HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Context-Aware Museum Guide for Mobile Devices

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: **Context-Aware Museum Guide for Mobile Devices**

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:	Context-Aware Museum Guide for Mobile Devices
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 23.11.1980in Singen*, dass ich

(1) meine Masterarbeit mit dem Titel

Context-Aware Museum Guide for Mobile Devices

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

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

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

Konstanz, 16.04.2015

(Unterschrift)

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

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

Acknowledgements

Lorem ipsum dolor

Contents

Abstract ii

Ehrenwörtliche Erklärung iii

Acknowledgements v

Contents vi

List of Figures ix

Abbreviations x

1 Introduction 1
- 1.1 The goal of this work . . . 1
- 1.2 Motivation . . . 2
- 1.3 The Approach . . . 2

2 Theoretical Background and Fundamentals 4
- 2.1 What is Context-Aware Computing? . . . 4
  - 2.1.1 Definition of Context . . . 4
  - 2.1.2 Categories of Context . . . 5
  - 2.1.3 Categories of Context-Aware Applications . . . 6
  - 2.1.4 Ubiquitous Computing . . . 7
- 2.2 Sensors of Mobile Devices . . . 8
- 2.3 Localization Techniques . . . 9
- 2.4 Geographic Coordinates . . . 9
  - 2.4.1 Introduction . . . 9
  - 2.4.2 Standards . . . 9
  - 2.4.3 WGS84 Coordinates . . . 10
- 2.5 Bluetooth Low Energy iBeacons . . . 10
- 2.6 Swift . . . 10
  - 2.6.1 What is Swift? . . . 10
  - 2.6.2 Comparison Scala and Swift . . . 11
  - 2.6.3 Major Differences to Scala . . . 12
    - 2.6.3.1 Implicit Type Conversions . . . 12
    - 2.6.3.2 Optionals . . . 13

2.6.3.3 Getter and Setter . . . 14
2.6.3.4 Exception Handling . . . 15
2.6.3.5 Named Parameters . . . 16
2.6.3.6 The Legacy of Objective-C . . . 16
2.6.3.7 Summary . . . 17
2.7 iOS and Cocoa Touch . . . 17
2.8 Couchbase NoSQL Database . . . 17
2.8.1 What is a NoSQL Database? . . . 17
2.8.2 Couchbase . . . 18
2.9 Typesafe Reactive Platform . . . 19
2.9.1 Overview . . . 19
2.9.2 Activator Tool . . . 20
2.9.3 Play Web Framework . . . 20

3 System Overview 22
3.1 Introduction . . . 22
3.2 Data Structure . . . 23
3.2.1 Exhibition Modelling . . . 23
3.2.2 Sensor Data . . . 24
3.3 Setting up the Couchbase Infrastructure . . . 25

4 CA Guide Mobile Application 26
4.1 Introduction and Goal . . . 26
4.2 The Target Platform . . . 26
4.3 Setting Up the Development Platform . . . 27
4.4 Architecture . . . 28
4.4.1 Setup of Couchbase Server and Sync Gateway . . . 32
4.5 Sensors Layers . . . 32
4.6 Application Logic . . . 33
4.7 Presentation Layer . . . 33

5 CA Guide Back End 34
5.1 Introduction . . . 34
5.2 The Target Platform . . . 35
5.3 Architectural Decisions . . . 35
5.3.1 The Client and Server Code Gap . . . 35
5.3.2 Solution Attempts . . . 36
5.4 Setting up the Development Platform . . . 37
5.4.1 Scala and Play . . . 37
5.4.2 Adding Reactive Couchbase as Scala Database Driver . . . 38
5.4.3 Adding Scala.js for Web Client Code . . . 38
5.4.4 Adding Bootstrap as UI Framework . . . 38
5.5 Architecture . . . 38
5.5.1 Overview . . . 38
5.5.2 The REST HTTP Interface . . . 38
5.5.3 Performing Database Queries using Scala Futures . . . 38

6 System Review **39**
6.1 Introduction . . . 39
6.2 Modelling the Tech Center . . . 39
6.2.1 Indoor Modelling . . . 39
6.2.2 Outdoor Modelling . . . 39
6.3 Testing the CA Guide Front End . . . 39
6.4 Analytics with the CA Guide Back End . . . 39

7 Conclusion **40**
7.1 Summary . . . 40
7.1.1 The CA Guide . . . 40
7.1.2 Swift and Scala . . . 40
7.2 Future Work . . . 41

A JSON Site Modelling File **42**

Bibliography **43**

List of Figures

2.1 Difference in meters between the geoid and the ellipsoid [NASA-Geoid] . . 9
2.2 Couchbase Lite and Sync Gateways. From [Couchbase-Mobile] 19
2.3 The Play framework stack. From [Hilton, 2014] with kind permission. . . . 21

3.1 CA Guite Architecture Overview 23

4.1 Layer Model of the Museum Guide Application 28
4.2 Layer model with virtual sensors 29
4.3 Data flow between the lower layers of the Sensorbase framework 30
4.4 Sequence Diagram for the Sensor Layers 31
4.5 Simplified Class Diagram for Sensor Data Processing 32

5.1 TypeScript in IntelliJ IDEA 36

Abbreviations

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

Chapter 1

Introduction

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

Various sensors 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 [1] after several conversations with museum directors in the tourist region of the Lake Constance in Summer 2014, who expressed interest in such a solution and encouraged us in pursuing the development of an affordable system based on commodity hardware and a highly configurable software platform.

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

1.3 The Approach

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

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

[1] 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 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 a 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 adapts 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".

2.1.1 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.

2.1.2 Categories of Context

There are many different types of context:

Computing Context

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

User Context

- The user's *identity*, including his age, gender, education, profession, interests. In today's age of social media the user's connections to other people and their identity are important and widely available as well.
- The *location* is the most obvious kind of user context. It can be subdivided in two different sub types of location:
  Logical location, e.g. "at home" or "at work" or slightly different "in a library, restaurant, cinema"
  Absolute location, e.g. "47°40'02.0"N 9°10'19.7"E" as geographic coordinates on earth or 2nd floor room 205 in the F-Building of HTWG Konstanz.
- The *emotional state* indicating if the user is happy, angry, worried etc. At which level is the personal stress level?
- *Fisical state*: Is the user rested or tired?
- The user's *health state*, determined by measuring some vital parameters (e.g. hearth rate, blood pressure and oxygen saturation). Many dedicated low-cost hardware items for this kind of measurements were released in 2014 or are announced for this year.
- Information about other people or mobile devices nearby forms the *co-location* context.
- The current *activity*. For example walking, driving a car, riding a bicycle or jogging. Some of this activities need to be further specificated, e.g. jogging to practice sport or jogging for catching a train.

Time Context

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

Physical Context

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

cf. [Paulo, 2014]

2.1.3 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).

2.1.4 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 beeing 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.".

2.2 Sensors of Mobile Devices

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.

2.3 Localization Techniques

2.4 Geographic Coordinates

2.4.1 Introduction

Since we need to work with gps data and model outdoor sites like parks, it is fundamental to understand how geographic coordinate systems work and which problems we may encounter.

2.4.2 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 WGS 84, 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.

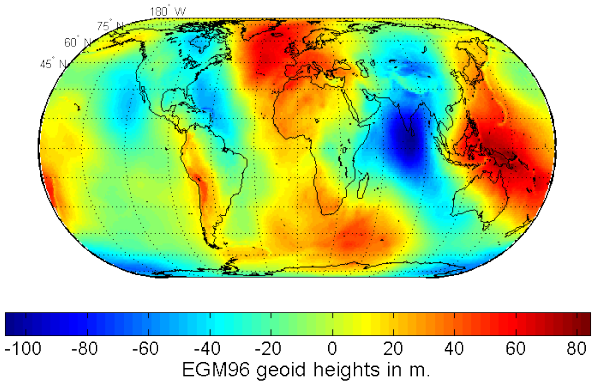

FIGURE 2.1: Difference in meters between the geoid and the ellipsoid [NASA-Geoid]

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

2.4.3 WGS84 Coordinates

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

For this work, the degree and decimal fraction representation was selected.

The maximum resolution using 6 fractional digits depends on the position on Earth. In Konstanz, the one degree of longitude equals

Google and Apple Maps 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 the center of his window on both maps showed a notable discrepancy of several meters. The cause is the different camera angle of the two pictures. Measuring the discrepancy using a prominent ground object like the marked lighting, the two map solutions

2.5 Bluetooth Low Energy iBeacons

Bluetooth Low Energy (BLE) is a standard

2.6 Swift

2.6.1 What is Swift?

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

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

Developers already familiar with the well designed functional programming language Scala will recognize several concepts. It has a concise Syntax avoiding big parts of

boilerplate code and syntactic noise, supports functional programming and is statically typed.

The following section is an overview comparison of Scala and Swift features, created during the familiarization with Swift for this thesis. Language features marked will a * will be discussed separately in section 2.6.3.

2.6.2 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[1]	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 `Option[T]` class	widely used, dedicated Syntax
Switch with pattern matching	yes	yes
String interpolation	yes `"Hello $nameVar"`	yes `"Hello \(nameVar)"`
Keyword for variable definition	var radius = 1	var radius = 1
Keyword for constant definition	let pi = 3.14	val pi = 3.14
Array literal	`Array(1,2,3)`	`[1,2,3]`
Map literal	`Map(1->"a", 2->"b")`	`[1:"a", 2:"b"]` [2]
If condition must be boolean	yes	yes
Tuples	yes, but without named elements	yes

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

[2] Maps are called dictionaries in Swift

Language Feature	**Scala**	**Swift**
Ranges	`for i <- 0 to 4` `0 until 4`	`for i in 0...4` `0..<4`
Constructor	def this()	init()
Extended getter/setter concept *	yes, no observers	yes, very flexible concept including observers (willSet() didSet() events)
Interfaces	trait	protocol
Extension of existing types	yes	yes
Struct	no	yes
Enum	via extending the Enumeration class	yes, dedicated keyword
"Any" Type	Any, AnyVal, AnyRef	Any (instance of any type, even function types) AnyObject (instance of any class type)

Qeury an instance of a type by a key in brackets, like arrays or maps	obj(index) calls obj.apply method
	obj(index) = newValue calls obj.update(0, newValue)
		`subscript(i:T) -> T2`
		`{`
		`  get {...}`
		`  set(newValue) {...}`
		`}`

Memory Management	JVM Garbage Collection	Automatic Reference Counting
Nested Functions	yes	yes
Generics	yes	yes

2.6.3 Major Differences to Scala

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

2.6.3.1 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. [3] 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.".

2.6.3.2 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
```

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

```
println(name!) // runtime error: unexpectedly found nil while
    unwrapping an Optional value
```

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

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

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

it is possible to write

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

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

Another concept in the context of optionals are failable initializers (known as constructors in other languages). When explicitly defining an initializer as failable appending a question mark (init?), it's return type is optional variant of the type it should initialize. That can be useful for handling invalid parameter values or other initialization problems.

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

2.6.3.3 Getter and Setter

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

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

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

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

This restricts write access to the current source file in Swift[4].

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.

2.6.3.4 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 {
```

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

```
    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.

2.6.3.5 Named Parameters

2.6.3.6 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[5], that makes it available in Objective-C code.

An example: Objective-C APIs sometimes use selectors, which are strings identifying a certain callback method by its method name and its named parameters that will be called on the passed object. When an API is used that expects a selector as parameter, the @objc attribute is needed on the passed object's class, so that that the library can

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

find and call the method, otherwise at runtime a fatal error occurs stating no method with that name can be found. This is a clear limitation of that automatic translation of old APIs. In my opinion, it would be much safer to rewrite this APIs in Swift and replace all selectors with function parameters or closures. This way, a misspelled method name is recognized at compile time.

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

2.6.3.7 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.[6]

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.

2.7 iOS and Cocoa Touch

iOS Cocoa Touch Framework Developing with XCode

2.8 Couchbase NoSQL Database

2.8.1 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.

[6] 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.

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

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

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.[7]

2.8.2 Couchbase

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

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

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

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

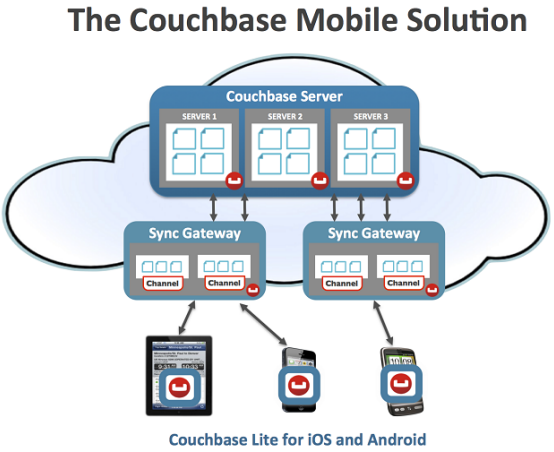

FIGURE 2.2: Couchbase Lite and Sync Gateways. From [Couchbase-Mobile]

While traditional relational database management systems (RDBMS) are typically hard to scale out[8], 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.

2.9 Typesafe Reactive Platform

2.9.1 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

[8] 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.

2.9.2 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.

2.9.3 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.

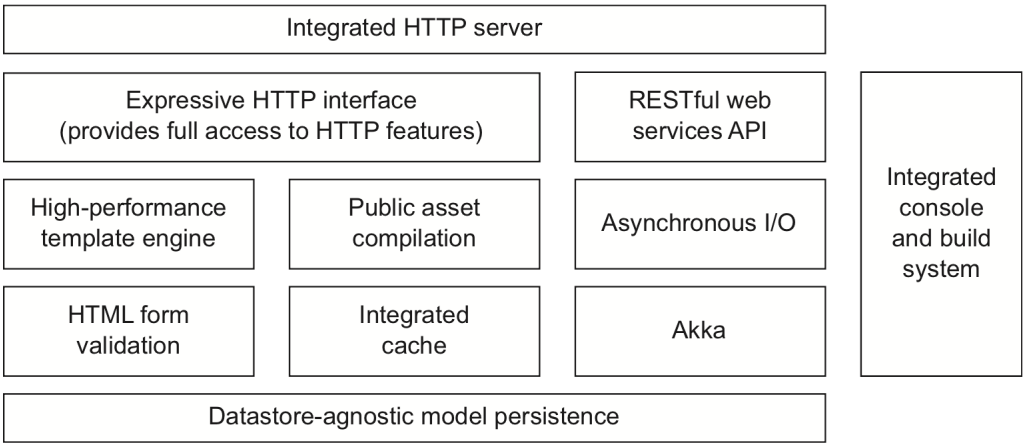

FIGURE 2.3: The Play framework stack. From [Hilton, 2014] with kind permission.

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 it's Platform instead of Lift or similar frameworks because it is completely written in Scala, uses Akka for asynchronous processing, providing a good concurrency handling, has the better type-safety even in html templates, and because it was the one with the greater momentum 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 font and back end. All data is exchanged over the database, enabling asynchronous communication.

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

[1] After actually having reached the limits of scalabity 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.

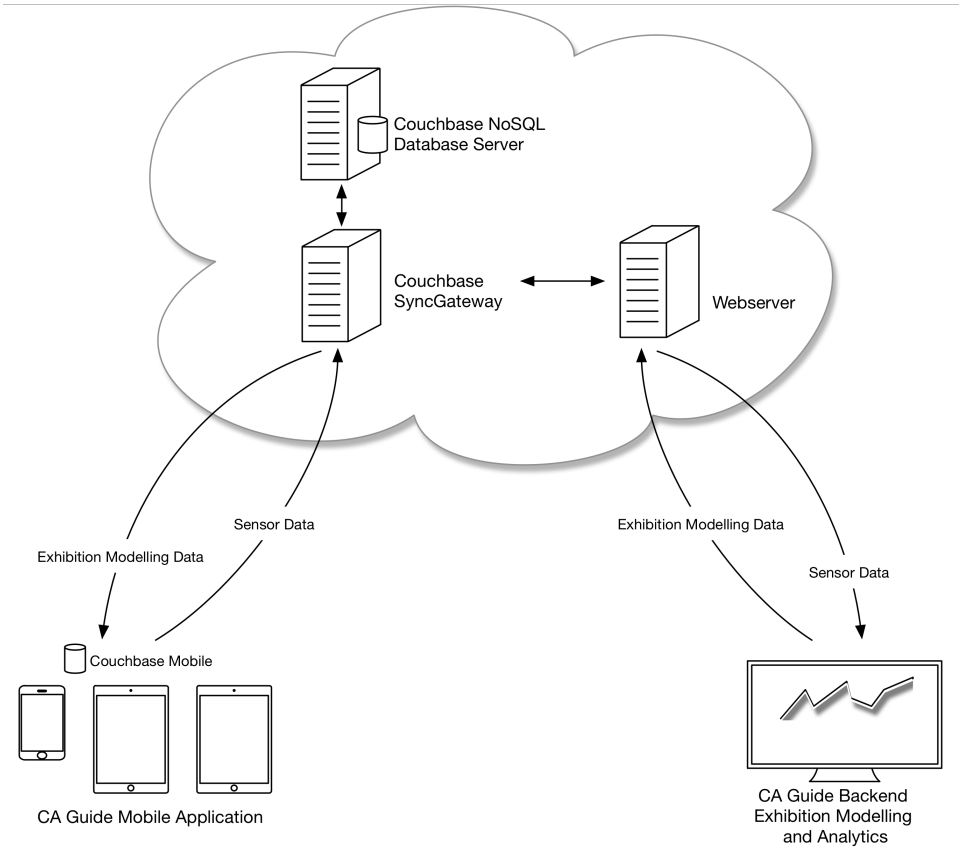

FIGURE 3.1: CA Guite Architecture Overview

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 entyty 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 millisecons. 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 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[1]. It can be downloaded at [Xcode-beta].

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

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

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

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

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

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

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

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

4.4 Architecture

For a clean The application is separated in 5 different layers.

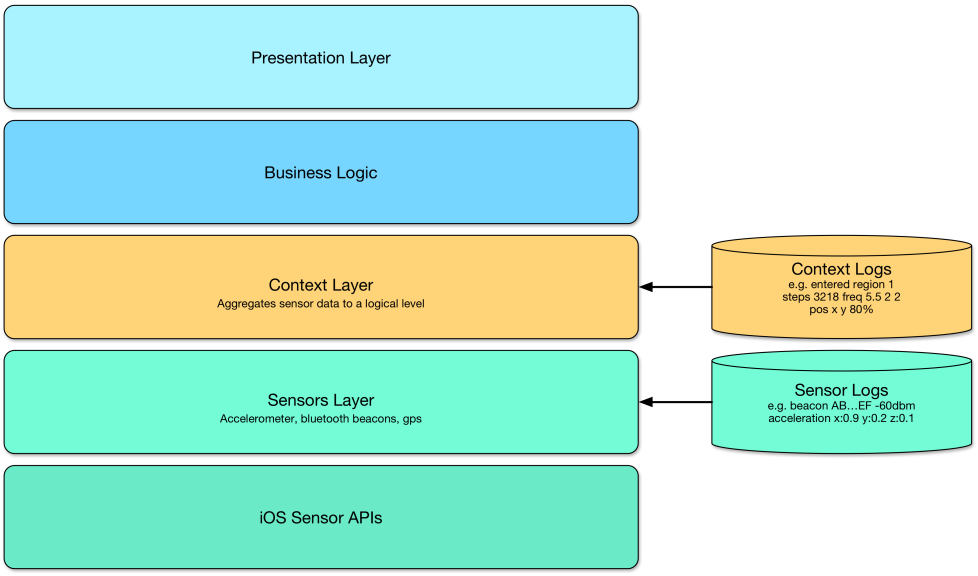

FIGURE 4.1: Layer Model of the Museum Guide Application

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 it's 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.

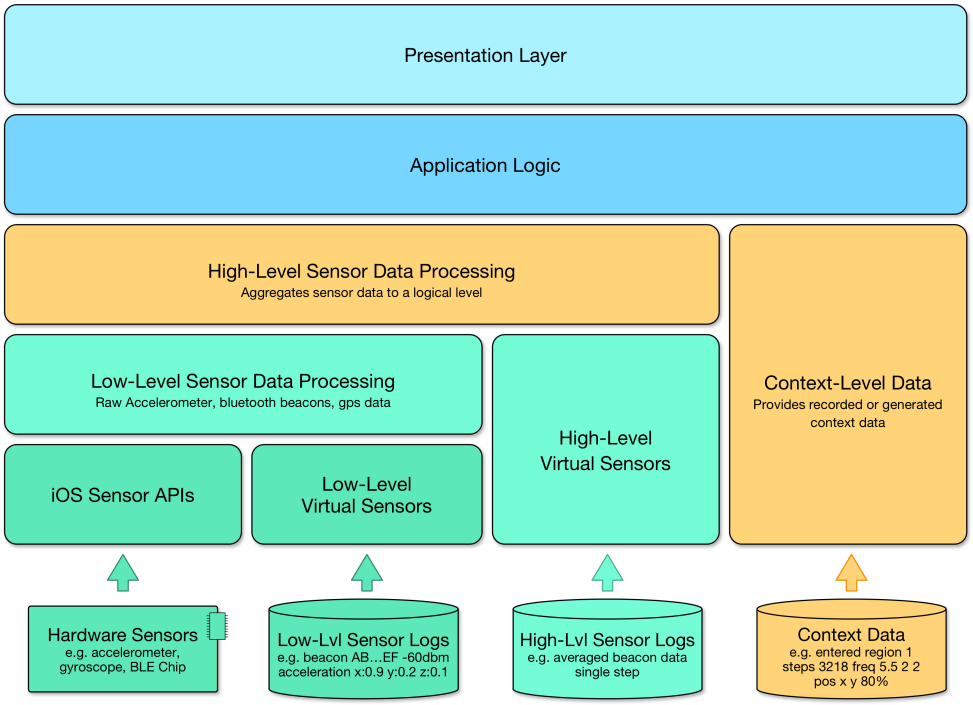

FIGURE 4.2: Layer model with virtual sensors

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.

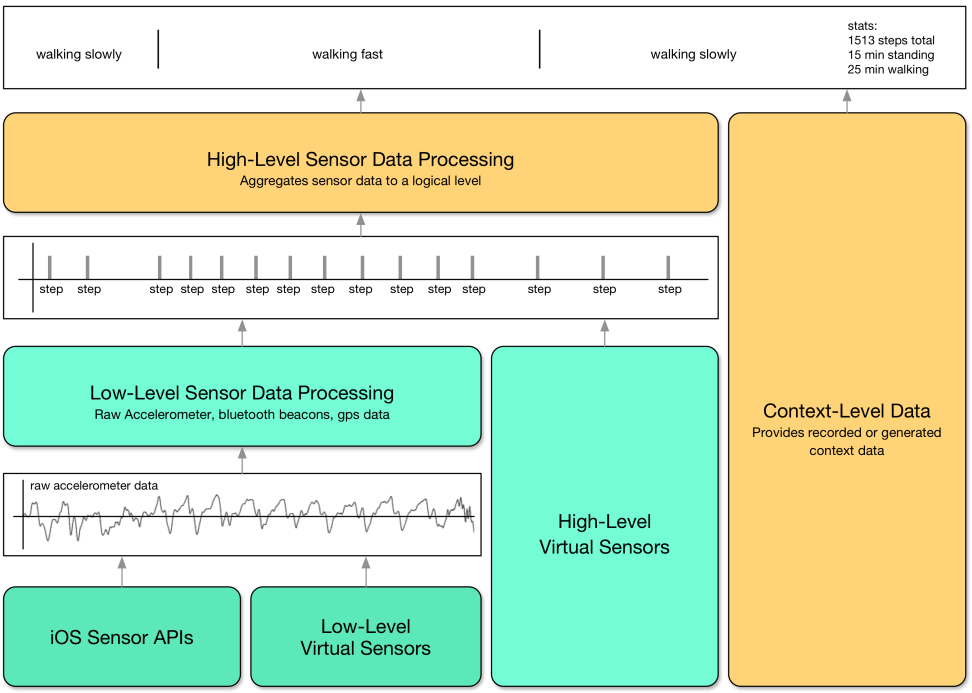

FIGURE 4.3: Data flow between the lower layers of the Sensorbase framework

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.

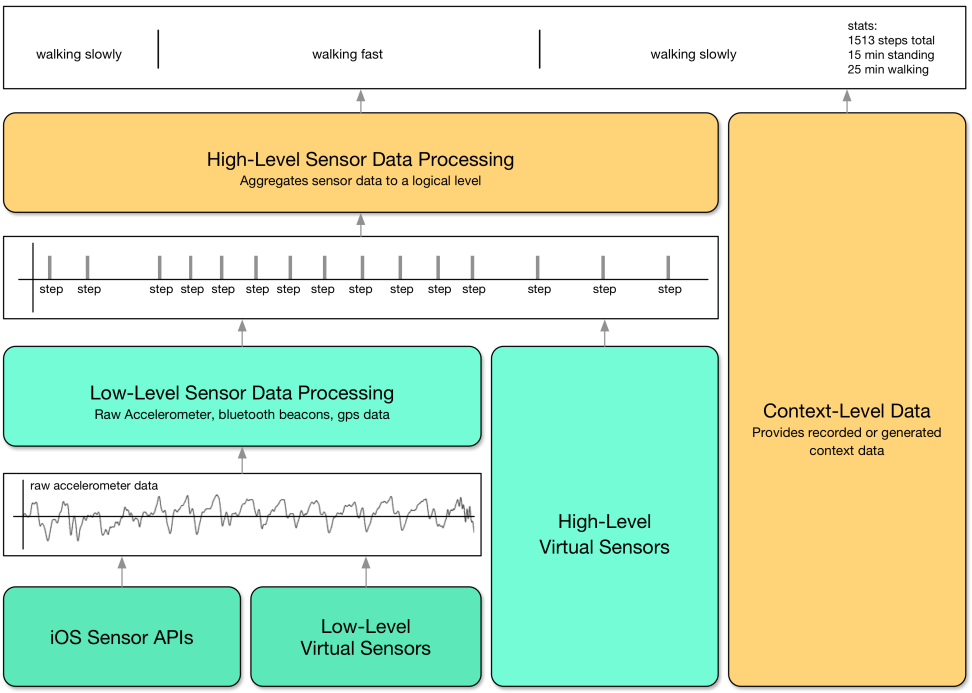

FIGURE 4.4: Sequence Diagram for the Sensor Layers

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, beeing 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.

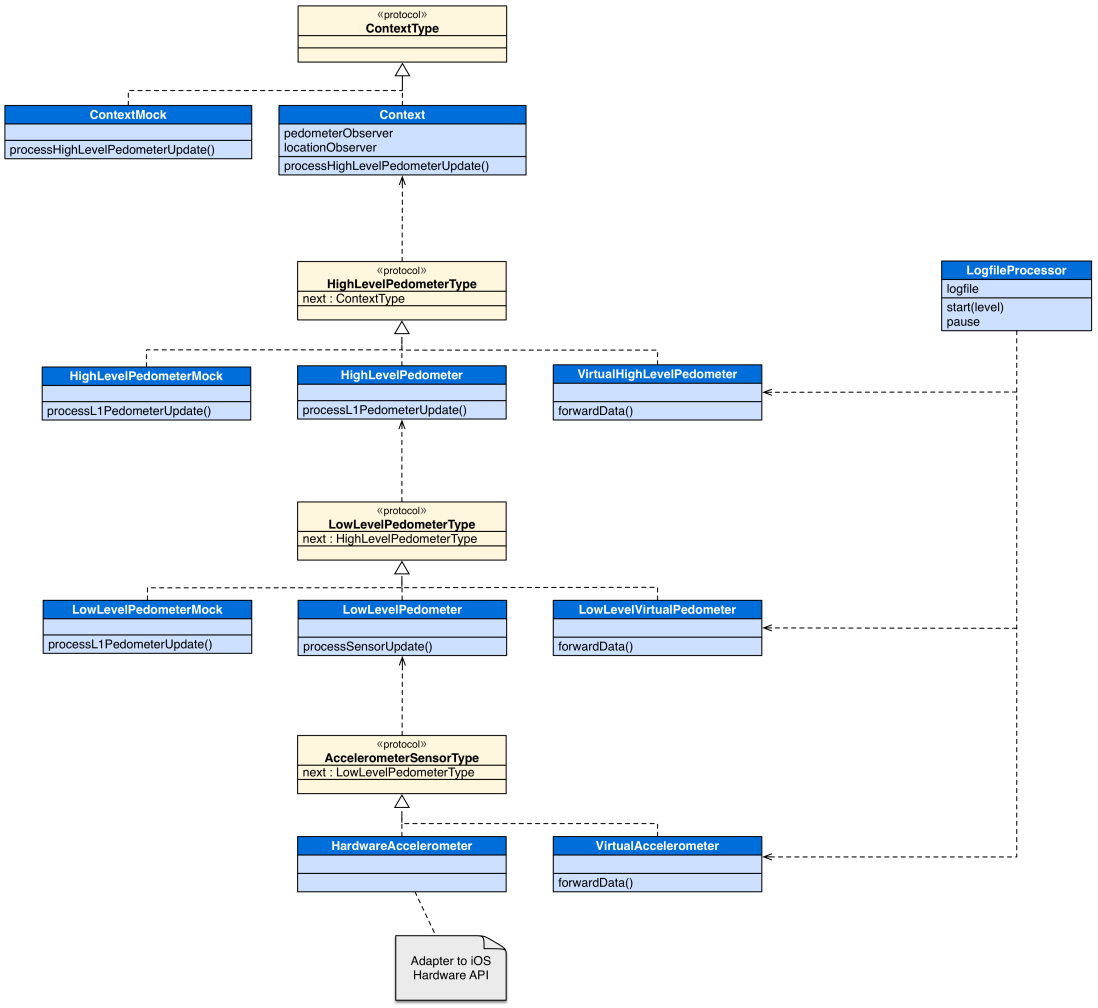

FIGURE 4.5: Simplified Class Diagram for Sensor Data Processing

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 trough 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[1].

[1] Of course, the privacy of the visitor has to be respected at any time. The data must only be collected anonymously and with the permission of the visitor.

5.2 The Target Platform

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

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

5.3 Architectural Decisions

5.3.1 The Client and Server Code Gap

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

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

This technique is commonly known as AJAX (Asynchronuous JavaScript and XML) and inside this acronym's phrase one can already spot the main problem. While AJAX gained popularity during the last years, it leveraged the massive use of JavaScript, a dynamically typed scripting language originally not designed for big projects. In fact, JavaScript 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.

So the two main problems to be solved are:

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

The next subsection focuses on different solution attempts.

5.3.2 Solution Attempts

Improved Javascript

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

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

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

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

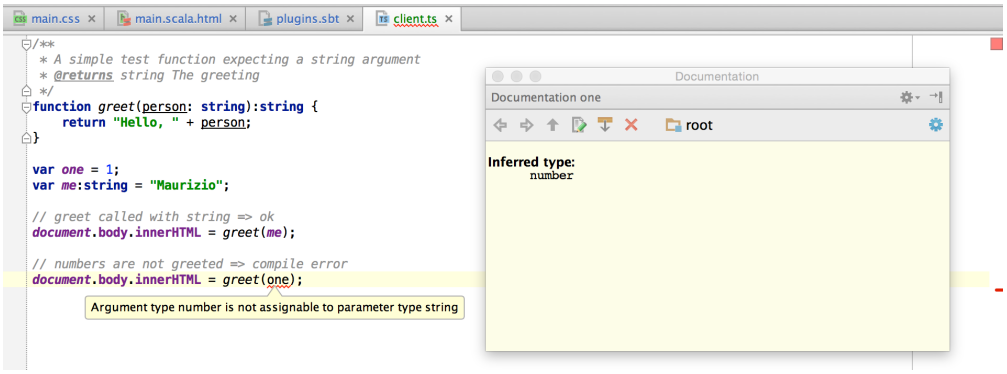

FIGURE 5.1: TypeScript in IntelliJ IDEA

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

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].

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 Guide back end client-side code.

5.4 Setting up the Development Platform

5.4.1 Scala and Play

Scala is used as programming language for the back end, with Play web framework as a basis for developing a web application in Scala.

5.4.2 Adding Reactive Couchbase as Scala Database Driver

5.4.3 Adding Scala.js for Web Client Code

5.4.4 Adding Bootstrap as UI Framework

5.5 Architecture

5.5.1 Overview

5.5.2 The REST HTTP Interface

5.5.3 Performing Database Queries using Scala Futures

The interface of the Reactive Couchbase Driver makes extensive use of Futures, a language feature introduced in Scala 2.10[2].

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.

[2]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 it's standard library. Currently, Scala has the better libraries, being better composed allowing to find familiar concepts and elements learned before and reused as part of other concepts.

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

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

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

7.2 Future Work

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

Appendix A

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.

[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.

[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.

[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).

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

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

[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 (visited on 02/23/2015).

[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).

[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).

[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).

[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).