<sup>1</sup>

---

<sup>1</sup>After actually having reached the limits of scalability of a single classical relational database server of a commercial project and painfully distributing the database on several servers of the productive system, the idea of an easy scale out using a document based database and map-reduce queries seemed even more ingenious to me.

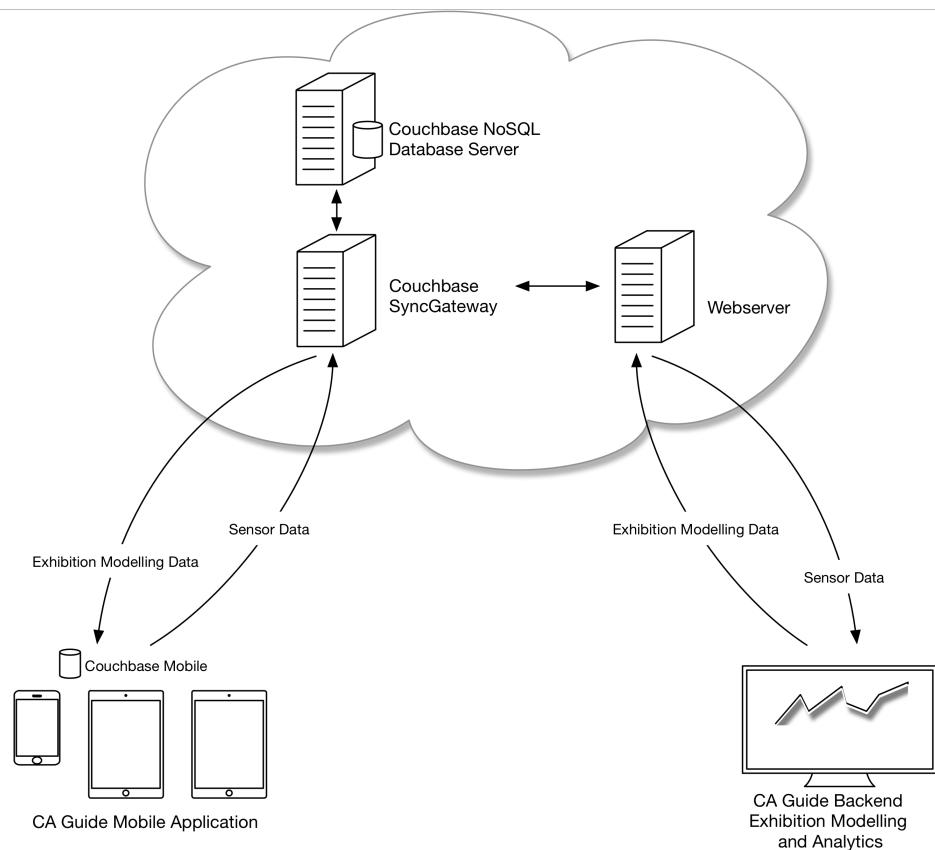


FIGURE 3.1: CA Guide Architecture Overview

Image Mockup CA Guide

Image A Backend Screen Mockup

## 3.2 Data Structure

### 3.2.1 Exhibition Modelling

A single exhibition is modeled in the JSON data format. Compared to XML, JSON is more readable and easier to produce without dedicated tools in a simple text editor.

The entities needed for modelling an exhibition are:

Image entity relationship diagram

With a normal relational database, pretty much of the entities would be saved in separate tables using primary keys and foreign keys to represent the relationships between the tables.

In a document-oriented database the whole exhibition can be stored in an aggregated way.

```
key: "exhibition - CXN01"
value: {
    "type": "exhibition",
    "city": "Konstanz",
    "pois": [
        {
            "name": "Dom",
            "locationDefinition": {
                "type": "circle",
                "location": [15.0020312, 2.3434322],
                "radius": 30,
                "floor": 1
            },
            "text": "",
            "images": [],
            "audio": {
                "level1": "",
                "level2": "",
                "level3": ""
            },
            "videos": []
        }
    ]
}
```

A JSON Schema is 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 is saved to the Couchbase database to allow replaying it for development, testing, presentations and for analytics presented in the system's web backend.

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

```
key: SENSORTRACE-CXN01
value: {
    "date": "2015-02-12",
    "time": "20:15",
    "accelerometer": [
        {"timestamp": 12345678.35, "data": [1.235, 0.364, 0.021]}
    ]
}
```

```
{"timestamp": 12345678.40, "data": [1.217, 0.378, 0.009]},  
...  
],  
"gyroscope": [  
    {"timestamp": 12345678.35, "data": [1.235, 0.364, 0.021]},  
    {"timestamp": 12345678.40, "data": [1.217, 0.378, 0.009]},  
    ...  
],  
"magnetometer": [  
    {"timestamp": 12345678.35, "data": [1.235, 0.364, 0.021]},  
    {"timestamp": 12345678.40, "data": [1.217, 0.378, 0.009]},  
    ...  
],  
"beacons": [  
    {"timestamp": 12345679.05, "data": [[ "A3F3", -51],[ "85B7", -68]]},  
    {"timestamp": 12345680.05, "data": [[ "A3F3", -51],[ "85B7", -68]]},  
    ...  
],  
"gps": [  
    {"timestamp": 12345679.35, "data": [1.44, 10.021]},  
    {"timestamp": 12345680.36, "data": [1.44, 10.009]},  
    ...  
],  
}
```

---

### 3.3 Setting up the Couchbase Server Infrastructure

Couchbase Server Admin Port

Couchbase Sync Gateway config.json

## 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<sup>1</sup>. 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 Objective-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"].

---

<sup>1</sup>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

The application is separated in 5 different layers.

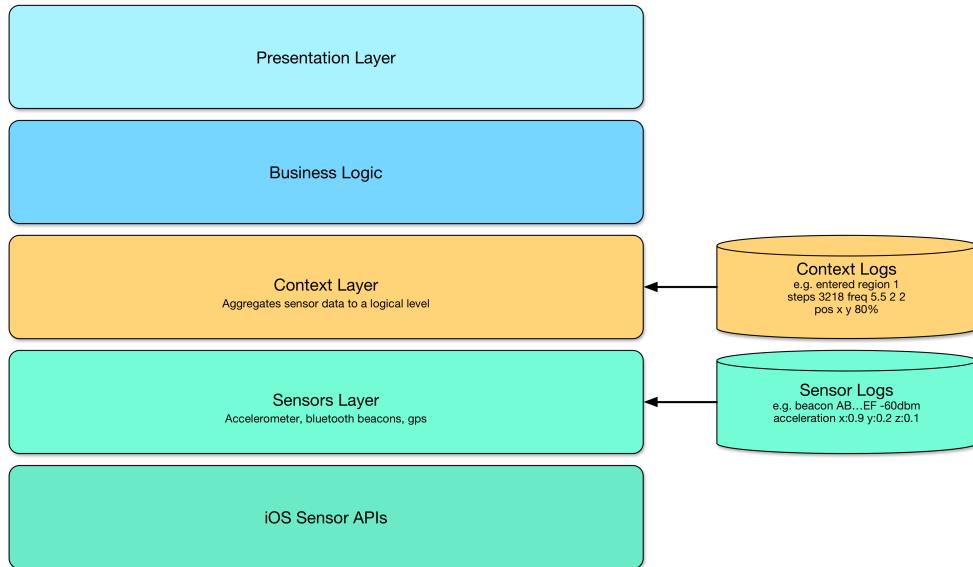


FIGURE 4.1: Layer Model of the Museum Guide Application

A fundamental requirement for the CA 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 the single layers using recorded sensor data and comparing it with its expected result.

To achieve this goal, virtual sensors are introduced at three different levels of abstraction inside the sensor data processing foundation that provides the context information for the application.

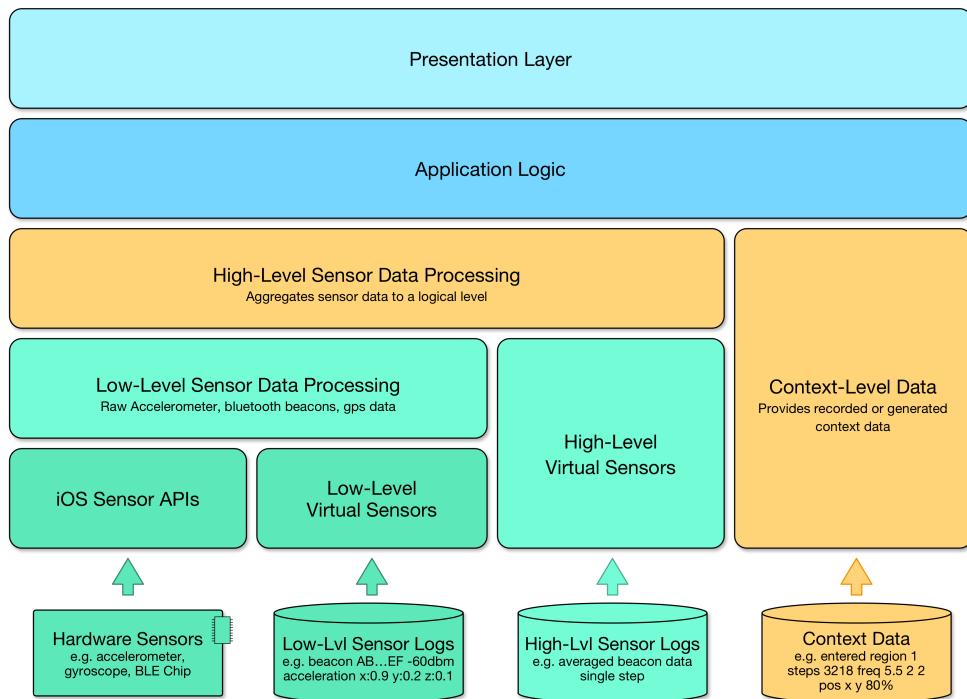


FIGURE 4.2: Layer model with virtual sensors

These three layers are important enough the deserve a name, so this foundation framework is called "Sensorbase" from now on. With a focus on clean and usable design it can evolve to a

The following figure 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.

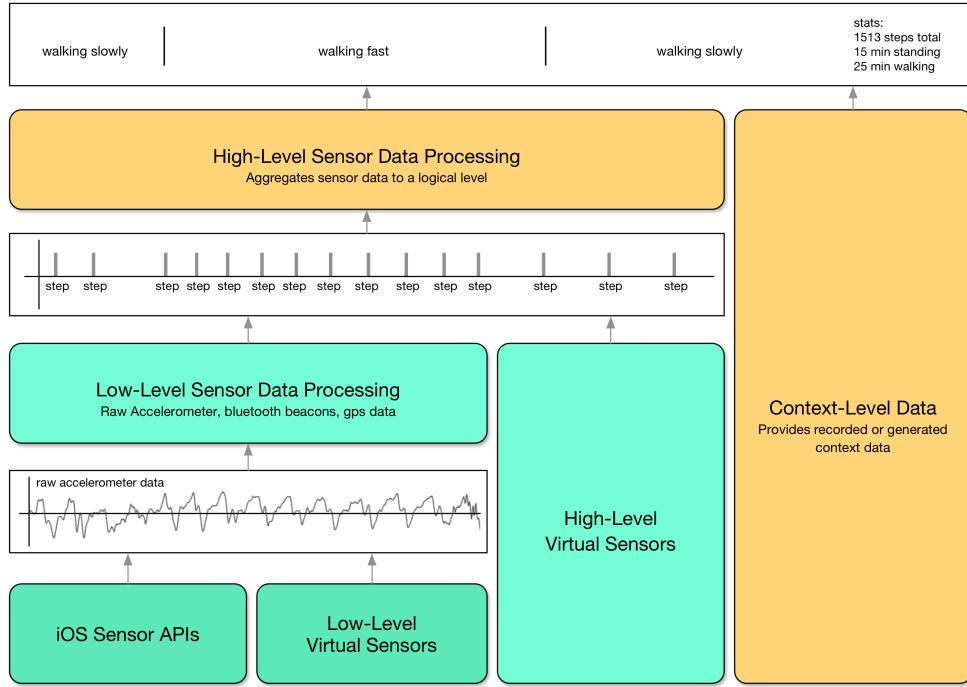


FIGURE 4.3: Data flow between the lower layers of the Sensorbase framework

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.

The frequency of events passed to the next layer decreases from real-time data (100 updates per second), demanding very fast algorithms, to some seconds or minutes.

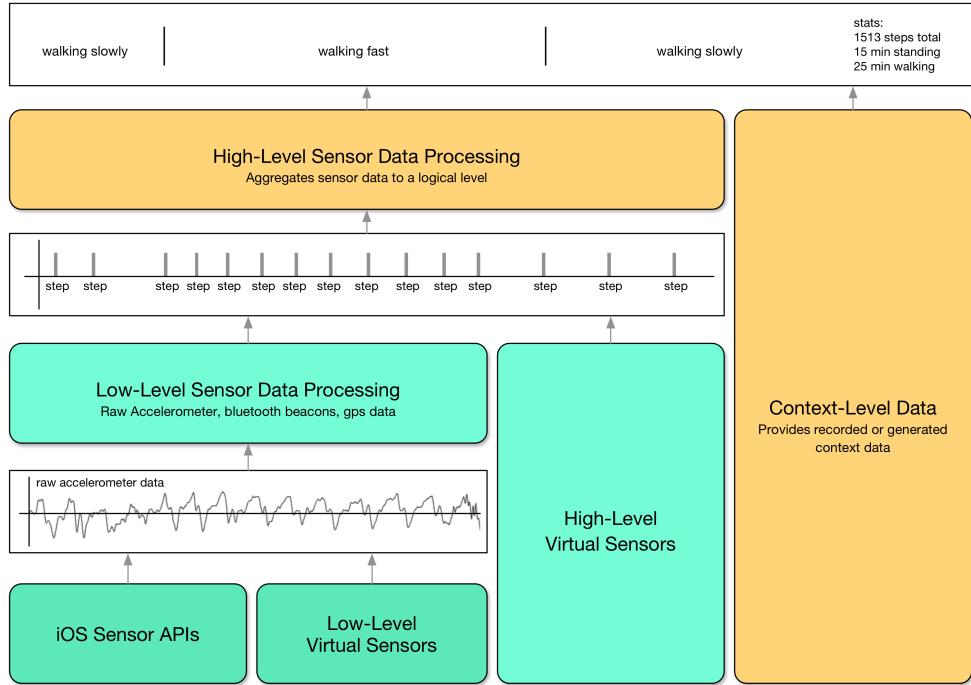


FIGURE 4.4: Sequence Diagram for the Sensor Layers

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 demonstration the guide to customers or prospects a walk through a park can be simulated at the desk on a real device running the guide, being much more intuitive than showing only screen shots.
- With an own class in the Sensor APIs layer acting as an adapter 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.

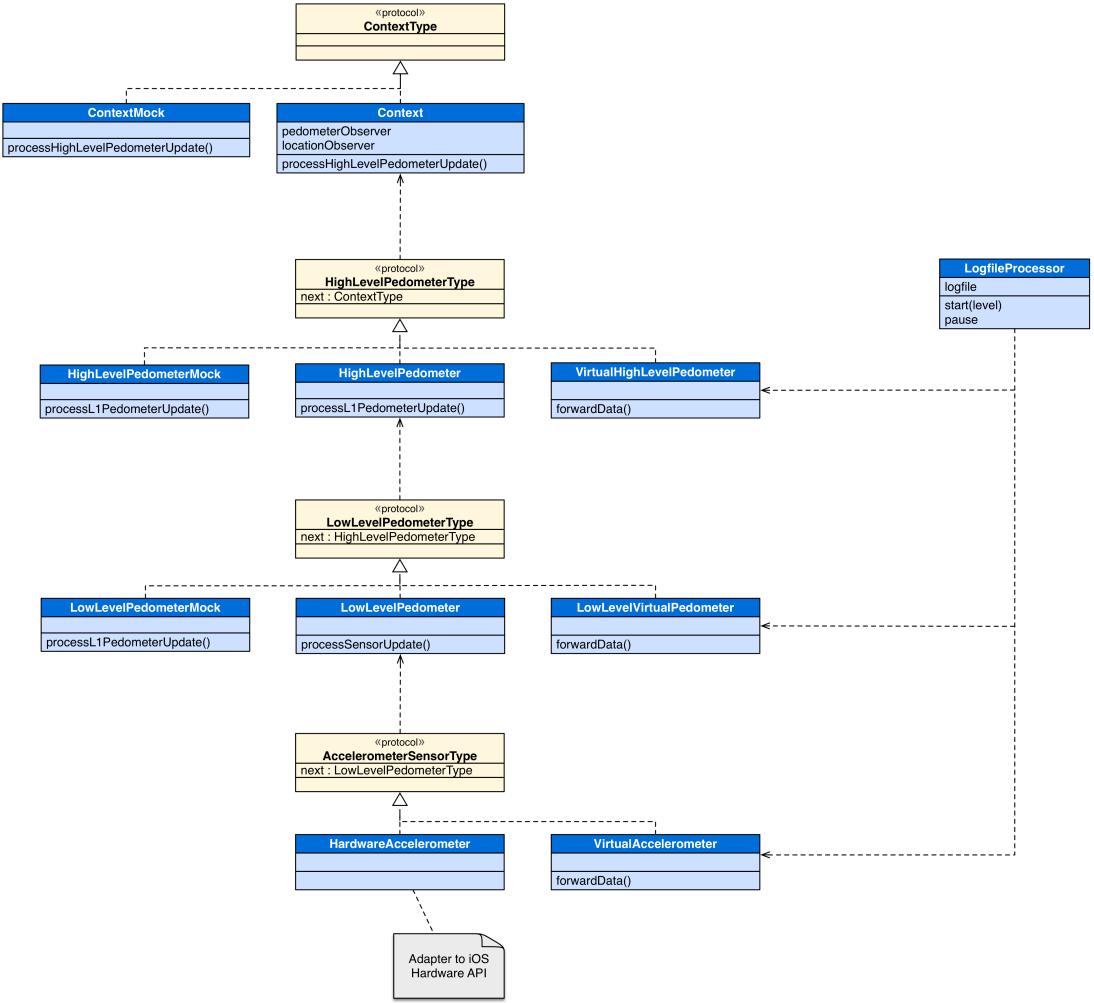


FIGURE 4.5: Simplified Class Diagram for Sensor Data Processing

Mock: useful for Testing the underlying layer, Adapter for iOS Sensor API, The Sensorbase class is a singleton and acts as facade for the whole framework. It provides a single interface to the set of interfaces of the sensorbase subsystem. Knowing which subsystem class is responsible for which call, it delegates client requests to the appropriate subsystem object (cf. [Gamma, 1995]).

#### 4.4.1 Setup of Couchbase Server and Sync Gateway

### 4.5 Sensors Layers

Sensor data is saved to couchbase database. Later access for replay (development), testing and presentations. The backend can access this saved data for later analysis.

Sensorbase debug view. [\[GCD-Reference\]](#)

## 4.6 Application Logic

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

Mockups XCode Storyboard Final App Screenshots

# Chapter 5

## CA Guide Back End

### 5.1 Introduction

The CA Guide 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 so the success of the whole product.

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

- Modeling the outdoor or indoor site by defining areas on a map or floor plan and adding content that later will be presented on the mobile device when entering this area. 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.
- Analytic functions for understanding how the visitors move through the exhibition and thus allowing to optimize it, similar to the way the behavior of website visitors is tracked and used to optimize the web presence<sup>1</sup>.

---

<sup>1</sup>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 its 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 (Asynchronous 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<sup>2</sup> 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.

---

<sup>2</sup>In the cited interview Eich says "The name is a total lie" - it was just a marketing decision. 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 be 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 a 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 its 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 being 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 try 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 is used as basis for developing the CA Guide 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 consisting of an app folder
- **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.

Using the activator command in a shell,

### 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=""
    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<sup>3</sup> 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.

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.

---

<sup>3</sup>serialization/deserialization is also called pickling/unpickling, explaining the framework's name

#### 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, Link evolution step of standard css files, 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

### 5.5.2 The Controller

#### 5.5.2.1 Designing the REST HTTP Interface

#### 5.5.2.2 Controller Actions

### 5.5.3 The Model

#### 5.5.3.1 Designing Classes to Cross-Compile

#### 5.5.3.2 Creating NoSQL Design Documents

#### 5.5.3.3 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<sup>4</sup>.

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.

### 5.5.4 The View

#### 5.5.4.1 Play HTML Templates

#### 5.5.4.2 Mockups

#### 5.5.4.3 The Client code in Scala.js

---

<sup>4</sup>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].

# Chapter 6

## System Review

### 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 its 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 being 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**

### **JSON Site Modelling File**

Write your Appendix content here.

# 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. URL: <http://dblp.uni-trier.de/db/journals/pieee/pieee100.html#AngermannR12>.
- [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.
- [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.
- [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](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).
- [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).
- [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).
- [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](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:

- [Bluetooth] <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 and Scala.js*. URL: <https://github.com/lihaoyi/upickle> (visited on 03/15/2015).
- [Xcode-beta] Apple Inc. *Xcode download page, containing pre-release versions*. URL: <https://developer.apple.com/xcode/downloads/> (visited on 03/10/2015).
- [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).
- [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).
- [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).