

UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
ELECTRONICS AND COMPUTERS FIELD



DESIGN AND DEVELOPMENT OF A MOBILE GUIDE
FOR MUSEUM BASED ON THE ANDROID PLATFORM

DIPLOMA THESIS OF

FRANCISCO DE BORJA SÁNCHEZ SANCHO

STUDENT OF THE UNIVERSIDAD DE VALLADOLID

SUPERVISORS: PROF. N. AVOURIS AND PROF. V. PALIOURAS

DIPLOMA THESIS NUMBER:
MARCH 2011

ABSTRACT

This paper presents the improvements of an electronic museum guide application, which brings the hand-held devices into museum to be visitor's own expert guide.

Therefore, the main purpose is to improve that application performance, features and usability for providing the visitors of the museum with a useful, easy and friendly application. This will allow to construct a mobile learning environment with *wireless* technology and modern portable devices. People who are in this environment would obtain the detailed information, and even the history of the exhibit displayed in text, streaming voice or even video clips.

Due to the fact that this project is closely related to a previous project, this thesis focuses into three main clearly different parts: an analysis about which aspects of that work could be improved; a new design of the *User Interface (UI)* developed with *Mockups* applications; and finally the implementation of some of the new design and other new methods (a *tour guide* system) based on the mobile *Android* platform.

The implementations of the demos are focused on the *Java Android* language. The choice of this platform reflects the general direction of the mobile industry, where an increasing amount of application development is done in high-level languages such as *Java*.

Finally, the paper present some possible extensions of the implementations as future work. These extensions specifically take advantage of the hardware and abilities of modern mobile devices, including orientation sensors, cameras and internal positioning *GPS*.

Key words: Museum guide application, *wireless* technology, mobile devices, *mockup*, *QR* Codes, *Android*.

Π Ε Ρ Ι Λ Η Ψ Η

Αυτή η εργασία παρουσιάζει τις βελτιώσεις στην εφαρμογή ενός ηλεκτρονικού οδηγού μουσείου, με σκοπό την αναβάθμιση του προσωπικού εξειδικευμένου οδηγού χειρός των επισκεπτών μουσείων.

Σ' αυτή την κατεύθυνση, ο κύριος στόχος είναι η βελτίωση των λειτουργιών, των χαρακτηριστικών και της χρηστικότητας που θα προσφέρουν στον επισκέπτη του μουσείου ένα χρήσιμο, εύχρηστο, και φιλικό εργαλείο. Αυτές οι βελτιώσεις θα επιτρέψουν την κατασκευή ενός κινητικού εκπαιδευτικού περιβάλλοντος που θα κάνει χρήση των *ασύρματων* τεχνολογιών και των σύγχρονων κινητών συσκευών. Οι χρήστες αυτού του περιβάλλοντος θα έχουν πρόσβαση σε λεπτομερείς πληροφορίες που αφορούν στην ιστορία των εκάστοτε εκθεμάτων, είτε με τη μορφή κειμένου, είτε μέσω μια καθοδηγητικής φωνής ή ακόμα και μέσω αποσπασμάτων βίντεο.

Λόγω του γεγονότος ότι η παρούσα εργασία είναι στενά συνδεδεμένη με ένα προηγούμενο project, εδώ εστιάζει σε τρία εντελώς διαφορετικά κύρια θέματα: σε μια ανάλυση των σημείων που θα μπορούσαν να βελτιωθούν, σε έναν νέο σχεδιασμό του *User Interface (UI)*, ο οποίος έχει προκύψει μέσω εφαρμογών *Mockups* και τέλος στην συμπλήρωση του νέου σχεδιασμού και άλλων μεθόδων (ένα *σύστημα ξενάγησης*) που στηρίζονται στην κινητή πλατφόρμα *Android*.

Οι συμπληρώσεις των δειγμάτων είναι εστιασμένες στη γλώσσα *Java Android*. Η επιλογή αυτής της πλατφόρμας αντικατοπτρίζει τη γενική κατεύθυνση της βιομηχανίας κινητών συσκευών, η οποία χρησιμοποιεί για όλο και μεγαλύτερο μέρος της εξέλιξης των εφαρμογών της, γλώσσες υψηλού επιπέδου, όπως η *Java*.

Τέλος, η εργασία παρουσιάζει κάποιες προεκτάσεις αυτών των συμπληρώσεων, οι οποίες θα μπορούσαν να έχουν χρήση στο μέλλον. Αυτές οι συμπληρώσεις συγκεκριμένα εκμεταλλεύονται τα υλικά και τις δυνατότητες των σύγχρονων κινητών συσκευών, που περιλαμβάνουν αισθητήρες προσανατολισμού κάμερες και *GPS*.

Λέξεις κλειδιά: εφαρμογή οδηγού μουσείου, *ασύρματες* τεχνολογίες, κινητές συσκευές, *Mockup*, *QR Codes*, *Android*.

ACKNOWLEDGMENTS

I would like to begin this report by thanking all the people who oneway or another contributed to the success of this work.

I am very much indebted to my supervisor *Christos Sintoris*, for his support, motivation, encouragement and guide during these six months. Furthermore, I am thankful to *Chris* for being not only my supervisor but also somebody close to me, for guiding me when I was lost and stressed and for giving me advices and freedom to express my own ideas and thoughts. I am also grateful for his patience with all the problems which arise during all this time. It was under his supervision I became interested in *Java*, *Android* and mobile devices development.

I would like to thank the other members of the *HCI Group* for allowing me to work in their laboratory: *Ioannis Ioannidis*, who was interested in the work I was developing, supported me with his advices and for providing me with everything I needed; *Irene Chounta*, who was everyday in a pleasant mood; *Filio Vogiantzi*, most of the mornings she opened the lab for me and she helped me when I had to make administrative papers. I want also to thank Professor *Nikolaos Avouris* (my *Erasmus* coordinator in Patra), *Dimitrios Raptis* and *Eleftherios Papachristos* for their good reception when I came to Patra.

I will like to thank all the extraordinary greek people I met here in Patra. Thanks as well to all my greek friends from the Univeristy Sport Centre, where I usually played basketball, my way of escaping everyday life...

Thanks to my several *Erasmus* friends, especially to *L*, an awesome and deep influence over me: "*We'r going through changes!*"

Finally, I want dedicate this report to my parents (and the rest of my relatives of course), *Francisco* and *Cristina*, for educated me, encouraging me, giving me everything, supporting me ... Dad and mum: *I did it!*

CERTIFICATION

It is certified that Diploma Thesis with the title:

DESIGN AND DEVELOPMENT OF A MOBILE GUIDE FOR MUSEUM BASED ON THE ANDROID PLATFORM

Of the student of the University of Valladolid

Sánchez Sancho
(Surnames)

Francisco de Borja
(Name)

was presented in public at the Department of Electrical and Computer Engineering of the University of Patras, Greece on March 31, 2011.

The supervisor

The head of the Electronics and Computers division

CONTENTS

INTRODUCTION	7
1.1 MOTIVATION	7
1.2 BACKGROUND	7
1.3 AIM OF THE THESIS	9
1.4 RELATED WORK	10
1.5 OVERVIEW	10
THEORY AND ANALYSIS	12
2.1 INTRODUCTION	12
2.2 EVALUATION	12
2.2.1 METHOD OF EVALUATION	12
2.2.2 TESTING DEVICE	13
2.2.3 PROCEDURE OF THE EXPERIMENT	14
2.2.4 ANALYSIS OF THE OBTAINED RESULTS	15
DESIGN	22
3.1 INTRODUCTION	22
3.2 STATE OF THE ART.....	22
3.2.1 MUSEUM OF ACROPOLIS AUDIO GUIDE	24
3.2.2 AMNH (AMERICAN MUSEUM OF NATURAL HISTORY) EXPLORER.....	25
3.2.2 MoMA (MUSEUM OF MODERN ART).....	26
3.3 MOCKING-UP	27
3.3.1 MOCKUPS	27
3.3.2 MOCKUP SOFTWARE APPLICATIONS	28
3.3.3 BALSAMIQ	28
3.3.3 UNDERSTANDING BALSAMIQ.....	29
3.4 DESIGNING THE NEW APPLICATION GUI	30
3.4.1 TITLE SCREEN.....	30
3.4.2 CHECKOUT (BARCODE SCANNER).....	31
3.4.3 LANGUAGE SELECTION.....	32
3.4.4 MAIN MENU SCREEN	33
3.4.5 DIRECTORY – IMAGE MAP.....	34
3.4.6 DIRECTORY – INSERT NUMBER (NUMERICAL CODES)	35
3.4.7 DIRECTORY – CAMERA (QR CODES)	36
3.4.8 DIRECTORY – HELP.....	37
3.4.9 MAP FEATURE	38
3.4.10 TOUR FEATURE.....	39
3.4.11 EXHIBIT SCREEN.....	40
3.5 FINAL COMMENT ABOUT THE DESIGN.....	40
DEVELOPMENT & IMPLEMENTATION	41
4.1 INTRODUCTION	41
4.2 FEATURES IMPLEMENTED	41
4.3 EXHIBIT SCREEN.....	42
4.3.1 GENERAL REVIEW	42
4.3.2 FULL SCREEN.....	45
4.3.3 COMPOUND CONTROL	45
4.3.4 AUDIO	47
4.3.5 WORKING WITH XML ON ANDROID	48
4.3.5.1 XML	48
4.3.5.2 XML PARSERS	49

4.3.5.3 USING SAX	51
4.4 TOUR FUNCTIONALITY	55
4.4.1 GENERAL REVIEW	55
4.4.2 CREATING THE TOUR LIST (AND THE EXHIBIT LIST)	57
EVALUATION, RESULTS AND CONCLUSIONS	63
5.1 EVALUATION OF THE APPLICATION	63
5.1.1 SCENARIO: A VIRTUAL MUSEUM.....	63
5.1.2 ACTIVITIES	64
5.1.3 USABILITY TESTINGS.....	64
5.1.4 HOW MANY USERS TO TEST?	65
5.1.5 SURVEYS	66
5.1.6 METHOD	66
5.2 APPLICATION RESULTS	67
5.2.1 DEMOGRAPHICS SURVEY	67
5.2.2 APPLICATION SURVEY	68
DEMOGRAPHICS SURVEY ANSWERS	72
APPLICATION QUESTIONNAIRE ANSWERS	73
5.3 CONCLUSIONS	75
REFERENCES.....	76
GLOSSARY	77
APPENDIX A	79
ANDROID	79
A.1 INTRODUCTION	79
A.2 THE LINUX KERNEL	79
A.3 THE SYSTEM LIBRARIES	80
A.4 ANDROID RUNTIME	80
A.4.1 CORE LIBRARIES	80
A.4.2 DALVIK VIRTUAL MACHINE	80
A.4.3 THE CODE VERIFIER	81
A.5 APPLICATION FRAMEWORK.....	81
A.6 THE STRUCTURE OF AN APPLICATION	81
A.6.1 ACTIVITY	82
A.6.2 SERVICE.....	82
A.6.3 BROADCAST RECEIVER	82
A.6.4 CONTENT PROVIDERS.....	82
A.7 THE LIFE CYCLE OF AN APPLICATION	83
A.7.1 INTENTS	83
A.7.2 KNOWING THE STATE OF AN APPLICATION	83
APPENDIX B	84
SOFTWARE DEVELOPMENT PROCESS	84
B.1 INTRODUCTION	84
B.2 PROCESS ACTIVITIES/STEPS	84
B.2.1 REQUIREMENTS AND ANALYSIS	84
B.2.2 DESIGN	85
B.2.3 IMPLEMENTATION	85
B.2.4 TESTING	85
B.2.5 OPERATION AND MAINTENANCE.....	85
B.3 SOFTWARE DEVELOPMENT MODELS	86
B.3.1 WATERFALL MODEL	86
B.3.2 SPIRAL MODEL.....	87

B.3.3 ITERATIVE AND INCREMENTAL DEVELOPMENT.....	87
B.3.4 AGILE DEVELOPMENT.....	87
APPENDIX C.....	88
SETTING UP THE DEVELOPMENT ENVIRONMENT.....	88
C.1 INTRODUCTION.....	88
C.2 CREATING AN ANDROID DEVELOPMENT ENVIRONMENT.....	89
C.2.1 INSTALL JDK.....	89
C.2.2 INSTALL ECLIPSE.....	90
C.2.3 CHECK FOR REQUIRED PLUG-INS.....	91
C.2.4 INSTALL ANDROID SDK.....	91
C.2.5 UPDATE THE ENVIRONMENT VARIABLES.....	91
C.2.6 INSTALL THE ANDROID PLUG-IN (ADT).....	92
C.3 THE FIRST PROGRAM: “HELLO, ANDROID”.....	93
C.3.1 STARTING A NEW ANDROID APPLICATION: HELLOWORLD.....	94
C.3.2 WRITING HELLOWORLD.....	97
C.3.3 RUNNING HELLOWORLD.....	99
APPENDIX D.....	101
INTEGRATE ZXING INTO ANDROID APPLICATIONS.....	101
D.1 INTRODUCTION.....	101
D.2 GETTING STARTED.....	101
D.2.1 Download ZXing.....	101
D.2.2 Download Apache Ant.....	101
D.2.3 Build.....	102
D.2.4 Integrate ZXing source lib into the Android Code project through Eclipse...	102
D.3 ALTERNATIVE METHOD FOR ANDROID.....	103
QUESTIONNAIRE FOR THE ANALYSIS.....	104
DEMOGRAPHICS SURVEY:.....	107
STEPS TO FOLLOW: A GUIDE TO THE MUSEUM.....	110
QUESTIONNAIRE FOR THE RESULTS:.....	118

LIST OF FIGURES

Figure 1 Abstract System Architecture	9
Figure 2 Overview flowchart.....	10
Figure 3 Museum of Science and Technology	12
Figure 4 HTC® Desire® (front and back sides)	13
Figure 5 Physical keys of the Desire®	13
Figure 6 Museum of Acropolis Audio Guide Maps	24
Figure 7 AMNH Explorer Pre-loaded tours feature	25
Figure 8 AMNH ExplorerMap and directions feature.....	25
Figure 9 MoMa Calendar, tour and info features	26
Figure 10 MoMa QR Code (Android).....	26
Figure 11 Balsamiq Mockup User Interface with Android elements and controls	29
Figure 12 Pinch zoom gesture	38
Figure 13 Exhibit Screen and menu	42
Figure 14 Full Portrait Screen	43
Figure 15 Help menu option.....	43
Figure 16 Checking QR Code	44
Figure 17 Exhibit Screen after scan a QR Code.....	44
Figure 18 SAX parser operation.....	50
Figure 19 DOM parser operation	50
Figure 20 List of Tours	55
Figure 21 List of Exhibits.....	56
Figure 22 Tour Exhibit Screen	57
Figure 23 Exhibit screen menu options	57
Figure 24 Exhibits in the HCI Laboratory Group.....	63
Figure 25 Virzi's Formula.....	66
Figure 26 The Android Software Stack.....	80
Figure 27 Life Cycle of an Android Activity	83
Figure 28 Waterfall model.....	86
Figure 29 Iterative and incremental development.....	87
Figure 30 Eclipse IDE for Java Developers Welcome page running on Windows 7.....	90
Figure 31 "Hello Android" Screenshot.....	94
Figure 32 Eclipse New Project Menu.....	94
Figure 33 Eclipse New Android Project dialog.....	95
Figure 34 Eclipse project listing after creation of the HelloWorld project	95
Figure 35 Hello Android String editing.....	99
Figure 36 Eclipse Application Type selection.....	99
Figure 37 Emulator and first try at Hello Android	100
Figure 38 <i>Zxing</i> Library contents	101
Figure 39 Apache Ant contents	101
Figure 40 Checking the installation of Ant	102
Figure 41 Building ZXing	102
Figure 42 Wireless shortcut and tray icon	111
Figure 43 Applications menu	111
Figure 44 The <i>tap</i> gesture (left) and the <i>drag</i> gesture (right)	111
Figure 45 The Tour List Screen.....	112
Figure 46 The Exhibit List Screen.....	113
Figure 47 The Exhibit Screen and the Full Screen portrait	114
Figure 48 Start and end of the tour buttons	114

Figure 49 The Exhibit Screen menu	115
Figure 50 Audio play/pause buttons	115
Figure 51 The Exhibit Screen Help option	116
Figure 52 How to scan a <i>QR Code</i>	117
Figure 53 Example of <i>QR Code</i>	117

LISTINGS

Listing 1 Full Screen Activity	45
Listing 2 Creating a new Compound Control.....	46
Listing 3 Example of layout for the new View	46
Listing 4 Inflating the new Compound Control.....	47
Listing 5 Some MediaPlayer methods.....	48
Listing 6 Example of XML file	49
Listing 7 XML File.....	51
Listing 8 The Exhibit POJO	52
Listing 9 Parse the data of the exhibit	52
Listing 10 <i>URL</i> in which is stored the <i>XML</i> file.....	52
Listing 11 Creating a new instance of a <i>SAXParser</i>	52
Listing 12 Creating the ContentHandler and apply it to the XMLReader	52
Listing 13 Start tag of one of the fields of the XML file.....	53
Listing 14 Method for starting the reading	53
Listing 15 End tag of one of the fields of the XML file.....	53
Listing 16 Method for closing the reading	53
Listing 17 Method for reading the data	53
Listing 18 Treating the data from the <i>URL</i>	53
Listing 19 Parsing Images on Android	54
Listing 20 Return the data	54
Listing 21 Setting the result to be displayed in the GUI	54
Listing 22 Layout for the ListView	58
Listing 23 Layout for the Tour List/Exhibit List.....	58
Listing 24 Elements of the ListView	58
Listing 25 Base feed parser	60
Listing 26 Tour List XML file.....	60
Listing 27 Simplified Android SAX Parser.....	61
Listing 28 Tour List Custom Adapter	62
Listing 29 Basic Activity Stub.....	82
Listing 30 HelloAndroid source code.....	97
Listing 31 main.xml source code.....	98
Listing 32 Scanning via Intent in Android	103

LIST OF CHARTS

Chart 1 Participants language selection.....	15
Chart 2 Android Platform knowledge and contact among the participants.....	16
Chart 3 Intuitiveness and Interest of the methods	17
Chart 4 Technology which is likely to produce more mistakes	18
Chart 5 Evaluation of the application contents: description and audio respectively.....	19
Chart 6 Appearance and UI of the application	20

LIST OF FLOW CHARTS

Flow Chart 1 BarCode Scanner Installation	31
Flow Chart 2 Scan QR Code	36

LIST OF MOCKUPS

Mockup 1 Title Screen	30
Mockup 2 Check out if the BarCode Scanner application	32
Mockup 3 Main Menu	33
Mockup 4 Image Map Gallery using TabWidget	34
Mockup 5 Insert Number with number keypad	35
Mockup 6 Help/Information Screen	37
Mockup 7 Map Feature	38
Mockup 8 Museum Tour functionality.....	39
Mockup 9 Tour menu	39
Mockup 10 Exhibit Screen	40
Mockup 11 Exhibit Compound Control	45

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Twenty years ago it was never clear that the mobile devices will have such an important status in today's society. These days is common to see everywhere people who are playing with their mobile devices, surfing the *internet*, listening to music, watching movies or using applications of every kind. That makes the traditional mobile phones more similar to a *PC*.

So not the mobile phone itself seems to be the most important thing. It is the operating system and the applications on it which can make the difference.

One example of this arising phenomenon is Japan; in Japan more people are connected to the *internet* via their mobile phone than there are *PC*'s with online connections.

This massive potential market has been discovered a few years ago by one of the biggest *internet* companies of the world: *Google*. *Google*'s approach was to develop an operating system which can run on every mobile device and not for the mobile device itself.

That shows what *Google* strategy is: reach as much people as possible and provide an affordable operating system for everybody.

1.2 BACKGROUND

A museum provides physical surroundings for touring people to acquire knowledge. Countries all over the world are using museums as a core facility to promote culture, art and tourism by widening their collections and services.

Exhibitions in museums generally have descriptions beside them in the form of written board or pamphlets. However, these media are often inconvenient for visually impaired people, children and the elderly...

As everybody can see in most of museums today, three different guide systems are provided. One of which is the traditional *tour guide*, the other is the *tape machine*; and finally, the *CD player*.

Many museums employ *tour guides* to provide vivid descriptions. However, the limited human resources mean that they can only provide group guidance, and they are unable to guide each visitor individually. This is done completely by manpower, suffering from high expenses on trainings and wages.

The use of *tape machines* and *CD players* seems to be cheaper, but they are both bounded by the storage capacity of those devices. People who have used it would also

notice that they are too big and too heavy to carry around. Not to mention that these means lack of the ability of interaction.

Therefore, this inconvenience gives the motivation to develop a museum tour guide system based on the modern *wireless* technologies and hand-held devices such as *Personal Digital Assistant (PDA)*, *Intelligent Mobile Devices* and *Smart Phones*.

Recent advances in information and networking technology, along with the increase in ownership of *wireless* network devices, have led museums to begin to construct *wireless* guidance systems.

Current *wireless* guidance systems in museum adopt *wireless RFID* technology to place *RFID* tags on their collections. For more information about *RFID*, see [2].

There are alternative methods of interaction apart from *RFID* like *QR Codes*, *Image Map* and *Numerical Codes* based on *wireless* technology [2]. As long *Android* devices (the devices used in this paper) don't support *RFID* methods, this thesis will focus on some of those methods.

Wireless networking technology, that is, *IEEE 802.11 b/g/n*, integrates *wireless* networking technology and established museum content in digital archives format, and easily applies geographic information systems to provide *Location Aware Tour Guide System*.

Visitors may enter their personal location, background (*e.g.* language), the content of the materials they plan to visit, and visiting timeframe to set up their personal handheld tour guide systems. The guide function for visiting moving line shortens the visitor's visiting line searching time, enable the visitor gains a wide range of appropriate information, thus, increasing the satisfactory level of the museum service.

Handheld devices present a paradigm shift in the museum visitor experience. Once, museums struggled to entice visitors to access centrally located repositories of information such as kiosk-based maps or exhibition didactic materials; today, visitors can carry an entire library of museum knowledge with them.

Capitalizing on a decade of web development in museums, the new generations of handhelds can now tap the sum of a museum's stored content, acting as a portable personal window to that information.

Assuming that visitors will bring their own hardware allows museums to concentrate on creating content experiences, while spending less time and money on specifying and maintaining tools.

This research is to use mobile devices as personal museum expert such that people may visit exhibits freely while exposing themselves to a rich knowledge at the same time.

1.3 AIM OF THE THESIS

The goal of the project is to build a digital repository for the “*The Museum of Science and Technology*” (M.S.T) of the University of Patras and a set of services concerning management of the collection and support for visitors and educational activities.

The systems include navigation and educational support through *PDA* and *smartphones* context aware application as well as a web site that will allow access to information about the museum and the surroundings, visit preparation, educational activities visit planning and also digital repository administration.

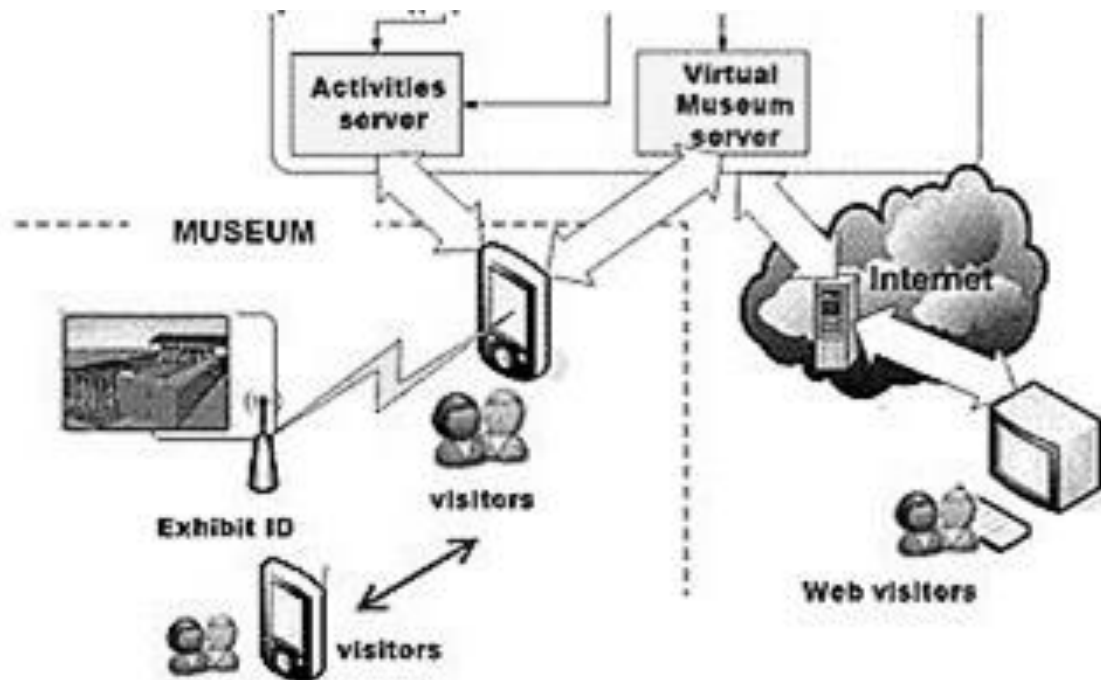


Figure 1 Abstract System Architecture

This thesis will aim to continue a previous project of an electronic museum guide for that developed in 2010 by a student of the same university [2]. That thesis focused in some human- computer interaction methods, so that shall be the starting point of this thesis, but not the final purpose.

The main goal of this thesis is to study and evaluate that electronic museum guide tour, improve its performance, features and usability. Also provide the user with a complete, easy and friendly application and implement some new features on handheld mobile devices, primarily based on the *Android* platform.

The way to do this is developing and evaluating new *Android* applications on the *Android* emulator from the official *Software Development Kit (SDK)*, and later test it on a real device which uses *Android* as native operating system.

For this purpose, an analysis and a design phases are essential, so this thesis also focuses in the design of a new user interface to make easier to the users the interaction with a handheld device.

1.4 RELATED WORK

For a comprehensive look into the development of an electronic guide museum see the thesis *Design and Development of a Mobile Application Based on Android for Physical-Digital Interaction* [2]. That thesis is the start-up for understanding the main goal of this thesis and all the work shown here is based on the analysis of that application.

1.5 OVERVIEW

This thesis is divided in 5 chapters corresponding to the *ADDIE* model. See Appendix B.

Chapter 2 introduces the application developed in [2] and the line of work. Therefore, the first step is analyze, test and study which aspects could be improved on it. For that purpose, it is essential to test the application in a real context (*The Museum of Science and Technology at the University of Patras*) and also check the results presented at the end of that thesis.

Chapter 3 proposes a new design of those aspects that should be improved. The use of mockups applications eases this work. This chapter also presents some of the new features the application could include.

Chapter 4 implements in *Android* language some of the new ideas presented in the previous chapter. First, it is presented the development environment, the device used for the testings and finally the code implementation.

The final experiment in which the reader can find the evaluation, testing of the new application, results, conclusions and future work of the application are discussed in Chapter 5.

The next figure shows how it is organized the document.

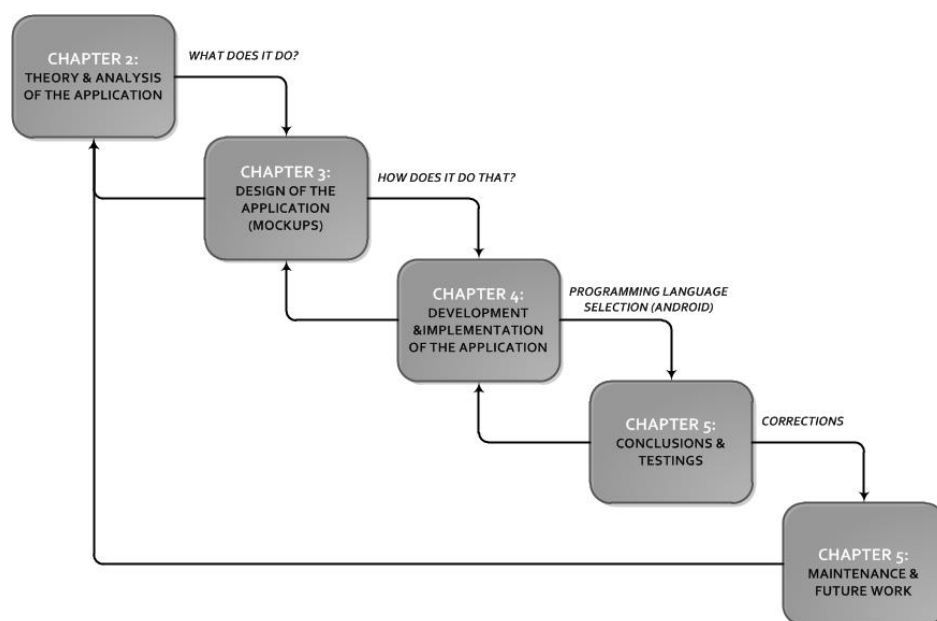


Figure 2 Overview flowchart.

The thesis also contains some Appendix the reader will find useful. By order, these are:

- Appendix A contains a general description of the *Android* Platform.
- Appendix B contains a general description of the *Software Development Process*.
- Appendix C describes how to set up the *Eclipse Development Environment*.
- Appendix D describes how to install the *ZXing* libraries for *BarCode Scanner* and *Apache Ant*.
- The questionnaire about the experiment proposed in chapter 2.
- The procedure, questionnaires and results about the final experiment for testing the new application in the chapter 5 are at the end of the document.

CHAPTER 2

THEORY AND ANALYSIS

2.1 INTRODUCTION

This section will evaluate the application developed in [2]. The main purpose is to analyze which aspects of the application should be improved for providing the users (visitors) a better interactive experience. This section also shows the results after the participants had completed the experiment and filled a small questionnaire about it.

This experiment took place on the “*Museum of Science and Technology*” of the University of Patras the second week of November 2010. For that, five random students from different countries with different studies were randomly selected.



Figure 3 Museum of Science and Technology

It is important to emphasize that some of the people surveyed were not experts on this field and most of them have not heard before of some of the technologies implemented.

2.2 EVALUATION

2.2.1 METHOD OF EVALUATION

The task of the testers is to fill a small questionnaire about the application. For that purpose, the scenario chosen for testing the application was the *Eminent Zakynthians Gallery* located on one of the walls of the *Museum of Science and Technology*.

First of all, the participants were provided with an *Android* device. Before running the application, it is important the participants become familiar and experiment with the device.

After that, the participants may take a look at the portraits and, at the same time, they may use the device and test the application.

Finally, and after testing all the interaction methods and options available in the application, testers will fill the questionnaire attending to the conclusions they have obtained after using the application.

2.2.2 TESTING DEVICE

The device provided to the participants to evaluate the application is the *HTC® Desire® Android Mobile* with operating system *Android 2.2* codename “*Froyo*”. This is a variant of *Google's Nexus One®*.



Figure 4 HTC® Desire® (front and back sides)

The device was kindly provided by the *HCI Group*. A brief description about the device could be found in the next paragraphs.

DESIGN

The dimensions of the device are *119 x 60 x 11.9 mm*, while its weight is *135g*. It is a quite compact and light device. Otherwise, the users will refuse to use it if the device is too big or heavy.

The front of the phone is taken up mostly by its *800 x 480 (WVGA) 3.7-inch* display. The size of the screen will provide the user a better experience.

The *Desire®* has physical keys for *Home*, *Menu*, *Back* and *Search*. Other buttons on the *Desire®* include the volume controls on the left edge and a power toggle on the top. Ports are standard with a *3.5mm* audio connector and micro *USB* port on the top and bottom, respectively.



Figure 5 Physical keys of the Desire®

FEATURES

Connectivity features on the *Desire*[®] are comprehensive, including *HSDPA*, *Wi-Fi* and *Bluetooth*. The *Desire*[®] also has *GPS* built-in, so satellite navigation is possible.

The camera found on the *Desire*[®] is a *5 megapixel* one with an attached *LED* flash for dark situations. This camera is compatible with recording video *WVGA* to *15 fps*. *Desire*[®] can reproduce video in *MPEG4*, *H.263*, *H.264* and *WMV9* formats and also to record sequences of video. It also has an integrated music reproducer that supports *WAV*, *MP3*, *AAC +*, and *WMA9*.

PERFORMANCE

With so many features and options to constantly stay connected, battery life will naturally be a concern. To test this out, every features included on the *Desire*[®] were turned on, including auto-sync of one e-mail account, weather updates and stock quotes.

The result was a flat battery from a full charge in about 20 hours. That was a pretty intensive trial that also included *web* browsing and the usual phone calls and messaging.

After adjusting the usage pattern, setting certain items to manual sync or reducing the update frequency, a single charge was managed slightly over a day.

Sound quality from the *Desire*[®] is good, providing clear voice for both the user and the other party.

The *1GHz Qualcomm Snapdragon* processor and *576MB* of *RAM* made sure things chugged along smoothly. No slowdowns were felt at all during the test period of the application.

2.2.3 PROCEDURE OF THE EXPERIMENT

The procedure of the experiment is the same as exposed in [2, section 7.3]. As a summary of that, the participants may follow the following steps:

1. *First contact with the testing device and Android platform:*
 - a. Become familiar with the device (e.g.: physical keys, touching keys on the screen).
2. *Start the application.*
3. *Language selection:*
 - a. English
 - b. Italian
 - c. Greek
4. *Welcome Screen:*
 - a. The *menu* physical key displays the available modes: *QR Codes*, *Image Map* and *Numerical Codes*.
5. *Test QR Codes option:*
 - a. Focus the device camera on the *QR Code*.

6. *Test Image Map option:*
 - a. Tap on one of the displayed portraits on the device screen.
7. *Test Numerical Codes option.*
 - a. Type the number of the exhibit.
8. *Test the Exhibit Screen.*
 - a. Description
 - b. Image
 - c. Audio
 - d. Related Items

2.2.4 ANALYSIS OF THE OBTAINED RESULTS

This section exposes the results of the experiment once complete. Although five participants may be insufficient, if coupled this results with the obtained in [2], this will be enough information for the analysis of the application.

STUDIES AND DEMOGRAPHICS

The five participants are students of the *University of Patras*. Each one studies in a different department: *Chemical Engineering, Business Administration, Mechanical Engineering, Primary Education* and *Aeronautical Engineering*. All of them, respectively, are from different countries: *Greece, Italy, Poland, Slovenia* and *Spain*.

The kind of the studies and the demographic questions will show the level of knowledge of the users have about *Android* and the modes of interaction implemented in the application. The age range of the participants is between 21 and 25.

About the language the participants are fluent in, all of them are at least fluent in english (as well as their native language). One participant is also fluent in russian language and another one in serbian and croatian languages.

For using the application, three of the participants chose english for testing the application, one chose italian and the last one chose greek. This shows that most users will use his native language if it is available on the device, even if they are fluent in another one.

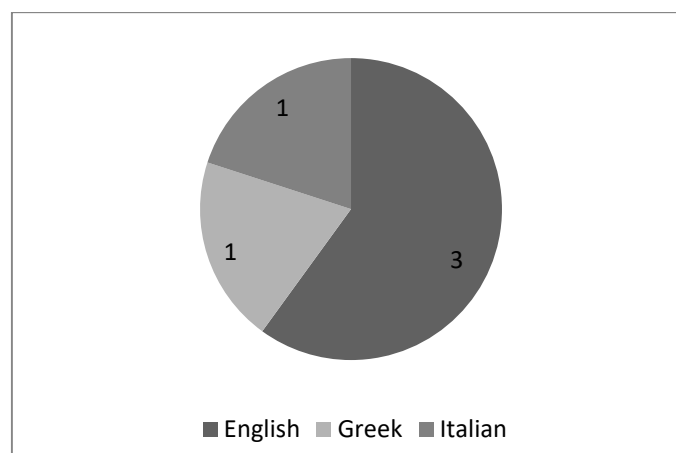


Chart 1 Participants language selection

ANDROID KNOWLEDGE

All of the participants have heard about what is the *Android* platform, although some of them do not know what it is exactly. This reveals that *Android* is, at least, a technology known by almost and is not dependent on technological knowledge of the participants.

As for who of them have had some kind of contact with some *Android* device, only one of them has interacted with an *Android* before this experiment. For the rest, this is the first time they interact directly with an *Android* device.

Although, nowadays there are a lot of low cost mobile devices which implements *Android* as operating system, most of the participants prefer the usual operating systems to the new ones.

Another reason is that, according to some studies and researches, *Android* is not an easy platform for people who are not interested in technology.

The conclusion is that *Android* is a well known technology, yet it has not had a remarkable penetration in the market, due to other available choices in the mobile devices market and because it is a recent technology.

The next chart shows the results about that:

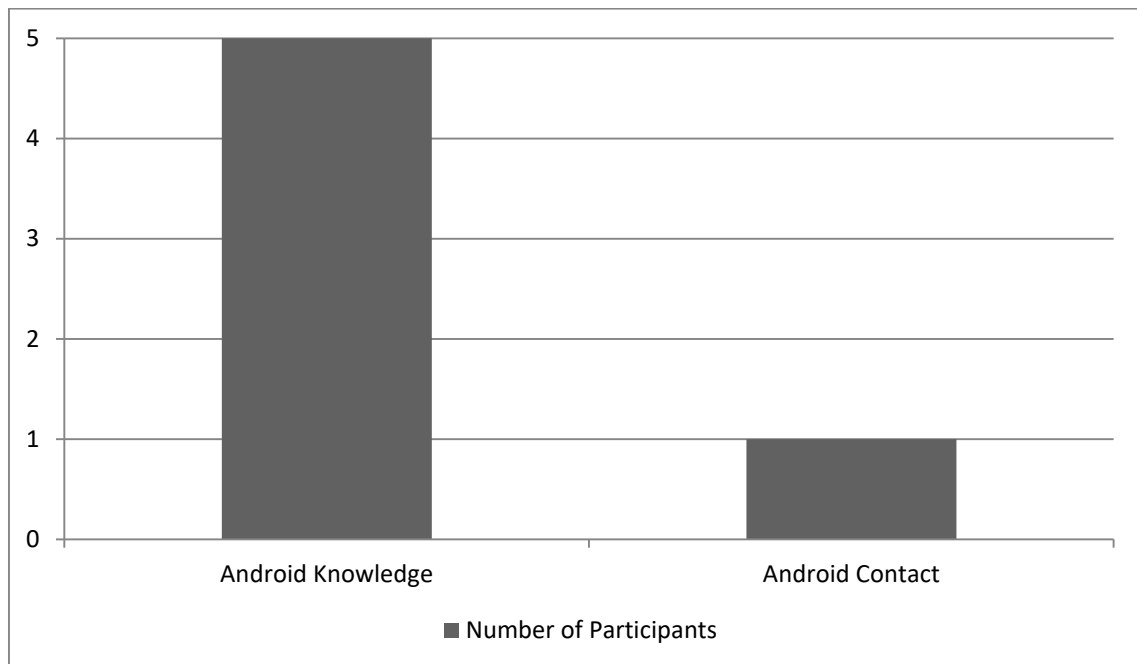


Chart 2 Android Platform knowledge and contact among the participants

TEST OF THE APPLICATION: TECHNOLOGIES

Once the participants have tried the three methods available in the application, almost of them think the *Image Map* is the most intuitive and interesting method. *QR Codes* method is also pretty close to *Image Map* results. *Numerical code* is the less intuitive and interesting method for the users.

Only two of all participants had contact with some of these technologies before they use the application: one of them with *Image Map* on the web, and the other with *Image Map* and *QR Codes*.

Although the participants answered they did not have contact before with an *Image Map* and they assured they did not what it is, they may have interact unconsciously with them sometime. This could be explained by the fact *Image Maps* is something usual on *internet* sites, but perhaps they do not know the technical name of that technology or they did not realize that they were using an *Image Map*.

That could be the explanation because for most of them, the *Image Map* was the most intuitive and interesting interaction mode. This means that they may use this method instead of others, confirming that for sure they have used it before.

The next graph shows the results; five means the most and zero the less.

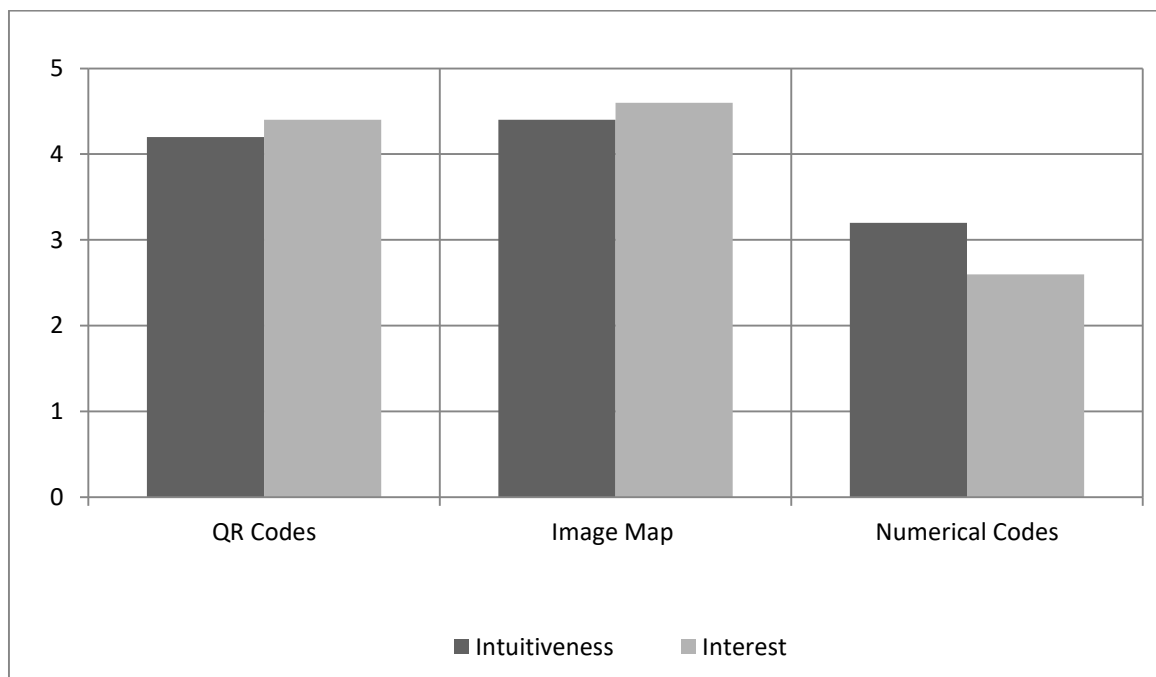


Chart 3 Intuitiveness and Interest of the methods

For the majority of the participants, like the experiment in [2], the *Image Map* is the easiest method. One of them suggested to display the exhibits on the screen in the order the visitor enters the room. Another one, while testing the application, tried to change the view of the device from portrait to landscape mode.

Some of the participants asked what would happen if there are a lot of portraits in the museum. Thus, this method will not be probably the adequate one.

QR Codes is also an interesting and intuitiveness option for the visitor. The problem of this method is that it is not well known among the participants and it may cause problems if they have never seen this kind of bar codes before. For example, some of the participants focused the camera in the portrait instead of focusing it on the *QR Code* (they think this method was something related with *Augmented Reality* and *Image Recognition*).

Finally, no participants have chosen *Numerical Codes* as the easiest option. One of the participants suggested that instead of use a keypad with symbols, it would be better to display only a keypad with the numbers. All participants were a bit confused while using this method because they did not know how to insert the numbers.

On the other hand, three of the five participants think that *Numerical Codes* is the method may produce more mistakes; the other two participants think *QR Codes* method will be.

This could be explained due to the *Numerical Codes* method requires the user types the number on the device; the problem is if the users are children or elderly people they would make more mistakes. Also, more errors could be made if the screen device is not big enough.

For *QR Codes*, the other two participants think there will be problems if they do not know how to positionate the device for scanning the tag.

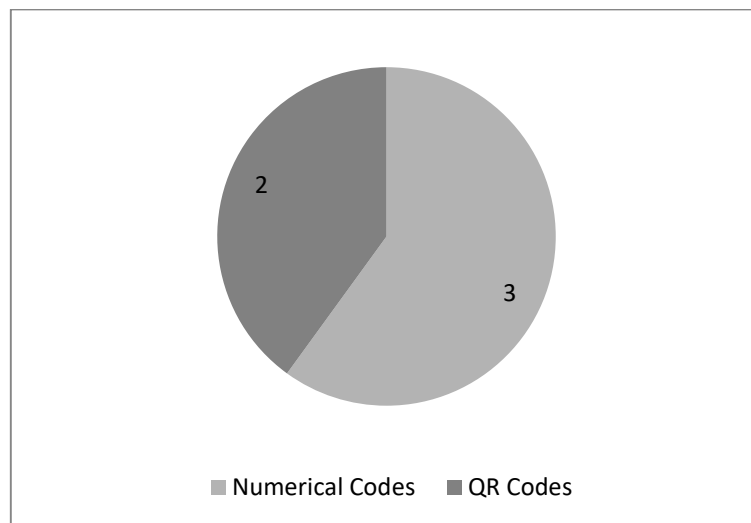


Chart 4 Technology which is likely to produce more mistakes

After checking each of the technologies, three of the participants would like to remove the *Numerical Codes* method from the application, due to they think this is a redundant method if they have available the *QR Codes* method.

The other two participants think on the principle “the more, the better”, so they think if the user has more choices, he/she will be happier with the application.

TEST OF THE APPLICATION: CONTENT OF THE APPLICATION

Almost all the participants are satisfied with the information provided with the exhibits: two of them think the information is enough, another two think the information is good and one think the information provided is too much because a lot of information could be boring for the visitor.

The same answer could be applied for the audio content. Although some of the participants think there are some problems for listening to the audio (due to a background noise), for all of them it was easy to understand.

Some have pointed out that if the application offers an audio content, which is the same as the description provided, it could be possible to offer the user a basic description, instead the whole description, and if the visitor would like to know more about one concrete exhibit, offer him the possibility of expanding the information with some kind of button.

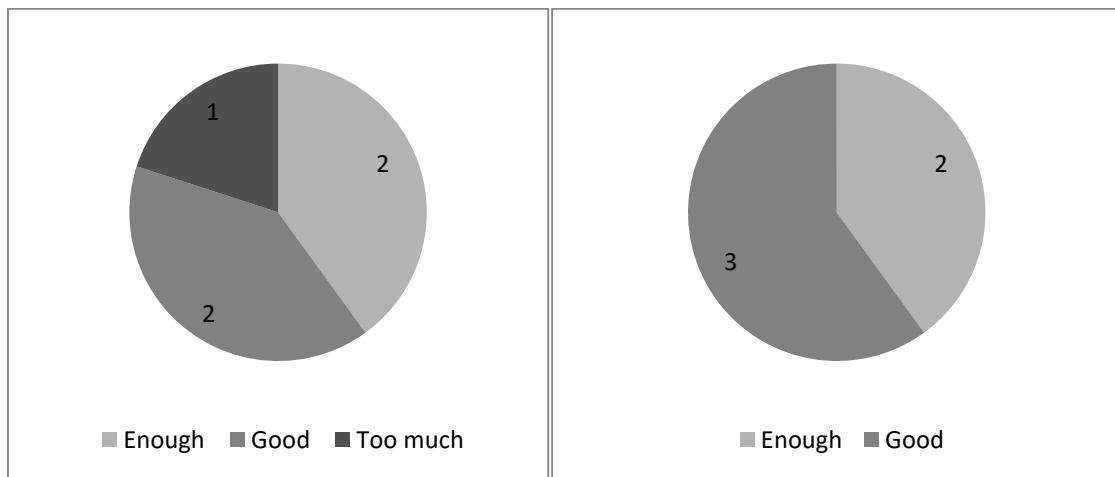


Chart 5 Evaluation of the application contents: description and audio respectively

All the participants would like the application includes other kind of media, like video clips, mini movies and history about the exhibits.

This means the participants are more interested in the media content rather the descriptions.

For all the participants, it will be useful to include some help that gives them additional information about the exhibits and how the guide works.

As for almost of them this is the first time they have used an *Android* device, they are not familiar with its interface and its buttons. This will be analysed in the next point.

TEST OF THE APPLICATION: USER INTERFACE

As seen above, participants would like the application was more intuitive. Some of them had some problems running the application because after they had selected the language, they did not know how to continue with the application in the “Welcome Screen”. All of them tried to tap on the screen device, but they did not notice they have a physical key on the device (*menu*). Therefore a help button, or launch the menu with the methods availables will be one of some solutions. The participants also want help features about how the provided methods work.

As regard, three of the five participants think the font size of the descriptions is not enough, they point out that for young people is enough, but it may not be sufficient for people with eyesight problems or for elderly people. All of them think that the portrait image displayed on the device with the description is not enough, and they would like a larger portrait.

Almost of the participants would like to read the description and view the portrait in full screen on the device. Most participants claim to have a “clean” window for the application.

On the other hand, three of the participants think the buttons in the application are necessary and two of them think the buttons should be bigger. The majority of participants think that if the application works by tapping and touching the screen device screen, the physical keys of the device are not useful. It would be easier for them to interact only with the screen device.

Four of five participants would like to have other languages available in the application, although they are not fluent in those. Four of them indicated they want spanish, two of them want german, and one of them also wants french, polish, russian and chinese. This could be explained because they think major languages should be offered in every application.

Finally, four of the participants liked the appearance of the application. They think it was friendly and pleasing (design, interface, images, colours, seize of the icons and buttons), so they were satisfied.

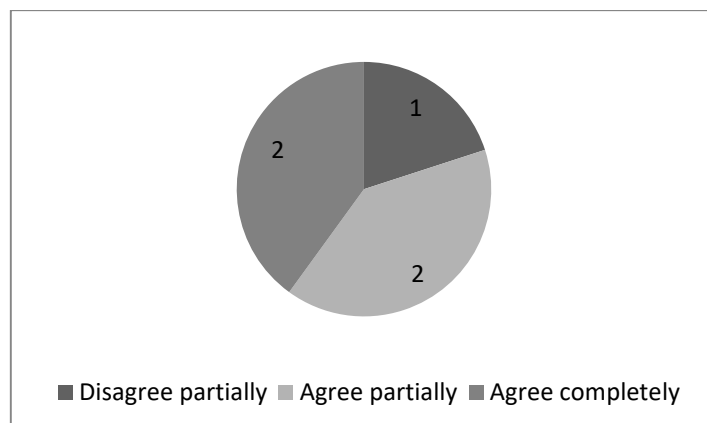


Chart 6 Appearance and UI of the application

NEW PERFORMANCES AND FEATURES

This point presents the improvements the participants would like to add to the existent application.

All of them think a *Tour Functionality* would be very useful, because the visitor may be only interested in some of the exhibits or he/she may have a limited time for visiting the museum.

They also think a *Map Functionality* would be useful for an orientation inside the museum. They pointed out that in a real museum with two or more floors, they usually get lost and confused if they do not have a map.

Two of the participants would like to include some kind of “*mini-games*”. They think it would be a nice idea for school trips to museums.

One of the participants would include also an *Exhibits Topic filter*, and *Options Menu* and *Background Music*.

FINAL COMMENTS

Finally, every participant in the experiment thinks that this electronic guide is useful for a real museum and they will use it.

Only one of the participants has used an application like this before in a museum, more specifically the *AMNH Explorer Application*.

Eventually, it is presented some feedbacks from each participant:

- More functionalities, help features, news about new exhibitions in the museum, calendars about special events.
- Offer the use the possibility to change between portrait and landscape mode. A more intuitive *User Interface* (do not use the physical buttons on the device).
- More buttons, display only the basic information and if the user wants, display extended information about the exhibit. Put some more media (videos, music), offer new languages.
- Offer the user zoom for reading the exhibits description and the images.

The whole questionnaire could be found on *Appendix C*, at the end of the document.

CHAPTER 3

DESIGN

3.1 INTRODUCTION

This chapter focuses in the new *UI* design of the application. The design will be based on the feedbacks reported by the users and the analysis of the results obtained in the questionnaire in the previous chapter.

In addition, it is necessary to evaluate similar applications on the market for *Android* devices (or other) in order to get some new and innovative ideas.

The *GUI* will be designed using a *mockup* application. There are several different ways to design a software application. Some developers like to start with programs like *Adobe® Photoshop®* and create the complete design. Others like to start with code, and build up the look at the same time they are creating functionality. Another technique is *wireframing*, where the developer “mockups” just enough of a page or screen’s layout to understand the functionality, then turn it over to a designer to create a finished look.

3.2 STATE OF THE ART

The *Museums & Mobile Survey* [8] is an ongoing international research initiative that tracks the museum community’s use of and ambitions for mobile technology tools.

It provides a unique insight into the community’s objectives with mobile guides, the challenges they face in developing and sustaining them and their vision for the future.

This survey launched in September 2010 had more than 700 responses, with professions ranging from museum employees to mobile technology vendors and researchers. The effort was international, but the majority of responses (80%) were from the United States, with the United Kingdom (5%) and Canada (4%) as the next largest group of respondents.

The survey points to some interesting trends that are cropping up with mobile development for museums. There is a ton of information in the survey, but here are some of the interesting things from it:

- Of the 738 survey participants, 30% already have some form of a mobile program in place and 23% planned to develop one. 36% of institutions had no plans to go mobile and the remaining 11% were responses from vendors and researchers.
- When asked which terms best described their current or planned mobile program, the most common responses were: some sort of audio tour, free for visitors, and visitors were expected to provide their own hardware (i.e. *smartphone*, *iPod®*).

- The goals for the mobile programs were most commonly described as providing supplementary information and diversifying the visitors experience.
- The most challenging for museums with mobile programs in place is encouraging adoption among visitors, producing the content, and keeping that content up to date.
- The largest challenge for those planning to develop a mobile plan is the implementation costs.
- Looking ahead five years, most institutions said that implementing in-house content development was a definite goal.
- What is least likely to be implemented in the next five years: institution-wide *wifi*, real time location services and *Augmented Reality*.
- The most requested areas of research were: guidelines for user experience design, methods for conducting visitor evaluations and analysis of different technology systems.

Mobile programs are becoming more common and less of an anomaly among museums. The survey also provides a snapshot of some of the concerns about development and adoption of a mobile program.

Also, the surveys pointed out that 50-55% of participants consider having a mobile friendly website as something that should be definitely implemented in the next five years.

An organization's website will often be the first portal of entry for a mobile user, and having a website that is optimized for mobile platforms should be a higher priority.

There are a lot of applications which provide the user a complete museum guide. This section overviews some of these applications for different kind of devices (mostly for *Apple® iPhone®* and *Touch®* and *Google® Android®* devices).

3.2.1 MUSEUM OF ACROPOLIS AUDIO GUIDE

This paid application is available only for *iPhone®*, but it contains some interesting features: it offers the visitor a conducted tour by navigating on the map or by touring each exhibit.

The application developed in [2] does not offer a touring option or a map in which the visitors could look up the place of a concrete exhibit or their own position. It is true to say that in [2] there is only one gallery in a small museum, and maybe it is not necessary a map, but if the application is used in a bigger museum, as this one, it is indispensable to offer the user some kind of orientation.

Other interesting features of the application are: virtual guide in the museum, photographs of the exhibits, presentation of exhibits per storey (*map gallery*), possibility to search for an exhibit by name (*numerical codes*), vocal description of all the exhibits, written text for all the exhibits, use of the program without *internet* connection or *wireless* (an advantage if there is not *internet* connection). Most of these features are implemented in [2], so the most interesting is the *touring* and *map* features.

Visitors can stop and continue the conducted tour whenever they wish; they can follow the suggested tour from the map or make their own tour. Moreover, they can repeat the vocal information of every exhibit as many times as they like. This option gives them the opportunity to go over the tour visit in order to review it or share it with others.

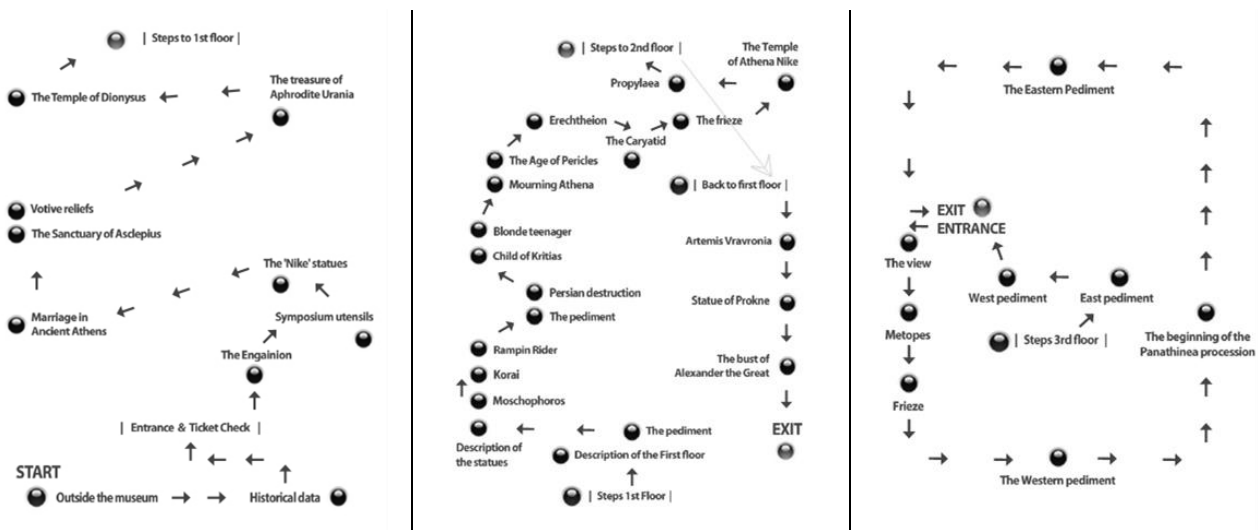


Figure 6 Museum of Acropolis Audio Guide Maps

The figure above shows the *map* implementation of the application. It looks very simple, but also intuitive and it implements the “*pinch zoom*” gesture. This gesture allows the user to zoom the map and then tap on the “*hot spots*” for displaying the information related to the exhibit, gallery or wall.

3.2.2 AMNH (AMERICAN MUSEUM OF NATURAL HISTORY) EXPLORER

This free application is available for *iPhone*® and *iPod*® *Touch* (and probably available on *Android*®).

The American Museum of Natural History has created an easy-to-use application with information about several star attractions, specific tours to guide the visitors through the museum, and a locator that will lead them to exhibits, cafés and restrooms. The most interesting point of this application is that it offers to the visitor five pre-loaded tours (the visitor can choose from *Museum highlights* or in *depth guided tours*) or even *custom tours* (the visitor may be able to plan his own tour before he arrives or on the spot). After hitting each stop and reading the two factoids provided, the visitor will be directed to the next checkpoint with clear maps and step by step directions.

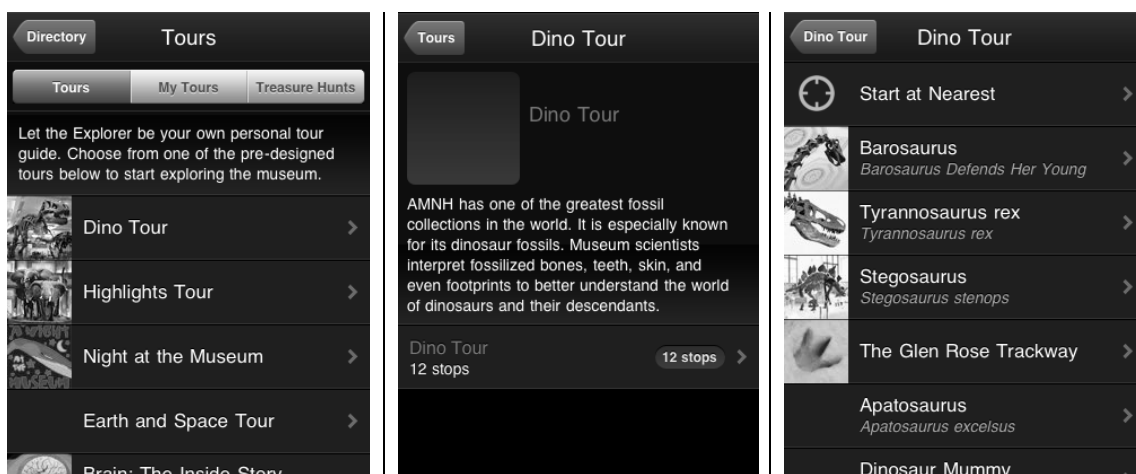


Figure 7 AMNH Explorer Pre-loaded tours feature

The tours help users narrow the focus of their visit and the facts included offer more information than can be found on the nearby plaques, particularly about conservation. However, some halls have no special exhibits highlighted, which can force visitors to learn about artifacts the old-fashioned way.

Other key features are: personal *GPS* that allows the users to find their current position in the museum and three dimensions map and turn-by-turn directions.

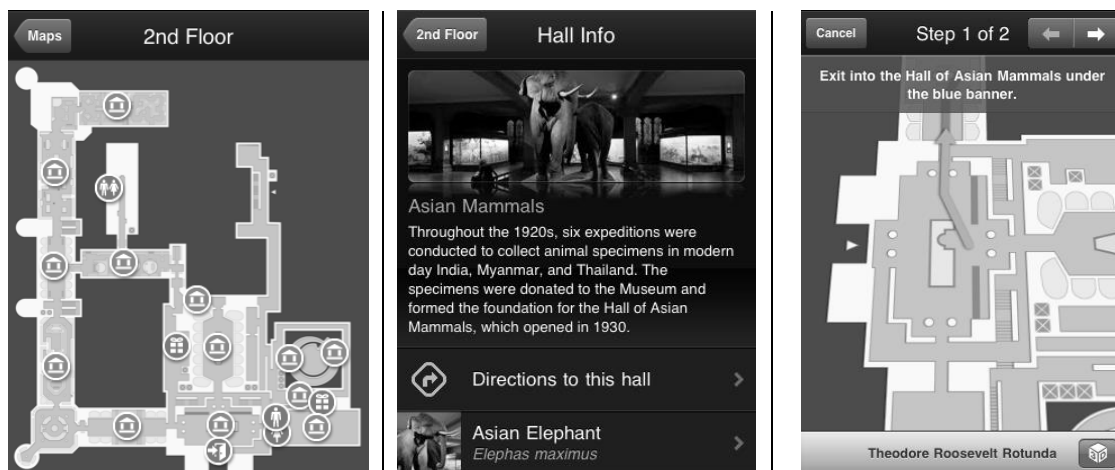


Figure 8 AMNH ExplorerMap and directions feature

3.2.2 MoMA (MUSEUM OF MODERN ART)

This free application is available for *iPhone*[®], *iPod*[®] *Touch* and *Google Android*[®]).

With this application, the visitors can browse images and information about exhibitions, artwork and artists in the collection.

There are also audio tours of specific parts of the collection, and a few playful extras of varying quality and usefulness.

The events calendar (which is updated daily) is the most useful tool. While the application has value as a navigational tool, it does not necessarily live up to the creative promise somebody would expect from a museum guide.

It also requires permanent connection to the internet.

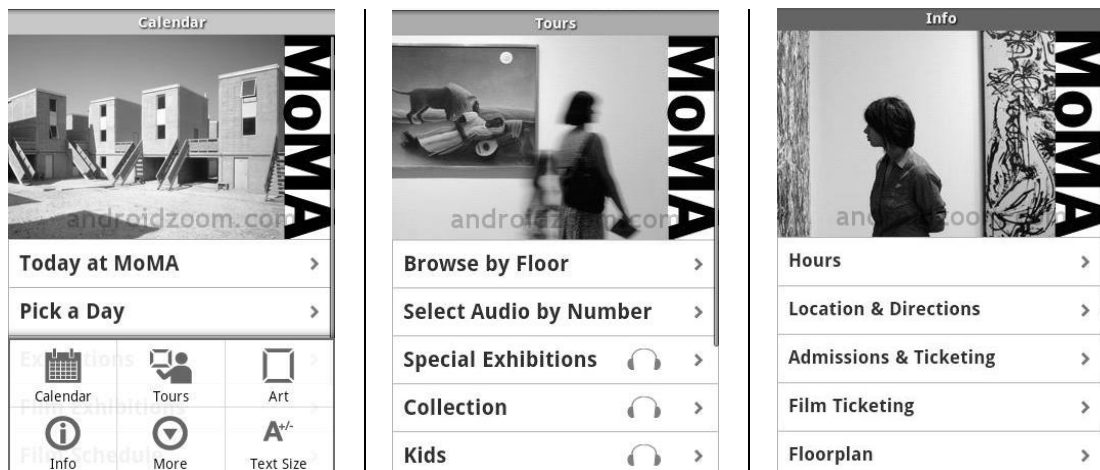


Figure 9 MoMa Calendar, tour and info features

The follow *QR code* allows the reader (if he/she has an *Android*[®] device and the *Barcode Scanner* application installed on his/her device).



Figure 10 MoMa QR Code (Android)

3.3 MOCKING-UP

There are a number of steps all developers have to go through when designing a piece of software or an application. After that moment of insight, the designer on duty needs to create an actual design and ‘*sketch*’ the look and feel of the application, essentially defining the graphical user interface. For this purpose, mockup conceptual drawings and wireframes are used.

3.3.1 MOCKUPS

Mockups are a way of designing user interfaces on paper or in computer images. A software mockup will thus look like the real thing, but will not do useful work beyond what the user sees. A software prototype, on the other hand, will look and work just like the real thing. In many cases it is best to design or prototype the user interface before source code is written or hardware is built, to avoid having to go back and make expensive changes. Software and systems designers use mockups mainly to acquire feedback from users about designs and design ideas early in the design process. Mockups are “very early prototypes”. The user, aided by the designer, may test the mockup (imagining that it works) and thus provide valuable feedback about the functionality, usability and understanding of the basic design or idea.

The advantages of mockups are numerous. Some of them are the following ones:

- Mockups incite criticism from users because they are low cost (the design of a page or screen is very quickly) and low fidelity. If a user is presented with an early version of a system that has required substantial work, the user is likely to be more reluctant (as well as able) to criticise it.
- When using mockups applications, the user and designer can collectively change the design; they only need a computer, the same mockup application, a basic knowledge about how that application works and an *internet* connection to share the *layouts*, and they don’t need to meet physically. As such, mockups are a discussion medium and a discussion facilitator between designer and user.
- Not only can the mockup function as a discussion medium between designer and user but also between the members of the design team. Thus, mockups may help facilitate work across disciplinary borders, bringing together a disparate team.
- Mockups make it possible to do usability testing early in the development process.
- Mockups incite and legalise experimentation as they are inexpensive to alter.
- Mockups focus on content and functionality and turn attention away from details of graphic design.
- Mockups designs can save time. By saving a mockup as a template, the designer can reuse it for other projects.

Mockups address the idea captured in a popular engineering one liner by *Frank Lloyd Wright*:

"You can fix it now on the drafting board with an eraser or you can fix it later on the construction site with a sledge hammer."

3.3.2 MOCKUP SOFTWARE APPLICATIONS

There a lot of *mockups* applications available on the *web*, paid or free. In general terms, when the designer is working on a project that involved designing several screens, receiving feedback on them and then rapidly reiterating soon after, the mockup application should:

- Be versatile; have many elements that let the designer create a mockup that truly reflects what the designer envisions.
- Let the designer export in *PNG* format (or something else that is high quality).
- Allow the designer to share the work with the final user and get direct feedback on it if possible.
- Let others work on the mockups if needed (team editing).
- Be affordable.

3.3.3 BALSAMIQ

Balsamiq is the selected *mockup* application used for designing the new application. Although this application is not for free, the designer can try it on the *web*, or download a trial for seven days (*Adobe*[®] *AIR*[®] is required for running the application). It also works on *Microsoft*[®] *Windows*[®], *Apple*[®] *Mac OS*[®] and *Linux*.

There were some reasons for choosing this application over other alternatives; it has an intuitive interface, an endless supply of elements and controls, plugins for lots of others applications and it can work *off line* (desktop version with *Adobe*[®] *AIR*[®]). It is fast and easy to work with and there were literally dozens of elements that the designers can customize and use in their mockup.

The *layout* is intuitive and makes designing both quick and efficient. The hand-drawn is simple, rough and letting the client know that they're not seeing the end product, rather just a mockup. Another great feature is that the designer can get more elements and/or controls from mockupstogo.net, where users contribute their own creations (an endless supply). Some of these elements are related to *Android* design, so the designer almost has every element and controls he/she will need to start the designs.

Of course, there are some cons: the application has not an easy way to share and it does not offer team support or editing for the current version the only way to share their work using *Balsamiq Desktop* is to export and *e-mail* it to recipients. When a designer is working with several mockups and have to go back and forth, this can quickly get annoying (anyway the *Balsamiq team* is currently working on a web version of the software that will offer much better collaboration features).

For the purpose of this project and for developers that just want to get their ideas down with minimum fuss, *Balsamiq Mockups* is the suitable tool.

3.3.3 UNDERSTANDING BALSAMIQ

The *Balsamiq UI* is very simple. There is workbook area, where the designer creates the mockups, and the available elements at the top. Dragging or *double clicking* one of those elements to add it to the mockup.

There's also a *UI library search* at the top, so if the element is not visible, the designer can quickly find it. All the elements have a *hand-drawn* look, which gives the whole process a feeling of sketching out an idea.

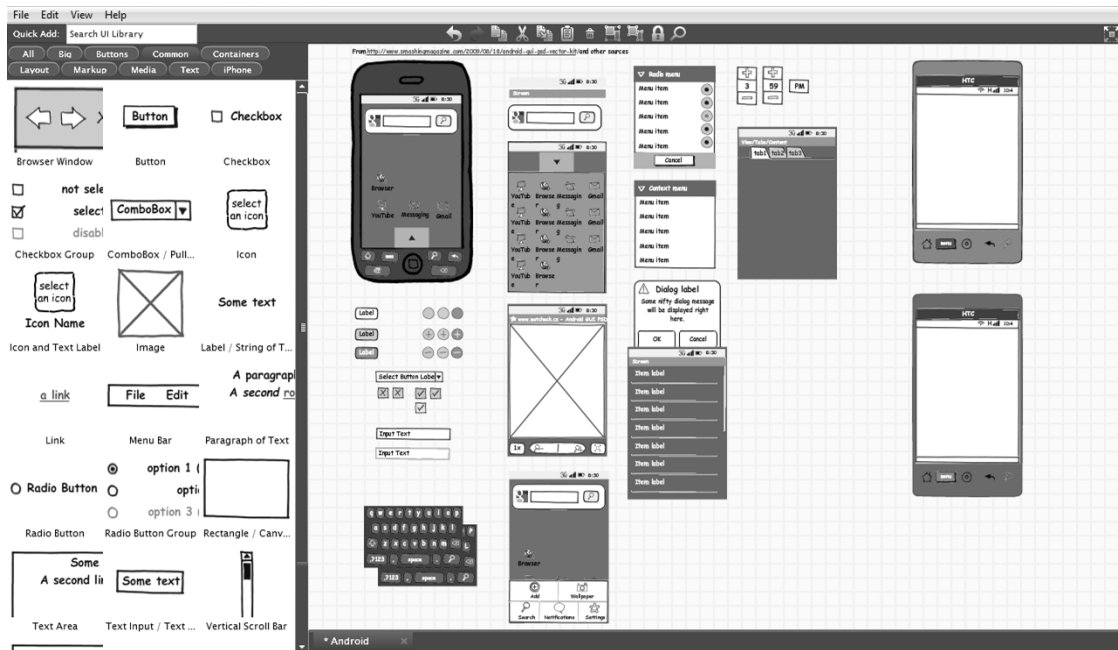


Figure 11 Balsamiq Mockup User Interface with Android elements and controls

Any element that has text can be *double-clicked* to change the text. For an element with multiple items, like a list, type in a comma-separated string to generate as many items as the designer wants. There is also a tool window that pops up when an element is selected that gives the designer context-sensitive tools for editing the element.

There is also a full screen view, which displays the mockup with a big blue arrow that it is possible to control with the mouse, something useful for presentations and pointing out specific elements.

3.4 DESIGNING THE NEW APPLICATION *GUI*

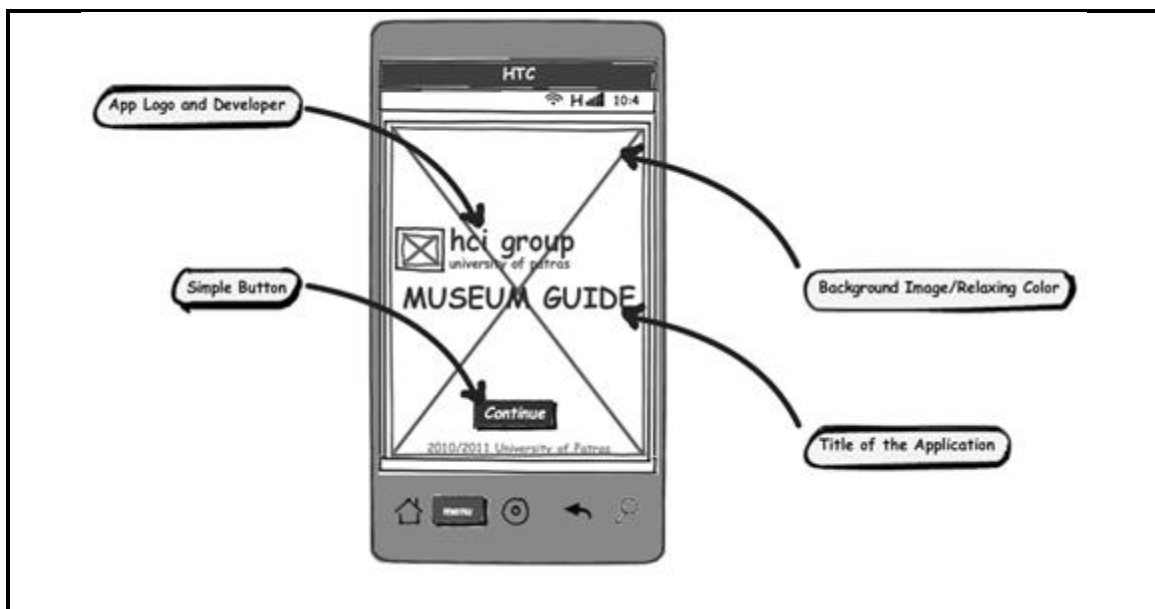
Once learnt how to use the previous tools, it is time to start designing an early prototype of the new application *GUI*.

3.4.1 TITLE SCREEN

Title Screen is a simple initial screen which serves as a splash page for enhancing the look and feel of the application. The information listed on the application screen usually includes:

- The name of the application
- The application developer
- An application logo
- The date of release
- The institution
- A simple button with the message “continue”, which the user may tap for starting the application.

The next mockup shows how this screen will look:



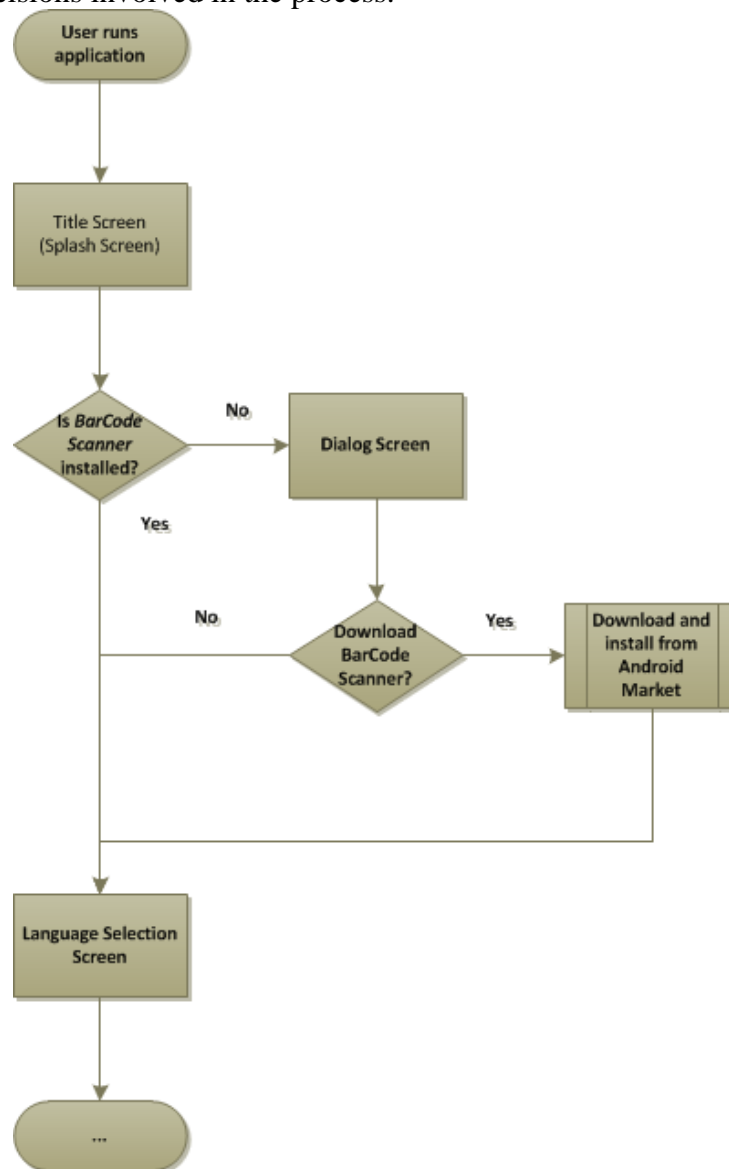
Mockup 1 Title Screen

3.4.2 CHECKOUT (BARCODE SCANNER)

The *Camera* option in the application developed in [2] implements some source code and some libraries from *Zxing* [6] for scanning the gallery portraits. For more details see Appendix C. *ZXing* is an open-source, multi-format *1D/2D* barcode image processing library implemented in *Java*. The focus is on using the built-in camera on mobile phones to photograph and decode barcodes on the device, without communicating with a server.

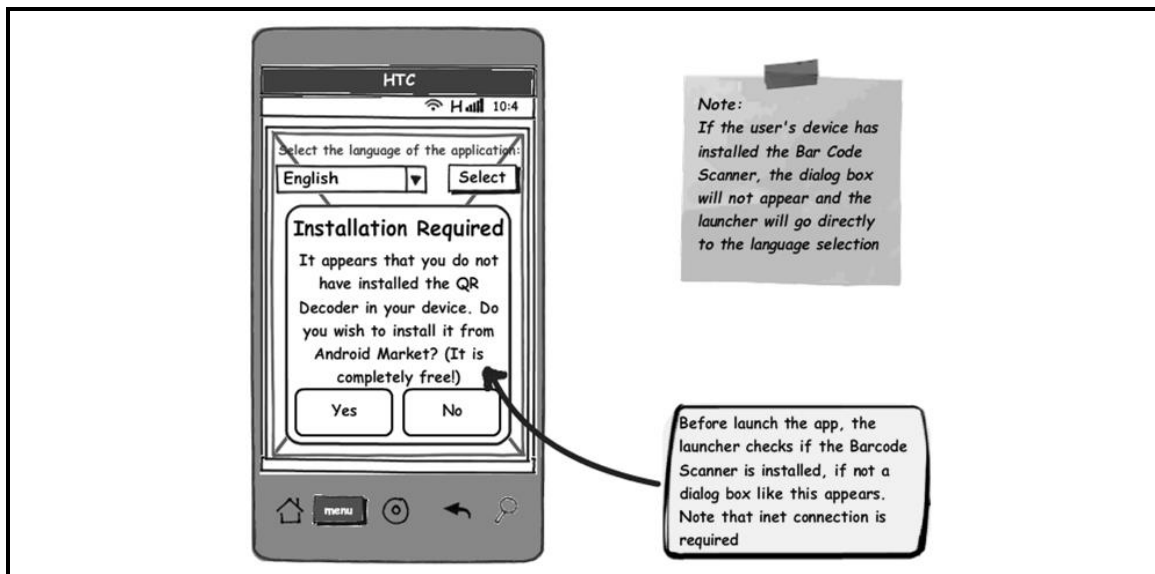
Considering that the museum has enough devices for providing the visitors, this feature will not be very useful, but maybe some of the visitors would like to use their own device, so the application may check if the device has installed the *BarCode Scanner* application.

So, when the user runs the application the first time, it should check if the device has installed that application or not. The flow chart in the next figure shows the sequence of actions and decisions involved in the process.



Flow Chart 1 BarCode Scanner Installation

It is supposed the visitor has *internet* connection on his device, otherwise the user could not install it and the application will throw an error (remember that the application requires a permanent connection to work properly).



Mockup 2 Check out if the BarCode Scanner application

3.4.3 LANGUAGE SELECTION

The application in [2] allows the user select among three different languages: *English*, *Italian* and *Greek*.

It is not tricky to add new languages to the application, but the most important feature the user would look for is the application starts in his/her native language (if that language is implemented). This is what it is called *Localization*.

Famous museums are frequently visited by people from different countries, so it is expected that not everyone may be fluent in other languages unless their native one (for example children, elderly people). The application should offer this variety of people the possibility of running the application in their native language (also the audio), otherwise the guide will be useless.

Implementing this feature, the museum will be able to offer their visitors the devices in their own language. For example, if the visitor has his own device set on greek language and he runs the application, this one will start on greek.

It is also possible to change the language of the application any moment the user wishes.

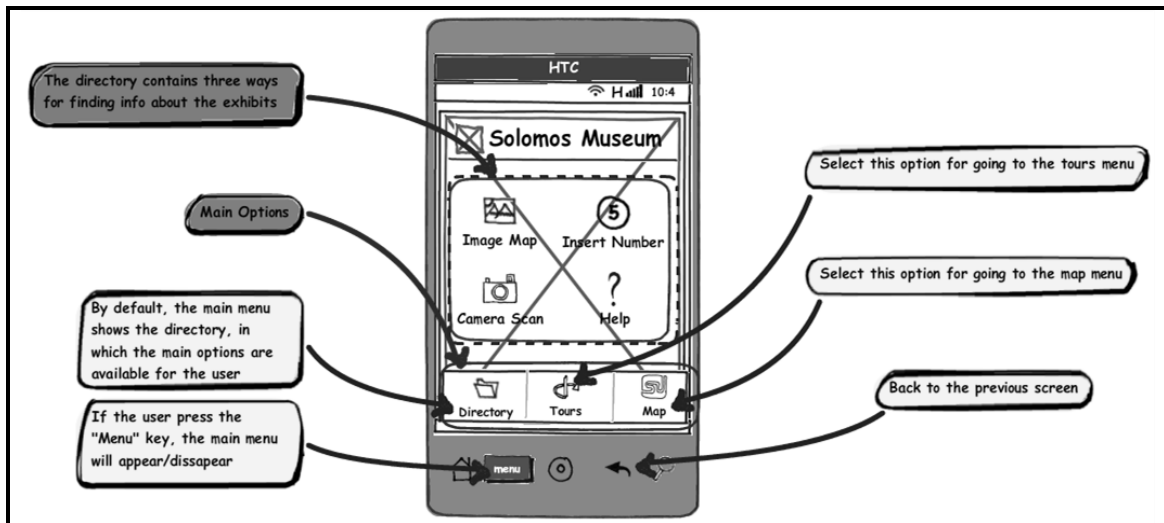
Therefore, one of the principal purposes of the new application is making it appocheable to everybody.

3.4.4 MAIN MENU SCREEN

Results shown in previous chapter show the welcome screen in [2] are not so intuitive for the people who fill the questionnaire, due to the fact that almost of them did not have contact with an *Android* device until they tested the application. This problem could be solved if this screen shows the menu options at first (without having to press the *menu key*), or redesigning the main screen.

Due to the new features the application may implement, one solution is to place some buttons on the main screen with the most important options, and also provide the user with some other options if the users press the *menu key*.

Provisionally, the application will show the three methods for get the information about the exhibitions: *Image Map*, *Numerical Codes* and *Camera Scan*. Tapping on each icon will drive the user to a new screen or/and mode.



Mockup 3 Main Menu

It is also important to provide the user with some kind of help (*help button*) about how the application works and the options it offers.

By pressing the *menu key* on the device, the application will display three modes: *Directory* (default screen), *Tours*; option in which the application suggests some possible paths through the museum among different preloaded tours and *Map*; option in which a simple map will be displayed.

3.4.5 DIRECTORY – IMAGE MAP

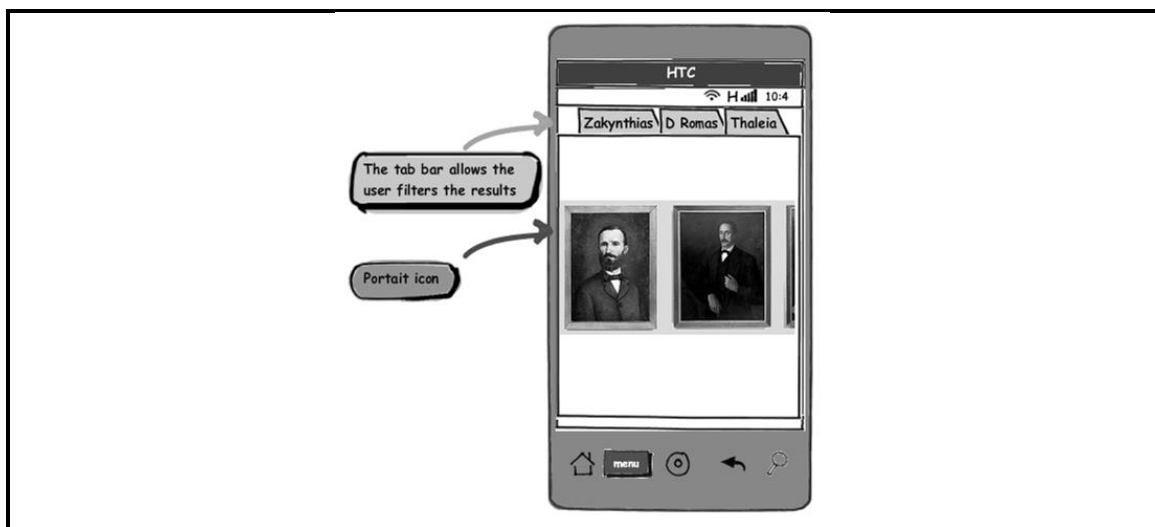
As showed in the results, an *Image Map* is a simple and an intuitive method for getting information about the galleries and exhibitions, so it is not necessary to change the *GUI*.

In addition, the *Image Map* matches with the same position the portraits are set out in the museum, so for the visitor would be easy to use this method.

On the other hand, the application in [2] focuses only in the portraits of “*Eminent Zakynthians*”, but that is not the only gallery the museum contains; it consists on the following: in the ground floor are the room of “*Dionysios Solomos*”, the room of “*Eminent Zakynthias*”, the “*D. Romas*” room and the “*Nicholasm and Thaleia Kolyva*” room.

The museum also has a library containing a large collection of documents, and also period furniture and archeological and folk art collection as well as a collection of old portraits, photos, pictures and woodcrafts.

Taking advantage of the widgets *Android* provides the developer, the use of a `TabWidget` will allow implement different activities corresponding to different galleries [5].



Mockup 4 Image Map Gallery using TabWidget

When the visitor tap on the different tabs, the device will display different *Image Maps* corresponding with the gallery the user wants check.

Another pro of using this widget, it is possible to implement the tab content in one of two ways: the developer can use the tabs to swap `Views` within the same `Activity`, or use the tabs to change between entirely separate `Activities`.

Which method the developer wants for the application will depend on his/her demands, but if each tab provides a distinct user activity, then it probably makes sense to use a separate `Activity` for each tab, so that he/she can better manage the application in discrete groups, rather than one massive application and *layout*.

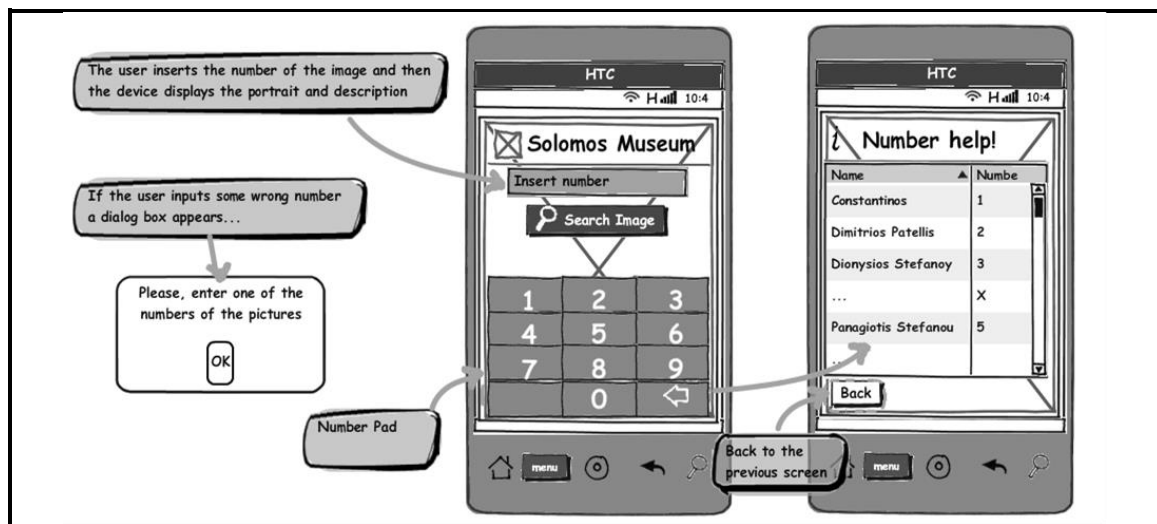
3.4.6 DIRECTORY – INSERT NUMBER (NUMERICAL CODES)

Beside every portrait in the museum, the visitor could see a label with a *QR Code* and also an “*id. Number*”.

The purpose of this modality is the user inserts that number on the device, tap on the “*Search Image*” button and the application will check if the number matches with some of the exhibits stored in the server. If yes, the device will show the information screen about the exhibit related to the number. If not, the device will ask the user for introducing a new number.

This method is already implemented in [2], but the user interface is inefficient due to the the keyboard the user may use. As far the user will only use numbers, the current keyboard has redundant keys (i.e. the symbols).

The aim is display only a “number keypad”, from 0 to 9 and a return button if the user would have a mistake typing the number.



Mockup 5 Insert Number with number keypad

This screen also implements a *help* button in case the labels are not available. The *Number help* screen would show the name of the exhibition and the *id. Number* which matches with that exhibit.

3.4.7 DIRECTORY – CAMERA (QR CODES)

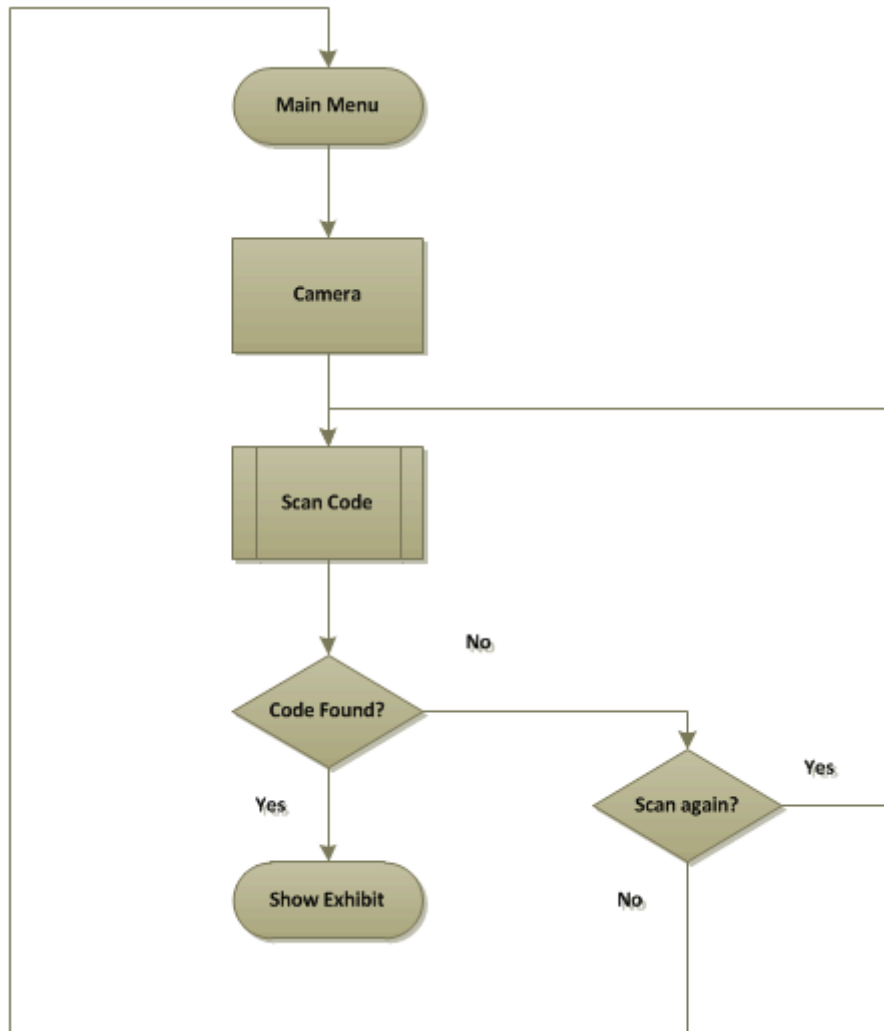
The last mode of operation for retrieving the information of the exhibits is using the *QR Codes*. As explained in 3.4.5, each exhibit has associated a *QR Code* and an *id. Number*.

This method operation is very easy: the user will scan with the device the *QR Code* and if the application found the exhibit related to the *QR Code* scanned, it will show on the device the information about the exhibit.

In case the *QR Code* scanned doesn't match with some of the stored in the database, the application will show the *Welcome Screen*.

One way of improving this, it is to allow the user scan a new *QR Code*.

The next Flow Chart displays that process.

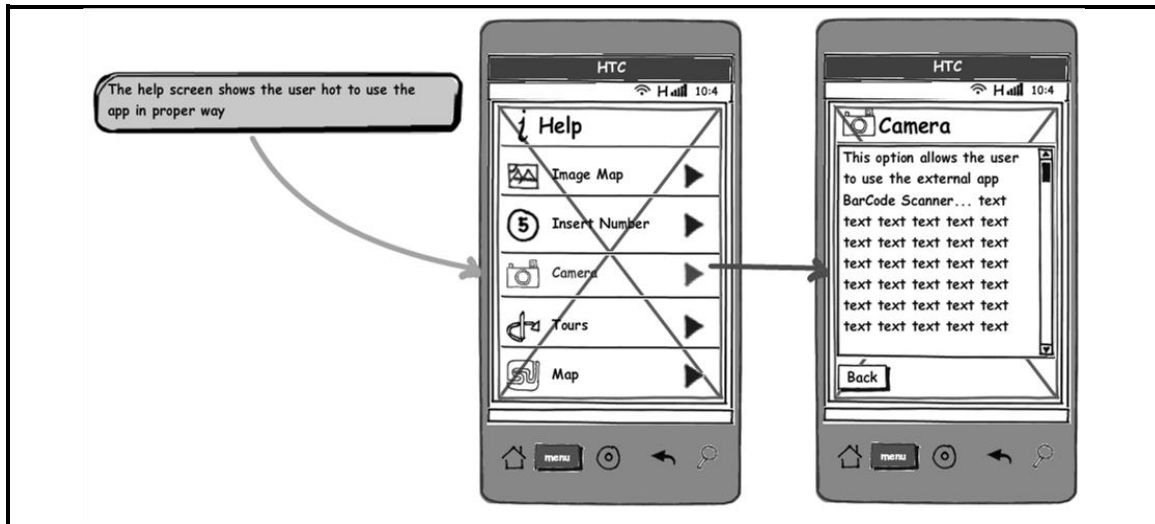


Flow Chart 2 Scan QR Code

3.4.8 DIRECTORY – HELP

The *Help Screen* is a simple screen where the user can check more information about how the application works.

This can be implemented using the `ListView widget` [5] from *Android*. A `ListView` allows the user select among of a list of items. Tapping on a concrete item, the user can read the related information about the mode he/she selected.



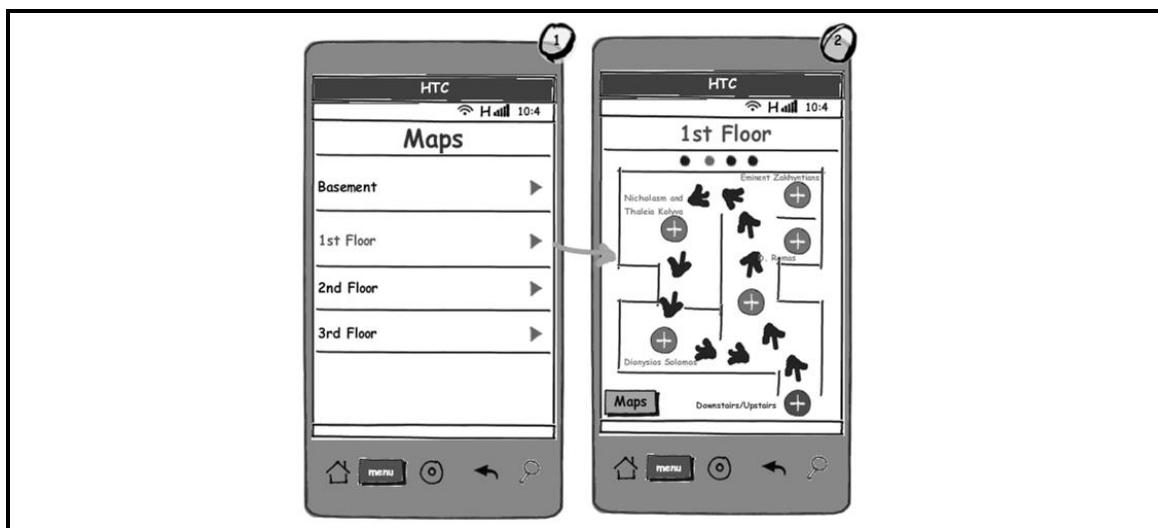
Mockup 6 Help/Information Screen

3.4.9 MAP FEATURE

The *Map* feature provides to the user an interactive map about the museum. A museum is usually divided into some floors and several rooms. For this reason, the first of all is offer the user a simple list in wich the visitor can select the floor of the museum. Tapping on the device the selected floor, the device will show a simple map (similar to the kind of maps the *Museum of Acropolis Audio Guide* offers).

A possible idea is reusing the *Image Map* technology. Creating an image map with a simple fixed background image of the floor will allow to hyperlink areas of the image to the different rooms/galleries in each floor.

The map floor also shows hot to cover the route around the floor. If the visitor taps on the map “hot spots”, a new screen will display with the correspondant image gallery.



Mockup 7 Map Feature

For example, if the visitor taps on the *Eminent Zakhyntians* “hotspot”, the device will display the *Image Map* shown on Mockup 4.

Another possible improvement is allow the user zoom in and zoom out the map using the *pinch zoom* gesture; with *pinch zoom*, the user place two fingers on the screen and squeeze them together to make the item you’re viewing smaller, or you pull them apart to make it bigger [1, Chapter 11].



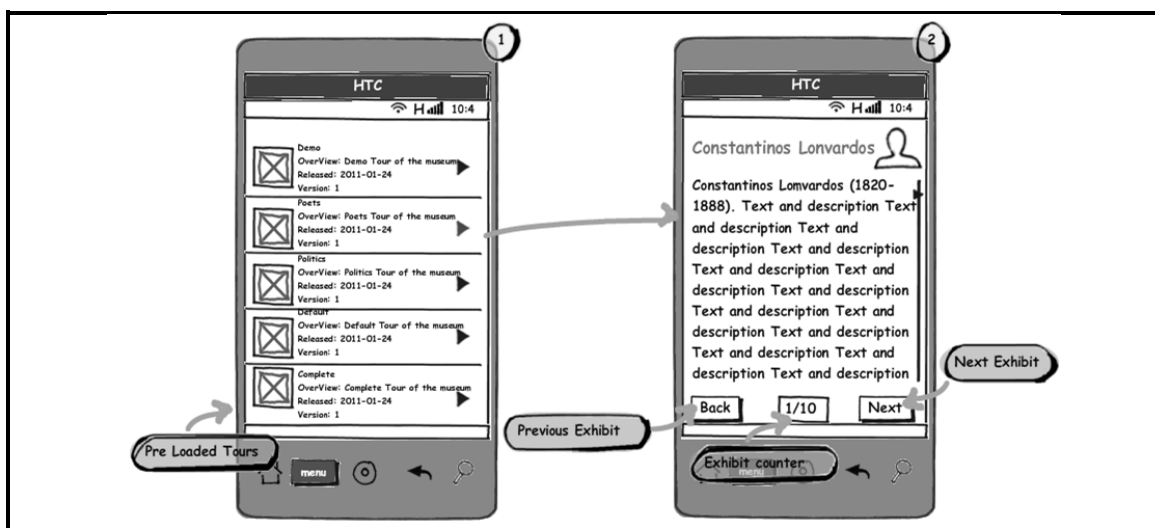
Figure 12 Pinch zoom gesture

3.4.10 TOUR FEATURE

The tour functionality offers the user some pre loaded and different kind tours of the museum; for example, if the visitor is solely interest in *Dionysios Solomos*, he/she will tap on the specific tour, and the device will automatically show all the exhibits related to the figure of *Dionysios Solomos*.

This feature would also include tours by time; if the visitor has not enough time for visit the whole museum, so the device implements “tours by time” with the most important exhibits in the museum for the required time the visitor selects.

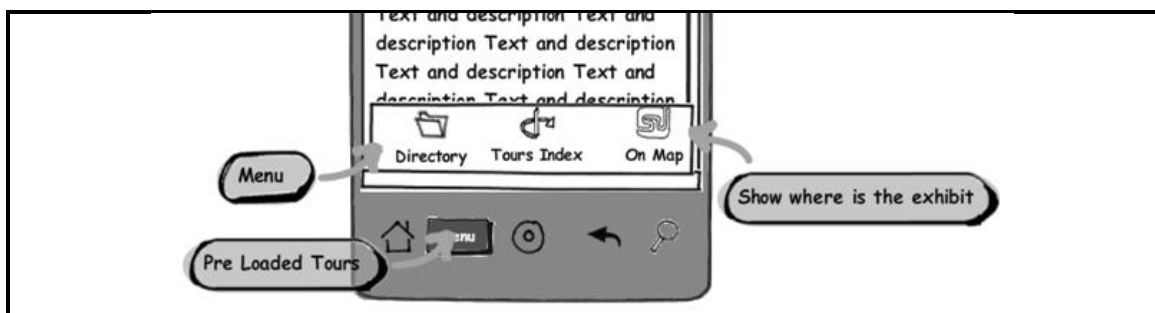
When the visitor tap on the desired *Tour* (from a `ListView`), the device will load the tour from the database of the museum and displays the first exhibit. By tapping in the “back” and “next” button, the user moves through the exhibits.



Mockup 8 Museum Tour functionality

As the tour functionality could display exhibits from different rooms, or even floors, it is necessary to provide the user an option in which he/she can easily find the location of the exhibit.

For that reason, if the user presses the physical “menu” key of the device, a menu will appear. A map option will appear, and if the user taps on it, it will display the floor where the exhibit can be found.



Mockup 9 Tour menu

3.4.11 EXHIBIT SCREEN

The *Exhibit Screen* is where all the information about the painting, portrait or document will be displayed. Without any doubt, this is where the visitor will spend more time using the guide, so it is necessary to assure the information on the display will be friendly (size of the text or images, colors, contrast, arrangement, sound, ...).

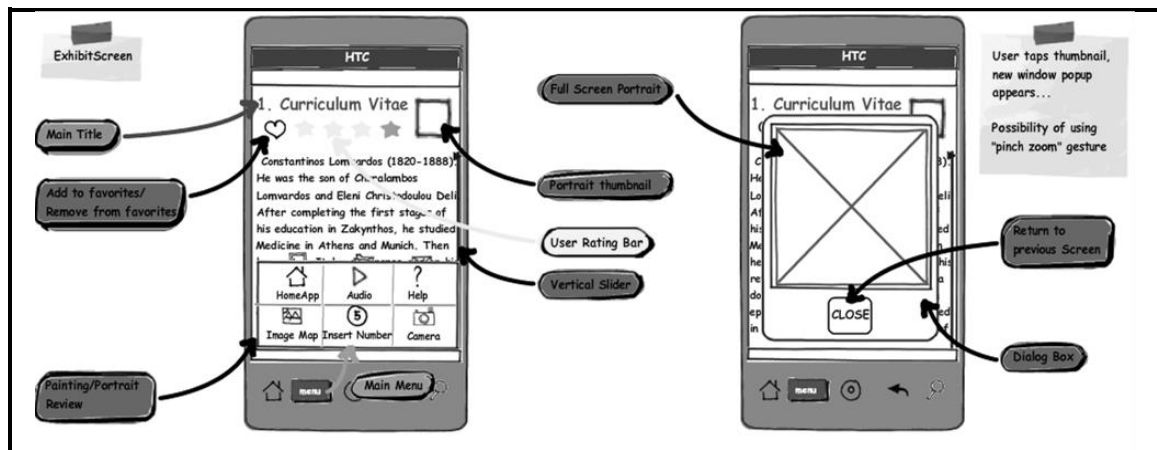
First of all and for maximizing the user experience, the exhibit will be displayed in full screen, so the system icons as the time, *wireless* connection and mobile coverage will disappear momentarily while the user is in the exhibit screen. This will allow the user to enjoy the screen device with an appropriate size for the text.

This screen consists of a title of the portrait, a small thumbnail and a description. Some features can be added like add the exhibit to favorites or rating the exhibit or rating the exhibit.

On the other hand, if the user taps the small thumbnail, a dialog box will appear with the portrait in large size or full screen. The “close” button will close the dialog box and return back to the previous screen.

It is also possible to implement the “*pinch zoom*” gesture for zoom in and zoom out the thumbnail, but the final look may not look so professional.

Finally, if the user presses the “*menu*” key on the device a new menu will appear. The options available are “*Home App*” (return back to the main menu of the application), “*Audio*” (pressing on it, an audio recorded voice will read the description of the exhibit), “*Help*”, and the three modes of retrieving information of the exhibits.



Mockup 10 Exhibit Screen

3.5 FINAL COMMENT ABOUT THE DESIGN

The design shown is subject to change at any time due to it is an initial prototype, according to the encoding difficulty and the available time.

CHAPTER 4

DEVELOPMENT & IMPLEMENTATION

4.1 INTRODUCTION

Chapter 4 focuses on the development and implementation of some of the designs shown in the previous chapter. As mentioned earlier, the application is coded with the *Google Android SDK*.

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The *Android SDK* provides the tools and *APIs* necessary to begin developing applications on the *Android* platform using the *Java* programming language [5].

The choice of *Android* as programming language for the application is based on that *Android* is designed to develop mobile applications (the main subject of this thesis). Also, *Android* is an *SDK* open source platform, which is not required to pay license fees to develop applications.

This means that this is platform which helps the developers to invest more in their time and understanding client needs. Also, the performance, stability and security are boosted as it is based on *Linux Kernel*. The *OS* is hence very smooth to operate and less chances of crashing down.

For more information about *Android*, Appendix A describes in detail all the features and functionality of this *Operating System*.

The development environment chosen for the application is *Eclipse* (a *Java* based *IDE*), which allows to fully integrate the *Android SDK*.

Appendix B describes in detail all steps to integrate *Google's Android SDK* and start developing applications in minimal time.

4.2 FEATURES IMPLEMENTED

Due to time constraints and the extent of the design proposed in the previous chapter, only some of the features described before will be implemented, leaving the other as the subject of next projects. Other features, like the *Map*, require a further study, since it is not a trivial subject.

The following sections focus on the implementation of the *Exhibit Screen*, the *Tour Functionality* and the *QR Codes* as method of interaction between the user and the device.

Because both features are closely related, *Exhibit Screen* was the first project to be developed. Second project will be implement the *Tour Functionality*, and finally integrate both projects in the same *Android* application.

4.3 EXHIBIT SCREEN

4.3.1 GENERAL REVIEW

Exhibit Screen is the screen where the device displays the description and information related to a portrait. This screen is basically:

- A main title (the title of the exhibit)
- A small thumbnail
- A description or review of the portrait
- Some buttons at the bottom of the screen which functionality is moving through the different exhibits or return back to the previous screen (activity).
- An emerging menu with some options (return back to the application main screen: *Home App*, *Audio*, *Help*, and the *QR Code* method implemented on [2]: *Camera*).

The next figure shows the implementation of the *Exhibit Screen*.



Figure 13 Exhibit Screen and menu

As the reader can appreciate, the exhibit screen does not finally display the content on the whole device screen.

If the user taps on the *small thumbnail* just to the right of the main title, a new dialog screen will appear with a larger portrait image as show in the next figure. For close it, the user should tap on the close button.

The *Help* menu option performs the same function as the user taps on the thumbnail.

Figure 12 shows the portrait in larger size:

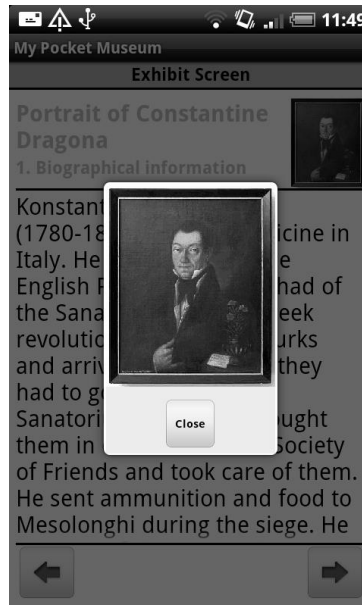


Figure 14 Full Portrait Screen

Figure 13 shows the help menu button dialog.

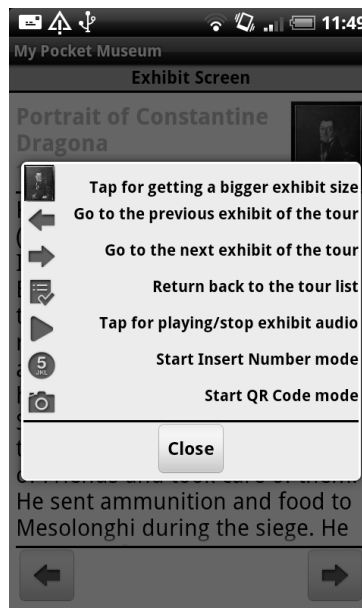


Figure 15 Help menu option

The *Camera* options is the same as shown in [2] except for a slight modification in the *Camera* option; if the user scans one code that not corresponds to any portrait, the application will not do anything and it will go back to the activity where it was.

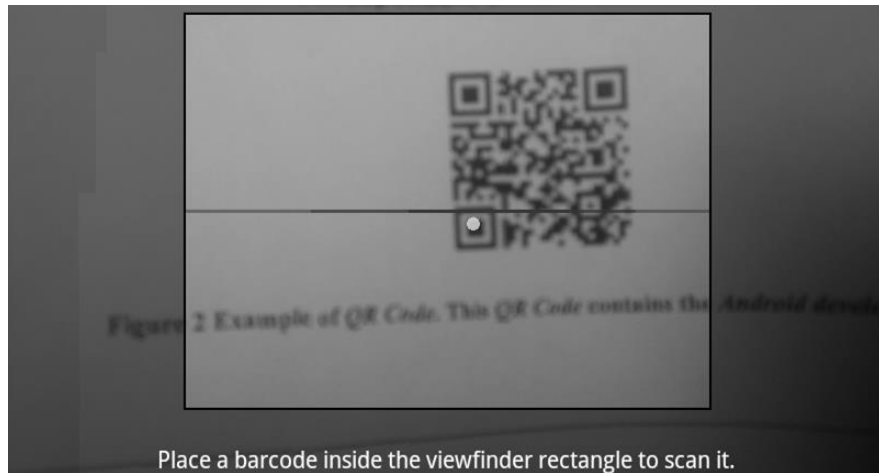


Figure 16 Checking QR Code

The next figure shows the portrait related to the *QR Code* scanned. The button on the bottom of the exhibit screen will drive the user to the previous exhibit screen. If the user scans a *QR Code* that does not correspond to any exhibit, the application will not display anything.

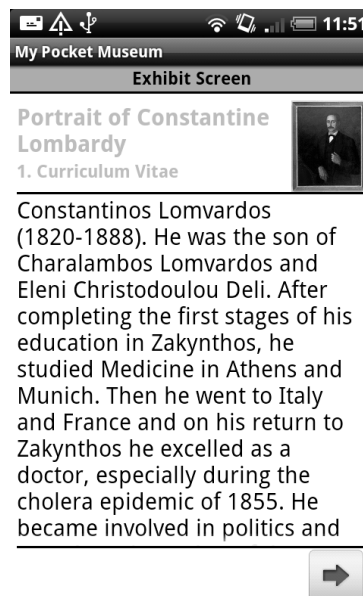


Figure 17 Exhibit Screen after scan a QR Code

Finally, the *Home App* option menu will show the main screen of the application, in this case the *List Tour* screen.

When the user taps the *Audio* option, the application will play the description related to the current portrait. Tapping again on the button, it will stop. If the user taps again on audio, it will resume the audio just the point it was stopped.

In [2], the audio record is played from a raw resource included in the application, but it will be more efficient to play it from web *URL* or a filesystem, due to the developers could update the descriptions and the audio without the user had to do anything.

4.3.2 FULL SCREEN

For a better experience while using the application, the user would like to have an appropriate font and image size. Taking advantage of the screen device for displaying only the application would increase user satisfaction. Although this feature is not implemented finally in the application, it is not very difficult to implement it.

The next Listing shows how to make the application (*Activity*) fullscreen:

```
...
requestWindowFeature(Window.FEATURE_NO_TITLE);

getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
    WindowManager.LayoutParams.FLAG_FULLSCREEN);
...
```

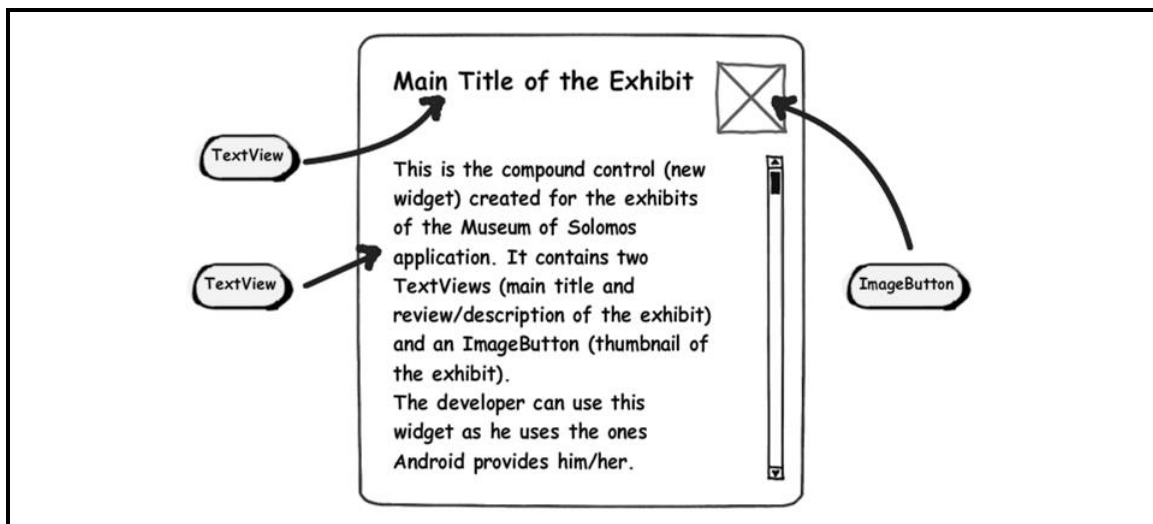
Listing 1 Full Screen Activity

4.3.3 COMPOUND CONTROL

It is clear that for all exhibits the structure will be the same: a **main title**, a **small thumbnail** and a **description**.

A *widget* in *Android* is an *UI* element that the developer can use in his/her application screen (for example a button, a listview, ...). *Android* offers a sophisticated and powerful componentized model for building an *UI*, based on the fundamental *layout* classes: *View* and *ViewGroup* [5].

The next *mockup* shows the new *widget*:



Mockup 11 Exhibit Compound Control

If none of the prebuilt *widgets* or layouts meets the developer needs, *Android* allows the developer to create his/her own *widgets* or compound controls.

On the other hand, it is possible that the developer only needs to make small adjustments to an existing *widget* or *layout*, therefore he/she can simply subclass the *widget* or *layout* and override its methods.

Compound controls are atomic, reusable `Views` that contain multiple child controls laid out and wired together. When the developer creates a *compound control* he/she defines the *layout*, appearance, and interaction of the `Views` it contains. The developer creates *compound controls* by extending a `ViewGroup` (usually a *layout*).

So it would be an interesting idea implement a new widget that contains a **main title**, a **small thumbnail** and a **review**. This new *widget* is a combination of two `TextView` (The **main title** and the **review**) and an `ImageButton` (the **small thumbnail**).

To create a new *compound control*, the developer may choose the *layout* class that's most suitable for positioning the child controls, and extend it as shown in the next listing:

```
public class ExhibitView extends LinearLayout {
    public ExhibitView(Context context, AttributeSet attrs) {
        super(context, attrs);
        ...
    }
}
```

Listing 2 Creating a new Compound Control

As with *Activities*, the preferred way to design *compound view layouts* is using an external resource instead of inside the code (but it is also possible).

The next Listing shows the *XML layout* definition the *compound control* consisting of the two `TextView` and the `ImageButton`. Attributes are hidden for brevity:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    ...>
    <TextView
        ...>
    </TextView>
    <ImageButton
        ...>
    </ImageButton>
    <ScrollView
        ...>
        <TextView
            ...>
        </TextView>
    </ScrollView>
</RelativeLayout>
```

Listing 3 Example of layout for the new View

To use the *layout* defined in the previous Listing for the new `View`, the developer may override the `View`'s constructor to inflate the *layout* resource using the `inflate` method from the `LayoutInflater` system service. The `inflate` method takes the *layout* resource and returns the inflated `View`.

For circumstances such as this, in which the returned `View` should be the class the developer is creating, he/she can pass in the parent `View` and attach the result to it automatically, as shown in the next Listing.

```
// Get a reference to the LayoutInflater Service
String infService = Context.LAYOUT_INFLATER_SERVICE;
LayoutInflater li = (LayoutInflater) getSystemService(infService);

// Inflate the view from the layout resource
li.inflate(R.layout.exhibit_view, this, true);

// Get references to the child controls
titleTextView = (TextView) findViewById(R.id.title);
contentTextView = (TextView) findViewById(R.id.review);
smallImage = (ImageButton) findViewById(R.id.thumbnail);
...
```

Listing 4 Inflating the new Compound Control

4.3.4 AUDIO

The *Android* platform offers built-in encoding and decoding for a variety of common media types. This means that the developer can easily integrate audio, video, and images into the applications. Accessing the platform's media capabilities is fairly straightforward the developer does so using the same intents and activities mechanism that the rest of *Android* uses.

Android lets the developer plays audio and video from several types of data sources. In [2], the user can play an audio file from media files stored in the application's resources (`raw` resources), but also it is possible to play this kind of media from standalone files in the filesystem, or from a data stream arriving over a network connection as in this application.

To play audio or video from an application, the developer may use the `MediaPlayer` class [5].

It would be interesting to play the audio files (or video in future versions) from a *web URL* or *database*; this will allow to improve and update the contents of the audio and the video without any user effort.

These are the steps the developer may follow:

- Create an instance of the `MediaPlayer` using `new`
- Call `setDataSource()` with a `String` containing the path (local filesystem or *URL*) to the file
- First `prepare()` then `start()` on the instance:

```

MediaPlayer mp = new MediaPlayer();
mp.setDataSource(PATH_TO_FILE);
mp.prepare();
mp.start();

```

Listing 5 Some MediaPlayer methods

The `stop()` and `pause()` methods work the same as discussed above.

4.3.5 WORKING WITH XML ON ANDROID

The *Android* platform is an open source mobile development platform. It gives the developer access to all aspects of the mobile device that it runs on, from low level graphics, to hardware like the camera on a phone. Consequently the developer can create powerful mobile applications.

With so many things possible using *Android*, developers might wonder why they need to bother with *XML*. It is not that working with *XML* is so interesting; it is working with the things that it enables.

XML is commonly used as a data format on the *internet*. If the developer wants access data from the *internet*, chances are that the data will be in the form of *XML*. If the developer wants to send data to a *web* service, they might also need to send *XML*. In short, if an *Android* application will leverage the *internet*, then developers will probably need to work with *XML*. Luckily, *Android* has a lot of options available for working with *XML*.

For example if each exhibit is encoded in an *XML* file the application could use the data of the *XML* and displays it on the screen device.

It is possible to encode each exhibit in individual files with different tags according to the *id*, *title*, *description*... read each file from the *internet* and finally show the information.

4.3.5.1 XML

XML is a markup language for documents containing structured information. Structured information contains every kind of content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

A markup language is a mechanism to identify structures in a document. The *XML* specification defines a standard way to add markup to documents.

XML specifies neither semantics nor a *tag* set. In fact *XML* is really a meta language for describing markup languages. In other words, *XML* provides a facility to define *tags* and the structural relationships between them. Since there's no predefined *tag* set, there can not be any preconceived semantics. All of the semantics of an *XML* document will either be defined by the applications that process them or by stylesheets.

In order to appreciate *XML*, it is important to understand why it was created. *XML* was created so that richly structured documents could be used over the *web*.

XML is not a replacement for *HTML*. In fact, *XML* and *HTML* were designed with different goals:

- *XML* was designed to transport and store data, with focus on what data is
- *HTML* was designed to display data, with focus on how data looks

HTML is about displaying information, while *XML* is about carrying information. *HTML* comes bound with a set of semantics and does not provide arbitrary structure.

The following example is a note to *Android*, from *Java*, stored as *XML*:

```

<note>
<to>Android</to>
<from>Java</from>
<heading>Reminder</heading>
<body>Don't forget the Eclipse!</body>
</note>
```

Listing 6 Example of XML file

The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this *XML* document does not do anything. It is just information wrapped in *tags*. Someone must write a piece of software to send, receive or display it.

The *tags* in the example above (like `<to>` and `<from>`) are not defined in any *XML* standard. These tags are "invented" by the author of the *XML* document. That is because the *XML* language has no predefined tags.

The tags used in *HTML* are predefined. *HTML* documents can only use *tags* defined in the *HTML* standard (like `<p>`, `<h1>`, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

It is important to understand that *XML* is not a replacement for *HTML*. In most *web* applications, *XML* is used to transport data, while *HTML* is used to format and display the data.

The best description of *XML* is that is a software and hardware independent tool for carrying information.

4.3.5.2 XML PARSERS

One of the greatest strengths of the *Android* platform is that it leverages the *Java* programming language. The *Android SDK* does not quite offer everything available to the standard *Java Runtime Environment (JRE)*, but it supports a very significant fraction of it.

Since *XML* tags are strictly hierarchical, and *XML* document can be mapped to a tree. *XML* parsing is the process of producing an *xml* parse tree from an *XML* document. As stated earlier, parsing is one of two low-level, basic operations one often performs on an *XML* document; the other is *XSL* transformation. But, in fact, transformation includes a parsing, which is normally hidden.

The *Java* platform has supported many different ways to work with *XML* for quite some time, and most of *Java*'s *XML* related *API*'s are fully supported on *Android*. For example, *Java*'s Simple *API* for *XML* (*SAX*) and the *Document Object Model* (*DOM*) are both available on *Android*. Both of these *API*'s have been part of *Java* technology for many years.

There are some differences between the two methods mentioned above: *XML* parsing based on the Simple *API* for *XML* (*SAX*) results in a virtual tree. On the other hand, *XML* parsing based on the *Document Object Model* (*DOM*) standard results in a literal tree.

SAX parsers provide access to a virtual parse tree through callbacks. As a *SAX* parser encounters the various elements that comprise a document, it calls methods defined in an interface and supplied by the user's program to process the data associated with the elements. Thus, it works incrementally and without having to have the entire document in memory.

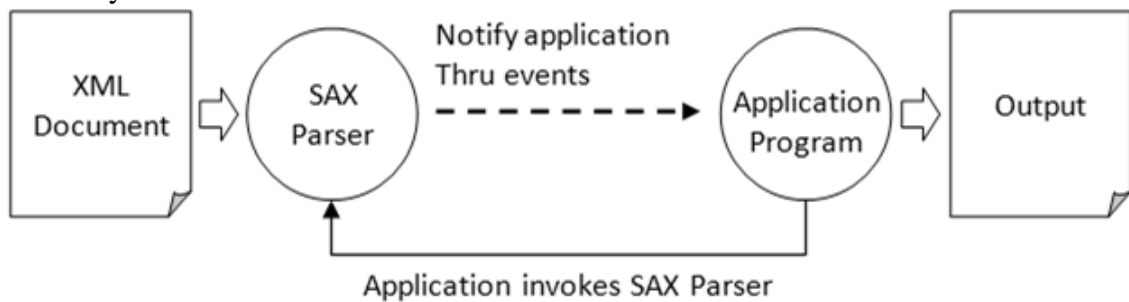


Figure 18 SAX parser operation

By contrast, a *DOM* parser requires that the entire document be read in before parsing can take place, and a complete parse tree is produced, regardless of the size of the document. Thus, both document and parse tree must reside in main memory. The *DOM* parser explicitly builds an object model, in the form of a tree structure, to represent an *XML* document. The application can then manipulate the nodes in the tree. *DOM* is a platform and language independent interface for processing *XML* documents. The *DOM API* defines the mechanism for querying, traversing and manipulating the object model built.

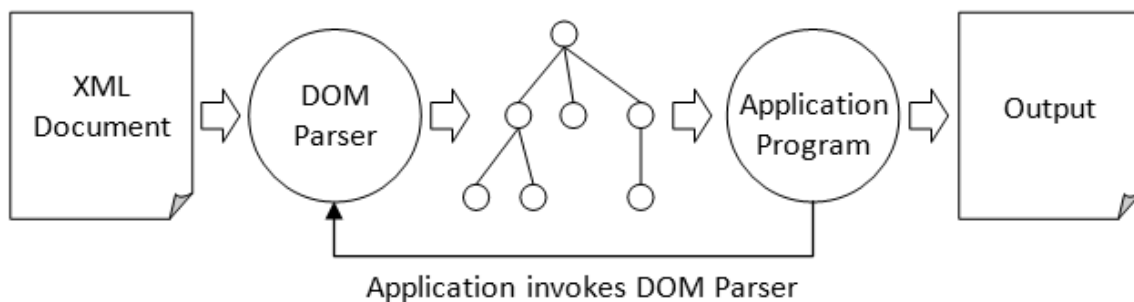


Figure 19 DOM parser operation

SAX parsers are generally faster than *DOM* parsers. There are other kinds of parsers as newer *Streaming API for XML (StAX)* that is not available in *Android*. However, *Android* provides a functionally equivalent library. Finally, the *Java XML Binding API* is also not available in *Android*.

In the following section, it is explained how the application take a source of *XML* available on the *Internet*, and then how to parse it within an *Android* application.

4.3.5.3 USING SAX

SAX is an *event driven API*. The *SAX API* defines a number of callback methods, which will be called when events occur during parsing. The *SAX parser* reads an *XML* document and generates events as it finds elements, attributes, or data in the document. There are events for document start, document end, element start *tags*, element end *tags*, attributes, text context, entities, processing instructions, comments and others.

In a *Java* environment, developers can often use the *SAX API* when they want a fast parser and want to minimize the memory footprint of the application. That makes it very well suited for a mobile device running *Android*. The developer can use the *SAX API* as-is from the *Java* environment, with no special modifications needed to run on *Android*.

The component *Exhibit Screen* will take an *XML* file from a server and parse it into a list of simple *Java* objects. These objects can be used to back the data the *compound control* (or other widgets) needs for displaying the information and the thumbnail of the exhibit.

[1] XML file structure

In order to treat an *XML* file, it is necessary to know its structure. For brevity, it will be a simple file on the *Internet* which contents the data. Some *tags* are omitted.

```
<exhibit>
<thumbnail>http://dl.dropbox.com/u/15614066/thumbnail.png</thumbnail>
<fullportrait>http://dl.dropbox.com/u/15614066/full_portrait.png</fullportrait>
<description>
  <maintitle>1. Curriculum Vitae</maintitle>
  <review>Constantinos Lomvardos.....</review>
</description>
</exhibit>
```

Listing 7 XML File

The data read from the *XML* will be shown on the the screen device using the *compound control* previously defined.

[2] POJO and Data Structures

In the next Listing, the Exhibit class is the classical *Plain Old Java Object (POJO)* that represents the data structure of an exhibit.

```
public class Exhibit {
    private String title;
    private String content;
    private String thumbnailUrl;
    private String fullportraitUrl;
    private Bitmap thumbnail;
    private Bitmap fullportrait;

    public Exhibit(String title, String content, String
thumbnailUrl,String fullportraitUrl) {
        this.title = title;
        this.content = content;
        this.thumbnailUrl = thumbnailUrl;
        this.fullportraitUrl = fullportraitUrl;
    }
    // getters and setters omitted for brevity
    ...
}
```

Listing 8 The Exhibit POJO

[3] Read the data and display it

To read the data from a *XML* file a *XMLReader* and a *ContentHandler* are needed. The *XMLReader*, as its name suggests, reads the data from the *XML* file and the *ContentHandler* “tells” the *XMLReader* how to treat that data.

The steps to follow are:

- Call the parser

```
anExhibit = parser.getExhibitData("someId");
```

Listing 9 Parse the data of the exhibit

- Open the *URL* that is hosting the file

```
String URL_Test = "http://dl.dropbox.com/u/" + testExhibitId;
URL url = new URL(URL_Test);
```

Listing 10 URL in which is stored the XML file

- Create the *SAXParser*

```
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
```

Listing 11 Creating a new instance of a SAXParser

- Add the *ContentHandler* to the *SAXParser*

```
XMLReader xr = sp.getXMLReader();
XMLHandler myXMLHandler = new XMLHandler();
xr.setContentHandler(myXMLHandler);
```

Listing 12 Creating the ContentHandler and apply it to the XMLReader

The *ContentHandler* is a *SAX-Handler*. Its work is read the tags of the *XML* file; when the *SAX-Handler* finds an opening tag like:

```
<maintitle>
```

Listing 13 Start tag of one of the fields of the XML file

The next method of the *Handler* will run. This is called every time an opening tag is encountered in the *XML* document.

```
public void startElement(String namespaceURI, String localName,
String qName, Attributes atts)
throws SAXException {}
```

Listing 14 Method for starting the reading

The same occurs when the *SAX-Handler* finds an end tag:

```
</maintitle>
```

Listing 15 End tag of one of the fields of the XML file

The *Handler* will run the next method:

```
public void endElement(String namespaceURI, String localName, String
qName)
throws SAXException {}
```

Listing 16 Method for closing the reading

The method for reading the data a tag contents:

```
public void characters(char ch[], int start, int length) {}
```

Listing 17 Method for reading the data

This method calls the methods that transfer data from the *XMLReader* to the class where the application stores the data from the file, in this case, the *ParsedXMLDataSet*.

The *ParsedXMLDataSet* provides the methods for collecting the data and a method that returns the values it contains. This method is called after reading the data from the main class.

- Pass the data from The *URL*

```
xr.parse(new InputSource(url.openStream()));
ParsedXMLDataSet parsedExampleDataSet = myXMLHandler.getParsedData();

exhibit = new Exhibit(parsedExampleDataSet.toTitle(),
parsedExampleDataSet.toReview(), parsedExampleDataSet.toThumbnailURL(),
parsedExampleDataSet.toFullportraitURL());

exhibit.setThumbnail(getBitmapFromUrl(parsedExampleDataSet
.getThumbnailURL()));

exhibit.setFullportrait(getBitmapFromUrl(parsedExampleDataSet
.getFullportraitURL()));

return exhibit;
```

Listing 18 Treating the data from the URL

As the reader can see in the Listing 16, it is also possible to parse images (in this case the thumbnail). The next Listing shows the method used for that purpose

```
private Bitmap getBitmapFromUrl(String url) {
    try {
        InputStream is = (InputStream) this.fetch(url);
        BitmapDrawable b = (BitmapDrawable) BitmapDrawable
            .createFromStream(is, "src");
        return b.getBitmap();
    } catch (MalformedURLException e) {
        e.printStackTrace();
        Log.d(MY_DEBUG_TAG, "ImageOperations malformed url",
e);
        return null;
    } catch (IOException e) {
        e.printStackTrace();
        Log.i(MY_DEBUG_TAG, "IO Exception -- Problem fetching
image", e);
        return null;
    }
}

private Object fetch(String address) throws MalformedURLException,
    IOException {
    URL url = new URL(address);
    Object content = url.getContent();
    return content;
}
```

Listing 19 Parsing Images on Android

- Display the data

```
myExhibitView.setExhibit(anExhibit);
```

Listing 20 Return the data

```
public void setExhibit(Exhibit anExhibit) {
    titleTextView.setText(anExhibit.getTitle());
    contentTextView.setText(anExhibit.getContent());
    smallImage.setImageBitmap(anExhibit.getThumbnail());
}
```

Listing 21 Setting the result to be displayed in the GUI

4.4 TOUR FUNCTIONALITY

4.4.1 GENERAL REVIEW

Tour Functionality is a new feature that provides two screens.

The first screen is a list of several tours from which the user may select one. This first screen is basically:

- A small icon representing the tour
- The name of the tour
- A small overview of the tour
- The duration and the distance of the tour
- The number of exhibits of the tour

The next figure shows the implementation of the first screen. This screen will be the main screen of the application.



Figure 20 List of Tours

The user can move through the list tour by dragging on the screen device and tapping on the desired tour or using the optical trackball of the device (if available).

Each tour contains more or less exhibits, depending on the exhibits available on the museum and also the available time. In instance, the *Unorganized Tour* may contain eight exhibits and the *ITLab Tour* only six, or *Complete Tour* may contain all the exhibits of the museum.

If the user presses the physical menu key an emerging menu will be displayed and it will offer the user the *Camera* option.

Once the user has selected one tour from available in the list, a new screen will be shown on the device.

This screen will display the name of the tour, some information about it, and finally a list of the exhibits it contains on the top of the screen.

Under that, the user can see the list of exhibits of that tour. Each exhibit displays the name of the portrait, its location, the information about that portrait and also the description of the portrait just in case the user wants to read it.



Figure 21 List of Exhibits

As before, if the user presses the “menu” key, the *Camera* option for scanning the *QR Codes* will be available.

Finally, when the user taps on one of the portraits, the last screen is displayed on the device. This screen is basically the Exhibit Screen shown in 4.3.

The exhibit screen is basically:

- The *Exhibit Screen* (title of the exhibit, the small thumbnail and the description or review of the portrait)
- Two Buttons to move through the selected tour exhibits
- An emerging menu with three: *Home App*, *Audio*, *Tour Index* and *Map*.

All the exhibits data on the selected tour is loaded once the user selects the tour.

The next figure shows the implementation of the exhibit screen:

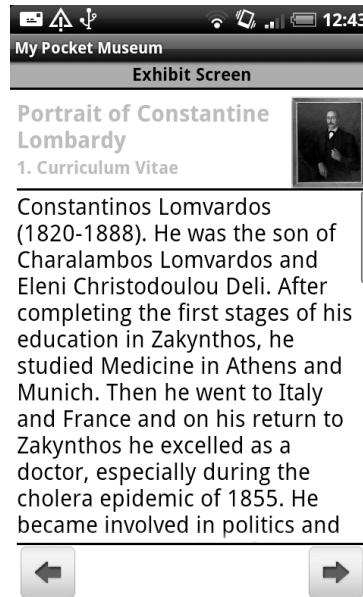


Figure 22 Tour Exhibit Screen

When the user finishes with the first exhibit of the tour, he may press the arrow buttons and he/she will drive to the previous/next exhibit. When the tour is complete, the arrow buttons will change its icon that will drive the user to the *Exhibit List* automatically.

If the user presses the menu key on the device, four options will be displayed: if the user taps on *Home App* the application will show the *Main Screen* of the application. *Home App* will go back to the first screen of the application: the *Tour List*; *Audio* will play the current portrait audio, *Help* will display some useful help for the user and *Camera* will start the *Bar Code Scanner* application for scanning *QR Codes*.



Figure 23 Exhibit screen menu options

4.4.2 CREATING THE TOUR LIST (AND THE EXHIBIT LIST)

The `ListView` [5] widget *Android* provides the developer is a good way for creating the *Tour List* and the *Exhibit List*. `ListView` is used to show a list of items in a vertically scrolling list. So, for the purpose of the *Tour List* and the *Exhibit List* the developer can use `ListView` to display the list of tours or exhibits.

The design of a default `ListView` is the first step. This is a very simple *XML* which will add a `ListView`.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  ...>
  <ListView
    android:id="@+id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
</LinearLayout>

```

Listing 22 Layout for the ListView

A `ListView`, by default, only allows the developer to add lines of text. Note that if the number of items is higher than the size of the screen device, `ListView` alone is the responsible for making a scroll, so it is not necessary to create a `Scroller`, because the control is automatic.

To add the icon a new layout has to be created as show in the next figure:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="?android:attr/listPreferredItemHeight">
  <ImageView
    ... />
  <LinearLayout
    ...
  >
    <TextView
      ...
    />
    <TextView
      ...
    />
    <TextView
      ...
    />
    <TextView
      ...
    />
  </LinearLayout>
</LinearLayout>

```

Listing 23 Layout for the Tour List/Exhibit List

The next step after the necessary `Layouts` are created is writing the code. In a new class file in which each line in the `ListView` contains an element of this class.

```

public class Tour {

    public String tourName;
    public String tourId;
    public String overview;
    public String released;
    public String version;
    public List<Exhibit> exhibits;
    // getters and setters omitted for brevity
    ...
}

```

Listing 24 Elements of the ListView

Like in the *Exhibit Screen*, the *Tour List* and the *Exhibit List* retrieve the data from the internet, specifically from an *XML* file.

So the next steps are the following:

- Parse the data
- Create a *Customized* `ArrayAdapter` for displaying the elements on the `ListView`.

As forward both screens use the same kind of parser, the paper may focus only on the *Tour List*. The *Exhibit List* was exactly the same work.

4.4.2 PARSING THE DATA

Before the *Tour Functionality* shows the available tours, the application must load the data from the *internet* (the icon, name, overview, duration, distance of the tour and the number of exhibits).

For that purpose, it is necessary to parse again the data. In this case, it is possible to use a simpler implementation based on the *SAX Parser* used for the *Exhibit Screen*.

The previous model, though fully operational and quite efficient, has a few disadvantages. On the one hand it is necessary to define a separate class for the *handler*. Additionally, the *SAX* model implies the need to put enough emphasis on defining the *handler*, due to the *SAX* events are not linked in any way with the tags in the *XML* document.

To avoid these problems, *Android* offers a variant of *SAX* that avoids defining a separate class for the *handler* and allows actions to directly associate specific tags within the *XML* document structure, which greatly alleviates the drawbacks mentioned.

The parse implementation will need to take a *URL* and use this to open an *HTTP* connection to the database. This common behavior is naturally modeled in *Java* code using an abstract base class as in the Listing 23.

The base class stores the `feedUrl` and uses it to open a `java.io.InputStream`. If anything goes wrong, it simply throws a `RuntimeException`, so that the application simply fails quickly. The base class also defines some simple constants for the names of the tags of the *XML* file.

```
public abstract class BaseFeedParser implements FeedParser {
    // XML Tags

    //TourList
    static final String TOURLIST_TOURS = "tourListTours";
    static final String TOURLIST_TOUR = "tourListTour";
    static final String TOURLIST_TOURNAME = "tourListName";
    static final String TOURLIST_TOURID = "tourListId";
    static final String TOURLIST_OVERVIEW = "tourListOverview";
    static final String TOURLIST_RELEASED = "tourListReleased";
    static final String TOURLIST_VERSION = "tourListVersion";
}
```



```

private final URL feedUrl;
protected BaseFeedParser(String feedUrl) {
    try {
        this.feedUrl = new URL(feedUrl);
    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    }
}

protected InputStream getInputStream() {
    try {
        return feedUrl.openConnection().getInputStream();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
}

```

Listing 25 Base feed parser

Next listing shows some sample content from the *Tours XML* file, so the reader can see the significance of these tags:

```

<tourList>

  <tourListTours>

    <tourListTour>
      <tourListThumbnail>http://dl.dropbox.com/u/15614066/
images/Demo%20Tour.png</tourListThumbnail>
      <tourListName>Demo</tourListName>
      <tourListId>0</tourListId>
      <tourListOverview>Demo Tour of the museum</tourListOverview>
      <tourListReleased>2011-01-24</tourListReleased>
      <tourListVersion>1</tourListVersion>
      <tourListLast_modified_at>2011-01-24</tourListLast_modified_at>
    </tourListTour>
    ...
  </tourListTours>

</tourList>

```

Listing 26 Tour List XML file

As the reader can see from the Listing above, a `tourListTour` corresponds to a `Tour` instance. The child nodes of item (`tourListName`, `tourListOverview`, and so on) correspond to the properties of the `Tour` instance.

The new *SAX* parsing code does not use a *SAX handler*. Instead it uses classes from the `android.sax` package in the *SDK*. These allow the developer to model the structure of the *XML* document and add an event listener as needed. In the beneath code, the developer may declare that his/her document will have a root element called `tourList` and that this element will have a child element called `tourListTours`. The `tourListTours` will have a child element called `tourListTour` and then it is possible to start to attach listeners. For each listener, it is required to use an anonymous inner class that implemented the interface that the developer was interested in (either `EndElementListener` or `EndTextElementListener`).

Notice there was no need to keep track of character data. Not only is this simpler, but it is actually more efficient. Finally, when the `Xml.parse` utility method is invoked, the developer now passes in a handler that is generated from the `root` element.

```
public class AndroidSaxFeedParser extends BaseFeedParser {

    static final String    TOURLIST    = "tourList";

    public AndroidSaxFeedParser(String feedUrl) {
        super(feedUrl);
    }
    public List<Tour> parseTourList() {
        final Tour currentTour = new Tour();
        RootElement root = new RootElement(TOURLIST);
        final List<Tour> tours = new ArrayList<Tour>();
        Element elementTours = root.getChild(TOURLIST_TOURS);
        Element elementTour = elementTours.getChild(TOURLIST_TOUR);
        elementTour.setEndElementListener(new EndElementListener()
{
            public void end() {
                tours.add(currentTour.copy());
            }
        });

        elementTour.getChild(TOURLIST_TOURNAME).setEndElementListene
r(new EndTextElementListener() {
            public void end(String body) {
                currentTour.setTourName(body);
            }
        });

        // More elementTour.getChild here. Ommited for brevity

        try {
            Xml.parse(this.getInputStream(), Xml.Encoding.UTF_8,
root.getContentHandler());
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return tours;
    }
}
```

Listing 27 Simplified Android SAX Parser

This kind of parser was used also for the *List of Exhibits Screen* as well as for the *Exhibit Screen* due to its simplicity.

4.4.3 CREATING A CUSTOM ADAPTER

Adapters are simple devices. On one side of the adapter is a data structure like a *Java* object storing data. `SimpleAdapter` handles *Java* objects that can be meaningfully translated into *Strings* by invoking the objects `toString()` method (every *Java* object supports that but for quite many of them, the `toString()` format is not meaningful for the end user).

On the other side of the adapter, there is a `View` that the data structure was transformed into. That `View` is displayed to the user. As it is possible to use *Adapters* to supports a

ListView, the *Adapter* handles lists of *Java* objects (that are eventually transformed into a list of Views).

Android's built in adapters are sufficiently versatile but it is often handy to create a custom adapter.

The `LayoutInflater` is get from the calling context, what this does is that it will inflate the *XML* to codes via `View` object. See `getView` function.

On `getView` function the `Layout` is inflated. At this point the `ListView` is ready. Finally, it is only necessary get the items from the *XML Layout* and returns the values.

The next Listing shows an example of the `CustomAdapter` developed for displaying the information and the icon on the `ListView` of the application.

```
public class TourArrayAdapter extends ArrayAdapter<Tour> {
    ...
    public View getView(int position, View convertView, ViewGroup parent)
    {
        View row = convertView;
        if (row == null) {
            // ROW INFLATION
            // Log.d(tag, "Starting XML Row Inflation ... ");
            LayoutInflater inflater = (LayoutInflater) this.getContext()

                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            row = inflater.inflate(R.layout.tours_data_row,
parent, false);
            // Log.d(tag, "Successfully completed XML Row Inflation!");
        }
        // Get item
        Tour tour = getItem(position);
        tourIcon = (ImageView) row.findViewById(R.id.tour_thumb_icon);
        tourType = (TextView) row.findViewById(R.id.type_text_view);
        tourOverview = (TextView) row.findViewById(R.id.overview_text_view);
        tourReleased = (TextView) row.findViewById(R.id.released_text_view);
        tourVersion = (TextView) row.findViewById(R.id.version_text_view);

        tourType.setText(tour.tourName);
        tourOverview.setText("Overview: " + tour.overview);
        tourReleased.setText("Released: " + tour.released);
        tourVersion.setText("Version: " + tour.version);
        String iconUrl=URL_DIR+tourType.getText().toString()+EXT;

        try {
            InputStream in = new java.net.URL(iconUrl).openStream();
            Bitmap bitmap = BitmapFactory.decodeStream(in);
            tourIcon.setImageBitmap(bitmap);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return row;
    }
}
```

Listing 28 Tour List Custom Adapter

CHAPTER 5

EVALUATION, RESULTS AND CONCLUSIONS

The main goal of this chapter will be to test the application implemented in the previous chapter in a real case study.

For that purpose, this chapter will present an experiment in a real environment in which the application developed could be tested.

5.1 EVALUATION OF THE APPLICATION

The next step after the application has been developed is to evaluate it. For that purpose, some experiments and people are required for testing the program. This is one of the most important steps in the moment of designing and implementing an application: verify that the results obtained have been the suitable ones. Therefore, it is necessary a real environment, some people who test the application and did not have contact with it before. This will provide an objective opinion about the project from the exterior and to realize the lacks and deficiencies the application has.

5.1.1 SCENARIO: A VIRTUAL MUSEUM

The scenario chosen for the study of the application was originally “*The Museum of Science and Technology*” at the University of Patras. Due to the museum could not provide the information about some of the collections, the scenario of the application changed to a custom “Virtual Museum” in the first floor of the *Department of Electrical and Computer Engineering*, where is located the *HCI Group*. The aim of the evaluation is to test the functionality of a museum guide which orients the user through some exhibits using a handheld device and the *wireless* technology.

The main advantage of the application is its dynamic contents, because all the information is stored on the *internet*, thus it also means that the application requires permanent connection to the *wireless* network.

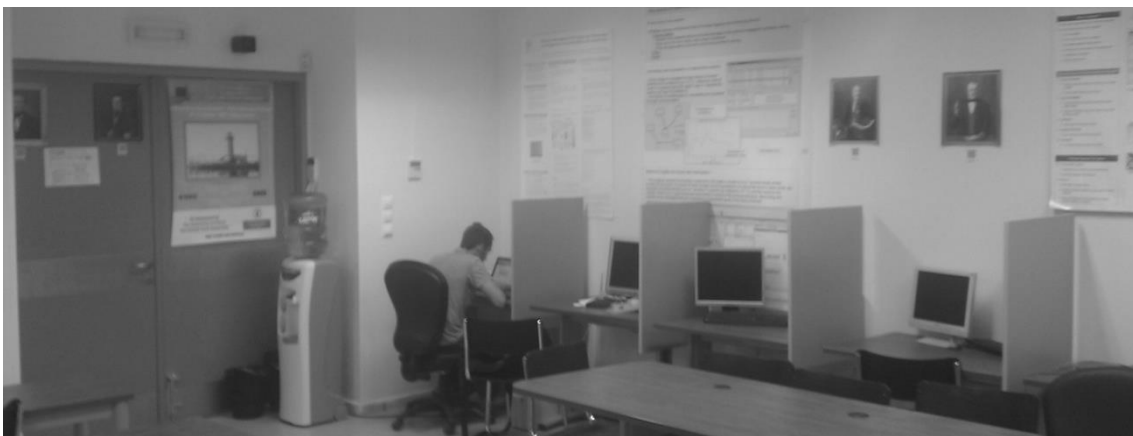


Figure 24 Exhibits in the HCI Laboratory Group

5.1.2 ACTIVITIES

The activities related to the application will take part on the first floor of the *Eastern Building* of the *Electrical and Computer Engineering Department*. In that floor, there are fourteen portraits distributed in three areas: the *HCI Group Lab*, the *corridor* of the floor and the *Mobile Group Office*.

In this context, the visitors will be given an *Android* device (the *HTC® Desire®*) with the application to experiment with it and evaluate how it operates. The device is the same used for the experiment of the analysis in the second chapter of this paper.

In the first activity, the users may fill a small demographic survey and after that some instructions will be given for showing them how to use the system and the application. With this guide, users will learn how to use the device and the application properly.

After that, the users will try and test the application according to the previous instructions: they may interact with the exhibits following a tour, or trying the *QR Codes* for interacting with them. This means that they will follow some procedure.

Finally, the users may fill a last survey about the application. The answers, replies and feedbacks of this survey will be very useful for analyze the behaviour of the application and show the final results.

5.1.3 USABILITY TESTINGS

Usability testing is a technique used to evaluate an application by testing it on users. This can be seen as an irreplaceable usability practice, since it gives direct input on how real users and testers use a system. This is in contrast with usability inspection methods where experts use different methods to evaluate a user interface without involving users.

Usability testing focuses on measuring a human made product's capacity to meet its intended purpose. Examples of products that commonly benefit from usability testing are *web* sites or *web* applications, computer interfaces, documents, and devices among others. Usability testing measures the usability, or ease of use, of a specific object or set of objects, whereas general human computer interaction studies attempt to formulate universal principles.

The usability testings are a “*black box*” testing technique. The aim is to observe people using the product to discover errors and areas of improvement. Usability testing generally involves measuring how well test subjects respond in four areas: efficiency, accuracy, recall, and emotional response. The results of the first test can be treated as a baseline or control measurement; all subsequent tests can then be compared to the baseline to indicate improvement.

- *Performance*: How much time, and how many steps, are required for people to complete basic tasks? (For example, find an exhibit, complete a tour...)
- *Accuracy*: How many mistakes did people make? (and were they fatal or recoverable with the right information?)

- *Recall*: How much does the person remember about the application afterwards or after periods of non use?
- *Emotional response*: How does the person feel about the tasks completed? Is the person confident, stressed? Would the user recommend this system to a friend?

It is important to point out that simply gathering opinions on an object or document is market research rather than usability testing. Usability testing usually involves systematic observation under controlled conditions to determine how well people can use the product.

Rather than showing users a rough draft and asking if the users understand the product, usability testing involves watching people trying to use something for its intended purpose.

The method used for setting up a usability test involves carefully creating a scenario, or realistic situation, wherein the person performs a list of tasks using the product being tested while an observer watch and take notes. Several other test instruments such as scripted instructions, paper prototypes, and pre/post-test questionnaires are also used to gather feedback on the product being tested.

For example, to test the application in this paper, a scenario would describe a situation where a person needs to make a tour through some exhibits in different areas of a museum and he is asked for that task.

The aim is to observe how people function in a realistic manner, so that developers can see problem areas, and what people like. Some techniques popularly used to gather data during a usability test include *think aloud protocol*, *Co-discovery Learning* and *eye tracking*.

5.1.4 HOW MANY USERS TO TEST?

Jakob Nielsen popularized the concept of using numerous small usability tests, typically with only five test subjects each, at various stages of the development process.

The claim of "*Five users is enough*" was later described by a mathematical model which states for the proportion of uncovered problems U :

$$U = 1 - (1 - p)^n$$

where p is the probability of one subject identifying a specific problem and n the number of subjects (or test sessions). This model shows up as an asymptotic graph towards the number of real existing problems.

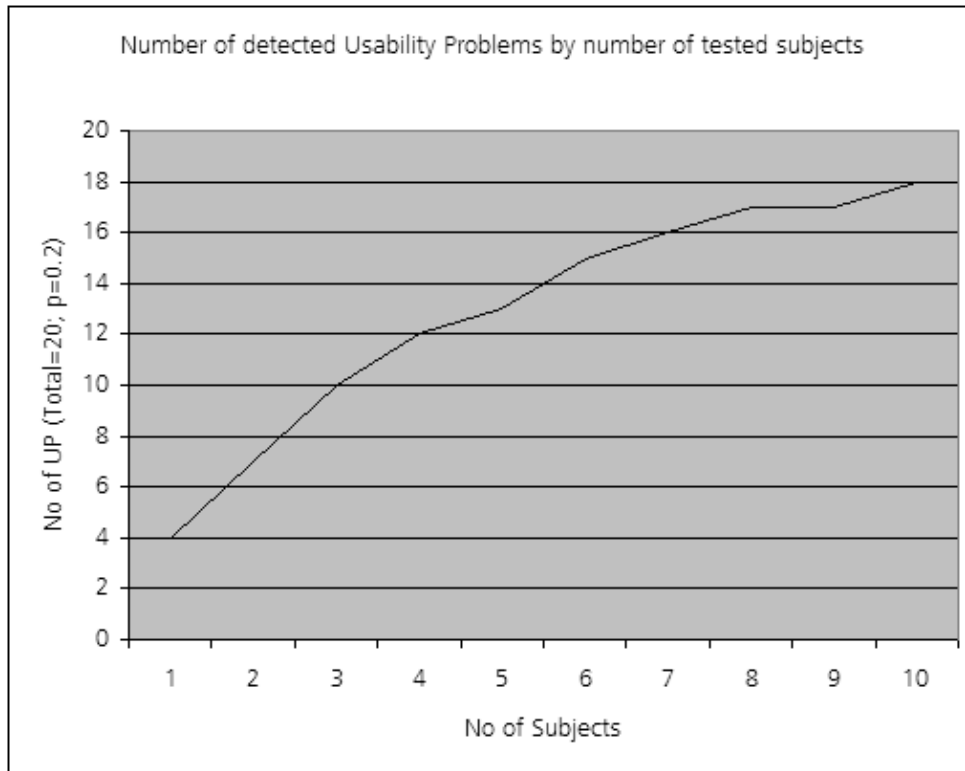


Figure 25 Virzi's Formula

For this paper, eight users will test the application showing that from five users onwards the results will be the same.

5.1.5 SURVEYS

Surveys and questionnaires are one of the most commonly used research methods. Surveys are frequently used to describe populations, to explain behaviours, to know about the use of some widgets...Surveys can be structured, well-tested, robust and result in data with high level validity, however they can be poorly done, resulting in data of questionable validity. The strength of the survey is the ability to get a large number of responses.

5.1.6 METHOD

The method of evaluation is simple; user may follow the next steps:

1. Fill a demographics survey
2. Read the procedure
3. Contact with the device
4. Try the application according to the procedure and the instructions
5. Fill a final survey

The method, tasks, surveys and the procedure could be found at the end of the paper.

5.2 APPLICATION RESULTS

The experiment took part on 24th March 2011 in the Eastern Building of the *Electrical and Computer Engineering Department* of the University of Patras. For that, eight people tested the application: some of them were students and others from the *HCI Group*.

Once they completed the method on the point above, it will be possible to analyze the results about the experiment.

5.2.1 DEMOGRAPHICS SURVEY

The demographics survey shows the level of knowledge the users have about some different topics: languages they are fluent in, knowledge about museums and knowledge about technology and environment used.

DEMOGRAPHICS QUESTIONS

Six of the participants are *Erasmus* students of the *University of Patras*. The field of study of three of them is *telecommunication* and *network engineering*, another one *aeronautical engineering*, one *journalism* and the last one *primary education*. Other of the participants is a student of the University Of Patras and collaborator in the *HCI Laboratory* and his field is *computer engineering*. The last participant is a PhD candidate and her field of study is *Human Computer Interaction* and *Collaborative Technologies*. There are five participants from Spain, two from Greece and one from Slovenia. Seven of the participants are males and only one is female.

The kind of the studies and the demographic questions will show the level of knowledge of the users have about *Android* and the technologies implemented in the application. The age range of the participants is between 21 and 32.

LANGUAGE

About the language the participants are fluent in, all of them are at least fluent in english (as well as their native language). One participant is also fluent in greek language (not his native language), another one in spanish language (not his native language) and another one in serbian and croatian languages.

MUSEUM

All of the participants in the experiment have been sometime in a real museum and they have also had contact with some kind of guide system (such a tour guide, tape machine or *CD Player*) while being there. The familiarity with this kind of guide will help them using the application later.

TECHNOLOGY

Six of the participants fell very comfortable using a mobile device, while one of them feels somewhat comfortable and the last one does not feel very comfortable using this

kind of devices. This shows that for most of the participants, the mobile device is something daily and familiar.

Besides from sending or receiving calls and messages, the participants use their mobile devices for other purposes and tasks: five of them use it for taking pictures, four of them for listening to music, two of them for *internet* browsing, three of them for playing games, one of them for checking his *e-mail* account and one of them for keeping track of calendar events. This proves that nowadays, people use their mobile devices not only for calls and messages.

The half of the participants has interacted sometime with an electronic guide in a museum, so for that people will be easier to use the new application due to they are more familiar to it.

With regard to the *Android* platform, seven of eight participants have heard about it. Six of these participants had contact with an *Android* device before this testing.

Finally, six of the participants have heard about what a *QR Code* is, but only two of them have used that mode of interaction technology.

This shows that for the participants these technologies are known, but on the other hand they do not know how it works and operates, or what could be its use.

5.2.2 APPLICATION SURVEY

This point shows the results of the questionnaires the users filled after reading the procedure and trying the application.

DEVICE AND PROCEDURES

For all the participants it was easy to become familiar with the device, even for those who do not feel comfortable with mobile devices. Therefore, if they feel comfortable with the device, accept it and they know how to use it properly, they will handle with the application easily.

For all the participants it was easy to follow the steps of the procedure and they understand what the main goal of the application is.

LANGUAGES

Six of eight participants would like the application implements other languages besides english. They say that not everyone who visits a museum is fluent only in english, so it would be a nice idea to offer the users other language choices. Participants who answered they want additional languages would like to implement french, german, greek and spanish. They also argue that if available, they will use their native language while running the application.

The other two participants think that english language would be enough, but also they said that at least it would be a nice to implement two or three major languages.

TOURS

All the participants think that the tour functionality was useful while trying the application because they have some kind of guide for visiting the exhibits they want to.

For five of the participants the information provided with each tour is good and for the other three is enough.

About a help button in the tour list screen, half of the participants think this button will be useful but the other half thinks it will be useless. This could be explained in accordance how comfortable the users feel with the interface and the application

All the participants agreed that the *Camera* option in the tour list screen and exhibits list screen is useful. They say that if they want just check some exhibits and avoid the load time of the application, this will save them a valuable time.

About the information displayed about the exhibits in the exhibit list screen, five of the participants think that information is good and the other three think it is enough.

As regards how easily was to find out the place of each exhibit in the scenario, six of the participants think it was easy, the others two had some troubles. This is an interesting point because the proposed scenario is only one floor with three different “areas”, not far among them. So it would be good to test the application in a larger museum.

According to the previous question all the participants, even it was easy for some of them to find out the place of each exhibit, think that a map functionality will be very useful, even if that functionality would be simple, because they argue it will help the people to find easily the exhibits.

EXHIBITIS

As for the exhibit screen, five of eight participants think the content provided about the exhibit was enough and the other three think it was good.

Seven of eight participants agree with the method implemented for moving through the exhibits, while one thinks it was difficult. One possible explanation is that this person also found difficult to find out the place of each exhibit.

For four participants, get a larger portrait on the device when they tap on the thumbnail was useful. The rest of participants think this feature is not useful, because they argue if they want to see the portrait in larger size they can look directly to the original one hanging on the wall.

Six of eight people consider that it will not be useful to include other media options besides the audio. They argue that maybe too much information could be boring for the visitor. This contrasts with the results of the questionnaire of the survey in the chapter 2. Generally, people prefer multimedia contents instead of text.

On the other hand, the people who think it will be useful say that the multimedia content is always more appealing than a text decription.

AUDIO

For testing the audio, some of the participants were provided with headphones and others tested the application without headphones. Apart from that, all the devices were given to the users with a low volume.

For three of all participants the audio was enough. They say that it would be nice if the audio was different from the text description. They also think that if it would be possible, while the audio is playing they would synchronize it with the text.

The volume of the audio was good for four people, enough for one and low for three. Participants who answered the sound was good used headphones or they increase the volume with the volume keys on the edge of the device. After checking the device of the people who answered the sound was low, some of them did not use the volume keys, so eventually they did not have a complete knowledge about the device. It is true that some of them did not have headphones, and maybe the audio record is not so good, so audio may be improved for sensitive persons. As for the quality of the sound, some participants complained about a background noise.

For seven of eight people it was easy to understand the audio. The participant who answered no said that in a real museum, with maybe a lot of visitors, crowded and with a lot of noise it would be difficult to listening to the audio. Every participant thinks in a real museum they will use headphones because the audio will not be good if there is a lot of noise or if in the museum is not allowed to speak loudly.

Finally, the half of the participants thinks the application may continue playing an audio track of one exhibit while moving through others would be better than stop the audio.

HELP

For all the participants the help button in the exhibit screen was useful. This could be explained due to the large number of options available in that screen.

MODES OF INTERACTION

As the main goal of this application is not study the human computer modes of interaction as in [2], it only implemented the *QR Codes* method. Only one of the participants did not have contact before with the *QR Codes*.

Over a mark of five about the intuitiveness of the *QR Codes*, the average mark among the participants was 4.125 and about the interest of them it was 4.5 showing that the *QR Codes* was a good interaction method choice. Six of all participants don't think it would be necessary to implement other interaction modes and two say that it would be nice because if the *QR Codes* may not available they think another "backup" interaction mode could be the solution. Some of the participants also noticed that the *Bar Code Scanner* application takes one second or two seconds to process the *QR Code* and decode it.

Some of the participants also asked about the *Augmented Reality* and *Image Recognition* interaction modes.

INTERFACE

Seven of eight participants agree with the font and image size, while one disagrees. This participant argues that it would be useful to implement some option for zoom in or zoom out the text and the images.

On the other hand five of eight participants would like the screen mode of the device changes from portrait mode to landscape mode. Five of them also agree with run the application in full screen mode, without the icons tray on the top of the device showing that the users prefer take advantage of the whole screen of the device for running the application.

For all the participants, the buttons of the application are necessary. When they were asked about the “multi touch gestures” most of them replied that they do not often use them or that they do not know about them. When asked them what kind of gestures they knew, some of them answered the “*tap*” gesture and the “*drag*” gesture (the basic ones).

About if the appearance and user interface of the application was friendly and/or pleasing five of the participants agreed completely and the rest agreed partially showing that the participants are satisfied with the interface, colors, images, backgrounds...

APPLICATION

As explained before, the application requires an *internet* connection in order to work. All the information about the tours, contents and exhibits has to be downloaded from the server, so while running the application the users have to wait some time until all the information is displayed on the device.

For two participants, the loading time is not acceptable. They argue that for example in case they want to make the complete tour of a museum they have to wait for at least one minute till all the information of the screen is shown. They also said that if they have to wait a long time visitors could be bored and they will reject the use of the application, but they also understand that the loading time also depends on the quality and strength of the *wireless* signal. Anyway, a better design in the *XML* files and the quality of images could improve the loading time.

All the participants agrees with the application handles dynamic contents, instead a static content. Six of the participants would also like to include a map functionality or other kind of orientation for the application: one of the participants suggested that in the exhibit list it will be useful if the user taps on the image related to the tour a map emerges with the position of each exhibit and the order of each exhibit. One of the participants would also like the application implements some games.

When the participants were asked if the application would be useful in a museum they all agreed. They also agreed when they were asked about if they will use it in a real museum. Finally, five of the participants have used some application like this before this test in some real museums or various relevant studies.

Next, there are the tables with the answers of the participants.

DEMOGRAPHICS SURVEY ANSWERS

	QUESTION/PARTICIPANT	1	2	3	4	5	6	7	8
1	<i>COUNTRY OF ORIGIN</i>	Spain	Spain	Spain	Spain	Spain	Greece	Greece	Slovenia
2	<i>AGE</i>	28	23	21	25	25	32	25	25
3	<i>GENDER</i>	Male	Male	Male	Male	Male	Female	Male	Male
4	<i>OCCUPATION</i>	Student	Student	Student	Student	Student	PHD Candidate	Student	Student
5	<i>EDUCATION/FIELD OF STUDY</i>	Telecommunication Engineering	Telecommunication Engineering	Aeronautical Engineering	Journalism	Telecommunication Engineering	Human Computer Interaction – Collaborative Technologies	Computer Engineering	Primary Education
6	<i>NATIVE LANGUAGE</i>	Spanish	Spanish	Spanish	Spanish	Spanish	Greek	Greek	Slovenian
7	<i>FLUENT IN</i>	English	English	English	English Greek	English	Spanish	English	English Croatian Serbian
8	<i>HAVE EVER BEEN IN A MUSEUM</i>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
9	<i>HAVE USED SOME KIND OF GUIDE SYSTEM IN A MUSEUM</i>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
10	<i>FEELINGS USING A MOBILE DEVICE</i>	Very comfortable	Very comfortable	Very comfortable	Somewhat comfortable	Very comfortable	Not very comfortable	Very comfortable	Very comfortable
11	<i>USE OF MOBILE PHONE FOR (ASIDE CALLS/MESSAGES)</i>	Taking photos	Listening to music Internet browsing	e-mail	Taking photos Playing games Listening to music	Taking photos Internet browsing	Playing games	Taking photos Playing games Listening to music Calendar events	Taking photos Listening to music
12	<i>HAVE EVER INTERACTED WITH SOME KIND OF ELECTRONIC GUIDE IN A MUSEUM</i>	Yes	No	No	No	Yes	Yes	Yes	No
13	<i>KNOWLEDGE ABOUT THE ANDROID PLATFORM</i>	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
14	<i>HAVE CONTACT WITH SOME ANDROID DEVICE BEFORE</i>	Yes	No	Yes	No	Yes	Yes	Yes	Yes
15	<i>HAVE EVER HEARD ABOUT QR CODES TECHNOLOGY</i>	Yes	Yes	Yes	No	Yes	Yes	No	Yes
16	<i>HAVE USED THIS TECHNOLOGY</i>	No	No	Yes	No	No	No	No	Yes

APPLICATION QUESTIONNAIRE ANSWERS

	QUESTION/PARTICIPANT	1	2	3	4	5	6	7	8
1	IT WAS EASY TO BECOME FAMILIAR WITH THE DEVICE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
2	IT WAS EASY TO FOLLOW THE STEPS OF THE PROCEDURE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3	IMPLEMENT OTHER LANGUAGES	Yes	Yes	Yes	No	Yes	Yes	No	Yes
4	LANGUAGES IF AVAILABLE	Spanish	Spanish	Spanish Greek	-	French Spanish	Greek	-	Slovenian
5	IT WAS USEFUL THE TOURS FUNCTIONALITY	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
6	PROVIDED CONTENT WITH THE INFORMATION OF THE AVAILABLE TOURS WAS	Enough	Good	Good	Good	Good	Enough	Enough	Good
7	A BUTTON HELP IN THIS SCREEN WOULD BE HELPFUL	Yes	Yes	No	Yes	No	No	No	Yes
8	IT WAS USEFUL THE CAMERA OPTION IN THE TOUR LIST SCREEN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
9	THE INFORMATION ABOUT THESE EXHIBITS WAS	Enough	Enough	Good	Good	Good	Enough	Good	Good
10	IT WAS EASY TO FIND OUT THE PLACE OF EACH EXHIBIT	No	Yes	Yes	Yes	Yes	Yes	Yes	No
11	IT WILL BE USEFUL TO IMPLEMENT SOME MAP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
12	THE CONTENT WITH THE INFO OF THE EXHIBIT WAS	Enough	Enough	Good	Good	Good	Enough	Enough	Enough
13	IT WAS EASY TO MOVE THROUGH THE EXHIBITS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
14	THE LARGER PROTRAIT WAS USEFUL	Yes	No	No	Yes	Yes	No	No	Yes
15	INCLUDE SOME OTHER MEDIA	No	No	Yes	Yes	No	No	No	No
16	THE AUDIO WAS EASY TO UNDERSTAND	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

DESIGN AND DEVELOPMENT OF A MOBILE GUIDE FOR MUSEUM BASED ON THE ANDROID PLATFORM

17	THE CONTENT OF THE AUDIO WAS	Enough	Enough	Good	Good	Good	Enough	Good	Good
18	THE AUDIO VOLUME WAS	Low	Low	Good	Good	Good	Low	Good	Enough
19	IT WOULD BE BETTER KEEP THE AUDIO OF ONE CONCRETE EXHIBIT WHILE MOVING THROUGH THE OTHERS	Yes	Yes	Yes	No	No	No	No	Yes
20	IT WAS USEFUL THE INFORMATION ABOUT THE EXHIBITS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
21	INTUITIVENESS OF QR CODES	3	5	5	5	4	2	5	4
22	INTEREST OF THE QR CODES	3	5	5	5	4	4	5	5
23	CONTACT WITH QR CODES	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
24	IMPLEMENT OTHER MODES OF INTERACTION	No	No	No	No	No	Yes	No	Yes
25	THE FONTS AND IMAGES SIZE IS ENOUGH	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
26	CHANGE THE SCREEN MODE OF THE DEVICE FROM PORTRAIT TO LANDSCAPE	No	No	Yes	Yes	Yes	Yes	Yes	No
27	FULL SCREEN FOR THE APP	Yes	No	Yes	Yes	Yes	No	No	Yes
28	THE BUTTONS IN APP ARE	Necessary	Necessary	Necessary	Necessary	Necessary	Necessary	Necessary	Necessary
29	THE APPEARANCE AND THE UI OF THE APPLICATION WAS PLEASING/FRIENDLY	Agree partially	Agree completeley	Agree completely	Agree partially	Agree completely	Agree partially	Agree completely	Agree completely
30	LOADING TIME IS ACCEPTABLE	Yes	Yes	No	Yes	Yes	No	Yes	Yes
31	INCLUDE SOME NEW PERFORMANCES	Map	Map	Map	Map	-	Map Mini Games	Map	-
32	THE GUIDE WILL BE USEFUL FOR A REAL MUSEUM	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
33	WILL YOU USE IT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
34	HAVE EVER USED SOME APPLICATION LIKE THIS	Yes	No	No	No	Yes	Yes	Yes	Yes

5.3 CONCLUSIONS

After the work presented in this paper and broadly speaking, the users who tried the application were satisfied, so the application works as it was expected; an electronic museum guide based on the *Android* platform which guides and provides its users information about the collections and exhibits of a museum.

It is obvious that the application does not implement a lot of options, but the main objective of the application was the creation of a simple, robust and friendly application with dynamic content which could be often updated without release a new version of the program.

All the users agreed with the operation of the application: select a tour, displays the exhibits of such tour and then begin from the first exhibit (or another the user wants) and interact with it through the available options: audio, help and *QR Codes*.

It is clear that for a big museum this application may be insufficient, but for the original museum of the application "*The Museum of Science and Technology*" at the University of Patras could be enough: it is not too big (only one floor), circular shaped, it contains different kind of collections that could be implemented as different tours, but of course a map of the museum with the exact place of the exhibits or collections would be very useful as well as some steps to following the tour. Providing the users with custom tours or audio tours would be another performance.

It is necessary to point out that the only responsibility of the employees in the museum is providing the information about the exhibits to the responsables of the program for updating the contents: new tours, new exhibits, news descriptions or new audio.

One important point is that one related to the loading time of the application; some users complain about this time, saying that it is a bit long and they do not want wait. This is not a problem of the application, but of course some methods could be implemented for improving the performance of this task, like for example, save the data on the device, and only when some contents changes, load the new data. Of course, the task of every museum should be to assure users a good *wireless* connection.

About the interaction mode chosen for the application, most of the participants were pleased with the use of *QR Codes* as method of interaction with the exhibits, although some of them did not know how to use it the first time, but once they have learnt it, it was very easy for them and it has a good acceptance between the users.

As future work, the application could be improved if it implements some new features like a "random tour" (each time the user tap on this tour, differents exhibits will be loaded). Another possible feature is providing the user events in the museum. Due to the fact that the museum collections or exhibits could change, a calendar track about the exhibits or special events would be useful.

Finally, it will be interesting some study about the implementation of new interaction methods; this was not the aim of the thesis, but some participants showed interest on the *Augmented Reality* and the *Image Recognition*.

REFERENCES

- [1] Ed Burnette. *Hello, Android Introducing Google's Mobile Development Platform*. Third Edition. The Pragmatic Bookshelf, July 2010.
- [2] María Teresa Manzano Rodríguez. *Design and Development of a Mobile Application Based on Android for Physical-Digital Interaction*. University of Patras, July 2010.
- [3] Reto Meier. *Professional Android 2 Application Development*. Wiley Publishing, 2010.
- [4] Rick Rogers, John Lombardo, Zigurd Mednieks and Blake Meike. *Programming with the Google SDK, Android Application Development*. O'Reilly, May 2009.
- [5] Android Open Source Project. *The Developer's Guide*. <http://developer.android.com/guide/index.html>. March 2011.
- [6] Zxing Project Hosting on Google Code. *Barcode Scanner*. <http://code.google.com/p/zxing/>
- [7] The state of the Mobile. The 2011 Museum & Mobile Survey <http://www.museums-mobile.org/survey/>. January 2011.
- [8] *Android Development Community & Android Tutorials*. <http://www.anddev.org/>. March 2011.
- [9] *Stackoverflow*, a collaboratively edited question and answer site for professional and enthusiast programmers. <http://stackoverflow.com/>. March 2011.
- [10] *Android-Spa*. <http://www.android-spa.com/>. March 2011.

GLOSSARY

AAC:	Advanced Audio Coding
ADT:	Android Development Tool
AMOLED:	Active Matrix Organic Light Emitting Diode
APP:	Application
API:	Application Programming Interface
CD:	Compact Disc
CPU:	Computer Processing Unit
DOM:	Document Object Model
FPS:	Frames Per Second
GNU:	GNU's Not UNIX
GPS:	Global Positioning System
GUI:	Graphical User Interface
HCI:	Human Computer Interaction
HSDPA:	High Speed Downlink Packet Access
HTC:	High Tech Computer Corporation
IDE:	Integrated Development Environment
IEEE:	Institute of Electrical and Electronics Engineers
JDK:	Java Development Kit
JDT:	Java Development Kit
JVM:	Java Virtual Machine
LED:	Light Emitting Diode
MP3:	Mpeg-2 Audio Layer-III
MPEG:	Movie Picture Experts Group
OHA:	Open Handset Alliance
PC:	Personal Computer
PDA:	Personal Digital Assistant
PNG:	Portable Network Graphics
RFID	Radio Frequency Identification
RTLS:	Real Time Locating System
POSIX:	Portable Operating System Interface for UNIX
QR:	Quick Response
RAM:	Random Access Memory
RFID:	Radio Frequency Identification
SAX:	Simple Api for XML
SD:	Secure Digital Card
SDK:	Software Developers Kit
SSL:	Secure Socket Layer
SQL:	Structured Query Language
STAX:	STreaming API for XML

UI:	User Interface
URL:	Universal Resource Locator
USB:	Universal Serial Bus
VM:	Virtual Machine
WAV:	WAVeform Audio File Format
WMA:	Windows Media Audio
WMV:	Windows Media Video
WST:	Web Standard Tools
WVGA:	Wide Video Graphics Array
XML:	eXtensible Markup Language
XSL:	eXtensible Style Language
ZXING:	Zebra Xing

APPENDIX A

ANDROID

A.1 INTRODUCTION

Android is a complete software platform for different sorts of devices, including mobile devices, which were used in this thesis. This means that it is an operating system and software stack to run *Android* specific software.

It was introduced by *Google Inc.* in 2007 and is aimed at changing the way mobile applications are developed. The system is based on *Linux* and runs *Java* programmed applications on top.

The big difference from normal mobile *Java* development is that *Android* supports a large part of the *Java SE*¹ through *Apache Harmony* instead of the crippled *Java ME package*. *Android* also does not use *Sun's Java Virtual Machine* or even *Java* byte code, but instead uses its own engine named '*Dalvik*' together with its own set of byte code. This means that *Android* does not run *Java* applications, but rather *Android* applications programmed in the *Java* language with *Android* extensions [3].

A.2 THE LINUX KERNEL

Android is based on the *Linux* 2.6 kernel. Just as in any other *Linux* system, the *Android* kernel provides core system functionalities mostly linked to the hardware abstraction [3].

Even though *Android* is based on a *Linux* kernel, it is not a *GNU/Linux*, which is what is normally thought of in relation to the word '*Linux*'. *Android* does not include all standard *Linux* utilities; it has no native window manager and it lacks compatibility with the standard *C/C++* libraries of the *GNU C* library (*glibc*).

Android includes its own *C* library (*libc*) called *Bionic*. *Bionic* is optimized for embedded applications, with small and fast code paths and a custom implementation of *POSIX* threads (*pthread*s). Furthermore, it has built-in support for some *Android* specific features like system properties and logging. The reason that *Bionic* is not fully compatible with that it lacks *glibc* some *POSIX* features which are not needed in *Android*. These missing features are mostly concerning *C++* libraries.

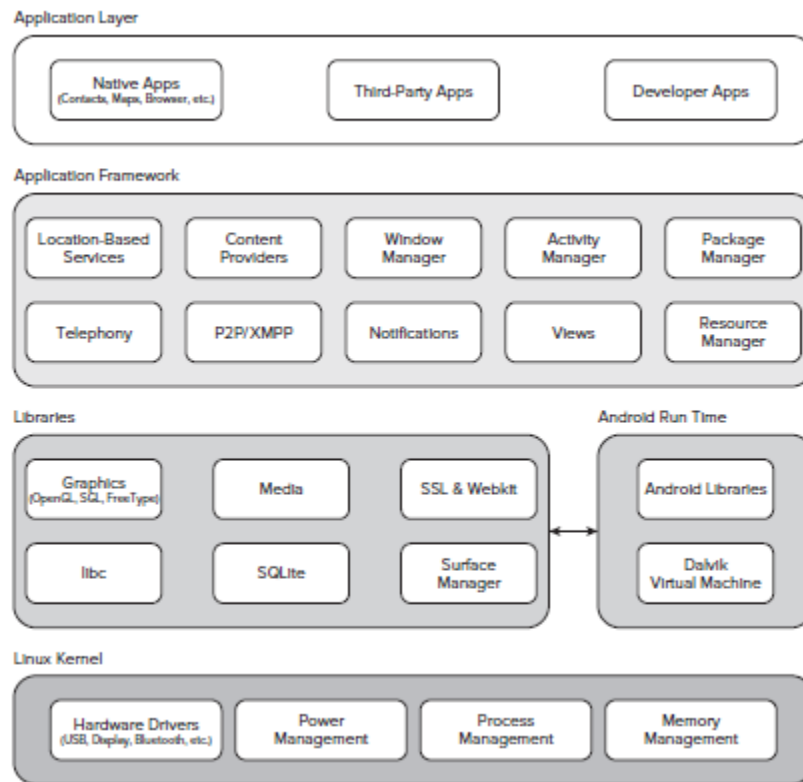


Figure 26 The Android Software Stack

A.3 THE SYSTEM LIBRARIES

System libraries (written in *C/C++*) are exposed through the application framework and provide graphics capabilities, hardware acceleration etc. and also includes a fully functional *SQL* server (*SQLite*), *SSL* libraries for security, an *OpenCORE* based media framework and *WebKit* for web browsing [3].

A.4 ANDROID RUNTIME

A.4.1 CORE LIBRARIES

Android implements a core set of standard *Java* libraries through *Apache Harmony*. This means that the *Android Java library set* differs from *Java ME* in that it is not sandboxed and stripped, but is rather 'desktop *Java* on a small screen'. It is almost fully compatible with *Java SE 1.6* except for graphical parts, but in return adds its own graphical environment and a large collection of fitting help classes and implementations to speed up *Android* development [3].

A.4.2 DALVIK VIRTUAL MACHINE

Dalvik is the *Virtual Machine (VM)* of the *Android* system. It is actually technically not a *Java Virtual Machine (JVM)* since it comes with a whole different instruction set than a *JVM* would interpret. *Android* applications are written in *Java* code, but compiled to *Dalvik* byte code.

Dalvik is a register based machine, as opposed to most *JVMs* which have a stack based structure. This structure choice will normally require larger instructions and thus often larger files, but that is not true in the case of *Dalvik* due to its trimmed down *VM*. *Dalvik* will also reduce the number of *CPU* instructions needed per *VM* instruction due to the fact that a register machine does not read from a stack and push the answer back but rather keeps addresses in instructions. This means that the *Dalvik* engine will generally be faster and load smaller files than a *JVM* would do.

It should be noted that the *Dalvik* engine will not do some optimizations such as loop unrolling in the manner that a normal *JDK/JVM* would. This means that the programmer will have to do some extra optimizations and smart design choices.

A.4.3 THE CODE VERIFIER

The local system (device) *VM* that loads the compiled *Dalvik* code (*.dex*) contains a system specific verifier. This verifier will check that the loaded code is type safe, is compatible with the device and also perform local and system specific optimizations of the byte code where possible.

A.5 APPLICATION FRAMEWORK

The application framework contains the code base for hardware interaction, the specialized design patterns and other parts of the *Android* development methodology. The main parts are

- **Activity manager** handles the activities of the application.
- **Views** are the building blocks of *UI* creation and manipulation.
- **Notification manager** provides different types of signaling alternatives to make the user and device aware of changes.
- **Content providers** provide a simple way of sharing data between applications. A normal *Android* application has its own file system which cannot be manipulated from other applications. Data channels are instead opened through content providers.
- **Resource manager** handles the resources of the application. Application resources are entities such as *UI* elements, strings and other parts that can be externalized from the code into *XML* files and fetched when needed [3].

A.6 THE STRUCTURE OF AN APPLICATION

”All applications are created equal.”

This means that there is no difference between built-in applications and third-party applications.

It is also possible to replace core applications such as the contact list, dialer or home screen. An application can be built by one or more parts. These parts are defined in *XML* and described in this section.

Each application defines its permissions in the application manifest. Typical permissions are *internet* access and camera usage. The user decides at install time if these needed permissions are acceptable. This means that the user will only agree to permission once, but it will be granted for as long as the application is installed.

A.6.1 ACTIVITY

An *activity* is a part of an application which displays a graphical interface of some sort and can retrieve input from the user. Most applications will be made of a collection of activities [1]. These activities contain *views* to display information. A very simple example of an *activity* can be seen in Listing 1.

```
import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Listing 29 Basic Activity Stub

A.6.2 SERVICE

Activities can contain services. Services are background processes which can run during a long period of time. These will most often sleep when there is nothing to be done and process information when wanted. This is somewhat like the concept of "server daemons" in *UNIX* [1]. An example of a service use case would be downloading weather forecast information for display in a widget.

A.6.3 BROADCAST RECEIVER

A *Broadcast Receiver* receives and may answer specific events. *Broadcast Receivers* must process very quickly and are used as small background services which can run when called. An example would be a *Broadcast Receiver* which receives an event which says that a photo has been taken and logs this to a specific file.

A.6.4 CONTENT PROVIDERS

Android applications do not share a file system beyond data saving on an *SD* card. Content providers are meant to serve applications with useful data of specific sorts. This data can for example be contacts in a phone book, bookmarks from a browser etc.

A.7 THE LIFE CYCLE OF AN APPLICATION

A.7.1 INTENTS

Applications are started from a system called "Intents". *Intents* define that an application is capable of handling a job. That job can be, for example "viewing a web page", "calling a phone number", "pick a photo" or "open the pod bay doors". When one application wants to perform a task that another application should implement it sends out an *Intent* [1].

If there is an application that can handle the *Intent* it will be started. If there are more than one application that can handle an intent, one will get a menu to choose from or the preferred one will be selected automatically depending on the situation.

Explicit starting of an application from for example an application menu is done by sending a specific intent to start said application.

A.7.2 KNOWING THE STATE OF AN APPLICATION

Android applications need to keep track of their state at all times. Instead of explicitly starting and stopping applications, the system is designed to be able to pause applications when they are not displayed and later possibly bring them back. This means that an application has to save its state when it gets a pause command or data might be lost if *Android* decides to shut down the application in order to save memory or battery power. During its lifetime, each activity of an *Android* program can be in one of several states, as shown in Figure 2.

If an application does not correctly pause its execution when told to, it will continue to run in background until it might be killed at any time without any further warning. This behavior makes it very simple to resume a session from the previously run since paused applications will keep their memory space until some other application claims it. It can also have the effect that applications will become useless when resumed unless first explicitly killed and later started again since clicking a "back" button or similar and starting the application again will not have the expected effect of an application restart unless the applications is designed for this behavior.

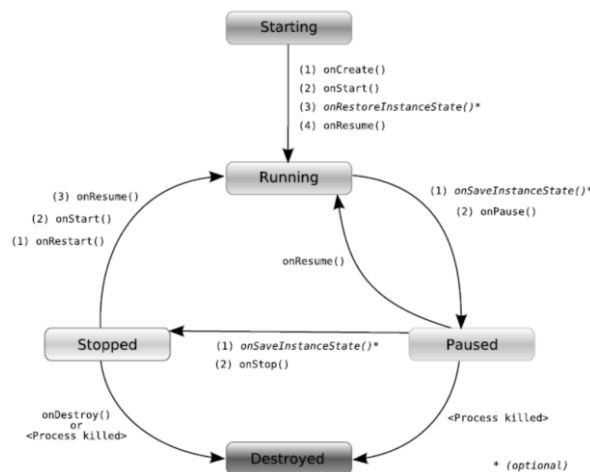


Figure 27 Life Cycle of an Android Activity

APPENDIX B

SOFTWARE DEVELOPMENT PROCESS

B.1 INTRODUCTION

A software development process is a structure imposed on the development of a software product. Synonyms include *software lifecycle* and *software process*. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

B.2 PROCESS ACTIVITIES/STEPS

Software Development Process model consists on five phases: *Requirements and Analysis, Design, Development, Implementation (or Coding), Testing (or Evaluation) and Operation and Maintenance.*

This model represents a dynamic, flexible guideline for building effective programs or other kind of software. The following sections describes briefly each phase.

B.2.1 REQUIREMENTS AND ANALYSIS

The most important task in creating a *software* product is extracting the requirements. Customers and clients typically know what they want, but not what *software* should do, while incomplete, ambiguous or contradictory requirements are recognized by skilled and experienced *software engineers*. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

The system's services, constraints and goals are established by consultation with system users. They are then defined in a manner that is understandable by both users and development staff.

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document. Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

This phase can be divided into:

- Feasibility study (often carried out separately)
- Requirements analysis
- Requirements definition
- Requirements specification

B.2.2 DESIGN

After the purpose and specifications of software are determined, software developers will design or employ designers to develop a plan for a solution. It includes low level component and algorithm implementation issues as well as the architectural view. This phase can be divided into two types: design of the system and design of the software.

- **System design:** partition the requirements to hardware or software systems. Establishes an overall system architecture. The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the program, as well as ensuring that future requirements can be addressed. The architecture step also addresses interfaces between the software system and other software products, as well as the underlying hardware or the host operating system.
- **Software design:** represent the software system functions in a form that can be transformed into one or more executable programs. Specification is the task of precisely describing the software to be written, possibly in a rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.

B.2.3 IMPLEMENTATION

Implementation (or Coding) is the part of the process where software engineers actually program the code for the project.

Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion. An important (and often overlooked) task is documenting the internal design of software for the purpose of future maintenance and enhancement. Documentation is most important for external interfaces.

B.2.4 TESTING

Testing (or Evaluation) is an integral and important part of the software development process. This part of the process ensures that defects are recognized as early as possible.

B.2.5 OPERATION AND MAINTENANCE

A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it.

People are occasionally resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, its very important to have training classes for the most enthusiastic software users (build excitement and confidence), shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software.

Users will have lots of questions and software problems which lead to the next phase of software.

Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer.

About two-thirds of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work. In comparison, about two-thirds of all civil engineering, architecture, and construction work is maintenance in a similar way.

B.3 SOFTWARE DEVELOPMENT MODELS

Several models exist to streamline the development process. Each one has its pros and cons, and it is up to the developer to adopt the most appropriate one for the project. A combination of the models may be more suitable.

B.3.1 WATERFALL MODEL

The waterfall model shows a process, where developers are to follow these phases in order:

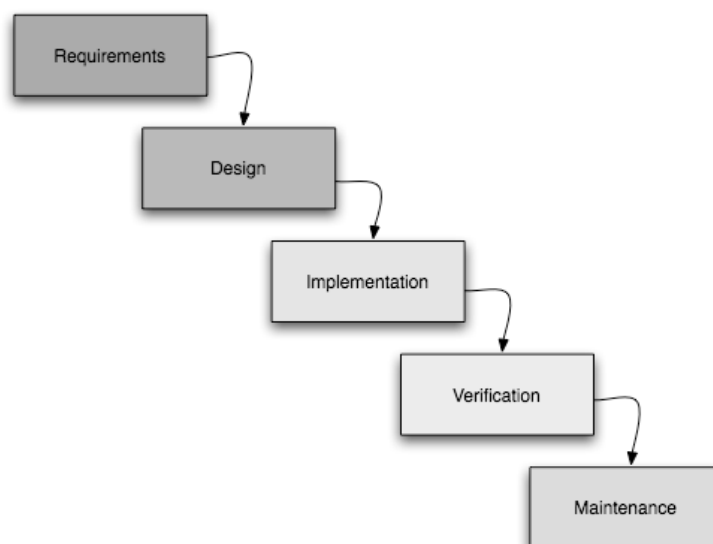


Figure 28 Waterfall model

In a strict waterfall model, after each phase is finished, it proceeds to the next one. Reviews may occur before moving to the next phase which allows for the possibility of changes (which may involve a formal change control process). Reviews may also be employed to ensure that the phase is indeed complete; the phase completion criteria are often referred to as a "gate" that the project must pass through to move to the next phase.

Waterfall discourages revisiting and revising any prior phase once it's complete. This "inflexibility" in a pure waterfall model has been a source of criticism by supporters of other more "flexible" models.

B.3.2 SPIRAL MODEL

The key characteristic of a spiral model is risk management at regular stages in the development cycle. This method combines some key aspect of the waterfall model and rapid prototyping methodologies, but provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems.

B.3.3 ITERATIVE AND INCREMENTAL DEVELOPMENT

Iterative development prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster.

Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what they want.

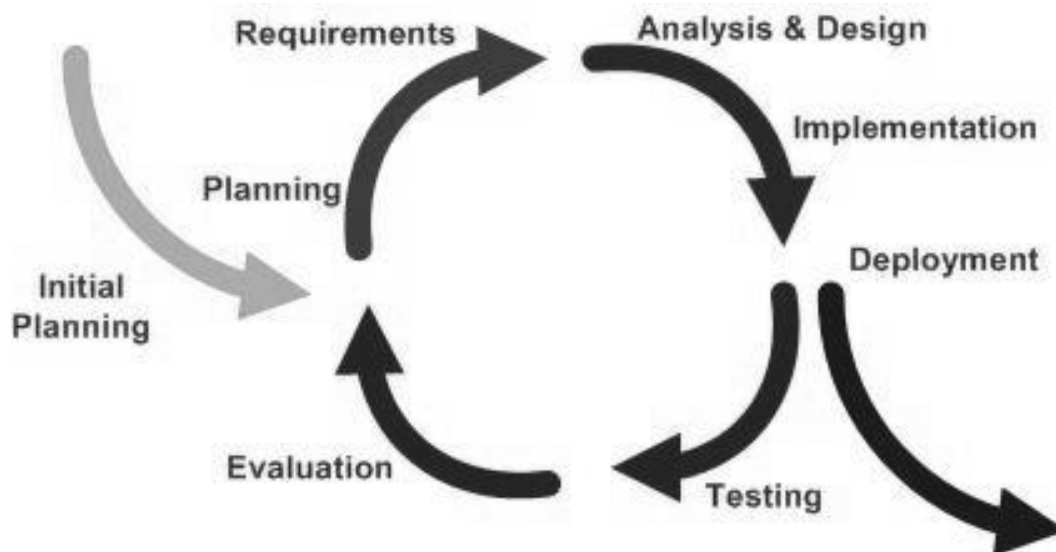


Figure 29 Iterative and incremental development

B.3.4 AGILE DEVELOPMENT

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches.

Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

APPENDIX C

SETTING UP THE DEVELOPMENT ENVIRONMENT

C.1 INTRODUCTION

Android applications, like most mobile phone applications, are developed in a host-target development environment. In other words, the application may be developed on a host computer (where resources are abundant) and download it to a target mobile phone for testing and ultimate use. Applications can be tested and debugged either on a real *Android* device or on an emulator. For most developers, using an emulator is easier for initial development and debugging, followed by final testing on real devices.

To write *Android* mobile phone applications, it is required to collect the required tools and set up an appropriate development environment on the *PC* (or *Mac*). This appendix shows how to collect those tools, download them and install them on the computer, and write a sample application that will let the developer gets the feel of writing and running *Android* applications on an emulator.

Linux, *Windows*, and *OS X* are all supported development environments. The developers will learn how to install the latest set of tools on each. After that, it is shown the configuration options after installing the tools (setting *PATH* environment variables and the like), again for each of the three operating systems.

Finally, the appendix ends with a short little “*Hello, Android*” application that demonstrates what needs to be done in order to get a generic application running.

The *Android SDK* supports several different integrated development environments (*IDEs*). This thesis focus on *Eclipse IDE* because it is the *IDE* that is best integrated with the *SDK*, freeware and also supported by *Google Inc.* However, the developer can opt for other *IDEs* like *NetBeans*, or even a simply *Notepad*. It all depends on the the developer and his/her familiarity with this.

No matter which operating system the developer is using, it will be necessary essentially the same set of tools:

- The *Eclipse IDE*
- *Sun’s Java Development Kit (JDK)*
- The *Android Software Developer’s Kit (SDK)*
- The *Android Developer Tool (ADT)*, a special *Eclipse* plug-in

Since the developers are probably going to develop on only one of the host operating systems, they can skip to the appropriate section that pertains to their selected operating system.

This guide is based on [4, Chapter 2]. More details can be found there.

C.2 CREATING AN ANDROID DEVELOPMENT ENVIRONMENT

The *Android Software Development Kit* supports *Windows (XP, Vista, 7)*, *Linux* (tested on *Ubuntu Maverick Markeet*, but any recent *Linux* distro should work), and *Mac OS X* (10.4.8 or later, *Intel* platform only) as host development environments. Installation of the *SDK* is substantially the same for any of the operating systems, and most of this description applies equally to all of them. Where the procedure differs, this guide will clearly show the developer what to do for each environment.

C.2.1 INSTALL JDK

The *Android SDK* requires *JDK* version 5 or version 6. If one of those is installed, skip to the next step. In particular, *Mac OS X* comes with the *JDK* version 5 already installed, and many *Linux* distributions include a *JDK*. If the *JDK* is not installed, the developer may go to <http://java.sun.com/javase/downloads>.

Windows (XP, Vista and 7)

- Select the distribution for “*Windows Offline Installation, Multi-language.*”
- Read, review, and accept *Sun*’s license for the *JDK*. (The license has become very permissive, but if there was a problem with it, alternative free *JDKs* exist.)
- Once the download is complete, a dialog box will ask for running the downloaded executable. When selected “*Run,*” the *Windows Installer* will start up and lead the developer through a dialog to install the *JDK* on the *PC*.

Linux

- Select the distribution for “*Linux self-extracting file.*”
- Read, review, and accept *Sun*’s license for the *JDK*. (The license has become very permissive, but if there was a problem with it, alternative free *JDKs* exist.)
- It is needed to download the self-extracting binary to the location in which the developer wants to install the *JDK* on his/her filesystem. If that is a systemwide directory (such as */usr/local*), it is necessary *root* access. After the file is downloaded, make it executable (`chmod +x jdk-6version-linuxi586.bin`), and execute it. It will self-extract to create a tree of directories.

Mac OS X

Mac OS X comes with *JDK* version 5 already loaded.

C.2.2 INSTALL ECLIPSE

The *Android SDK* requires *Eclipse* version 3.3 or later. If that version of *Eclipse* is not installed yet, go to <http://www.eclipse.org/downloads> to get the latest one, and the developer might as well get version 3.6.1 (also known as *Helios*), since that package includes the required plug-ins mentioned in the next step. Download the version of the *Eclipse IDE* labeled “*Eclipse IDE for Java Developers*” for the desired operating system. Eclipse will ask to select a mirror site, and will then start the download.

Windows (XP, Vista and 7)

The *Eclipse* download comes as a big *ZIP* file that it is installed by extracting the files to the developer’s favorite directory. It is assumed that it is extracted to *C:/eclipse*. *Eclipse* is now installed, but it will not show up in the *Start* menu of applications. This can be solved creating a *Windows* shortcut for *C:/eclipse/eclipse.exe* and place it on the *PC* desktop, in the *Start* menu, or someplace else where it can be easily found.

Linux and Mac OS X

Note that, as of this writing, the version of *Eclipse* installed if you request it on *Ubuntu Maverick Merkaat* is 3.5.2 codename *Galileo*, which does not contain all the plug-ins needed for *Android*. The *Eclipse* download comes as a big *tarball* (.gz file) that the developer may install by extracting the files to his/her favorite directory. It is assumed that it is extracted to */usr/lib/eclipse*. The executable itself is located in that directory and is named *eclipse*.



Figure 30 Eclipse IDE for Java Developers Welcome page running on Windows 7

C.2.3 CHECK FOR REQUIRED PLUG-INS

This step can be skipped if the developer just downloaded a current version of *Eclipse* as it was recommended. If using a preinstalled version of *Eclipse*, it is needed to make sure check if the the *Java Development Tool (JDT)* and *Web Standard Tools (WST)* plug-ins are installed. This can be easily checked to see whether they are installed by starting *Eclipse* and selecting menu options “*Windows → Preferences...*”. The list of preferences should include one for “*Java*” and one for either “*XML*” or “*Web and XML*”. If they aren’t on the list, the easiest thing to do is reinstall *Eclipse*, as described in the previous step. Installing “*Eclipse IDE for Java Developers*” will automatically get the needed plug-ins.

C.2.4 INSTALL ANDROID SDK

Start here if the right versions of *Eclipse* and the *JDK* are installed and loaded. The *Android SDK* is distributed through *Google’s Android* site, <http://developer.android.com/sdk/index.html>. There will be a list of downloads, and also a table of distributions. Select the one for the appropriate operating system.

For versions 3.3 and later of *Eclipse*, the *Android* download site provides directions about how to install the plug-in through *Eclipse’s* software updates utility. If using *Eclipse* 3.2 or the software update technique doesn’t work, download the *SDK* from the *Android* site and install it using instructions in the next paragraph.

The downloaded file is another archive file, as with *Eclipse*: a *ZIP* file on *Windows*, a tar-zipped file for *Linux* and *MacOS X*. Do the same thing as for *Eclipse*: extract the archive file to a directory where *Android* will be installed, and make a note of the directory name needed it in step B.2.6. The extraction will create a directory tree containing a bunch of subdirectories, including one called *tools*.

C.2.5 UPDATE THE ENVIRONMENT VARIABLES

To make it easier to launch the *Android* tools, add the *tools* directory to an appropriate path.

- On *Windows XP*, click on *Start*, then right-click on *My Computer*. In the popup menu, click on *Properties*. In the resulting *System Properties* dialog box, select the *Advanced* tab. Near the bottom of the *Advanced* tab is a button, “*Environment Variables*,” if pressed an *Environment Variables* dialog will appear. *User environment variables* are listed in the top half of the box, and *System environment variables* in the bottom half. Scroll down the list of *System environment variables* until “*Path*”; select it, and click the “*Edit*” button. Now an *Edit System Variable* dialog will allow you to change the environment variable “*Path*.” Add the full path of the *tools* directory to the end of the existing *Path* variable and click “*OK*.” Now the new version of the variable is in the displayed list. Click “*OK*” and then “*OK*” again to exit the dialog boxes.
- On *Windows Vista* and 7, click on the *Microsoft “flag”* in the lower left of the desktop, then right-click on *Computer*. At the top of the resulting display, just below the menu bar, click on “*System Properties*.” In the column on the left of

the resulting box, click on “*Advanced system settings.*” The *OS* will warn the user with a dialog box that says “*Windows needs your permission to continue*”; click “*Continue.*” Near the bottom of the *System Properties* box is a button labeled “*Environment Variables*”. Press it to go to an *Environment Variables* dialog. *User environment variables* are listed in the top half of the box, and *System environment variables* in the bottom half. Scroll down the list of *System environment variables* until “*Path*”; select it, and click the “*Edit*” button. Now an *Edit System Variable* dialog allows the developer to change the environment variable “*Path.*” Add the full path of the *tools* directory to the end of the existing *Path variable*, and click “*OK.*” Now the new version of the variable in the displayed list is configured. Click “*OK*” and then “*OK*” again to exit the dialog boxes.

- On *Linux*, the *PATH environment variable* can be defined in the *~/.bashrc* *~/.bash_profile* file. If having either of those files, use a text editor such as *gedit*, *vi*, or *Emacs* to open the file and look for a line that exports the *PATH* variable. If finding such a line, edit it to add the full path of the *tools* directory to the path. If there is no such line, it is possible add a line like this:

```
export PATH=${PATH}:your_sdk_dir/tools.
```

- On *Mac OS X*, look for a file named *.bash_profile* in the home directory (note the initial dot in the filename). If there is one, use an editor to open the file and look for a line that exports the *PATH* variable. If there is such a line, edit it to add the full path of the *tools* directory to the path. If there is no such line, add a line like this:

```
export PATH=${PATH}:your_sdk_dir/tools.
```

C.2.6 INSTALL THE ANDROID PLUG-IN (ADT)

Google supplies *Android Development Tool* plug-in that is necessary for use in building *Android* applications. The plug-in is installed in much the same way as any other *Eclipse* plug-in:

- a. Start *Eclipse*, if it’s not already running
- b. From the menu bar, select “*Help → Software Updates → Find and Install...*”.
- c. In the *Install/Update* dialog, select “*Search for new features to install*” and click on “*Next.*”
- d. In the *Install* dialog, click on “*New Remote Site.*” . A “*New Update Site*” dialog pops up. Enter a name for the plug-in (“*Android Plugin*” will do), and the *URL* for updates: <https://dl-ssl.google.com/android/eclipse>. Click “*OK.*”
- e. The new site should now appear in the list of sites on the *Install* dialog. Click “*Finish*”.

- f. In the Search Results dialog, select *Plugin* the checkbox for “*Android →/ Developer*” and click *Tools* “*Next*”.
- g. The license agreement for the plug-in appears. Read it, select “*Accept terms of the license agreement*” and click “*Next*”. Click “*Finish*”.
- h. A warning message will appear with the following saying that the plug-in is not signed. Choose to install it anyway by clicking “*Install All*”.
- i. Restart *Eclipse*.
- j. After *Eclipse* restarts, it is needed to tell it where the *SDK* is located. From the menu bar, select “*Window → Preferences*.” In the Preferences dialog, select “*Android*” in the left column.
- k. Use the “*Browse*” button to navigate to the place where the *Android SDK* is installed, and click on “*Apply*,” then on “*OK*”.

The *Android* development environment is now installed. The environment includes a very sophisticated set of tools to make *Android* programming easier, including:

- An *Integrated Development Environment* based on *Eclipse*, arguably the premier *IDE* for Java development. *Eclipse* itself brings many valuable development features. *Google* and *OHA* have taken advantage of *Eclipse*’s extensibility to provide features customized for *Android*, including debugging capabilities that are tuned to the needs of mobile application developers.
- A *Java development environment* and *Dalvik virtual machine* that build on *Sun*’s *JDK* foundation to provide a very sophisticated programming environment for *Android* applications.
- A complete mobile phone emulator that allows the developers to test their applications without having to download them to a target mobile phone. The emulator includes features for testing applications under different mobile phone communication conditions (fading, dropped connections, etc.).
- Test tools, such as *Traceview*, which allow developers to tune their applications to take best advantage of the limited resources available on a mobile phone.

C.3 THE FIRST PROGRAM: “HELLO, ANDROID”

A “*Hello World!*” program is a traditional one, so start with something similar to that to demonstrate what the developer needs to do to create, build, and test an *Android* application. *Android API* will not be explored so much for this program, but here the developers will get a taste for the development environment and the steps they go through to create an application for *Android*.

There is not much functionality in this program. The main aim is displaying some text on the *Android* emulator window that says “*Hello Android!*” (see the follow Figure).



Figure 31 "Hello Android" Screenshot

C.3.1 STARTING A NEW ANDROID APPLICATION: HELLOWORLD

Several components are needed to build an Android application. Fortunately, the *Eclipse IDE* with the *Android* plug-in automates a lot of the work needed to create and maintain these components.

Start up *Eclipse* and select “*File* → *New* → *Project...*” from the menu bar (be sure to select “*Project...*” not “*Java Project*”). A list of project types will be displayed, similar to the menu.

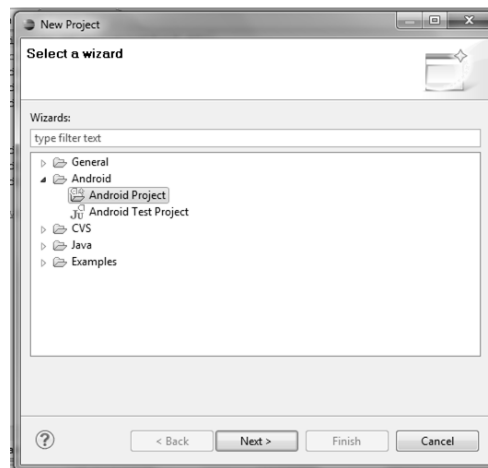


Figure 32 Eclipse New Project Menu

Select “*Android Project*” and click “*Next*” to get the “*New Android Project*” dialog box. *Hello Android*” will be the name for both the *Project* and the *Application*. It is not necessary to change the button or checkbox selections. Select “*Android 2.2*” as “*Build Target*”. The package name will be called “*appendix.b.helloandroid*” as shown. Every Android application has to have at least one *Activity* (an executable that usually has a user interface), so finish the project by creating the main activity “*HelloAndroid*”, and selecting the minimum *SDK* version; “*8*”.

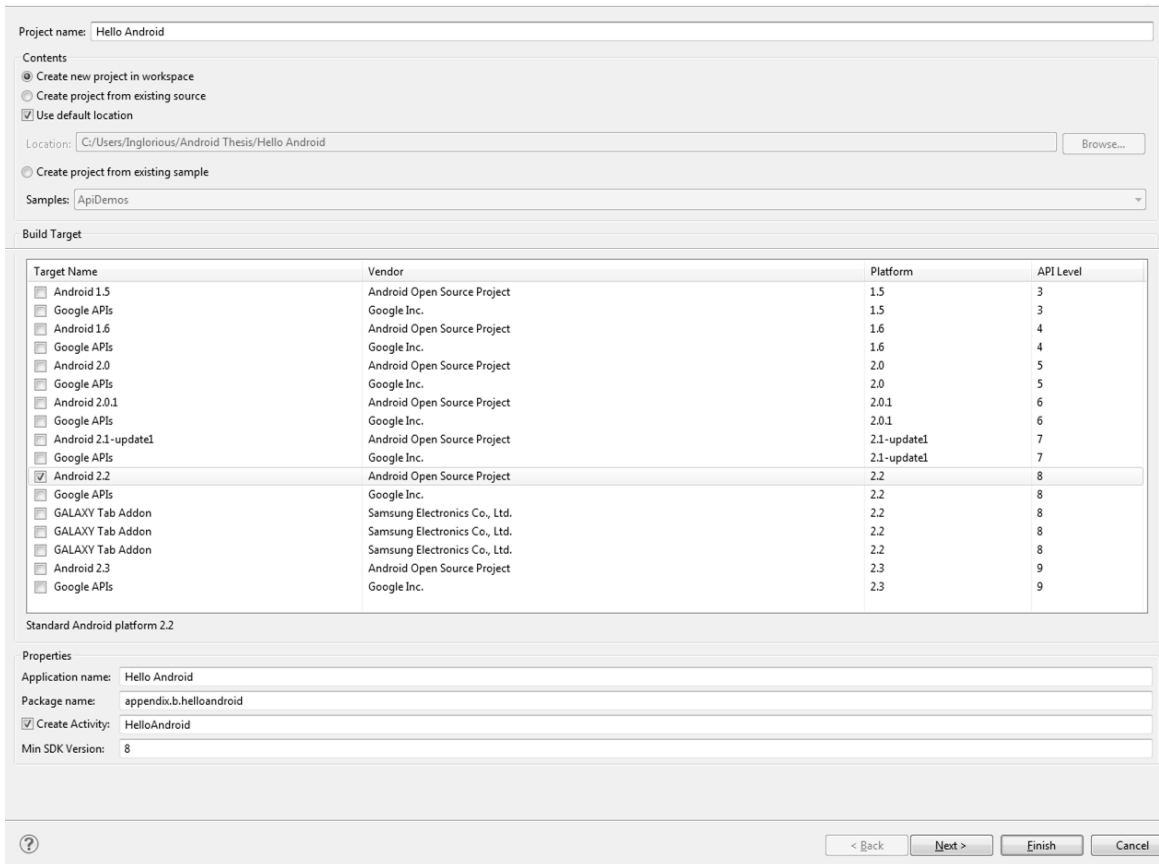


Figure 33 Eclipse New Android Project dialog

Click “*Finish*,” and the *Android Software Development Kit* does a number of things for the developer. In the follow Figure, It is expanded the tree in the *Package Explorer* window to show some of the files and directories that the *Android SDK* created.

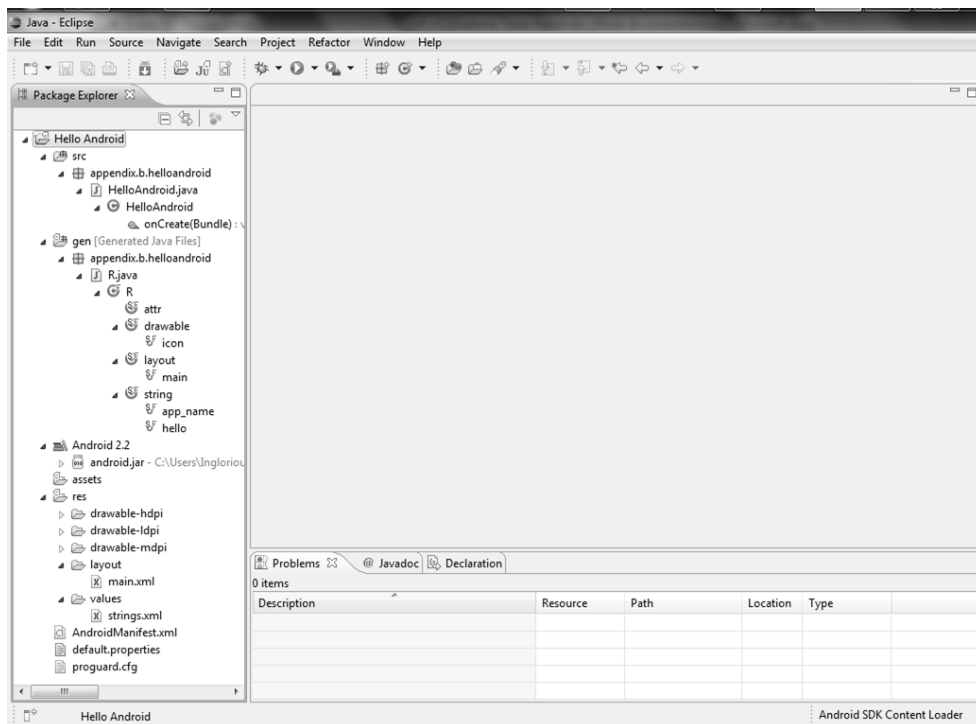


Figure 34 Eclipse project listing after creation of the HelloWorld project

The *Android SDK* created a *Hello Android* directory in the default *Eclipse* workspace for the project. It also created subdirectories for the source files (*.src*), references to the *Android Library*, *assets*, *resources* (*.res*), and a *manifest file* (*AndroidManifest.xml*). In each of the subdirectories it created another level of subdirectories as appropriate:

Sources (under src)

- Contains a directory structure that corresponds to the package name given for the application: in this case, *appendix.b.helloandroid*.
- Contains a *Java* template for the *Activity* it was indicated in the application (*Hello Android*) and may contain a directory of resource references (*R.java*). *R.java* is actually generated by the *Android SDK* the first time the application is compiled; it contains the *Java* version of all the resources defined in the *res* directory (covered later).

Android Library

This is just what it says. Expand the *android.jar* tree and see the names of the modules included in the library. This is where the application will go for *Android library* references.

assets

Files wanted to bundle with the application.

Resources (under res)

- *Drawable* resources are any images, bitmaps, etc., the developer may need for the application. For *Hello Android*, the *Android SDK* supplied the developer with the default *Android* icon.
- *Layout* resources tell *Android* how to arrange items on the screen when the application runs. These resources are *XML* files that give the developer quite a bit of freedom in laying out the screen for different purposes. For *Hello Android*, we'll just use the defaults generated by the *Android SDK*.
- Values are constants, strings, etc., available for use by the application. Keeping them outside the sources makes it easier to customize the application, such as adapting it for different languages.

Manifest (AndroidManifest.xml)

This is another *XML* file that tells the *Android* build system what it needs to know to build and package the application so it can be installed on an *Android* phone or the emulator. This file has its own specialized editor, which it will be described when developing more complicated applications.

C.3.2 WRITING HELLOWORLD

In the *Eclipse Package Explorer* window, double-click on `HelloAndroid.java`. This opens the source file of that name in the center window, ready for editing:

```
package appendix.b.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Listing 30 HelloAndroid source code

Looking quickly at the template code that the *Android SDK* has provided for the developer, it can be noted several things:

- The *Android SDK* has included the package reference the developer asked for, which is consistent with the directory structure it created.
- It has also created a (collapsed) set of imports for the library references it knows the developer needs.
- It created a class definition for the *Activity* (*HelloAndroid*), including a method called `onCreate`. The `savedInstanceState Bundle` is a way of passing data between activities and storing data between instantiations of the same *Activity*. It won't be needed to use this for *HelloAndroid*.
- One special line of code has been included in `onCreate`:

```
setContentView (R.layout.main);
```

The developer may remember that *Android* uses layouts to define screen layouts on the target, and that *main.xml* was the name of the default layout file that the *Android SDK* created for us under *.res/layout*. The *R.java* file is generated automatically and contains *Java* references for each of the resources under *.res*. It will not be necessary edit the *R.java* file by hand; the *Android SDK* takes care of it as the developer adds, change, or delete resources.

Again in the *Eclipse Package Explorer* window, double-click on *main.xml* and default layout screen will appear in the center window. There are two tabs at the bottom of the panel that say “*Layout*” and “*main.xml*”. Click on the one that says “*main.xml*” to bring up the code version:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>

```

Listing 31 main.xml source code

Again, let's look at the key features of this template code:

- Like any other *XML* file, this one starts with a reference to the *XML* version and encoding used.
- *LinearLayout* is one of the screen layout formats provided by the *Android SDK*. There are several others, which can be combined hierarchically to create very complex screen layouts. For this example, a simple linear layout is fine. More Layout types are covered later in the book in [1, Chapter 3], [2, Chapter 4], [3, Chapter 11].

- The *LinearLayout* definition:

```
xmlns:android=http://schemas.android.com/apk/res/android
```

identifies the *XML* schema being used.

- This code:

```

android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"

```

defines an orientation, width, and height for the entire scope of the layout.

- *TextView* describes an area where text can be displayed and edited. It resembles the text boxes you may have encountered when programming in other graphical environments.

- Within the *TextView* definition:

```

android:layout_width="fill_parent"
android:layout_height="wrap_content"

```

define a width and height for the *TextView* box.

- This code:

```
android:text="@string/hello"
```

provides some text to display in the *TextView*. The actual string is defined in a separate file, *res/values/strings.xml*. If that file is opened (again by clicking on it in the *Package Explorer*), a specialized string editor added by *ADT* will be displayed. If “*hello (String)*” is selected by clicking on it, the current value for that string will be displayed. By a stroke of luck, the *Android SDK* has already included text that is close to what developers wanted to display anyway. For example, the developer may change the value of the String *hello* to say “*Hello Android!*”, or something else equally clever.

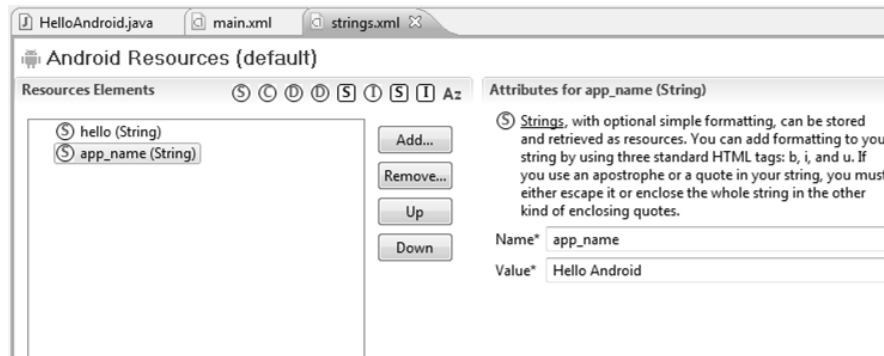


Figure 35 Hello Android String editing

Save the *Project* either from the *Eclipse File* menu (*File* → *Save*) or by clicking on the disk icon in the menu bar.

C.3.3 RUNNING HELLO WORLD

From the *Eclipse* menu bar, select *Run* → *Run*. A “*Run As*” dialog box will pop up. Select “*Android Application*” from the list, which displays the dialog shown in the next Figure.

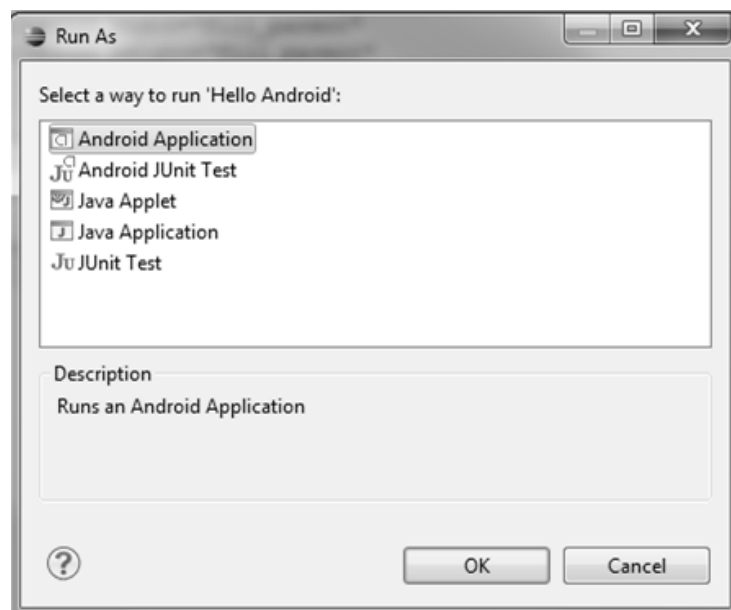


Figure 36 Eclipse Application Type selection

A command window will pop up, followed quickly by an emulator window that looks just like a mobile phone. The emulated phone will then go through its boot sequence, which takes a few minutes. After a few minutes a screen as shown in the next Figure will be displayed.



Figure 37 Emulator and first try at Hello Android

This is enough for creating the first *Android* program.

APPENDIX D

INTEGRATE ZXING INTO ANDROID APPLICATIONS

D.1 INTRODUCTION

ZXing is an open-source, multi-format 1D/2D barcode image processing library implemented in *Java*. This library focuses on using the built-in camera on mobile phones to photograph and decode barcodes on the device, without communicating with a server.

D.2 GETTING STARTED

D.2.1 Download ZXing

Grab the latest stable distribution from <http://code.google.com/p/zxing/downloads/list>. After that, extract the contents of the .zip file in the root directory of the hard disk.

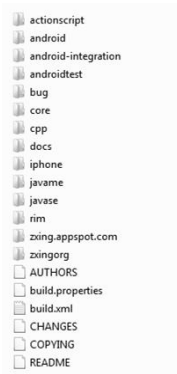


Figure 38 *Zxing* Library contents

D.2.2 Download Apache Ant

Apache Ant is a *Java* library and command-line tool which's mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of *Ant* is the build of *Java* applications. Download the last version from <http://ant.apache.org/bindownload.cgi> and extract the content.

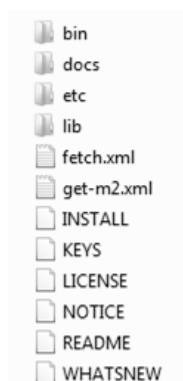


Figure 39 Apache Ant contents

To install the Apache Ant is necessary to update the environment variables. This step is similar to the step shown in B.2.5. Thus, depending on Operating System used, the developer may add the `..\apache-ant-1.8.1\bin` to an appropriate path.

To verify the environment variables are correctly updated, from a console window run the following command:

```
ant -version
```

If it displays the version information as the next figure

```
C:\Users\Inglorious>ant -version
Apache Ant version 1.8.1 compiled on April 30 2010
C:\Users\Inglorious>
```

Figure 40 Checking the installation of Ant

Ant is correctly configured and installed.

D.2.3 Build

The code lives in several subdirectories like it is shown in Figure 27, corresponding to the various subcomponents, like "*core*" and "*javase*".

Within each, there is a `build.xml` *Ant* build file which controls building of that component. As *Apache's Ant* tool is already present on the system, simply type `ant -f core/build.xml` in the *ZXing* directory to build that component.

To build everything, simply use the `build.xml` file found in the top-level directory, above all components.

```
C:\ZXing-1.6\zxing-1.6>ant -f core/build.xml
Buildfile: C:\ZXing-1.6\zxing-1.6\core\build.xml
clean:
  [delete] Deleting directory C:\ZXing-1.6\zxing-1.6\core\build
  [delete] Deleting: C:\ZXing-1.6\zxing-1.6\core\core.jar
build:
init:
compile:
  [mkdir] Created dir: C:\ZXing-1.6\zxing-1.6\core\build
  [javac] C:\ZXing-1.6\zxing-1.6\core\build.xml:36: warning: 'includeantruntime'
e' was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds
  [javac] Compiling 171 source files to C:\ZXing-1.6\zxing-1.6\core\build
  [jar] Building jar: C:\ZXing-1.6\zxing-1.6\core\core.jar
BUILD SUCCESSFUL
Total time: 6 seconds
```

Figure 41 Building ZXing

D.2.4 Integrate ZXing source lib into the *Android Code* project through *Eclipse*

In *Eclipse* right-click the Project Folder → *Properties* → *Java Build Path* → *Libraries* → *Add External JARs* → *Navigate to the ZXing folder and open the core directory*. Select *core.jar*.

Finally, it is necessary to correct a few errors in the translations and the `AndroidManifest.xml` file. After that, it is possible to compile, and the developer will have a working standalone barcode scanner application based on the *ZXing* source.

D.3 ALTERNATIVE METHOD FOR ANDROID

D.3.1 ScanningViaIntent

In *Android*, there is an alternative method for simple access to barcode scanning, via `Intents` rather than direct use of project code.

D.3.1 ScanningViaIntent

- Manually

If the *Barcode Scanner* is installed on an *Android* device, the developer can have it scan for him and return the result, just by sending it an `Intent`. For example, it is possible to hook up a button to scan a *QR* code like the next Listing shows:

```
public Button.OnClickListener mScan = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new
Intent("com.google.zxing.client.android.SCAN");
        intent.setPackage("com.google.zxing.client.android");
        intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
        startActivityForResult(intent, 0);
    }
};

public void onActivityResult(int requestCode, int resultCode, Intent
intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            String contents = intent.getStringExtra("SCAN_RESULT");
            String format =
intent.getStringExtra("SCAN_RESULT_FORMAT");
            // Handle successful scan
        } else if (resultCode == RESULT_CANCELED) {
            // Handle cancel
        }
    }
}
```

Listing 32 Scanning via Intent in Android

For more options, like scanning a product barcode, or asking *Barcode Scanner* to encode and display a barcode, developers may see the source file from <http://code.google.com/p/zxing/source/browse/trunk/android/src/com/google/zxing/client/android/Intents.java>

- `IntentIntegrator`

ZXing team provides a small library of classes that encapsulate some of the details of above. See `IntentIntegrator` for a possibly easier way to integrate. In particular this will handle the case where *Barcode Scanner* is not yet installed. <http://code.google.com/p/zxing/source/browse/trunk/android-integration/src/com/google/zxing/integration/android/IntentIntegrator.java>

QUESTIONNAIRE FOR THE ANALYSIS

QUESTIONNAIRE

(This questionnaire is confidential and will be handled with the strictest of confidentiality)

Country of origin:

Studies:

1. Do you know what is the *Android Platform*? (Mark with a cross)

- Yes**
 No

2. Did you have contact with some *Android* device?

- Yes**
 No

3. What of the following languages are you basically fluent in? (Mark with a cross)

- English**
 Greek
 Italian
 Other

4. Which of the three modes of interaction did you find more intuitive to use? (Mark with a cross. 5 means more intuitive)

	1	2	3	4	5
QR Codes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Image Map	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Numerical Codes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

5. Which modality did you find more interesting? (Mark with a cross. 5 means more interesting)

	1	2	3	4	5
QR Codes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Image Map	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Numerical Codes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

6. Which one did you think was likely to produce more mistakes? (Wrong *clicks/taps*, wrong scans...)

- QR Codes**
 Image Map
 Numerical Codes

7. Did you have contact with some of these technologies before?

- Yes**
 No

8. Would you withdraw from the application some of these technologies?

- Yes**
- No**

If you have marked **Yes**, indicate which one:

- QR Codes**
- Image Map**
- Numerical Codes**

9. The provided content in the application with the information of the pictures in the gallery was (mark with a *cross*):

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Insufficient | Enough | Good | Too Much |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

10. The content of the *audio* was:

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Insufficient | Enough | Good | Too Much |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

11. Was it easy to understand the *audio* with the descriptions provided?

- Yes**
- No**

12. Would you include some other media (like *video clips* about the exhibits)?

- Yes**
- No**

13. Would you like a help button that gives you additional information about the exhibit or how the guide works?

- Yes**
- No**

14. Do you think the description font/image size is enough?

- Yes**
- No**

15. Would you like to read the description/view the portrait in full screen on the device?

- Yes**
- No**

16. Would you like the application implements other languages?

- Yes**
- No**

If you have marked **Yes**, indicate which ones:

- Spanish**
- French**

- German**
- Other:**

17. The buttons in the application are

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Small | Big | Necessary | Unnecessary |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

18. Would you like to include some new performances to the guide? (Mark with a cross which ones)

- Tour Functionality**
- Map Functionality**
- Exhibits Topic Filter**
- Options Menu**
- Background Music**
- Museum Mini-Games**
- Other**

19. The appearance and the UI (*User Interface*) of the electronic guide was pleasing/friendly (e.g. : *design, interface, images, colours, size of icons*):

- | | | | | |
|--------------------------|----------------------------|---------------------------|--------------------------|--------------------------|
| DK/NA/REF | Disagree completely | Disagree partially | Agree partially | Agree completely |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

20. Do you find the electronic guide useful for a real museum?

- Yes**
- No**

21. Have you ever used applications like this?

- Yes**
- No**

If you have marked **Yes**, indicate which one and when:

.....

.....

22. How would you improve this application?

.....

.....

Thank you for your participation.


DEMOGRAPHICS SURVEY:


INTRODUCTION

The aim of the following experiment is study the functionality of a *Mobile Museum Guide Application: "My Pocket Museum"* in a "simulated museum" placed at the *Eastern Building* of the *Electrical and Computer Engineering Department* of the University of Patras, 1st floor, where is placed also the laboratory of the *HCI Group*

For this purpose, you will use a smart mobile device running *Google Android* Operating System. If you are not familiar with these kind of mobile devices or technology, this test will show you the advantages of that technology in such different fields.

The test is divided into two parts:

 The first one is a survey about demographics you may fill **before testing the application**. After that, some instructions about how the application runs will be given to you. Once you read them, you may start the application and follow the steps in the instructions.

 The second part is another survey with some questions about the usability, performance, versatility, etc. of the application. This survey will be given to you **after you finish testing the application**.

The surveys are confidential and will be handled with the strictest of confidentiality.

The information gained from both parts will be used for the purpose of research, results and conclusions about my Diploma Thesis.

Francisco de Borja Sánchez Sancho, Erasmus student of the Department of Electrical and Computer Engineering at the University of Patras.

Thank you a lot for your participation.

DEMOGRAPHICS SURVEY

(This survey is confidential and will be handled with the strictest of confidentiality)

Country of origin:

Age:

Gender: Male

Female

Occupation:

Education/Field of study:

Languages

1. What is your native language?

.....

2. What of the following languages are you basically fluent in? (Mark with a cross as many as you want)

English

Greek

Italian

Other(s)

Museums

3. Have you ever been in a museum? (Mark with a cross)

Yes

No

4. Have you used some kind of guide system such a tour guide, tape machine or CD Player? (Mark with a cross)

Yes

No

Other(s)

Technology

5. Generally speaking, how comfortable do you feel using a mobile device? (Mark with a cross)

Very comfortable

Somewhat comfortable

Not very comfortable

Not at all comfortable

6. Aside from making and sending/receiving calls or messages, what do you mostly use your mobile phone for? (Mark with a *cross* as many as you want)

- Taking photos
- Playing games
- Listening to music
- Internet browsing
- Keep track of calendar events
- Other:

7. Have you ever interacted with some kind of electronic guide in a museum? (Mark with a *cross*)

- Yes
- No

If you have marked **Yes**, indicate which one and where:

.....

8. Do you know what is the *Android Platform*? (Mark with a *cross*)

- Yes
- No

9. Did you have contact with some *Android* device before this test? (Mark with a *cross*)

- Yes
- No

10. Have you ever heard about *QR Codes* technology? (Mark with a *cross*)

- Yes
- No

11. Have you used this technology? (Mark with a *cross*)

- Yes
- No

If you have marked **Yes**, indicate when and where:

.....


STEPS TO FOLLOW: A GUIDE TO THE MUSEUM


STEPS TO FOLLOW: A GUIDE TO THE MUSEUM

1. THE DEVICE: FIRST CONTACT WITH ANDROID

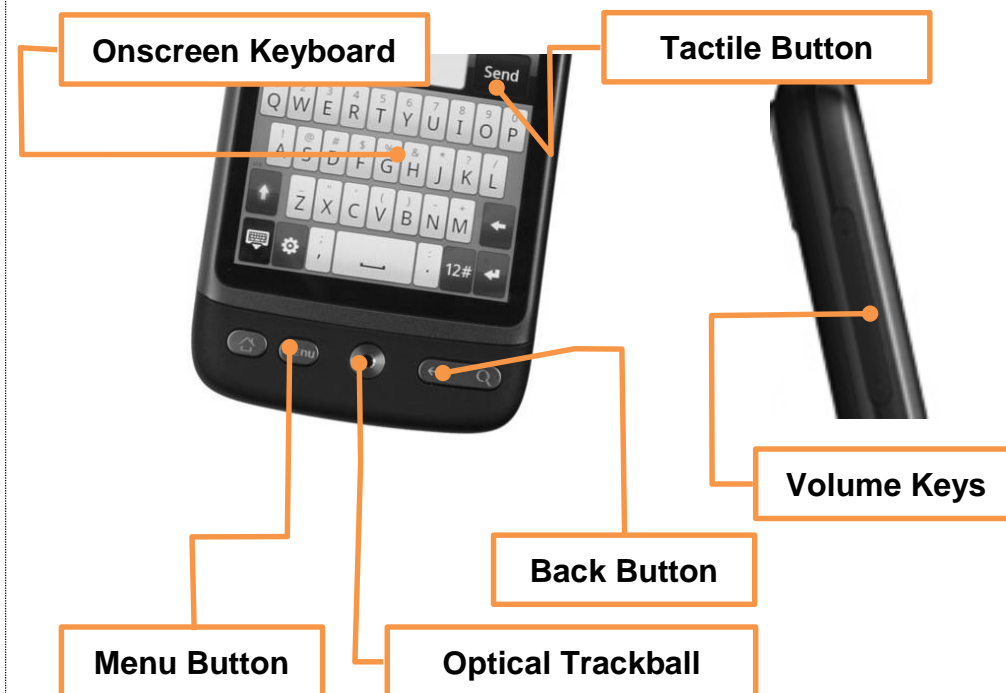
You are provided with a last generation intelligent mobile device made by *HTC*: the *HTC Desire*. It runs the *Google's Android 2.2* Operating System.

As you can see, the device has a few physical buttons and the most important is that it has not a physical keyboard. Thus you will be able to use two modes of interaction:

 The **tactile mode** on the device screen: you interact with the application by *tapping* the *virtual onscreen keyboard, buttons, images, etc.* or *dragging* the *information*. Once you are running the application you will often use this mode for displaying the information, moving through the exhibits, playing the audio, ...

 The **physical mode**: you interact with the application by pressing some of the buttons the device has: the *menu key*, the *back key*, the *optical trackball*, the *home key*, the *browse key* and the *volume keys* on one of the sides of the device. You will use the *menu key* in the exhibits for displaying the different options (go back to the main screen of the application, play the audio, display some help and tips or display a concrete exhibit using the *QR Codes*). The *back key* allows you to return to the immediately previous screen. The *optical trackball* allows you to move through the tour list as well as the exhibit list. Finally the volume keys allow you to increase or decrease the exhibit audio.

The next figure shows these two modes of interaction:



2. RUNNING THE APPLICATION

While in the *Home* screen of the device, you may follow the following steps to start the application:

Before running the application, make sure the device has access to the wireless network due to the application needs it to run. You can check this if in the *Home* screen you see the next icon on the icon tray.

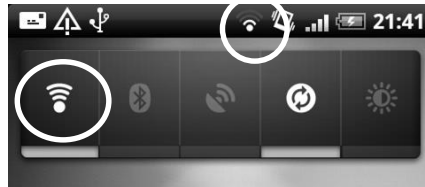


Figure 42 Wireless shortcut and tray icon

If not, you can switch on the wireless by tapping on the next icon on the *Home* Screen.

1. Press the physical key *menu*. A new menu will emerge.
2. In that menu, tap on “Applications”.
3. Move through the applications by dragging the screen device or using the optical trackball and find out the “**My Pocket Museum**” application.
4. Tap on the icon and the application will start.



Figure 43 Applications menu

(Note: The application is currently only in English language, it is recommended you have a basic knowledge of that language).



Figure 44 The *tap* gesture (left) and the *drag* gesture (right) .

3. THE MAIN SCREEN: TOUR LIST SCREEN

The main screen is the *Tour Screen*. This screen will welcome you to the application and show you a list of tours in which you may select one of them. This screen is used also as the *Home* screen for the application.

The list also provides you basic information about the tour: a small representative *icon*, the *name* of the tour, a small *overview* about it, the number of exhibits it contains, the estimated *duration* and the *distance* of the tour.

You can move through the list by *dragging* on the device screen or, if you prefer, with the *optical trackball*.

Tap on the desired tour to select the tour you want display and go to the next screen.



Figure 45 The Tour List Screen

You can press the “menu” physical key for displaying the “Camera” mode. This will allow you to scan the *QR Code* placed under the exhibit for displaying it without selecting a tour.

4. THE TOUR AND THE EXHIBITS LIST

Once you have selected the tour, the second screen will be loaded. This new screen will show you the following information: on the top of the screen some useful information about the tour you have selected: the representative *icon* of the tour, the *name* and the number of exhibits it contains. Under this, the application will load the list of the exhibits that tour contains.

The list of the exhibits shows you a small *thumbnail* of the exhibit, the *name* of the portrait, the *info* about it, the *review* (you can read it in this screen) and the *location* of the exhibit in the building.

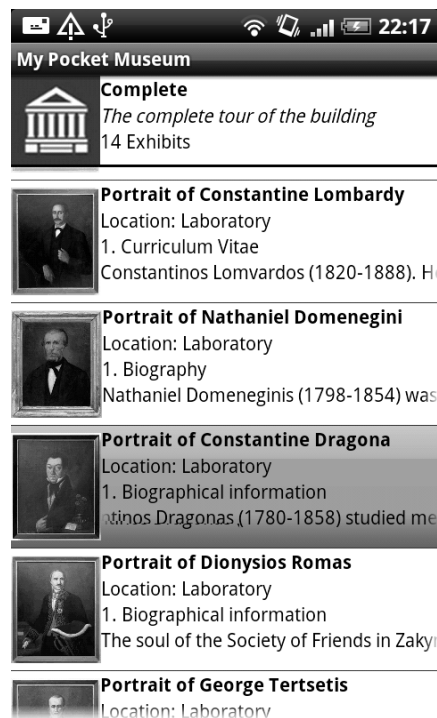


Figure 46 The Exhibit List Screen

The order of the exhibits in the list indicates the order you should visit the exhibits. Anyway, if you want you can skip some of the exhibits and start on the desired exhibit.

For that, you may only tap on the desired exhibit and the device will show you the information related to that exhibit. For example, if you want to complete the whole tour, you should tap on the first exhibit of the list, but if you do not have enough time to complete the tour or you are not interest in some exhibits the tour contents, you can tap on the exhibit you want and start the tour from that one.

If you decide you do not want to make that tour, you can go to the *Tour Screen* simply by pressing the *back* physical button.

Again, you can press the “menu” physical key for displaying the “Camera” mode. This will allow you to scan the *QR Code* placed under the exhibit for displaying it without selecting a tour.

5. THE EXHIBIT SCREEN

This screen will display when you tap on one of the *exhibits* displayed on the *Exhibit List*. The *exhibit screen* will display the following elements: the *title*, the info of the exhibit, a small *image*, the *description* related to the exhibit and two buttons in the bottom of the screen.

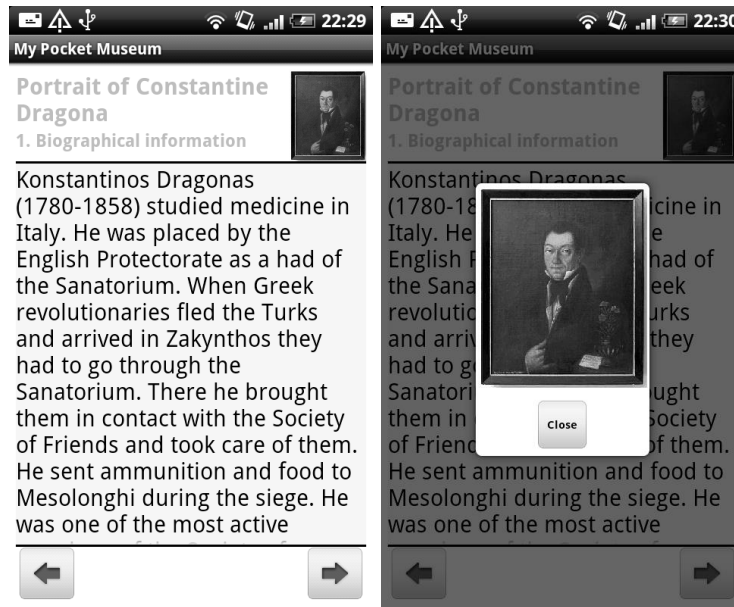


Figure 47 The Exhibit Screen and the Full Screen portrait

If you tap on the small image a new window will appear and it will display the image of the exhibit in a bigger size. If you want to close the window, just tap the *close* button under the exhibit or press the *back* button.

For moving through the description (if necessary) you can use the drag gesture on the screen or use the *optical trackball*.

The two arrows on the bottom of the device screen allow you to move through the exhibits of the tour, so if you want go to the previous exhibit you may tap on the *left bottom button*, and if you want to the next exhibit you may tap on the *right bottom button*. When the tour finishes or it starts from the first exhibit of the tour, you may notice that the *arrows buttons* will change into a *list button*. If you tap on them, you will go back to the Exhibit List Screen.

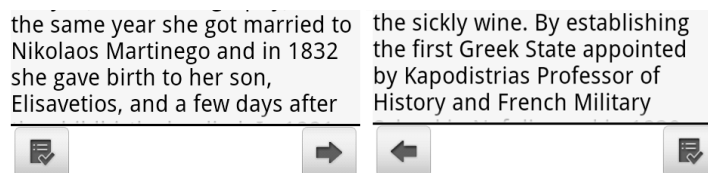


Figure 48 Start and end of the tour buttons

Finally, in this screen you will be able to use the physical *menu* key. If you press it, a new options menu will emerge from the bottom of the screen with four possible choices: *Home App*, *Audio*, *Help* and *Camera*. The options are explained in the next point.

6. THE EXHIBIT SCREEN MENU

The *Exhibit* screen allows the user to choose among four different choices. The choices available are: *Home App*, *Audio*, *Help*, and *Camera*.

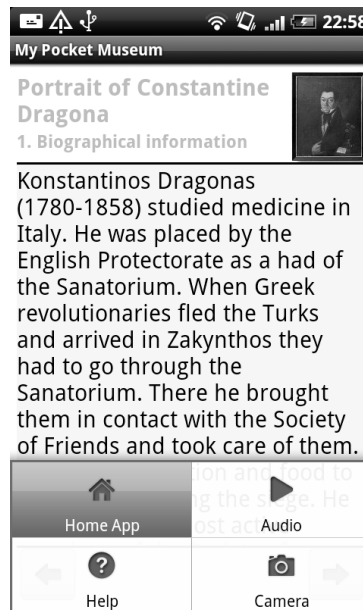


Figure 49 The Exhibit Screen menu

Here is a small explanation about each one:

6.1 HOME APP

Tap here if you want to go back to the main screen: the *Tour Screen*.

6.2 AUDIO

Tap here if you want to listen to the description of the current exhibit. When you tap on the *audio* option the audio record will be loaded. If you want to stop the audio, press *menu* and you will see that the audio icon has changed. Tap on it for pause the audio of the current exhibit. If you tap again on audio, the soundtrack will resume. Note that if you move to other exhibit or other screen, the audio will stop.

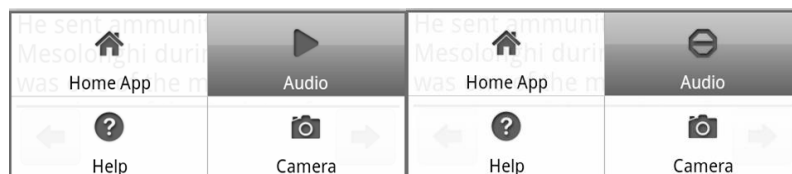


Figure 50 Audio play/pause buttons

If you want to increase/decrease the volume of the audio, you may use the *volume* keys on the left side of the device.

6.3 HELP

Tap here if you need some help about the exhibit screen. Basically it will display the same information you are reading in this manual.

When you tap on *Help* a new window will emerge with some useful information. For close it, tap on the *close* button under the text, or press the *back* key.

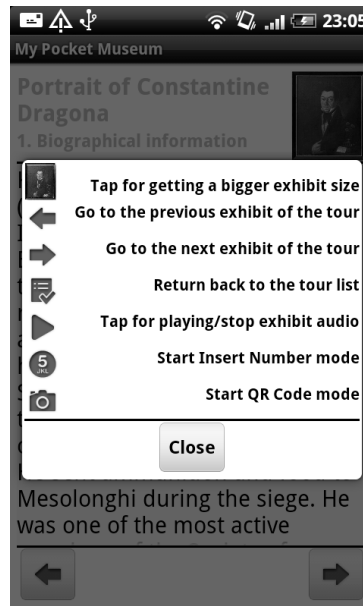


Figure 51 The Exhibit Screen Help option

6.4 CAMERA

The next and last option you will test to obtain the information of the exhibits in the museum will be *QR codes*.

A *QR code* (short for *Quick Response*) is a specific matrix barcode (or two-dimensional code), readable by dedicated *QR barcode* readers and camera phones. The code consists of black modules arranged in a square pattern on a white background. The information encoded can be *text*, *URL* or other *data*.

In the *exhibit* screen, press on the *menu* key, and the menu will be shown again. Tap on *Camera* and the *barcode scanner* application will start.

To be able to see the exhibit, you may focus the camera of the mobile device on the *QR code*.

The application can take one or two seconds to recognize the code, depending on the position of the camera, so try to maintain the camera in a fix position.

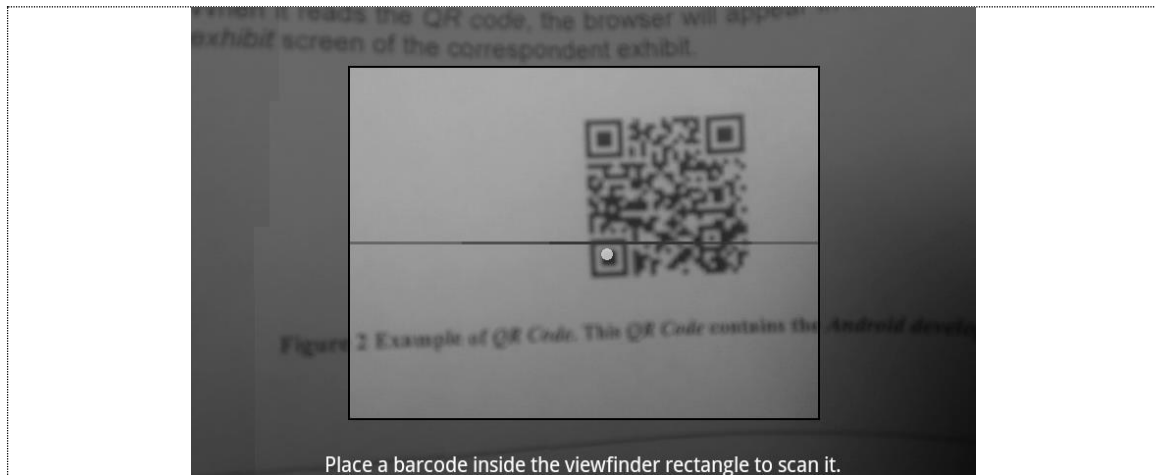


Figure 52 How to scan a *QR Code*

When it reads the *QR code*, the browser will appear in the screen, showing the *exhibit* screen of the correspondent exhibit and button on the right bottom of the screen. If you tap on that button you will return to the Exhibit Screen where you were previously.



Figure 53 Example of *QR Code*

Now, you can start with the test and try the *Android* mobile following the instructions above.

QUESTIONNAIRE FOR THE RESULTS:

QUESTIONNAIRE

Device and procedures

1. Was it easy for you to become familiar with the device? (Mark with a *cross*)
 Yes
 No
2. Was it easy to follow the steps to complete the procedure of the application? (Mark with a *cross*)
 Yes
 No

Languages

3. The application you tried is currently only available in English, would you like the application implements other languages? (Mark with a *cross*)
 Yes
 No

If you have marked **Yes**, indicate which ones (Mark with a *cross* as many as you want):

- French**
- German**
- Greek**
- Italian**
- Spanish**
- Other:**

4. Which of the languages you marked in the previous question would you use if it will be available in the application? (Skip this question if you answer **No** in the question #3)
.....

Tours

5. Did you find useful the tours functionality for finding information about the exhibits? (Mark with a *cross*)
 Yes
 No
6. The provided content in the application with the information of the available tours was (Mark with a *cross*):

Insufficient

Enough

Good

Too Much

7. Do you think a button help in this screen will be helpful?

- Yes**
- No**

8. Was it useful for you the Camera option in the Tour List screen?

- Yes**
- No**

9. When you select a tour, the application displays on the screen a list of the exhibits that tour contains. The information about these exhibits was (Mark with a *cross*):

Insufficient

Enough

Good

Too Much

10. Was it easy for you to find out the place of each exhibit (while you are following a tour)? (Mark with a *cross*):

- Yes**
- No**

11. Do you think it will be useful to implement some map (or other kind of orientation) for finding the place of each exhibit? (Mark with a *cross*)

- Yes**
- No**

Exhibits

12. The provided content in the application with the information of the exhibit was (Mark with a *cross*):

Insufficient

Enough

Good

Too Much

13. Was it easy for you to move through the exhibits? (Mark with a *cross*):

- Yes**
- No**

14. When you tap on the small image, a new one emerges with bigger size, was it useful for you? (Mark with a *cross*)

- Yes**
- No**

15. Would you include some other media (like *video clips* about the exhibits)? (Mark with a *cross*)

- Yes**
- No**

Audio

16. Was it easy to understand the *audio* with the descriptions provided?
(Mark with a *cross*)

- Yes**
- No**

17. The content of the *audio* was (Mark with a *cross*):

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Insufficient | Enough | Good | Too Much |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

18. The audio volume was (Mark with a *cross*):

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Low | Enough | Good | Too Loud |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

19. Would it better keep the audio of one concrete exhibit while you move through the other exhibits?

- Yes**
- No**

Help

20. Was it useful the information about the exhibits? (Mark with a *cross*)

- Yes**
- No**

Modes of Interaction

21. Did you find the *QR Codes* method intuitive to use? (Mark with a *cross*. 5 means more intuitive)

- | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 1 | 2 | 3 | 4 | 5 |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

22. Did you find interesting the *QR Codes* method? (Mark with a *cross*. 5 means more interesting)

- | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 1 | 2 | 3 | 4 | 5 |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

23. Did you have contact with this technology before? (Mark with a *cross*)

- Yes**
- No**

24. Would you like the application implements other modes of interaction?
(Mark with a *cross*)

- Yes**
- No**

Interface

25. Do you think the fonts and images size is enough? (Mark with a *cross*)

- Yes**
- No**

26. Would it be useful for you to change the screen mode of the device from portrait to landscape while running the application?

- Yes**
- No**

27. Would you like the application runs on the full screen of the device (hide the top icon tray)?

- Yes**
- No**

28. The buttons in the application are (Mark with a *cross*. Mark as many as you want):

- | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|
| Small | Big | Necessary | Unnecessary |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

29. The appearance and the UI (*User Interface*) of the application was pleasing/friendly (e.g. : *design, interface, images, colours, size of icons*) (Mark with a *cross*):

- | | | | | |
|--------------------------|----------------------------|---------------------------|--------------------------|--------------------------|
| DK/NA/REF | Disagree completely | Disagree partially | Agree partially | Agree completely |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Application

30. Do you think the loading time of the application (and among the screens) is acceptable? (Mark with a *cross*)

- Yes**
- No**

31. Would you like to include some new performances to the guide? (Mark with a *cross* which ones)

- Map Functionality**
- Museum Mini-Games**
- Other**

32. Do you think this guide will be useful for a real museum? (Mark with a *cross*)

- Yes**
- No**

33. Will you use it?

- Yes**
- No**

34. Have you ever used some application like this? (Mark with a *cross*)

- Yes**
- No**

If you have marked **Yes**, indicate which one and when:

.....
.....

35. How would you improve this application? (Mark with a *cross*)

.....
.....

Thank you for your participation.