HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Context-Aware Museum Guide for Mobile Devices

Maurizio Tidei

Konstanz, 16.04.2015

MASTERARBEIT

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)

Acknowledgements

Lorem ipsum dolor

Contents

List of Figures

Abbreviations

INS	**I**nertial **N**avigation **S**ystem
MEMS	**M**icro**e**lectro**m**echanical **S**ystem
DSL	**D**omain**S**pecific**L**anguages

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.

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. 1] *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.

The system consists of two main parts:

The first one is a kind of a content management system that allows to define all relevant textual and audio information describing the exhibitions areas and single exhibits using a web interface.

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.

1.2 Motivation

This work is a basis 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 P. Lorenz [1] after several conversations with museum directors in the area of the Lake of Constance in Summer 2014, which 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.

1.3 The Approach

Since the location context is key for the system that has to be designed, first, different methods for providing the location context are evaluated (Chapter 2.1).

Chapter 2.2 describes the chosen programming languages and tools.

Out of the possible context providing methods, the best fitting ones are tested in several technology prototypes (Chapter 3).

The system design will be discussed in Chapter 4.

The first results using the application in a real-life scenario in the "Schloss Arenenberg" Park and World War I exposition are presented in Chapter 5.

Chapter 6 handles the conclusion and future work.

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

Chapter 2

Theoretical Background

2.1 What is Context-Aware Computing?

Since context-aware computing emerged as a subcategory of ubiquitous computing, the latter one is described fist. Ubiquitous computing means ...

The term "context-aware computing" was first used by B. Schilit et. al. 1994 in the paper "Context-Aware Computing Applications" [2]. 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".

They 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).

There are many different types of Context:

- Location

The Location is the most obvious kind of context in modern applications. It can be subdivided in two different sub types of location:

Kind of Location

For example "Home" or "At Work" or slightly different "In a Library" or "Inside a restaurant"

Absolute Position

For example "47.667230, 9.172134" as geographic coordinates on earth or 2nd floor room 205 in the F-Building of HTWG Konstanz.

- Time

  Season

  Daytime

  Date

- Activity

  E.g. Walking, Driving a Car, Jogging...

- Environment

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

A context aware application collects context information without explicitly asking the user for input. The goal should always be improving the user experience by providing the right information or taking the right actions at the right time without distracting or annoying the user. Basic tasks like automatically dimming mobile phone displays are well-known automatically performed actions. Others like remembering the parking location without user interaction are new .

By enhancing the sensing for further

- Emotional State Is the user happy, angry, worried? At which level is the personal stress level?
- Fitness State Is the user rested or tired?
- Health State Are all vital parameters of the user inside reasonable limits?

2.2 Sensors of Mobile Devices

2.2.1 Accelerometer

2.2.2 Gyroscope

Pedometer ref.

2.2.3 Magnetometer

2.2.4 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.

2.2.5 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.

2.2.6 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 Bluetooth Low Energy iBeacons

Bluetooth Low Energy (BLE) is a standard

2.5 Swift

2.5.1 What is Swift?

Swift is a modern programming language released by Apple in 2014. In [3] 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 2.5.3.

2.5.2 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

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

Language Feature	**Scala**	**Swift**
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
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)

[2]Maps are called dictionaries in Swift

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

2.5.3 Major Differences to Scala

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

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 [5, 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.".

Optionals

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

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

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

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

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

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

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

[3] A good example for using implicit conversion to build a DSL query language can be found at [4]

```
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.

Getter and Setter

Swift has a very well designed property access system, with a concise syntax due to built-in language support.

It addresses the two main problems, encapsulation and dependency management, classical getter and setter methods solve, without the overhead of writing separate access methods for object 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, or to the class if it is the only one in the source file.

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

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

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

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

Exception Handling

2.5.3.1 test

2.6 iOS and Cocoa Touch

iOS Cocoa Touch Framework Developing with XCode

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

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

[2] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

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

[4] Jan Macacek. Type-safe dsl.

[5] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide.* Artima Incorporation, USA, 2nd edition, 2010. ISBN 0981531601, 9780981531601.